Change Detection and Notification of Web Pages: A Survey

VIJINI MALLAWAARACHCHI, The Australian National University, Australia
LAKMAL MEEGAHAPOLA, Idiap Research Institute & École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
ROSHAN MADHUSHANKA, ERANGA HESHAN, and DULANI MEEDENIYA, University of Moratuwa, Sri Lanka
SAMPATH JAYARATHNA, Old Dominion University, USA

The majority of currently available webpages are dynamic in nature and are changing frequently. New content gets added to webpages, and existing content gets updated or deleted. Hence, people find it useful to be alert for changes in webpages that contain information that is of value to them. In the current context, keeping track of these webpages and getting alerts about different changes have become significantly challenging. Change Detection and Notification (CDN) systems were introduced to automate this monitoring process and to notify users when changes occur in webpages. This survey classifies and analyzes different aspects of CDN systems and different techniques used for each aspect. Furthermore, the survey highlights the current challenges and areas of improvement present within the field of research.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Information systems** → **Web searching and information discovery**; *Web applications*; *Document structure*; *Similarity measures*; *Information extraction*; *Relevance assessment*;

Additional Key Words and Phrases: Distributed digital collections, scheduling, change detection, change notification, webpages, websites, web search, search engine

ACM Reference format:
Vijini Mallawaarachchi, Lakmal Meegahapola, Roshan Madhushanka, Eranga Heshan, Dulani Meedeniya, and Sampath Jayarathna. 2020. Change Detection and Notification of Web Pages: A Survey. *ACM Comput. Surv.* 53, 1, Article 15 (February 2020), 35 pages.
https://doi.org/10.1145/3369876

1 INTRODUCTION

The World Wide Web (WWW or Web in simpler terms) is being evolved at a rapid pace, and keeping track of changes is becoming more challenging. Many websites are being created and updated daily with the advancement of tools and web technologies. Hence, websites at present have become more dynamic, and their content keeps changing continuously. Many users are interested

Authors' addresses: V. Mallawaarachchi, The Australian National University, Canberra ACT 0200, Australia; email: vijini.mallawaarachchi@anu.edu.au; L. Meegahapola, Idiap Research Institute, Rue Marconi 19, 1920 Martigny, Switzerland & École Polytechnique Fédérale de Lausanne (EPFL), Route Cantonale, 1015 Lausanne, Switzerland; email: lakmal.meegahapola@epfl.ch; R. Madhushanka, E. Heshan, and D. Meedeniya, University of Moratuwa, Moratuwa, Bandaranayake Mawatha, Moratuwa 10400, Sri Lanka; emails: {alwisroshan.13, eranga.13, dulanim}@cse.mrt.ac.lk; S. Jayarathna, Old Dominion University, 5115 Hampton Blvd, Norfolk, VA 23529, United States; email: sampath@cs.odu.edu.

0360-0300/2020/02-ART15 $15.00
https://doi.org/10.1145/3369876

in keeping track of the changes occurring on websites, such as news websites, booking websites, wiki pages, and blogs. Back in the 1990s, people used to register to Really Simple Syndication (RSS) feeds, originated from the Resource Description Framework (RDF) specification [97] to keep track of frequently updated content. Later in 2005, RSS was replaced by Atom [83]. Currently, the majority of the users keep track of websites and get the latest updates using bookmarks in web browsers.

Web syndication technologies (e.g., RSS and Atom) emerged as a popular means of delivering frequently updated web content on time [51]. Users can subscribe to RSS or Atom feeds, and get the latest updates. However, when considered from a perspective of webpage change detection, RSS feeds have many potential issues. A study carried out to characterize web syndication behavior and content [51] shows that the utilization of fields specified in the XML specification of RSS is less, which can result in missing information, errors and uncategorized feeds. Furthermore, services such as Google Reader have been discontinued due to the declining popularity of RSS feeds [47] caused by the rising popularity of *microblogging* (also known as *social media*), shifting of formats from XML to JSON, and market forces promoting proprietary interfaces and de-emphasizing interoperability.

In the current context, managing and using bookmarked websites and RSS feeds have become a significant challenge, and people are continuously seeking better and convenient methods. Change Detection and Notification (CDN) systems [70] make it easy for users to get notified about changes that have occurred on webpages, without having to refresh the webpage continuously. Google Alerts [45], Follow That Page [43] and Visualping [118] are some of the most popular CDN services that are used by many users to get updates about content changes that occur on webpages.

1.1 Change Detection and Notification Systems

CDN systems [73] automatically detect changes made to pages in the web, and notifies about the changes to interested parties. The significant difference between search engines and CDN systems is that search engines are developed for searching webpages, whereas CDN systems are developed for monitoring changes that occur on webpages. In theory, most of the search engines also have an underlying change-detection mechanism to determine which sites to crawl and keep their search results up-to-date [46]. The use of CDN systems allows users to reduce the browsing time, and facilitates users with the ability to check for changes on webpages of their interest [125].

CDN systems emerged in the WWW with the introduction of Mind-it (currently discontinued) [82], the first CDN tool that was developed by NetMind in 1996. Since then, several services were introduced such as ChangeDetection (introduced in 1999, now known as Visualping [118]), ChangeDetect [20] (introduced in 2002), Google Alerts [45] (introduced in 2003), Follow That Page [43] and Wachete [120]. CDN systems have evolved throughout the past few decades, with improvements in detection rates, efficient crawling mechanisms and user-friendly notification techniques.

CDN systems available at present have become easier to use, and are more flexible to incorporate user requirements. The majority of the currently available CDN systems such as Wachete [120] and VisualPing [118] provide various monitoring options, such as:

(1) Multiple webpage monitoring: multiple parts of a webpage, an entire webpage, multiple webpages or an entire website.
(2) Content to monitor: text, images, links, documents (PDF, DOC, DOCX).
(3) The threshold of changes: the percentage of changes occurring on a given webpage.
(4) Frequency of monitoring: hourly, daily, weekly, monthly or on-demand monitoring.
(5) Frequency of notification: twice a day, once a day, once a week or as changes occur.

1.2 Categories of Change Detection and Notification

Based on the architecture involved, change detection can be segregated into two main subdomains. The first branch is server-side change detection, and the other is client-side change detection [70]. Server-side change-detection systems use servers that poll webpages, track changes, and notify them to users. The client-side change-detection systems make the client-side infrastructure poll the webpages, and track changes on their own.

CDN systems obtain versions of webpages by crawling them, and saving the data to version repositories. These data are saved in an unstructured manner, mostly in the format of documents with tags, to allow easy storage and retrieval. Then, changes are detected by comparing a previously saved version with the latest version of a particular webpage using similarity computations. The majority of the change-detection mechanisms convert the data of a saved version into an XML-like format where an element represents opening and closing HTML tags (e.g., <*h*> and </*h*>). Certain CDN systems allow the user to define a threshold of change, and the user gets notified about a change, only if the change exceeds this threshold.

The majority of the CDN systems operate on predefined monitoring schedules, that are specified by the user or determined by the system itself. Detected changes are visualized using many methods. A common means of visualizing text changes is by showing the newly added text in green color, and the deleted text in red color (often with strikethrough formatting) [120].

Another prominent factor discussed in CDN systems is their crawling schedules. Most of the currently available CDN systems crawl webpages under predefine schedules [71]. However, webpages can be updated at different time schedules (certain webpages may be frequently updated, whereas some webpages may get updated rarely), thus how often they change can vary. Hence, CDN systems require mechanisms for estimating the change frequency to create efficient checking schedules for webpages. This will reduce the number of page checks where no changes were detected, and maximize the probability of achieving optimum resource utilization.

1.3 Survey Motivation

According to our study, only a limited number of surveys have been carried out regarding webpage CDN techniques. Additionally, it is challenging to find comprehensive evaluations of existing CDN systems that discuss different aspects of such systems. Oita et al. [84] have reviewed major approaches used for the change-detection process in webpages. They have reviewed temporal aspects of webpages, types of webpage changes, models used to represent webpages to facilitate change detection and various similarity metrics that are used for detecting changes. Shobhna and Chaudhary [104] discuss about a selected set of CDN systems with different types of change-detection algorithms in a summarized manner. However, there is a requirement to explore and improve CDN systems by comprehensively considering the various aspects of CDN such as the architecture, monitoring frequency, estimation of change frequency, change notification methods, and change visualization mechanisms.

Several CDN systems have been developed, and are available for public use [73]. However, we discovered that still there are issues related to these systems, and limited evaluations have been carried out. Hence, the first objective of this survey would be to deliver a comprehensive overview of the different characteristics of CDN systems. The second objective is to study existing CDN systems, and evaluate their features and various performance aspects. Our final objective is to evaluate the different aspects of CDN, study new trends and highlight the areas that require improvement. We believe that this survey can shed light on the relevant research areas, and possibly, pave the way for new scientific research fields.

The organization and the aspects discussed in this survey are summarized in Figure 1. Section 2 discusses the dynamic nature of webpages, and the experiments that have been conducted

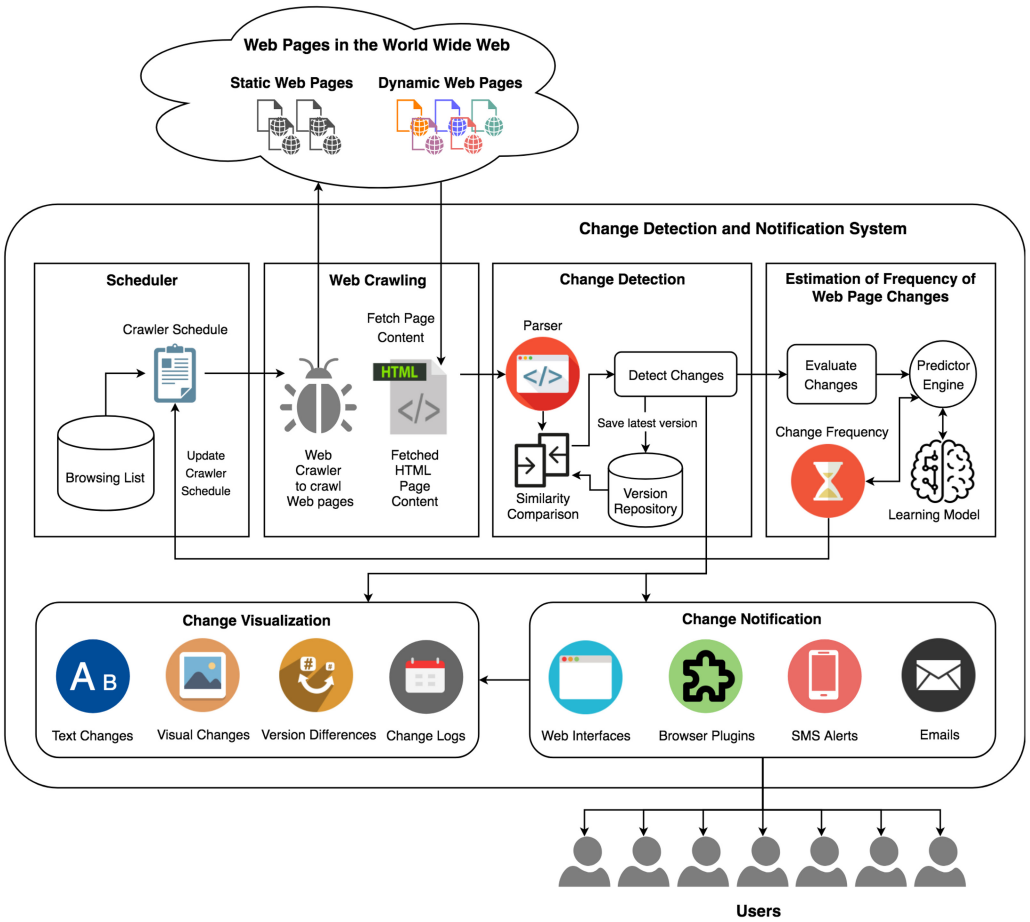


Fig. 1. Organization of the survey and the aspects discussed.

to understand the changing behavior of webpages. Section 3 presents different architectural approaches, which have been introduced to develop CDN systems. Further, various traditional architectures and several architectures that have been customized to improve the efficiency of the CDN mechanisms are presented. Section 4 confers about the techniques used for detecting changes on webpages. It includes different crawling techniques, change-detection techniques, scheduling algorithms and methods to detect the frequency of webpage changes. Section 5 presents different notification techniques to notify users when changes have been detected on webpages of their interest, whereas Section 6 describes how these changes are visualized to the user. Section 7 compares and evaluates the different features of publicly available CDN systems and Section 8 discusses current trends and issues in the field of CDN. Finally, Section 9 concludes the survey article with further improvements that can be incorporated into existing CDN systems, and presents the identified future research directions for CDN systems.

2 DYNAMICS OF WEB-BASED CONTENT

The World Wide Web (WWW) keeps on growing larger and more diverse every day as new content is being added with the advancement of web content creation tools. The most common units of information on the web are pages, documents, and resources [102]. These units can include textual

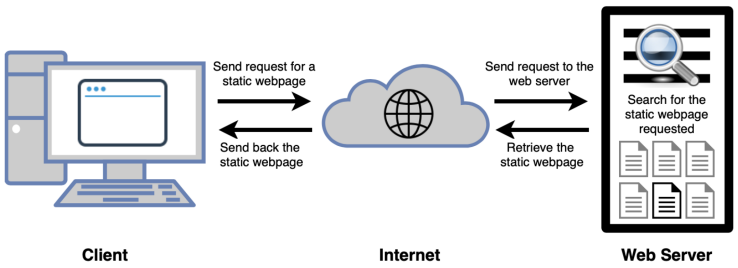


Fig. 2. Process of retrieving a static webpage.

as well as non-textual information, such as audio files, video files, and images. They can undergo various changes since the time they were added to the WWW. Hence, it is crucial to understand the changing frequency and the dynamic nature of webpages to provide efficient solutions to detect such changes.

2.1 What are Webpages and their Models of Change?

Webpages are individual files that consist of text, graphics and other features, such as links to scripts and style sheets [107]. Webpages are implemented using HyperText Markup Language (HTML) or a comparable markup language. The WWW is considered as a collection of such webpages. Webpages are linked together to form websites. Once a webpage is requested on a web browser, the web browser obtains the relevant content, coordinates the resources and presents the webpage. The web browser uses Hypertext Transfer Protocol (HTTP) to send such requests to retrieve webpages. Webpages fall into two broad categories, namely, (1) static and (2) dynamic.

2.1.1 Static Webpages. Static webpages have content that does not change after the developer has saved the webpage to the web server [60]. The webpage remains the same until the developer replaces it with an updated static webpage in the server. Static webpages are not tailored to each visitor. They appear the same to all the users who view it. Figure 2 depicts how a static webpage is displayed once the client requests it.

Static webpages can be created easily as existing web development tools allow us to drag and drop HTML elements as required. Similarly, it is easy to host, because only a web server is required to host, without requiring any additional software. Furthermore, static webpages have the advantages of fast loading and improved performance for end-users. However, if dynamic functionalities such as personalized features are required, then they have to be added separately.

2.1.2 Dynamic Webpages. Dynamic webpages are pages that do not exist until it is generated by a program in response to a request from a client while guaranteeing that the content is up-to-date [60]. Their content changes in response to different contexts or conditions. As the information obtained from the client is used to generate the webpage to be shown, it can be tailored according to the client. Figure 3 illustrates the process of generating and displaying a dynamic webpage when a request is made by a client.

Dynamic webpages pave the way for more functionality and better usability, which is not available via static webpages. They allow the creation of web applications that can be stored on a central server, and can be authorized to access from any place via a web browser. Details related to the user and context can be retrieved from external databases to tailor the webpage. Moreover, it reduces the cost and burden for updating the entire webpage whenever changes occur frequently [108]. However, dynamic webpages may face security risks as corporate applications and databases can be exposed to unintended parties. Furthermore, additional software should be installed and maintained, which is required to generate tailored websites to clients.

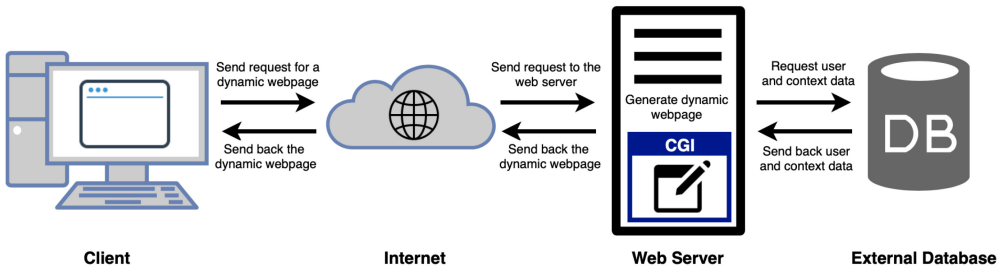


Fig. 3. Process of retrieving a dynamic webpage.

The Common Gateway Interface (CGI) [60] is the software, which runs in the server that is invoked when a client requests to retrieve a webpage. It is responsible for generating webpages dynamically. PHP, ASP.NET and JSP are some of the common languages that are often used to create webpages, and they use the CGI to generate dynamic webpages. The majority of the webpages available at present are dynamic. Dynamic webpages have become popular as many services (e.g., Content Management Systems (CMS), such as WordPress [124] and Drupal [14]) are available at present, where anyone, even a person with limited programming knowledge, can create a website with a few webpages via a control panel, update pages as required and deploy them instantly.

The content found in webpages may get outdated or maybe no longer required. Hence, timely maintenance should be done to ensure that the webpages are up-to-date. Three key events, creations, updates and deletions, are considered to cause webpages to change [4, 64].

(1) *Creations*: Once a webpage is created, it cannot be seen publicly on the Web until it is linked by an existing webpage. Hence, adding a new webpage causes at least an update of an existing one; adding a new link to the newly created webpage. However, at present, search engines such as Google provide the facility to add the Uniform Resource Locator (URL) of a newly created webpage, so that it can be indexed, and made available to the public [48].
(2) *Updates*: Updates can be made to the existing content of webpages. Updates can be of two types. The first type is a minor change, where the content of the webpage is almost the same as its previous version, but slight modifications may have been done, such as at the sentence or paragraph level. The second type is a major change, where all the content of the webpage is drastically different from its previous version.
(3) *Deletions*: A page is considered to be deleted if there are no links available to traverse to that particular page or if the page itself is removed from the Web. However, there may be instances where the webpage has been removed but still the link to that webpage exists (known as a *broken link*). Furthermore, content can be deleted from a webpage as well.

2.2 Detecting the Dynamic Nature of Webpages

Identifying the amount of change and changing patterns of webpages has been of great interest to researchers. Many studies have been carried out to understand the changing nature of webpages. The content of webpages and their change frequencies have been highly focused areas in this research scope. The available literature demonstrates the ever-changing nature of webpages and various reasons for those changes. Different factors have been considered regarding the dynamic behavior of web content.

Cho and Garcia-Molina [24] have conducted an experiment with 720,000 webpages from 270 websites to study how webpages evolve over time. They have crawled these webpages every day

for 4 months. Based on the results, the researchers have found that more than 20% of the webpages had shown changes whenever they visited them, and 40% of the webpages had changed in a week. Over 50% of the .com and .edu webpages had not changed at all during the time frame of the experiment. A massive-scale experiment that extended the study done by Cho and Garcia-Molina was performed by Fetterly et al. [40]. They have studied how frequently webpages change, and the quantity of change occurred, using approximately 150 million webpages over eleven weeks. According to their findings, approximately 65% of the pages had zero change over the considered time. Further, it shows the relationships in-between certain top-level domain types (e.g., .org, .gov, .com), and their respective frequencies of changing. It was revealed that .org and .gov domains are less likely to change than .edu domains. Furthermore, it was shown that pages that including spams would change more frequently.

Olston and Pandey [86] have crawled 10,000 web URLs from the Yahoo crawled collection and 10,000 web URLs from the Open Directory listing. According to the results, from the dynamic content on webpages of the Open Directory listing, about a third has shown a scroll behavior. Adar et al. [1] have crawled and analyzed approximately 55,000 webpages, which are revisited by users, to characterize the various changes occurring in them. The authors have tracked the frequency and amount of change that has occurred in the webpages individually. Thirty-four percent of the webpages had not shown any changes whereas the remaining pages had displayed at least one change every 123 hours. This study has shown that popular webpages change more frequently when compared to less popular webpages. Webpages falling under categories such as sports, news and personal pages change most frequently, and webpages with government and educational domain addresses have no frequent changes.

Furthermore, the work carried out by Elsas and Dumais [38] describes the temporal aspects of changing web documents. The authors describe news websites to consist of highly changing webpages. Whenever new stories are available, or the old stories are updated, the news websites would change. These changes occur in different amounts and at different frequencies. To observe how documents are changed, the authors have created a set of approximately 2,000,000 HTML pages, and crawled them every week for 10 weeks. Over the sampled period, over 62% of pages had no virtual difference. They have also pointed out that highly relevant documents are both more likely to change, and contain a higher amount of change than general documents. As the final outcome, the authors have proposed a ranked retrieval model for dynamic documents based on the temporal aspect of content that lets differential weighting of content. Work done by Saad and Gançarski [99] has monitored over 100 webpages from the France Televisions (FranceTV) archive, which depicts the evolution of changes within the French Web. Each page was crawled every hour, and over 3,000 versions were obtained daily. The results have shown that the pages at the root level of the archive, such as homepages changed significantly, whereas pages in the deepest levels did not change.

Table 1 presents a summary of the various research work carried out to detect the dynamic nature of web content. From Table 1, it can be seen that webpages belonging to popular websites such as sports and news websites tend to change frequently whereas webpages belonging to government and educational domains change less frequently. Hence, it can be concluded that webpages belonging to popular websites tend to change more frequently than those of less popular websites that target specific functions and niche audiences.

3 ARCHITECTURAL ASPECTS

Several studies have proposed different architectures for change-detection systems. The two main architectures that are being used widely within current CDN systems are server-based architecture and client-based architecture [70] and they have their advantages and disadvantages.

Table 1. A Summary of the Work Done to Detect the Dynamic Nature of Webpages

Work	Websites crawled	Pages that have changed significantly	Pages that have not changed
Cho and Garcia-Molina 2000 [24]	720,000 webpages from 270 websites	40% of the crawled webpages	Over 50% of .com and .edu webpages
Fetterly et al. 2003 [40]	150 million webpages	Webpages of .edu domain and spam	Webpages of .org and .gov domains
Adar et al. 2009 [1]	55,000 webpages	Popular webpages (e.g., sports, news, etc.)	Webpages of .gov and .edu domains
Elsas and Dumais 2010 [38]	2,000,000 HTML pages	Highly relevant documents	62% of the crawled webpages
Saad and Gançarski 2012 [99]	100 webpages from FranceTV archive	Webpages at the root level of the archive	Webpages in deepest levels of the archive

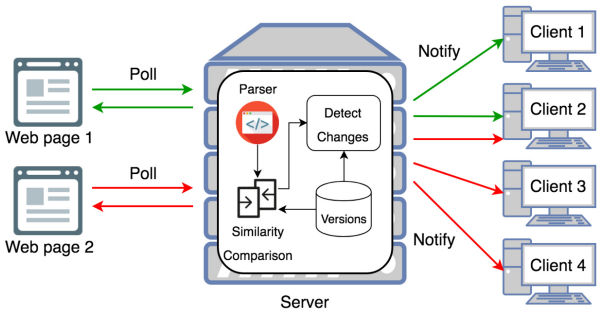


Fig. 4. Server-based architecture for CDN systems.

3.1 Server-based Architecture

The server-based architecture, as depicted in Figure 4 consists of the main server, which polls webpages periodically to track changes, and sends alerts about these changes to the subscribed users (clients) by email notifications. If a large number of webpages exist, then the computational load for the server will increase as the server must identify changes in each of the webpages added by users. This can also lead to reduced detection frequencies. The process of scaling such tools with the server-based architecture becomes expensive and makes the system less efficient. Due to these issues, the frequency in which changes are detected on webpages will not be sufficient and the server might fail to observe some changes that have occurred in frequently-changing webpages.

Sitemaps [106] is used by servers to inform search engines about the changes that it thinks are important, and are made available for crawling. This allows crawlers to identify webpages that have changed recently without having to download and process HTML pages [11]. Support for Sitemaps protocol by the search engine industry was announced by Google, Yahoo and Microsoft in the year 2006 [101]. Since then, many websites, such as Amazon, CNN and Wikipedia, have begun supporting Sitemaps. Studies have shown that the use of sitemaps has improved crawling results [101]. Data from Sitemaps can be used by CDN systems to get updated content. The use of Sitemaps helps to eliminate difficulties faced by crawlers and expose data regarding changes only.

3.2 Client-based Architecture

The client-based architecture involves clients who wish to track changes occurring on webpages, and these machines poll webpages in iterations to identify changes. Users having extra

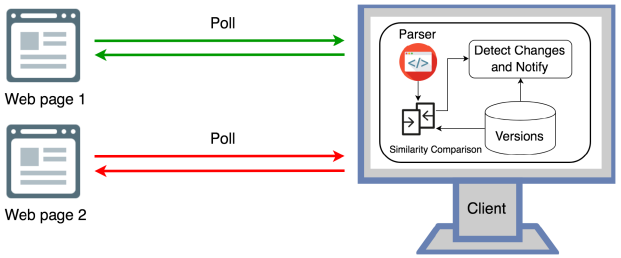


Fig. 5. Client-based architecture for CDN systems.

computational resources can detect frequent changes occurring on webpages, and the other users might be unable to do so. The client-based architecture has been implemented in the form of browser plugins, and hence, may get bottlenecks due to network connectivity and machine performance. Figure 5 illustrates the client-based architecture for a CDN system.

Out of the currently available CDN systems, a limited number of them are built using the client-based architecture. They come in the form of browser plugins/extensions, such as Distill [34] and Check4Change [128]. Once the extension is installed and enabled successfully, the user can add webpages to monitor. It will regularly check the webpages the user has added to be monitored, within the user's browser. If any changes are detected, then the user will get a browser notification.

3.3 Customized Architectures

Several customized architectures of CDN can be found in which researchers have tried to improve the efficiency from an architectural perspective. A design to crawl webpages in parallel using several machines, and integrate the problems with crawling has been proposed by Yadav et al. [125]. They have designed a new architecture to build a parallel crawler. The proposed architecture consists of two main components. They are the Multi-threaded Server (MT-Server) and the client crawlers. MT-Server is the principal component that manages a collection of client machine connections, which downloads and crawls webpages. It has a special methodology to distribute URLs among the clients when their "priority index" is determined. The client crawlers are considered as the different machines that interact with the server. The client count that can be supported may vary according to the available resources and their implementation. Furthermore, there exists no communication between clients, and the clients interact only with the server. However, scaling the system can result in high costs as the count of server nodes has to grow.

Work carried out by Prieto et al. [94] presents a system with a collaborative architecture to detect changes in distributed systems. The authors have proposed the Web Change Detection system (WCD). The four major components of this system's architecture are the web client, web server, WCD agent, and WCD server. Web client is a general browser of the web, which loads the webpages. Web server is a general server that caches the webpages that were monitored. WCD agent is an application operating in the browser that sends information about modifications that have occurred on webpages to the WCD server. WCD server stores and sends the WCD agents to clients, and stores the data about monitored webpages. To detect the near-duplicates, PageRank [89] values have been considered along with the shash tool [29]. High change-detection rates and a low cost of maintenance have been produced by this tool. However, in times of excessive usage, if the system gets many requests, then it may fail to process them in real-time.

An "approach for distributed aggregation, faster scalable content classification and change detection in web content" has been proposed by Nadaraj [81]. The author has presented an algorithm to determine webpage changes, and a method to identify what field has been altered with the help of a data structure named Bloom filters [7]. Web crawlers are being used to collect details about

Table 2. A Summary of the Architectures Used/proposed for CDN Systems

Architecture	Advantage	Disadvantage	Inter-Client Communication	Parallel Processing	Cost
Server-based [70]	Centralized monitoring	High load on the server	No	No	High
Client-based [70]	Clients have control	The network bottleneck	No	No	Medium
Parallel crawling [125]	Prioritize URLs to crawl	High overheads in communication	No	Yes	High
Collaborative WCD [94]	Operate in distributed networks	Can fail to process requests in real-time	No	No	Medium
Distributed aggregation [81]	Distributes the workload	Overhead of hashing	Yes	Yes	Low
Hybrid [70, 73]	Optimal use of client resources	Need to co- ordinate clients	Yes	Yes	Low

pages and URLs. It uses Bloom filters to identify the duplicate URLs in a webpage. Hash keys for every visited URL are saved in the bloom filter. Bloom filter entries will be validated when new URLs are discovered. This prevents the crawling mechanism from repeating in loops. The system creates a hash key for the content that has been crawled, and checks the presence of the hash within the Bloom lookup. If present, then the content is the same as the existing content; otherwise, the content has been updated. If the hash key for the URL is not found, then the URL is added to the Bloom filter lookup file, and a hash for the crawled content is created and inserted. This method increases the efficiency of the crawling process as it distributes the workload. Furthermore, the strong association of crawlers to working machines will be minimized, and the crawlers will be able to function without any restrictions in distributed networks.

A hybrid architecture for CDN is proposed by Meegahapola et al. [70, 73]. This architecture is a hybrid of server-based and client-based architectures. It has two independent crawling schedules for the server and the clients. The server will crawl all the webpages it has registered, and the clients will crawl the webpages that they want to track. The change-detection process occurs independently in the server and clients. If the server detects a change, then it will be notified to the interested clients. If a change is detected by a client, then it will directly report back to the server, and the server will notify the interested clients. According to this architecture, the time elapsed in between two consecutive server poll actions is divided among the available clients, which in turn speeds up the detection process. Table 2 presents a summary of the various architectures that are being used by commercially available CDN systems and that have been proposed by researchers.

4 DETECTING CHANGES OF WEBPAGES

4.1 Web Crawlers and Crawling Techniques

A web crawler (also known as a spider) is "a software or a programmed script that browses the WWW in a systematic, automated manner" [58], and systematically downloads numerous webpages starting from a seed URL [9]. Web crawlers date back to the 1990s, where they were introduced when the WWW was invented. The World Wide Web Worm [68] and MOMspider [41] were among the early web crawlers. Moreover, commercial search engines developed their own web crawlers as well, such as Google crawler [13] and AltaVista [105]. Later on, web crawlers that could efficiently download millions of webpages were built.

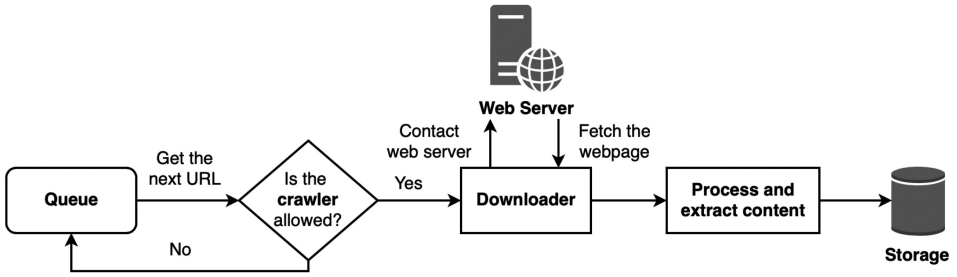


Fig. 6. Overview of the web crawling process.

We can consider the Internet to be a "directed graph" where each node represents a webpage, and the edges represent hyperlinks connecting these webpages [58]. Web crawlers traverse over this graph-like structure of the Internet, go to webpages, and download their content for indexing purposes. It is crucial to identify which crawling technique should be used according to the purpose of the application.

Web crawling can be considered as the main process behind web caching, search engines and web information retrieval. A web crawler begins crawling from a seed URL and visits pages. Then it downloads the page, and retrieves the URLs in it. The discovered URLs will be kept in a queue, and this process repeats as the crawler travels from page to page. Figure 6 shows an overview of the web crawling process. Many crawling techniques are being used by web crawlers at present [58], such as (1) general-purpose crawling, (2) focused crawling, and (3) distributed crawling.

4.1.1 General-purpose Crawling. In the general-purpose crawling technique, the web crawlers collect all the possible webpages from a set of URLs and their links to a central location. It can fetch numerous pages from many different locations. However, this technique can slow down the crawling process and reduce the network bandwidth as all the pages are being fetched.

4.1.2 Focused Crawling. Focused crawling is used to collect pages and documents that belong to a specific topic. This can reduce the workload and network traffic during download as the required pages are only downloaded. It crawls only the relevant regions of the Web. This method saves hardware and network resources significantly.

Initially, focused crawling was introduced by Chakrabarti et al. [18] as "a new approach to topic-specific Web resource discovery." The focused crawler has three major components. They are as follows.

(1) "Classifier decides whether to expand links on the webpages crawled."
(2) "Distiller determines visit priorities of crawled pages."
(3) "Crawler consists of dynamically reconfigurable priority controls, which are controlled by the classifier and distiller" [18].

Diligenti et al. [32] have highlighted the importance of assigning credits to different documents across crawling paths that produce larger sets of topically relevant pages. The authors have proposed a focused crawling algorithm with a model that captures link hierarchies containing relevant pages. Later on, Menczer et al. [74] have discussed different ways to compare topic-driven crawling techniques, whereas "a general framework to evaluate topical crawlers" was presented by Srinivasan et al. [109]. Pant and Menczer [92] have introduced a topical crawling technique to gather documents related to business intelligence, which can support organizations to identify competitors, partners or acquisitions. The authors have tested four crawlers; Breadth-First, Naive Best-First, DOM, and Hub-Seeking. The results of precision and recall on a given topic show

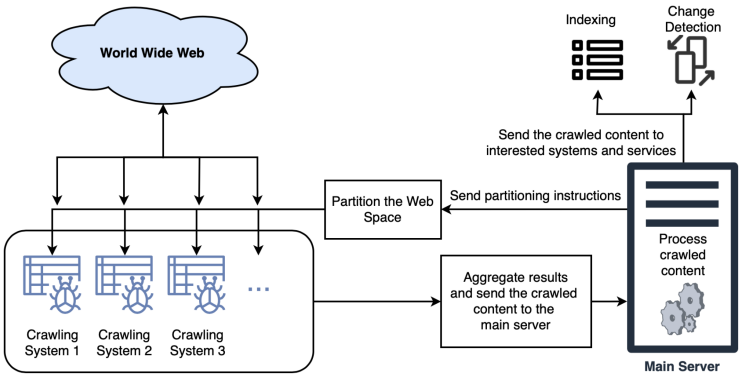


Fig. 7. Arrangement of components in a distributed crawling system.

that the Hub-Seeking crawler outperforms other crawlers. A popular technique to design-focused crawlers is the utilization of the "link structure" of the documents. Li et al. [65] have proposed a focused crawling technique using a decision tree created by anchor texts found in hyperlinks. Jamali et al. [54] have presented a novel "hybrid focused crawler," which utilizes the "link structure" and similarity of the content to a particular topic.

Mali and Meshram [67] have proposed another approach for a web crawler with focused crawling features. Three layers are present in this architecture; "page relevance computation," "determination of page change," and "updating the URL repository." During the crawling mechanism, all the pages are not downloaded. Instead, it extracts the URLs and the words of interest. Frequency of related words and the number of forward links and backward links to and from the webpage collaboratively decide the importance of that webpage being parsed. Certain parameters, such as topic vectors and relevance scores, are used to check the importance of the page. If the relevance level exceeds a predefined threshold, then it is downloaded for extraction. Otherwise, the page will be discarded.

Work done by Bhatt et al. [6] has studied focused web crawlers with their advantages and disadvantages. Additionally, the authors have suggested methods to further improve the efficiency of web crawlers. With the advancement in optimization algorithms, researchers have turned their focus on optimizing web crawlers. Optimizations allow web crawlers to select more suitable webpages to be fetched. Focused crawlers have been optimized to increase the efficiency and performance of the crawler using search algorithms such as Breadth-First Search [6], Fish-Search [31], and Shark-Search [50] and evolutionary algorithms such as Genetic algorithms [126, 127] and Ant algorithms [127].

4.1.3 Distributed Crawling. This method uses multiple processes to crawl webpages and download their content. Web crawlers are distributed over different systems, allowing them to operate independently. Distributed crawling techniques were introduced due to the inherent problems faced by centralized crawling solutions, such as reduced throughput of the crawl and link congestion [59].

Figure 7 denotes the general arrangement of components in a distributed crawling system. The crawlers are distributed across different systems, and they crawl webpages in different parts of the Web. The Web can be partitioned using different graph partitioning algorithms, and each crawler is provided with a seed URL to start the crawling process [2]. All the crawled content is then aggregated and sent to the main server. This content can be processed within the main server or they can be sent to other services for different purposes, such as indexing and version comparison

Table 3. A Summary of the Main Crawling Techniques

Crawling Technique	Crawling Pattern	Prioritize links	Partition link space	Advantage	Disadvantage
General purpose [58]	Crawl all the available webpages by going through link URLs	No	No	Can fetch pages from many different locations	Affects the network bandwidth
Focused crawling [18]	Crawl pages belonging to a given topic	Yes	No	Can retrieve content related to a given topic	Becomes more time consuming as the topics expand
Distributed crawling [59]	Crawlers are distributed across systems independently	Varies	Yes	Provide more throughput	Difficult to manage and coordinate crawlers

within CDN systems. One of the most popular high-performance distributed crawlers found at present is the Googlebot [46]. Googlebot is the web crawler used by Google to fetch webpages for indexing. It is designed to be distributed on several machines so that the crawler can scale as the Web grows. Table 3 summarizes the three main crawling techniques with their advantages, disadvantages and comparison of features.

Detection of crashed agents is one of the important concerns of distributed web crawlers. Several distributed crawler solutions have been presented during the past [8, 103]. To address this concern, crawlers can be designed as reliable failure detectors [19], in which timeouts can be used to detect crashed agents. UbiCrawler [8] is an example of a web crawler with a reliable failure detector. It is a distributed web crawler, which consists of several agents that are responsible to crawl their own share of the Web. However, it does not guarantee that the same webpage is not visited more than once. Based on the experience with UbiCrawler, the authors have built BUbiNG [9], a fully distributed web crawler that can detect (near-)duplicate webpages based on the content. However, it does not support URL prioritization.

Another aspect that has drawn the attention of researchers is the efficient partitioning mechanisms of the Web space. Work done by Exposto et al. [39] has presented a multi-objective approach for partitioning the Web space by modeling the Web hosts and IP hosts as graphs. These graphs are partitioned, and a new graph is created with the weights calculated using the original weights and the edge-cuts. Results show that efficient partitioning techniques have improved download times, exchange times and relocation times.

Kc et al. [59] have introduced LiDi Crawl (which stands for Lightweight Distributed Crawler), which is a centralized crawling application with limited resources. It consists of a central node and several individual crawling components. The distributed crawling nature results in the reduced dependence on expensive resources. Kumar and Neelima [63] have proposed a scalable, fully-distributed web crawler, without a central node. It consists of multiple agents, where each agent is coordinated so that they crawl their own region on the Web. An agent runs several threads, where each thread is made responsible to visit a single host. The main objective of having multiple agents is to break down the task of crawling into separate parts, and allow specialized modules to carry them out efficiently. Anjum et al. [3] state that web crawlers should be aware of webpage modifications, and have discussed techniques to retrieve information on such modifications. However, the authors have found that the presence of multiple JavaScript and CSS files can reduce the efficiency of certain retrieval techniques.

Efficient crawling of Rich Internet Applications (RIA) is an open problem as RIAs consist of many characteristics, such as the use of JavaScript and browser plugins, which makes the crawling

Table 4. A Summary of Various Crawling Solutions Presented in the Literature

Crawler	Distributed	Advantage	Disadvantage
UbiCrawler [8]	Yes	Failure detection	Does not guarantee that duplications are not present
BUbiNG [9]	Yes	No coupling to machines	Does not allow prioritization of URLs
LiDi Crawl [59]	No	Can work with limited resources	Affected by a single point of failure
Dist-RIA crawler [77]	Yes	Can crawl Rich Internet Applications in parallel	Lack of adaptive load-balancing for diverse nodes
MTCCDW [69]	No	Optimized for the process of change detection	Affected by a single point of failure

process complex. Model-based Crawling [5] was introduced to determine an optimal crawling strategy for RIA. An extension of this model is presented by Dincturk et al. [33], which uses a statistical approach in determining a crawling strategy. The recent work carried out by Mirtaheri et al. [77], intends to lower the time taken to crawl RIAs by introducing Dist-RIA crawler, which is a distributed crawler to crawl RIAs in parallel. However, it assumes that all the nodes have equal processing power, and assigns an equal number of tasks to each node. This can be problematic when there is heterogeneity.

Multi-Threaded Crawler for Change Detection of Web (MTCCDW) has been introduced by Meegahapola et al. [69] to optimize the change-detection process of webpages. Many threads were used to carry out the tasks of (1) "retrieving current versions of webpages," (2) "retrieving old versions from a version repository," and (3) "comparison of the two versions to detect the changes" [69]. Two thread-safe queues were used in between these three tasks to optimize them. The authors have determined the optimum number of threads to be used in each of these tasks, to ensure that the CDN system works optimally without idling. This multi-threading-based algorithm differs from standard process optimization tasks because of the I/O operations that are involved in fetching a webpage from the Web. Hence, this algorithm provides a more significant improvement in processing times over doing I/O operations and processing in a central processing unit. Table 4 summarizes the various crawling solutions presented in this subsection.

4.2 Scheduling Algorithms

Among the important research problems observed in the web information retrieval domain, the scheduling process of visiting webpages along hierarchies of links has become significant. The frequency of change may differ in different webpages as they get updated at different time schedules. Certain webpages, such as wiki pages, may get updated rarely whereas news websites and blogs may get updated more frequently. To crawl these webpages more efficiently, dynamic mechanisms are required to detect the change frequency, and create checking schedules accordingly. Users will be notified immediately after changes have occurred on webpages that they are interested in. This will ensure computational resources are used optimally. However, most of these algorithms are kept proprietary, and a limited amount of details are published about them to prevent being reproduced [17].

Various solutions have been proposed to determine optimal webpage crawling schedules based on different assumptions and different statistical frameworks. Work done by Coffman et al. [28] has described crawler scheduling policies by assuming independent Poisson page-change processes.

Table 5. A Summary of Various Scheduling Solutions

Scheduling Solution	Advantage	Limitations
Coffman et al. 1998 [28]	Schedule crawls when webpages change and reduce unnecessary crawls	Can be sensitive to parameter changes in the Poisson process
Cho and Garcia-Molina 2000 [23]	Significantly improves the freshness of webpage indexes	Affected when resource synchronization is not uniform
Wolf et al. 2002 [123]	Minimizes the number of non-relevant webpages for a user query	Not suitable to be used online with large input dimensions
Pandey and Olston 2005 [91]	Improves the quality of user experience with fewer resources	Does not revisit webpages based on change frequency
Santos et al. 2013 [100]	Prioritizes crawls according to how frequently webpages change	Has not considered other features such as PageRank and crawling cost

Studies carried out by Cho and Garcia-Molina [23] have addressed the problem of determining the optimal number of crawls per page by using a staleness metric and Poisson process, where they assume uniform synchronization over time. Work done by Wolf et al. [123] has proposed a technique to detect optimal crawl frequencies and theoretical optimal times to crawl webpages based on probability, resource allocation theory and network flow theory. Pandey and Olston [91] have proposed a scheduling policy based on the "user-centric search repository quality."

Various scheduling strategies for web crawling have been studied and compared in previous studies [17, 42]. Among these strategies are optimal, depth, length, batch, partial [17], depth-first search, breadth-first search, best-first search and best n-first search [42]. However, some of the strategies that have been considered cannot determine the temporal aspects, such as how frequently webpages are undergoing changes, and prioritize the crawls accordingly. Evolutionary programming techniques, such as genetic programming [100], can be used to solve this issue by facilitating schedulers to rank webpages according to how likely they are being modified. Moreover, certain algorithms may get affected as webpages are crawled even though nothing has changed, where computational resources are used inefficiently. Table 5 compares the various scheduling solutions presented previously.

4.3 Change-detection Algorithms

Changes occurring on webpages can be divided into five categories as discussed by Oita and Senellart [84]. They are (1) content changes, (2) structural (or layout) changes, (3) attribute (or presentation) changes, (4) behavioural changes and (5) type changes. Content changes include changes occurring in the text, images, etc., whereas structural (or layout) changes consist of changes occurring to the placement of HTML tags. Attribute (or presentation) changes include changes done to the design and presentation of a webpage, such as changes in the fonts, colors or captions. Behavioral changes occur when active components, such as embedded applications and scripts of webpages, are changed. Type changes occur when modifications are done to the existing HTML tags, such as when a *p* tag becomes a *div* tag. Studies have been focused on all of these types of changes [84] and different methods and algorithms have been proposed [10, 53].

A major concern of change detection in webpages is the ability to identify relevant and irrelevant changes. The research carried out by Borgolte et al. [10] has focused to ignore irrelevant changes,

such as change of advertisements, and try to detect important changes occurring on webpages using different methods. The Delta framework introduced in this research, consists of four precise steps. In the initial step, it retrieves the currently available version of a website, and normalizes the DOM tree. Then in the next step, similarities are measured in comparison to the base version of the website. Similar changes are clustered to lower the analysis engine submissions. The final step is the "analysis of identified and novel modifications with a potential computationally costly analysis engine." Compared to many other pieces of research that focus on detecting changes that have been done to a website, the Delta framework focuses more on detecting significant changes occurring on a website.

Changes occurring within information collections can be difficult to track due to the vast amount of data being available. Unexpected changes within the collection can make the content unorganized and outdated, which can cause burdens, such as the removal of outdated resources and replacement for lost resources. Jayarathna and Poursardar [55] have presented a study that focuses on a categorization and classification framework to manage and curate distributed collections of web-based resources. The selected pages were categorized with their relationships between the anchor text to identify the relevant information of the target page. They have proposed a digital collection manager that addresses webpages with textual content only. It has been shown that due to unexpected changes, documents in these collections can get problematic. Further research is necessary to detect changes occurring to other resources in digital collections, such as audio files, video files and documents.

Various change-detection approaches have been proposed across the available literature to detect the different changes occurring on webpages. Elaborated in Table 6 are a few popular algorithms that were considered in this survey. One of the common approaches that have been used for change detection in hierarchically structured documents such as webpages is the use of tree-structured data with *diff* algorithms [27]. Many variations of these diff algorithms have been presented in the available literature [10, 122]. The basic idea on top of which all of these algorithms are built on is the process of modeling the hierarchically structured documents in the form of a tree with nodes, and compare the trees to identify changes that have occurred. Similarly, two trees can be developed; one with nodes from the parsed content of a previous version and the other with nodes from the parsed content of the currently available version. The two trees could be compared to identify which nodes have changed so that the changes can be determined.

4.3.1 Fuzzy Tree Difference Algorithm. The Delta framework proposed in Reference [10] involves tree difference algorithms. First, the modifications that have occurred in a website are extracted using fuzzy tree difference algorithms. Second, a machine learning algorithm is used to cluster similar modifications. The authors have employed a tree difference algorithm in which they have generalized to remove irrelevant modifications during the early phases of the analysis.

4.3.2 BULD Diff Algorithm. The BULD Diff algorithm [27] has been used in computing the differences among the given two XML documents. It matches nodes in the two XML documents, and constructs a delta that represents the changes between the two compared documents. BULD stands for "Bottom-Up, Lazy-Down" propagation [27], which has been derived from its matchings that are propagated "bottom-up" and "lazily down." This algorithm can run in linear time. First, the algorithm matches the largest identical parts of both the XML documents. Then the algorithm tries to match subtrees and the parents of the matched subtrees. This process is repeated until no more matches are made. Then the remaining unmatched nodes can be identified as insertions or deletions.

Table 6. A Summary of Existing Change-detection Algorithms

Algorithm	Methodology/Function	Advantages	Disadvantages	Associated Work
Shingling algorithm	Group a sequence of terms in a given document and encode by a 64-bit Rabin fingerprint, referred as a "shingle". Use the Jaccard coefficient between the shingle vectors to compute the similarity between the two documents.	Fast detection due to the speed of the algorithm.	If the content considered in the document is small, then it will not be able to generate enough shingles.	Detecting duplicate web documents using click- through data [95]
Johnson's algorithm	Computes the distance between two documents based on paragraphs, headings and keywords. Each signature difference is calculated, and the sum is taken as the total distance.	Categorizes the change type. Easy identification of the mostly changed content type.	Does not include links when determining changes.	Walden's Paths Path Manager [44]
Proportional algorithm	Computes the distance that is normalized and symmetric using each individual signature. Proportional change of each signature is used with regards to the total number of changes.	Can evaluate changes efficiently in the page without analyzing all the related pages in a given path.	There is a slight performance trade-off when compared to Johnson's Algorithm.	Walden's Paths Path Manager [44]
Cosine algorithm	Compute a cosine value between two vectors of the page considered and all the pages for the similar topic except the page considered. Then measure the change in cosine value.	Provides a more meaningful cut-off to identify the level of change.	The context-based algorithm gives a mid- level outcome as it tends to generate false positives and false negatives.	Managing distributed collections [30]
Fuzzy tree difference algorithm	A tree difference algorithm that is generalized into a fuzzy-notion.	Can eliminate trivial changes momentarily.	Have to define a suitable (fuzzy) hash function.	Delta framework [10]
BULD Diff algorithm	Match subtrees of two XML documents till no more matches can be made. Unmatched subtrees are considered to be insertions or deletions.	The difference can be computed in linear time.	Reduce performance proportionally to the tree-depth.	Detect changes in XML documents [27]
CX-DIFF	Identifies user specific changes on XML documents. Objects of interest are extracted, filtered for unique insertions and deletions, and the common order subsequence is computed.	Gives notifications to users on time. Optimized space usage. Efficient monitoring of types.	Can lead to overloading of the servers due to the highly expensive computational cost.	WebVigil [52]
X-Diff	Compares two XML documents depending on their equivalent trees. It generates a "minimum-cost edit script" including a series of fundamental edit operations that transform a given tree to the other at a minimum cost.	Provides accurate results by maintaining an unordered view of the document where ancestor links add value to the result.	May require a large amount of statistical information for the detection process.	X-Diff [122]
Vi-DIFF	Detects content and structural changes including the visual representation of webpages.	Can detect semantic differences between two versions.	May cost more resources and time when compared to other methods.	Vi-DIFF [93]
Level order traversal	This is a breadth first traversal algorithm and considers the changes in the document-tree to detect the changes.	Simple. Low cost of computation. Helps to reduce network traffic due to the usage of HTTP metadata.	May not give sufficient information to the user.	Change Detection in Webpages [125]

4.3.3 X-Diff Algorithm. The research carried out by Wang et al. [122] has investigated how XML documents change and methods to perform change detection of XML documents. The authors have pointed out the fact that XML is becoming important as it is at present the de-facto standard in publishing web applications and data transfer/transportation. Internet query systems, search engines and continuous query systems heavily rely on efficiently detecting webpage changes in XML-documents because of the frequent rate at which webpages change in present days. The authors have described an algorithm named X-Diff [122], which is an algorithmic model used to compute the difference in-between two versions of an XML-document. Unordered trees, structural information of XML-documents and high performance are among the main features of X-diff algorithm. The authors have tested the X-Diff algorithm for its accuracy and efficiency of change detection by considering three main nodes in the DOM tree, namely, element nodes, text nodes, and attribute nodes. Going beyond what has currently being done, the authors have compared the ordered tree model of change detection and unordered tree model that they have used. They have also discussed the characteristics of XML domain, and established few key concepts, such as "node signature" and "XHash."

4.3.4 Tree Difference Algorithms. The research carried out by Jain and Khandagale [53], has focused on detecting changes in a specific location on a website or any document. This method involves tree comparison techniques. The majority of the existing techniques/systems check for changes in the whole webpage without allowing the user to select specific areas to monitor. When considering frequent changes, the cost of communication (or the information exchange) will cause inefficiencies by disturbing the focus to a given context. Hence, the authors have proposed a tree comparison mechanism to overcome these difficulties. They have considered the management of zone changes (changes in a particular place on a webpage), and it is quite achievable using a tree mechanism, because once a webpage is converted into XML format, it can be converted into a tree according to the HTML tags and attributes. When it has to localize the detection mechanism, the focus is given only to a particular node, and the detection process will continue to their child nodes. However, the limit for the depth of the tree is not specified, which can be inefficient with large trees.

Yadav et al. [125] describe a methodology to build a model to efficiently monitor webpage changes. The authors have suggested an algorithm that efficiently identifies and extracts the changes on various versions of a particular webpage, and evaluates the significance/importance of the change. This algorithmic model has three main sections. Initially, a document-tree is created for the webpage, and then it encodes the tree. Finally, it matches the current version with the previous version of the encoded tree. The algorithm searches for two change types occurring in web content. They are, namely, structural changes and content changes.

Tree models can be of two types: the first type is the ordered tree model where the left-to-right order between nodes is crucial, and this order directly contributes to the result of change detection; the second type is unordered tree model, where only the ancestor relationships are significant. Different studies have been carried out using these two tree models. Level Order Traversal [125] is a form of breadth-first search that uses an ordered tree model. First, it constructs a document tree by taking in an HTML file and parses the elements. Then, the opening tags are identified as tree nodes, and the tree is constructed for the webpage as illustrated in Figure 8, while maintaining parent-child relationships and left-to-right order in-between siblings. Then the algorithm traverses through the tree to detect changes, and identify at which level of the tree the changes have occurred. However, some researches [122] have argued that the unordered tree model can generate more accurate results.

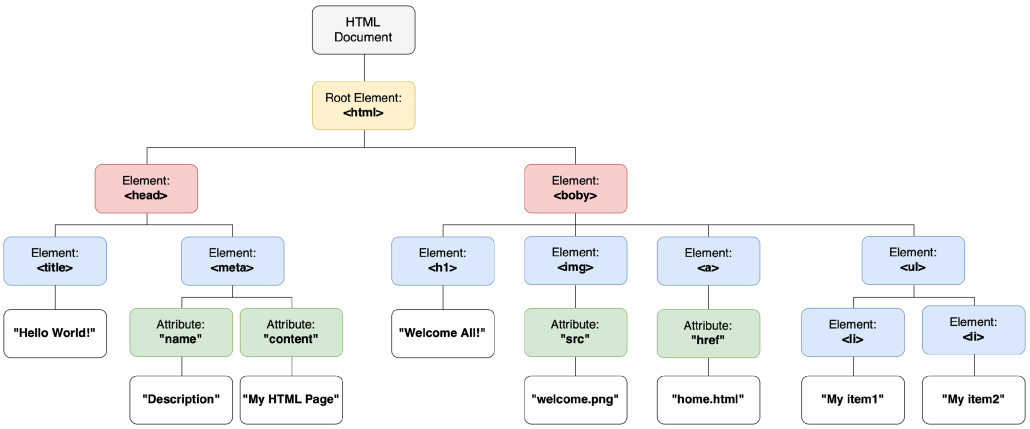


Fig. 8. HTML document tree hierarchy of a sample webpage.

4.3.5 Johnson's Algorithm. The Johnson's algorithm [56] was originally proposed as a method to "find the shortest paths between all the pairs of vertices in a sparse weighted directed graph." The same algorithm has been introduced by Johnson and Tanimoto [57] to detect changes to content. The authors have tested a prototype to manage and deliver the latest tutorial content on the web to students. The system was designed to anticipate changes to documents. The difference between documents stored is evaluated and updated accordingly. This algorithm has been used in a tool named as Walden's Paths Path Manager [44]. It computes the distance between two documents based on paragraphs, headings and keywords. Each signature difference is calculated, and the sum is taken as the total distance. It categorizes the change type, and it is easy to identify which type of content is mostly changed on a webpage. However, it does not consider the changes that occur to links. Furthermore, the results produced by this algorithm can be hard for a normal user to understand without the assistance of a computing device.

4.3.6 Proportional Algorithm. The Proportional algorithm [44] is based on signatures, and gives a much simple calculation of the distance. It computes a distance that is normalized and symmetric using each signature (paragraphs, headings, keywords and links). The proportional change of each signature is used with regards to the total number of changes. The measured distance has properties such as normalized properties and symmetric properties. These properties help the user of the algorithm significantly by providing a listing or visualization of various webpage changes. Having such listings, and visualizations makes it possible for the user to easily analyze changes of the webpage without necessarily reading and reviewing all the pages in the path. However, there is a slight performance trade-off when compared to the Johnson's Algorithm as changes to each signature are computed individually.

Table 6 summarizes the change-detection algorithms discussed in this subsection, and compares the methodology/functions, advantages, disadvantages and associated research work. According to Table 6, it can be observed that there are many change-detection algorithms based on difference calculation and traversal techniques. Different algorithms detect different changes, such as changes occurring to text, visual representation and XML structure. The majority of the algorithms perform at fairly good speeds, and provide faster detection rates along with efficient resource utilization and low computational costs. Algorithms such as level order traversal have a low overhead, which will in return reduce the network traffic in communication. Algorithms such as the Cosine algorithm and the Fuzzy Tree Difference algorithm provide different levels of changes so that the

threshold can be decided upon the specific usage of the algorithms. Algorithms such as the Johnson's algorithm can be used to categorize the change types so that it is easy to identify which type of content is mostly changed in a webpage.

However, there are certain shortcomings in each of these algorithms. If the content being compared is very small, then the Shingling algorithm will not be able to generate sufficient shingles. The Johnson's algorithm does not identify changes occurring in links. Algorithms such as CX-DIFF and Vi-DIFF consist of highly expensive computations, and can even lead to overloading of the servers. Level order traversal algorithms do not provide sufficient information to the user about changes occurring.

4.4 Frequency of Webpage Changes

Since the 1970s, several studies have been carried out based on statistics, to estimate the change frequency [15, 78]. Nevertheless, the majority of these studies have been done under the assumption that the history of changes is available for the webpage, which might not be true always in the area of change detection of webpages. When analyzing CDN systems, it is visible that the complete change histories will not be available for a webpage being tracked. In several related studies [40, 49, 86], the Poisson model has been introduced as a model that can be used to estimate the rate of change of a webpage (also known as change frequency). Most of the work that has been carried out is with an assumption: "changes arrive as a Poisson process, and that the average rate of change can be estimated under this model." Brewington and Cybenko [12] have used an exponential probabilistic model to infer the times between changes occurring in individual webpages. The ages of webpages that have gone through many changes over a time period can be closely modeled using an exponential distribution. However, it models all the webpages as dynamic, even if the webpages change rarely and their only changes are their removal from the Web.

According to Olston and Najork [85], the binary freshness model can be used to measure the freshness in a webpage. This model, which is also known as obsolescence, is a function that is of the form

$$f(p,t) \in \{0,1\} \tag{1}$$

where $f(p,t)$ denotes the freshness of a particular webpage p over a t time period. It compares the live copy of a specific webpage p with a cached version of the webpage across a time period t to check if they are identical (or near-identical). Under this model, if $f(p,t)$ equals to one, then the webpage p can be called fresh over the time t, whereas otherwise it is considered as a stale webpage that has gone through some change. This model is simple, but effective, and provides readers with a very good understanding of webpage changes. The first most study regarding the freshness maximization problem was done by Coffman et al. [28], where the authors have proposed a Poisson model for webpage changes. A set of random and independent events that occur in a fixed-rate can be modeled using the Poisson Process. A webpage can undergo changes that cause the cached copy of the web crawler to go stale. If $\lambda(p)$ is the rate parameter of a Poisson distribution where p is the webpage, then that specific distribution can be used to denote the occurrence of changes in that specific webpage. This also suggests that the changes happen independently and randomly with a mean rate of $\lambda(p)$ changes per unit of time.

However, the binary freshness model lacks the ability to determine whether one page is fresher than the other, since the model outputs a binary value; fresh or stale. Hence, Cho and Garcia-Molina [25] have introduced a non-binary freshness model known as the temporal freshness metric, which is

$$f(p,t) \propto age(p,t) \tag{2}$$

in which $age(p, t)$ represents the age of a page p up to time t and $age\ (p, t) \in \{0, a\}$ where a is the time duration the copies differed. If a cached-copy of webpage p is indistinguishable from its live-copy, then $age(p, t) = 0$. The intuition of this methodology is that "the more time a cached page stays unsynchronized with its live-copy, the more their content tends to drift away."

Cho and Garcia-Molina [26] have proposed several frequency estimators for different online applications that require frequency estimation with different accuracy levels. The authors have proposed frequency estimators for scenarios, where the existence of a change is known, and the last date of change is known. Furthermore, the authors have proposed a model to categorize elements into different classes based on their change frequencies. These aspects can be modeled in CDN systems to categorize webpages once their change frequencies have been estimated.

Grimes and O'Brien [49] state that for every webpage, the hourly update rate can be represented by a Poisson distribution together with $\lambda = 1/\Delta$ where λ is a parameter and Δ is the mean time between changes. The authors have also described a mechanism to identify webpage changes and a model for the computation of the rate of change of a given webpage.

Work carried out by Meegahapola et al. [71] has proposed a methodology to pre-predict the update frequency of webpages (the rate at which changes happen), and reschedule the crawling schedule to make the process of crawling webpages more efficient in search engines and web crawlers used for change detection. The authors have introduced a change frequency detection system named Change Frequency Estimator Module (CFEM). It includes two processes. Whenever a fresh webpage is added, that webpage will be crawled to detect a suitable change frequency, and it will be recorded in the system. Then these values are sent to a machine learning model [72] that will predict the time interval between two crawls for a particular webpage. The change values together with change frequencies for a webpage is sent to the machine learning model, it would output a time interval called a *loop time* corresponding to that particular webpage. This value is an estimation of the average time taken by the webpage between two changes or in other terms, the *refresh rate*. It has also been observed by the authors that frequently changing websites obtained lower loop times in comparison to webpages that do not change often. Table 7 summarizes the different change frequency detection techniques with their characteristics and limitations.

5 CHANGE NOTIFICATION

CDN systems provide facilities to notify users about information changes or occurrence of events in a particular webpage of interest. From our studies, we have determined three main characteristics that should be considered when designing a notification system. They are (1) when to notify, (2) how to notify and (3) what to notify.

When to notify changes is an important aspect to decide on when developing a change notification system. Users may want to get notifications as soon as they occur, or some may want to get periodic notifications as they may not want to have a clutter of notifications. The way notifications are sent decides how useful the notification system will be. The system should be able to send the notifications to the user in a way that the user will not be overwhelmed with the notifications. The content to be notified may depend on the user as they may have different information needs. Hence, it is wise to allow the user to customize the content.

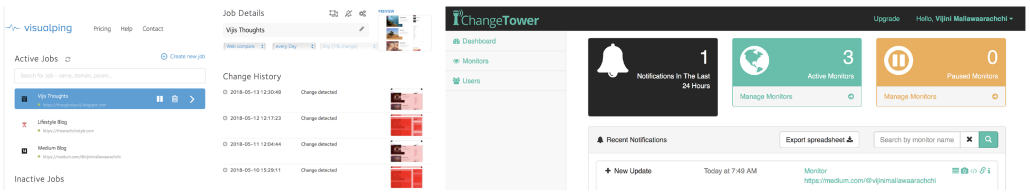
The change notification process of CDN systems available at present consists of a wide range of notification methods. A few of the popular methods are web interfaces, browser plugin notifications, emails, SMS alerts and popup alerts [34].

5.1 Web Interfaces

WebCQ [66] is a CDN system that is among the earliest to become popular back at the beginning of the 2000s. The notification system of WebCQ runs on the server-side, and hence, it uses periodic

Table 7. A Summary of Existing Change Frequency Detection Techniques

Method	Citation	Characteristics	Limitation
Poisson distribution	Coffman et al. 1998 [28]	Webpage changes occur as a Poisson process and determine the mean rate at which changes occur.	Sensitive to parameter changes
Exponential probabilistic model	Brewington and Cybenko 2000 [12]	The ages of webpages that have gone through many changes over a time period are modeled using an exponential PDF.	Sensitive to parameter changes. Assumes all webpages are dynamic.
Temporal freshness metric	Cho and Garcia-Molina 2003 [25]	Determine the level of freshness of a page	Cannot incorporate information about the type of changes
Binary freshness model	Olston and Najork 2010 [85]	Determine whether a page is fresh or stale (provides a binary value)	Cannot determine the level of freshness
Change Frequency Estimator Module (CFEM)	Meegahapola et al. 2017 [71]	Use a machine learning model to determine the time interval between two crawls	Can result in overfitting



(a) (b)

Fig. 9. Dashboard of (a) Visualping [118] and (b) ChangeTower [22].

notifications so that it can run efficiently with a large user base and webpage entries. The interval to send notifications is defined by the user when registering a webpage to track. WebCQ allows users to view changes on a web interface, where they can query to find reports detailing the changes or reports with a summary of changes.

Throughout the past two decades, CDN systems have evolved, by utilizing modern front-end development frameworks, to provide more appealing and user-friendly interfaces with improved user experience. These interfaces have been able to convey useful information to the user in a more efficient and readable manner. Recently introduced change-detection systems, such as Visualping [118] and ChangeTower [22], provide more advanced features for notifying changes within the web interfaces (as shown in Figure 9). The majority of the systems provide a dashboard for the user with summaries of recent changes that have occurred to the webpages they are tracking.

5.2 Browser Plugins

Distill Web Monitor [34] is a CDN system, which is available as a browser plugin. Figure 10 illustrates the browser plugin of Distill Web Monitor. It allows users to view changes highlighted

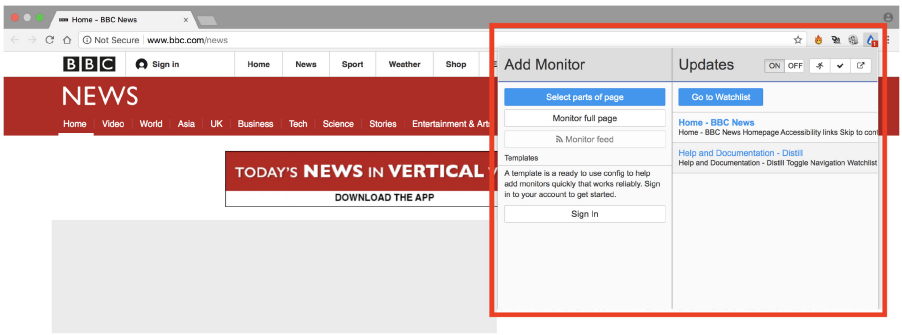


Fig. 10. Browser plugin of Distill Web Monitor [34].

Table 8. A Summary of the Change Notification Techniques

Change Notification Method	Amount of information delivered	Payload size	Level of accessibility	Cost to implement
Web interfaces	High	High	Medium	Low
Browser plugins	High	Low	Medium	Low
Emails	Medium	Medium	Medium	Medium
SMS Alerts	Low	Low	High	High

on the webpage itself. Furthermore, this system provides various notification options for users including, emails, browser notifications, SMS, sounds and popup alerts.

5.3 Emails

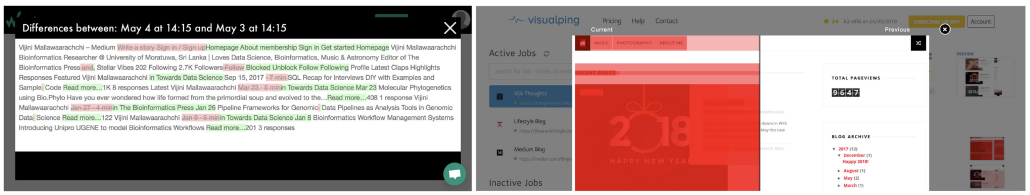
Email notifications have become popular in all the online services as a means to convey new updates and news to the users. Similarly, most of the CDN systems provide the facility to get notified via emails, once changes occur in monitored webpages. Emails generally contain links that when clicked by the user, will be redirected to a page with further information about the relevant change.

5.4 SMS Alerts

Certain CDN systems provide the facility to get notifications via Short Message Service (SMS). The system requests the user to enter a mobile phone number to which the user wants to have the notifications delivered. Google Alerts [45] and Distill Web Monitor [34] are two popular services that provide this facility for its users. SMS alerts are very useful for users who wish to get notifications about webpage updates while traveling and when they do not have access to the Internet to log into the online system. However, the SMS alert feature may require the user to upgrade to their paid versions. Table 8 compares the different features of the various change notification techniques.

6 CHANGE VISUALIZATION

Visualization of changes is an important aspect of CDN systems. Proper visualization techniques will allow the users to easily identify the changes of the webpages being tracked. Publicly available CDN systems visualize changes occurring on webpages in different ways [118, 120]. Most of the changes are depicted in the original interface itself that is loaded by the CDN system.



(a) (b)

Fig. 11. (a) Text change visualization in Wachete [120] and (b) visual comparison of changes in Visualping [118].

Fig. 12. Sample change log of Versionista [117].

6.1 HTML Differencing

One popular means of visualizing textual content is by graphically annotating the differences in the HTML content. This was first introduced as the HtmlDiff tool [35], which marks up the HTML text to indicate how it has changed from a previous version. This tool was introduced in the AT&T Internet Difference Engine (AIDE) [36], which was developed to detect and visualize changes to webpages. Most of the currently available CDN tools use the HtmlDiff technique or its variants that highlight the changes in different colors. The most commonly used color convention is that deleted text is highlighted in red color with strike-through formatting, whereas newly added text is highlighted in green color. The text that has not changed is not highlighted. This method of change visualization is more straightforward as the changes have already been highlighted and are shown to the user. Figure 11(a) shows the visualization used in Wachete [120].

6.2 Visual Comparison

Another interactive method of visualizing changes is by showing the current version and previous version of a webpage side by side on an interface allowing a user to observe the changes for himself. Visualping [118] provides this facility for its users, as shown in Figure 11(b). The two versions are shown side by side, and the cursor can be moved horizontally to slide the visible window to view and compare a particular area on the webpage in the two versions. This method is more interactive as the user is involved in identifying the changes that cannot be seen at once. However, certain users may find this method too tedious and time-consuming as the user himself has to find the changes by making comparisons between the two versions of a webpage.

6.3 Change Logs

Versionista [117] is a CDN tool, where the user can view the modifications in the form of a log. Figure 12 illustrates a sample change log from Versionista. However, some users may find this format hard to read and understand as the changes are not visible immediately.

Table 9 denotes a feature comparison of the various change visualization methods. It can be seen that different techniques have different levels of information representation, understandability, ease of identifying, and cost.

Table 9. A Summary of the Change Visualizations Techniques

Change Visualization Technique	Amount of information shown	Ease of understanding	Ease of identifying changes	Cost
HTML differencing	Medium	High	High	Medium
Visual comparison	Medium	Medium	Medium	High
Change logs	High	Low	Low	Medium

Table 10. Comparison of Features of Publicly Available CDN Systems

CDN System	Monitor a single page	Monitor multiple pages	Server-side detection	Client-side detection	Fixed interval checks	Email	Browser plugin	SMS alerts
Google Alerts [45]	X	X	X	-	X	X	-	X[c]
Distill [34]	X	X	-	X	X	X	X	X[a]
Visualping [118]	X	X	X	X	X	X	X	-
FollowThatPage [43]	X	X	X	X	X	X	X	-
Trackly [113]	X	X	X	-	X	X	-	-
Versionista [117]	X	X[b]	X	-	X	X	-	-
ChangeDetect [20]	X	X	X	-	X	X	-	-
Wachete [120]	X	X[b]	X	X	X	X	X	-
ChangeTower [22]	X	X[b]	X	-	X	X	-	-
OnWebChange [87]	X	X[a]	X	-	X	X	-	-
ChangeMon [21]	X	X	X	-	X	X	-	-
Pagescreen [90]	X	X[b]	X	-	X	X	-	-

[a] This feature is available in the paid version only.
[b] Only a number of limited webpages can be tracked with the free version.
[c] SMS alerts provided by third party applications.

7 PUBLICLY AVAILABLE CDN SYSTEMS

Different CDN systems are available at present, and each of them has its own supported set of features. Table 10 denotes twelve popular CDN systems, and compares their features, such as pages monitored, detection architecture and notification methods.

Most of the systems support change detection of a single page and multiple pages for a single user freely. However, systems such as Versionista [117], Wachete [120] and PageScreen [90] offer a limited number of webpages that can be checked under the trial version. If a user wants to track more webpages than the given limit, then the user has to upgrade to the paid version where unlimited tracking is provided.

The majority of the systems use server-side change-detection approaches, whereas a few systems use client-side change-detection approaches. According to studies, most of the commercial systems available at present use server-side change detection due to the easy implementation in a central location, where users can call the functionality as a service via a web interface. There are a few tools, such as Visualping [118] and Follow that Page [43], where they use both server-side and client-side change-detection approaches. However, tools such as Distill [34] use client-side change detection via browser plugins.

It is evident that all the systems support fixed interval checks. This can cause issues, because the user has no knowledge of how often a particular webpage will get changed, and the user may fail to observe important updates by selecting arbitrary checking intervals. Hence, dynamic scheduling mechanisms should be addressed to enhance the efficiency of CDN systems.

Most systems support a wide range of fixed interval checks, such as twice a day, daily, weekly and monthly. More frequent checks such as hourly checks, three hourly checks and six hourly checks are provided in the paid versions of many CDN systems. However, the browser plugin of Distill supports checks varying from every 5 seconds up to 29 days. Such high checking frequencies are possible as the Distill system runs on the client, where the client may have ample resources. However, server-based systems may not support such high checking frequencies as the server can get overloaded with the growing user base and the number of pages to be tracked.

When considering the notification methods, it can be seen that all the tools provide email alerts. Emails have become popular due to its simplicity, easy implementation and extensive use among the clients. A few systems provide browser plugin alerts and SMS alerts. However, with the development of mobile devices, certain services, such as Watchete and Visualping, have provided mobile applications that can be installed on smartphones. This has allowed the user to get updates and manage monitors via his/her smartphone. Some systems, such as ChangeDetect [20] and PageScreen, provide free trials for users to try out their features for a limited time. After the trial period has passed the users must upgrade to continue to get the services. Trackly [113] provides a free plan where it allows a user to track three webpages. Trackly also provides 30-day free trials for all its plans and consists of the same features as the paid version. ChangeMon [21] provides a 7-day free trial where a user is not required to sign up for an account and can create up to 1,000 monitors.

8 DISCUSSION

In the modern world, webpage change detection has become very complicated due to many reasons, such as (1) evolution of technologies used in webpage creation, (2) addition of rich and dynamic content to webpages, and (3) privacy concerns and regulations. In this section, we will discuss some of the trends, concerns and insights as to what modern researchers/developers should pay attention to when building solutions related to webpage CDN.

8.1 Ethical Aspects and Security Concerns of Web Crawling

Change detection of webpages primarily relies on web crawling. Ethical aspects and security concerns of web crawling are important considerations when building web crawler-based applications although they are mostly neglected. Research and guidelines in this area are limited, and work done by Thelwall and Stuart [112] is one of the modern works regarding this topic. After conducting an extensive analysis regarding applications of web crawlers, they have come up with four main types of issues that web crawlers may cause to the society or individuals. They are, namely, (1) denial of service—due to repetitive requests to web servers, failures may occur causing disturbances to normal users of the website; (2) cost—increasing number of requests may incur additional costs to the website owner depending on the cost of the web hosting plan that is used; (3) privacy—even though data in webpages are public, gathering of information in large scale might lead to privacy violations; (4) copyright—crawlers create copies of webpages/datasets from websites without prior consent, which may directly involve copyright violations. The authors have further explained how Robots Exclusion Protocol (robots.txt) [62] can be used to overcome the above issues from the perspectives of website owners and web crawler owners. In doing so, they have emphasized the limitations of the set of guidelines, and elaborated on alternative techniques to overcome the limitations.

Even though many commercial search engines and CDN systems have adopted the Robots Exclusion Protocol to various extents, the degree to which each web crawler abides by the guidelines set by the protocol differs. A study carried out by Sun et al. [110] has come up with a mechanism to quantify the ethicality of web crawlers using a vector space model. While acknowledging the fact that the unethicality of crawling may differ from webpage to webpage, they have used a

common set of ethical guidelines set by Eichmann [37] and Thelwall et al. [112] in creating this mechanism. More research on this avenue would be interesting as a lot of governments, and policy regulating agencies have shown an increasing interest regarding ethical aspects and data privacy of online content during the past 5 years. To come up with widely adopted guidelines and regulations regarding web crawling, having such ethicality quantification measurements of web crawlers would be crucial. Further, regardless of the guidelines set by the Robots Exclusion Protocol that has been adopted by over 30% of webpages by 2007 [111], many web crawlers do not abide by the regulations. Because of this, many websites with crucial and private data deploy anti-crawling techniques, such as (1) IP address ban; (2) captcha; (3) obfuscated JavaScript code; (4) frequent structural changes; (5) limit the frequency of requests and downloadable data allowances; and (6) mapping important data like images, which is considered one of the most effective ways.

8.2 Change Detection in Dynamic Webpages and Webpages with Rich Content

Modern webpages developed using technologies such as HTML5 with rich content, such as images and videos, are constantly changing their elements to provide rich and interactive functionality to users [98]. Cascading Style Sheets (CSS) is used to manage their layouts, and JavaScript is used to manage their user interactions. The content and layouts of such webpages have become more dynamic to adapt to different user actions and devices. Moreover, dynamic webpages can have temporary information, such as help pop-ups, stored in a set of stacked layers apart from the basic two-dimensional layout, hence giving the name three-dimensional webpages [119]. These layers can undergo many changes. Hence, detecting changes in such dynamic webpages has become challenging.

Currently, a limited amount of research work can be found for change detection in dynamic webpages within the available literature. However, web tripwires [96], CRAWLJAX [75], ReDeCheck [121], and detection of visibility faults caused by dynamic layout changes [79, 98] can be considered as significant work for crawling and detecting changes in dynamic webpages. These make use of the client-side code to determine state changes occurred within the browser's dynamically built DOM tree when user events are triggered. The DOM structure of webpages created using JavaScript frameworks, such as angular, ember, and react, can change as events are triggered, and relevant DOM elements are rendered. Crawler scripts can determine such changes by accessing the rendered content via calling the innerHTML property of the elements, as the HTML content is not readily visible [96]. Since accessing the client-side code is a critical aspect in detecting changes of dynamic webpages, it is worth to explore more efficient methods to perform this task.

When considering from an industry perspective, not many commercially available CDN systems support the monitoring of dynamic webpages. A handful of CDN systems, such as Wachete [120], Visualping [118] and Versionista [117], allow users to monitor dynamic and JavaScript rendered pages. Moreover, the algorithms used are kept as trade secrets by these companies, and are not available publicly. However, the monitoring process of highly dynamic JavaScript webpages can timeout due to large amounts of data that have to be processed. Hence, there is an opportunity for researchers to contribute for the development of efficient algorithms to determine changes occurring in highly dynamic webpages.

Modern webpages have rich content, such as images, audio, and video, to enhance the user experience. Webpages may be changed by changing these rich content. Most of the currently available CDN systems can detect changes within the HTML content. In the case of images, a change can be detected if the contents of the <*img*> tag change. Sometimes the actual image may be changed but the image has the same file name as before, and hence, such changes are not detected. However, to the best of our knowledge, currently available systems do not digitally analyze images,

and detect whether an image is actually changed or not. However, several image change-detection algorithms, such as image differencing, image ratioing and change vector analysis, can be found in the research domain of image processing. These algorithms are used for applications such as aerial analysis of images of forest areas and urban environments [76]. Similar ideas can be utilized in CDN systems for webpages to detect changes occurring in images. However, if such sophisticated methods are implemented within CDN systems, then they will require more computational resources to operate efficiently.

8.3 Resource Synchronization

The resource synchronization problem has been identified as an issue with frequently changing web sources [114]. It is defined as the "need for one or more destination servers to remain synchronized with (some of the) resources made available by a source server." From a CDN perspective, we can state this problem as clients wanting to stay up to date with the new content of frequently changing webpages. Previous work includes the use of Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) [88] to harvest digital resources [115], where new and updated resources are collected incrementally, but it was not adopted broadly.

ResourceSync is an emerging framework [61, 116] that describes means for both client-pull and server-push of change notifications. It consists of several modules that allow intended parties to stay in sync with changing resources. The framework allows source servers to publish up-to-date lists of resources and changes to the resources. Destination servers can access this published data and synchronize the resources. This framework can be adapted for large repositories. The framework represents changed resources or differences between the previous and new representations as a bitstream, allowing to obtain changes to different media types. The use of the ResourceSync framework in CDN systems can make the synchronizing process of clients more efficient and deliver change notifications in a timely manner.

8.4 Linked Data Notifications

With the emergence of User-Generated Content (UGC) proprietorship and anti-scraping tools, commercial crawlers are not allowed to crawl content, and bots are blocked [80]. Hence, the details about webpages are unknown before crawling. Despite these challenges, if the changes of webpages can be made available to web crawlers in a standard manner, web crawlers can easily access these data to detect changes that have occurred. An emerging idea that can be used by CDN systems is the Linked Data Notifications (LDN) protocol.

The LDN protocol describes how servers (termed as receivers) can make applications (termed as senders) push messages to them, and how other applications (termed as consumers) can retrieve those messages [16]. This protocol allows us to share and reuse notifications across different applications while paying less attention to how they were created or what their contents are. Notifications are implemented in such a manner that they can run on different technology stacks, and continuously work together while supporting decentralized communication in the web. The idea was first brought forward by Capadisli [16], and has been considered as a W3C recommendation. LDN identifies a notification as "an individual entity with its own Uniform Resource Identifier (URI)," and hence, they can be retrieved and reused. The protocol stores notifications in a manner so that they are compatible with the Linked Data Platform (LDP) standard.

The overview of LDN is illustrated in Figure 13. A sender wants to deliver a notification to the server that is intended to a receiver. The sender selects a target source, finds the location of the target's inbox, and sends the notification to that particular inbox. Then the receiver allows consumers to access the notification in its inbox. Similarly, when considering the perspective of CDN, CDN systems can make use of the LDN protocol to implement notifications sent to subscribed users.

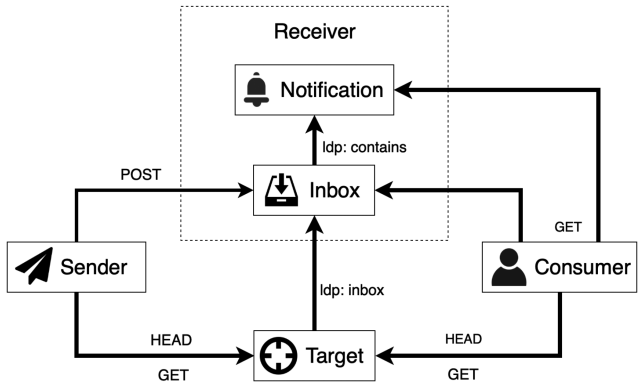


Fig. 13. Overview of Linked Data Notifications [16].

Since the notifications are reused, the system becomes more efficient, and improves the system productivity. LDN will become the next trend in CDN systems.

9 CONCLUSION

The history of CDN systems dates to the 1990s when they were introduced to automate the detection process of webpage changes and notify interested users. Since then, various CDN systems and techniques have been introduced to improve the efficiency of the CDN process. This article presented a survey on CDN systems for webpages and various related techniques. We have reviewed and compared different techniques in the literature involving various aspects of CDN systems. Among them, techniques used in web crawler scheduling, change detection and change frequency were identified as significant research areas, where extensive research has been carried out. The most common change-detection algorithms are based on difference calculation between tree structures of documents. The process of identifying the frequency of changes occurring on webpages plays a significant role in optimizing crawler schedules to retrieve updated content.

We have also compared different change notification and change visualization techniques that are being used by currently available CDN systems. Most of such systems show notifications on a web interface, and use email notifications as their main method of notifying the user, whereas some systems provide the facility to get notifications via SMS alerts or browser plugin notifications. Additionally, a majority of applications use HTML differencing techniques for change visualization between two variants of a webpage, whereas some systems provide a visual separation among versions, which allows the user to identify the changes by observing the two versions. Moreover, we have compared different features of twelve popular CDN systems that are publicly available at present. According to the comparison results, it is evident that most of the systems support checks at fixed intervals, but not checks at random intervals. These systems can be improved by introducing intelligent crawling schedules to optimize the crawling process by crawling webpages at their estimated change frequency.

Finally, we have discussed new trends such as LDN and issues such as determining changes in dynamic webpages, privacy concerns and ethical aspects in CDNs. Throughout this survey, we have identified four important directions of research. The first research direction focuses on improving the architecture of CDN systems, where computing resources and temporal resources can be utilized efficiently while overcoming the limitations of traditional server-based and client-based architectures. The second research direction focuses on improving change-detection algorithms to track webpage changes quickly with high accuracy. The third research direction

focuses on identifying the change frequency of webpages and designing optimized crawler schedules so that computing resources can be used efficiently by deploying crawlers when required. The final research direction is improving and developing methods and algorithms to detect changes in dynamic and JavaScript rendered webpages that can efficiently handle large amounts of data.

REFERENCES

[1] E. Adar, J. Teevan, S. T. Dumais, and J. L. Elsas. 2009. The web changes everything: Understanding the dynamics of web content. In *Proceedings of the 2nd ACM International Conference on Web Search and Data Mining (WSDM'09)*. ACM, New York, NY, 282–291. DOI: https://doi.org/10.1145/1498759.1498837

[2] F. Ahmadi-Abkenari and A. Selamat. 2012. An architecture for a focused trend parallel Web crawler with the application of clickstream analysis. *Info. Sci.* 184, 1 (2012), 266–281. DOI: https://doi.org/10.1016/j.ins.2011.08.022

[3] A. Anjum and A. Anjum. 2012. Aiding web crawlers; projecting web page last modification. In *Proceeding sof the 15th International Multitopic Conference (INMIC'12)*. 245–252. DOI: https://doi.org/10.1109/INMIC.2012.6511443

[4] R. Baeza-Yates, C. Castillo, and F. Saint-Jean. 2004. *Web Dynamics, Structure, and Page Quality*. Springer, Berlin, 93–109. DOI: https://doi.org/10.1007/978-3-662-10874-1_5

[5] K. Benjamin, G. von Bochmann, M. E. Dincturk, G.-V. Jourdan, and I. V. Onut. 2011. A strategy for efficient crawling of rich internet applications. In *Web Engineering*, S. Auer, O. Díaz, and G. A. Papadopoulos (Eds.). Springer, Berlin, 74–89.

[6] D. Bhatt, D. A. Vyas, and S. Pandya. 2015. Focused web crawler. *Adv. Comput. Sci. Info. Technol.* 2, 11 (Apr. 2015), 1–6.

[7] B. H. Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13, 7 (July 1970), 422–426. DOI: https://doi.org/10.1145/362686.362692

[8] P. Boldi, B. Codenotti, M. Santini, and S. Vigna. 2004. UbiCrawler: A scalable fully distributed Web crawler. *Softw.: Pract. Exper.* 34, 8 (2004), 711–726. DOI: https://doi.org/10.1002/spe.587 arXiv: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.587

[9] P. Boldi, A. Marino, M. Santini, and S. Vigna. 2018. BUbiNG: Massive crawling for the masses. *ACM Trans. Web* 12, 2 (June 2018). DOI: https://doi.org/10.1145/3160017

[10] K. Borgolte, C. Kruegel, and G. Vigna. 2014. Relevant change detection: A framework for the precise extraction of modified and novel web-based content as a filtering technique for analysis engines. In *Proceedings of the 23rd International Conference on World Wide Web (WWW'14)*. ACM, New York, NY, 595–598. DOI: https://doi.org/10.1145/2567948.2578039

[11] O. Brandman, J. Cho, H. Garcia-Molina, and N. Shivakumar. 2000. Crawler-friendly web servers. *SIGMETRICS Perform. Eval. Rev.* 28, 2 (Sept. 2000), 9–14. DOI: https://doi.org/10.1145/362883.362894

[12] B. E. Brewington and G. Cybenko. 2000. How dynamic is the Web? *Comput. Netw.* 33, 1 (2000), 257–276. DOI: https://doi.org/10.1016/S1389-1286(00)00045-1

[13] S. Brin and L. Page. 1998. The anatomy of a large-scale hypertextual Web search engine. *Comput. Netw. ISDN Syst.* 30, 1 (1998), 107–117. DOI: https://doi.org/10.1016/S0169-7552(98)00110-X

[14] D. Buytaert. 2000. Drupal—Open Source CMS | Drupal.org. Retrieved from https://www.drupal.org/.

[15] G. C. Canavos. 1972. A Bayesian approach to parameter and reliability estimation in the poisson distribution. *IEEE Trans. Reliabil.* R-21, 1 (Feb. 1972), 52–56. DOI: https://doi.org/10.1109/TR.1972.5216172

[16] S. Capadisli, A. Guy, C. Lange, S. Auer, A. Sambra, and T. Berners-Lee. 2017. Linked data notifications: A resource-centric communication protocol. In *The Semantic Web*, E. Blomqvist, D. Maynard, A. Gangemi, R. Hoekstra, P. Hitzler, and O. Hartig (Eds.). Springer International Publishing, Cham, 537–553.

[17] C. Castillo, M. Marin, A. Rodriguez, and R. Baeza-Yates. 2004. Scheduling algorithms for Web crawling. In *Proceedings of the WebMedia and LA-Web Conference*. 10–17. DOI: https://doi.org/10.1109/WEBMED.2004.1348139

[18] S. Chakrabarti, M. van den Berg, and B. Dom. 1999. Focused crawling: A new approach to topic-specific Web resource discovery. *Comput. Netw.* 31, 11 (1999), 1623–1640. DOI: https://doi.org/10.1016/S1389-1286(99)00052-3

[19] T. D. Chandra and S. Toueg. 1996. Unreliable failure detectors for reliable distributed systems. *J. ACM* 43, 2 (Mar. 1996), 225–267. DOI: https://doi.org/10.1145/226643.226647

[20] ChangeDetect. 2002. ChangeDetect Web Page Monitoring. Retrieved from https://www.changedetect.com/.

[21] ChangeMon. 2015. ChangeMon—Monitor Any Web Page For Changes. Retrieved from https://changemon.com/.

[22] ChangeTower. 2017. ChangeTower—Monitor Website Changes, Get Alerts, Archive Website History. Retrieved from https://changetower.com/.

[23] J. Cho and H. Garcia-Molina. 2000. The evolution of the web and implications for an incremental crawler. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB'00)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 200–209. Retrieved from http://dl.acm.org/citation.cfm?id=645926.671679.

[24] J. Cho and H. Garcia-Molina. 2000. Synchronizing a database to improve freshness. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'00)*. ACM, New York, NY, 117–128. DOI: https://doi.org/10.1145/342009.335391

[25] J. Cho and H. Garcia-Molina. 2003. Effective page refresh policies for web crawlers. *ACM Trans. Database Syst.* 28, 4 (Dec. 2003), 390–426. DOI: https://doi.org/10.1145/958942.958945

[26] J. Cho and H. Garcia-Molina. 2003. Estimating frequency of change. *ACM Trans. Internet Technol.* 3, 3 (Aug. 2003), 256–290. DOI: https://doi.org/10.1145/857166.857170

[27] G. Cobena, S. Abiteboul, and A. Marian. 2002. Detecting changes in XML documents. In *Proceedings of the 18th International Conference on Data Engineering*. 41–52. DOI: https://doi.org/10.1109/ICDE.2002.994696

[28] E. G. Coffman Jr., Z. Liu, and R. R. Weber. 1998. Optimal robot scheduling for Web search engines. *J. Schedul.* 1, 1 (1998), 15–29. DOI: https://doi.org/10.1002/(SICI)1099-1425(199806)1:1⟨15::AID-JOS3⟩3.0.CO;2-K

[29] D3S. 2015. Shash Tool. Retrieved from http://d3s.mff.cuni.cz/ holub/sw/shash/.

[30] Z. Dalai, S. Dash, P. Dave, L. Francisco-Revilla, R. Furuta, U. Karadkar, and F. Shipma. 2004. Managing distributed collections: Evaluating Web page changes, movement, and replacement. In *Proceedings of the Joint ACM/IEEE Conference on Digital Libraries, 2004.* 160–168. DOI: https://doi.org/10.1109/JCDL.2004.240012

[31] P. De Bra and L. Calvi. 1997. Creating adaptive hyperdocuments for and on the web. In *Proceedings of the WebNet Conference (WebNet'97)*. 149–165.

[32] M. Diligenti, F. Coetzee, S. Lawrence, C. Giles, and M. Gori. 2000. Focused crawling using context graphs. In *Proceedings of the 26th Very Large Data Base Conference (VLDB'00)*. 527–534.

[33] M. E. Dincturk, S. Choudhary, G. von Bochmann, G.-V. Jourdan, and I. V. Onut. 2012. A statistical approach for efficient crawling of rich internet applications. In *Web Engineering*, M. Brambilla, T. Tokuda, and R. Tolksdorf (Eds.). Springer, Berlin, 362–369.

[34] Distill.io. 2013. Monitor Websites for Changes, Get SMS Alerts and Email Alerts | Distill.io. Retrieved from https://distill.io.

[35] F. Douglis and T. Ball. 1996. Tracking and viewing changes on the web. In *Proceedings of the USENIX Annual Technical Conference (USENIX'96)*. USENIX Association, Berkley, CA.

[36] F. Douglis, T. Ball, Y.-F. Chen, and E. Koutsofios. 1998. The AT&T internet difference engine: Tracking and viewing changes on the web. *World Wide Web* 1, 1 (Mar. 1998), 27–44. DOI: https://doi.org/10.1023/A:1019243126596

[37] D. Eichmann. 1995. Ethical web agents. *Comput. Netw. ISDN Syst.* 28, 1 (1995), 127–136. DOI: https://doi.org/10.1016/0169-7552(95)00107-3

[38] J. L. Elsas and S. T. Dumais. 2010. Leveraging temporal dynamics of document content in relevance ranking. In *Proceedings of the 3rd ACM International Conference on Web Search and Data Mining (WSDM'10)*. ACM, New York, NY, 1–10. DOI: https://doi.org/10.1145/1718487.1718489

[39] J. Exposto, J. Macedo, A. Pina, A. Alves, and J. Rufino. 2008. Efficient partitioning strategies for distributed web crawling. In *Information Networking. Towards Ubiquitous Networking and Services*, T. Vazão, M. M. Freire, and I. Chong (Eds.). Springer, Berlin, 544–553.

[40] D. Fetterly, M. Manasse, M. Najork, and J. Wiener. 2003. A large-scale study of the evolution of web pages. In *Proceedings of the 12th International Conference on World Wide Web (WWW'03)*. ACM, New York, NY, 669–678. DOI: https://doi.org/10.1145/775152.775246

[41] R. T. Fielding. 1994. Maintaining distributed hypertext infostructures: Welcome to MOMspider's Web. *Comput. Netw. ISDN Syst.* 27, 2 (1994), 193–204. DOI: https://doi.org/10.1016/0169-7552(94)90133-3

[42] K. Filipowski. 2014. Comparison of scheduling algorithms for domain specific web crawler. In *Proceedings of the European Network Intelligence Conference*. 69–74. DOI: https://doi.org/10.1109/ENIC.2014.14

[43] Follow That Page. 2008. Follow That Page—Web Monitor: We Send You an Email When Your Favorite Page has Changed. Retrieved from https://www.followthatpage.com.

[44] L. Francisco-Revilla, F. Shipman, R. Furuta, U. Karadkar, and A. Arora. 2001. Managing change on the web. In *Proceedings of the 1st ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'01)*. ACM, New York, NY, 67–76. DOI: https://doi.org/10.1145/379437.379973

[45] Google. 2003. Google Alerts—Monitor the Web for Interesting New Content. Retrieved from https://www.google.com/alerts.

[46] Google. 2013. Googlebot: Search Console Help. Retrieved from https://support.google.com/webmasters/answer/182072?hl=en.

[47] Google. 2013. Official Google Blog: A Second Spring of Cleaning.Retrieved from https://googleblog.blogspot.com/2013/03/a-second-spring-of-cleaning.html.

[48] Google. 2018. Submit URL. Retrieved from https://www.google.com/webmasters/tools/submit-url.

[49] C. Grimes and S. O'Brien. 2008. Microscale evolution of web pages. In *Proceedings of the 17th International Conference on World Wide Web (WWW'08)*. ACM Press, New York, NY, 1149–1150.

[50] M. Hersovici, M. Jacovi, Y. S. Maarek, D. Pelleg, M. Shtalhaim, and S. Ur. 1998. The shark-search algorithm. An application: Tailored Web site mapping. *Comput. Netw. ISDN Syst.* 30, 1 (1998), 317–326. DOI: https://doi.org/10.1016/S0169-7552(98)00038-5

[51] Z. Hmedeh, N. Vouzoukidou, N. Travers, V. Christophides, C. du Mouza, and M. Scholl. 2011. Characterizing web syndication behavior and content. In *Proceedings of the Conference on Web Information System Engineering (WISE'11)*, A. Bouguettaya, M. Hauswirth, and L. Liu (Eds.). Springer, Berlin, 29–42.

[52] J. Jacob, A. Sanka, N. Pandrangi, and S. Chakravarthy. 2004. *WebVigiL: An Approach to Just-In-Time Information Propagation in Large Network-Centric Environments*. Springer, Berlin, 301–318. DOI: https://doi.org/10.1007/978-3-662-10874-1_13

[53] S. Jain and H. Khandagale. 2014. A web page change-detection system for selected zone using tree comparison technique. *Int. J. Comput. Appl. Technol. Res.* 3, 4 (Apr. 2014), 254–262.

[54] M. Jamali, H. Sayyadi, B. B. Hariri, and H. Abolhassani. 2006. A method for focused crawling using combination of link structure and content similarity. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI 06)*. IEEE Computer Society, Washington, DC, 753–756. DOI: https://doi.org/10.1109/WI.2006.19

[55] S. Jayarathna and F. Poursardar. 2016. Change detection and classification of digital collections. In *Proceedings of the IEEE International Conference on Big Data (Big Data'16)*. 1750–1759. DOI: https://doi.org/10.1109/BigData.2016.7840790

[56] D. B. Johnson. 1977. Efficient algorithms for shortest paths in sparse networks. *J. ACM* 24, 1 (Jan. 1977), 1–13. DOI: https://doi.org/10.1145/321992.321993

[57] D. B. Johnson and S. L. Tanimoto. 1999. Reusing web documents in tutorials with the current-documents assumption: Automatic validation of updates. In *Proceedings of the EdMedia + Innovate Learning Conference*, Betty Collis and Ron Oliver (Eds.). Association for the Advancement of Computing in Education (AACE), Seattle, WA, 74–79. Retrieved from https://www.learntechlib.org/p/17401.

[58] M. A. Kausar, V. S. Dhaka, and S. K. Singh. 2013. Web crawler: A review. *Int. J. Comput. Appl.* 63, 2 (2013), 31–36.

[59] M. Kc, M. Hagenbuchner, and A. C. Tsoi. 2008. A scalable lightweight distributed crawler for crawling with limited resources. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, Vol. 3. 663–666. DOI: https://doi.org/10.1109/WIIAT.2008.234

[60] J. E. Keogh. 2005. *ASP.NET 2.0 Demystified* (1st ed.). McGraw Hill Professional, New York.

[61] M. Klein, R. Sanderson, H. Van de Sompel, S. Warner, B. Haslhofer, C. Lagoze, and M. L. Nelson. 2013. A technical framework for resource synchronization. *D-Lib Mag.* 19, 1/2 (Jan. 2013).

[62] M. Koster. 1993. Guidelines for Robot Writers. Retrieved from http://www.robotstxt.org/guidelines.html.

[63] M. Kumar and P. Neelima. 2011. Design and implementation of scalable, fully distributed web crawler for a web search engine. *Int. J. Comput. Appl.* 15, 7 (Feb. 2011), 8–13.

[64] M. Levene and A. Poulovassilis. 2013. *Web Dynamics: Adapting to Change in Content, Size, Topology and Use (2013 ed.)*. Springer Science & Business Media, Berlin. DOI: https://doi.org/10.1007/978-3-662-10874-1

[65] J. Li, K. Furuse, and K. Yamaguchi. 2005. Focused crawling by exploiting anchor text using decision tree. In *Proceedings of the of the 14th International Conference on World Wide Web (WWW'05)*. ACM Press, New York, NY, 1190–1191.

[66] L. Liu, C. Pu, and W. Tang. 2000. WebCQ-detecting and delivering information changes on the web. In *Proceedings of the 9th International Conference on Information and Knowledge Management (CIKM'00)*. ACM, New York, NY, 512–519. DOI: https://doi.org/10.1145/354756.354860

[67] S. Mali and B. B. Meshram. 2011. Focused web crawler with page change-detection policy. In *Proceedings of the 2nd International Conference and Workshop on Emerging Trends in Technology (ICWET'11)*. 51–57.

[68] O. McBryan. 1994. GENVL and WWWW: Tools for taming the web. In *Proceedings of the 1st International World Wide Web Conference (WWW'94)*.

[69] L. Meegahapola, R. Alwis, E. Heshan, V. Mallawaarachchi, D. Meedeniya, and S. Jayarathna. 2017. Adaptive technique for web page change detection using multi-threaded crawlers. In *Proceedings of the 7th International Conference on Innovative Computing Technology (INTECH'17)*. 120–125. DOI: https://doi.org/10.1109/INTECH.2017.8102430

[70] L. Meegahapola, R. Alwis, E. Nimalarathna, V. Mallawaarachchi, D. Meedeniya, and S. Jayarathna. 2017. Optimizing change detection in distributed digital collections: An architectural perspective of change detection. In *Proceedings of the 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'17)*. IEEE, 277–282. DOI: https://doi.org/10.1109/SNPD.2017.8022733

[71] L. Meegahapola, R. Alwis, E. Nimalarathna, V. Mallawaarachchi, D. Meedeniya, and S. Jayarathna. 2017. Detection of change frequency in web pages to optimize server-based scheduling. In *Proceeding sof the 17th International Conference on Advances in ICT for Emerging Regions (ICTer'17)*. IEEE, 165–172. DOI: https://doi.org/10.1109/ICTER.2017.8257791

[72] L. Meegahapola, V. Mallawaarachchi, R. Alwis, E. Nimalarathna, D. Meedeniya, and S. Jayarathna. 2018. Random forest classifier-based scheduler optimization for search engine web crawlers. In *Proceedings of the 7th International*

Conference on Software and Computer Applications (ICSCA'18). ACM, New York, NY, 285–289. DOI: https://doi.org/10.1145/3185089.3185103

[73] L. B. Meegahapola, P. K. D. R. M. Alwis, L. B. E. H. Nimalarathna, V. G. Mallawaarachchi, D. A. Meedeniya, and S. Jayarathna. 2017. Change-detection optimization in frequently changing web pages. In *Proceedings of the Moratuwa Engineering Research Conference (MERCon'17)*. IEEE, 111–116. DOI: https://doi.org/10.1109/MERCon.2017.7980466

[74] F. Menczer, G. Pant, P. Srinivasan, and M. E. Ruiz. 2001. Evaluating topic-driven web crawlers. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'01)*. ACM, New York, NY, 241–249. DOI: https://doi.org/10.1145/383952.383995

[75] A. Mesbah, A. van Deursen, and S. Lenselink. 2012. Crawling AJAX-based web applications through dynamic analysis of user interface state changes. *ACM Trans. Web* 6, 1 (Mar. 2012). DOI: https://doi.org/10.1145/2109205.2109208

[76] S. Minu and A. Shetty. 2015. A comparative study of image change-detection algorithms in MATLAB. *Aquatic Procedia* 4 (Mar. 2015), 1366–1373.

[77] S. M. Mirtaheri, D. Zou, G. V. Bochmann, G. Jourdan, and I. V. Onut. 2013. Dist-RIA crawler: A distributed crawler for rich internet applications. In *Proceedings of the 8th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*. 105–112. DOI: https://doi.org/10.1109/3PGCIC.2013.22

[78] P. Misra and H. Sorenson. 1975. Parameter estimation in Poisson processes (Corresp.). *IEEE Trans. Info. Theory* 21, 1 (Jan. 1975), 87–90. DOI: https://doi.org/10.1109/TIT.1975.1055324

[79] M. A. Moyeen, G. G. M. N. Ali, P. H. J. Chong, and N. Islam. 2016. An automatic layout faults detection technique in responsive web pages considering JavaScript defined dynamic layouts. In *Proceedings of the 3rd International Conference on Electrical Engineering and Information Communication Technology (ICEEICT'16)*. 1–5. DOI: https://doi.org/10.1109/CEEICT.2016.7873146

[80] Z. Muscovitch. 2012. Who Owns Your Craigslist Advert? Retrieved from http://www.dnattorney.com/056-057-IPM_October_2012.pdf.

[81] S. Nadaraj. 2016. Distributed content aggregation & content change detection using bloom filters. *Int. J. Comput. Sci. Info. Technol.* 7, 2 (Mar. 2016), 745–748.

[82] NetMind. 1996. NetMind Mind-it. Retrieved from http://www.netmind.com/.

[83] M. Nottingham and R. Sayre. 2005. RFC 4287—The Atom Syndication Format. Retrieved from https://tools.ietf.org/html/rfc4287.

[84] M. Oita and P. Senellart. 2011. Deriving dynamics of web pages: A survey. In *Proceedings of the Temporal Workshop on Web Archiving TWAW'11)*. HAL-Inria, Hyderabad, India. Retrieved from https://hal.inria.fr/inria-00588715.

[85] C. Olston and M. Najork. 2010. Web crawling. *Found. Trends Info. Retriev.* 4, 3 (2010), 175–246. DOI: https://doi.org/10.1561/1500000017

[86] C. Olston and S. Pandey. 2008. Recrawl scheduling based on information longevity. In *Proceedings of the 17th International Conference on World Wide Web (WWW'08)*. ACM, New York, NY, 437–446. DOI: https://doi.org/10.1145/1367497.1367557

[87] OnWebChange. 2009. OnWebChange—Track Web Page Changes and Get Notified. Free Sign-up. Retrieved from https://onwebchange.com/.

[88] Open Archives Initiative. 2017. Open Archives Initiative Protocol for Metadata Harvesting. Retrieved from https://www.openarchives.org/pmh/.

[89] L. Page, S. Brin, R. Motwani, and T. Winograd. 1999. *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report 1999-66. Stanford InfoLab. Retrieved from http://ilpubs.stanford.edu:8090/422/.

[90] Pagescreen. 2018. Monitor Website Changes: Automated Alerts and Archives. Retrieved from https://pagescreen.io/.

[91] S. Pandey and C. Olston. 2005. User-centric web crawling. In *Proceedings of the 14th International Conference on World Wide Web (WWW'05)*. ACM, New York, NY, 401–411. DOI: https://doi.org/10.1145/1060745.1060805

[92] G. Pant and F. Menczer. 2003. Topical crawling for business intelligence. In *Research and Advanced Technology for Digital Libraries*, T. Koch and I. T. Sølvberg (Eds.). Springer, Berlin, 233–244.

[93] Z. Pehlivan, M. Ben-Saad, and S. Gançarski. 2010. Vi-DIFF: Understanding web pages changes. In *Database and Expert Systems Applications*, P. G. Bringas, A. Hameurlain, and G. Quirchmayr (Eds.). Springer, Berlin, 1–15.

[94] V. M. Prieto, M. Álvarez, V. Carneiro, and F. Cacheda. 2015. Distributed and collaborative web change-detection system. *Comput. Sci. Info. Syst.* 12, 1 (2015), 91–114.

[95] F. Radlinski, P. N. Bennett, and E. Yilmaz. 2011. Detecting duplicate web documents using clickthrough data. In *Proceedings of the 4th ACM International Conference on Web Search and Data Mining (WSDM'11)*. ACM, New York, NY, 147–156. DOI: https://doi.org/10.1145/1935826.1935859

[96] C. Reis, S. D. Gribble, T. Kohno, and N. C. Weaver. 2008. Detecting in-flight page changes with web tripwires. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, San Francisco, CA, 31–44.

[97] RSS-DEV Working Group. 2000. RDF Site Summary (RSS) 1.0.Retrieved from http://web.resource.org/rss/1.0/spec.

[98] Y. Ryou and S. Ryu. 2018. Automatic detection of visibility faults by layout changes in HTML5 web pages. In *Proceedings of the IEEE 11th International Conference on Software Testing, Verification and Validation (ICST'18)*. 182–192. DOI: https://doi.org/10.1109/ICST.2018.00027

[99] M. B. Saad and Stéphane Gançarski. 2012. Archiving the web using page changes patterns: A case study. *Int. J. Dig. Libr.* 13, 1 (Dec. 2012), 33–49. DOI: https://doi.org/10.1007/s00799-012-0094-z

[100] A. S. R. Santos, N. Ziviani, J. Almeida, C. R. Carvalho, E. S. de Moura, and A. S. da Silva. 2013. Learning to schedule webpage updates using genetic programming. In *String Processing and Information Retrieval*, O. Kurland, M. Lewenstein, and E. Porat (Eds.). Springer International Publishing, Cham, 271–278.

[101] U. Schonfeld and N. Shivakumar. 2009. Sitemaps: Above and beyond the crawl of duty. In *Proceedings of the 18th International Conference on World Wide Web (WWW'09)*. ACM, New York, NY, 991–1000. DOI: https://doi.org/10.1145/1526709.1526842

[102] R. W. Sebesta. 2011. *Programming the World Wide Web* (4th ed.). Addison-Wesley Publishing Company, Boston, MA.

[103] V. Shkapenyuk and T. Suel. 2002. Design and implementation of a high-performance distributed Web crawler. In *Proceedings of the 18th International Conference on Data Engineering*. 357–368. DOI: https://doi.org/10.1109/ICDE.2002.994750

[104] Shobhna and M. C. Chaudhary. 2013. A survey on web page change-detection system using different approaches. *Int. J. Comput. Sci. Mobile Comput.* 2, 6 (June 2013), 294–299.

[105] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz. 1999. Analysis of a very large web search engine query log. *SIGIR Forum* 33, 1 (Sept. 1999), 6–12. DOI: https://doi.org/10.1145/331403.331405

[106] Sitemaps.org. 2008. sitemaps.org—Home. Retrieved from https://www.sitemaps.org/index.html.

[107] B. E. Smith. 2008. *Creating Web Pages For Dummies* (9th ed.). John Wiley & Sons Inc, New Jersey.

[108] K. Sproul. 2009. *The Dao of SEO (2009 ed.)*. Lulu.com, Morrisville, North Carolina.

[109] P. Srinivasan, F. Menczer, and G. Pant. 2005. A general evaluation framework for topical crawlers. *Info. Retriev.* 8, 3 (Jan. 2005), 417–447. DOI: https://doi.org/10.1007/s10791-005-6993-5

[110] Y. Sun, I. G. Councill, and C. L. Giles. 2010. The ethicality of web crawlers. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, Vol. 1. 668–675. DOI: https://doi.org/10.1109/WI-IAT.2010.316

[111] Y. Sun, Z. Zhuang, and C. L. Giles. 2007. A large-scale study of Robots.Txt. In *Proceedings of the 16th International Conference on World Wide Web (WWW'07)*. ACM, New York, NY, 1123–1124. DOI: https://doi.org/10.1145/1242572.1242726

[112] M. Thelwall and D. Stuart. 2006. Web crawling ethics revisited: Cost, privacy, and denial of service. *J. Amer. Soc. Info. Sci. Technol.* 57, 13 (2006), 1771–1779. DOI: https://doi.org/10.1002/asi.20388 arXiv: https://onlinelibrary.wiley.com/doi/pdf/10.1002/asi.20388

[113] Trackly. 2016. Trackly | Website Change Detection. Retrieved from https://trackly.io/.

[114] J. Umbrich, M. Hausenblas, A. Hogan, A. Polleres, and S. Decker. 2010. Towards dataset dynamics: Change frequency of linked open data sources. In *Proceedings of the 3rd International Workshop on Linked Data on the Web (LDOW'10), in Conjunction with 19th International World Wide Web Conference WWW'10)*. CEUR.

[115] H. Van de Sompel, M. L. Nelson, C. Lagoze, and S. Warner. 2004. Resource harvesting within the OAI-PMH framework. *D-Lib Mag.* 10, 12 (2004).

[116] H. Van de Sompel, R. Sanderson, M. Klein, M. L. Nelson, B. Haslhofer, S. Warner, and C. Lagoze. 2012. A perspective on resource synchronization. *D-Lib Mag.* 18, 9/10 (Sep. 2012).

[117] Versionista. 2007. Versionista: Monitor Website Changes. Retrieved from https://versionista.com/.

[118] Visualping. 2017. Visualping: #1 Website Change Detection, Monitoring and Alerts. Retrieved from https://visualping.io/.

[119] w3computing.com. 2017. Dynamic and Three-Dimensional Web Pages. Retrieved from https://www.w3computing.com/systemsanalysis/dynamic-three-dimensional-web-pages/.

[120] Wachete. 2014. Wachete—Monitor Web Changes. Retrieved from https://www.wachete.com.

[121] T. A. Walsh, G. M. Kapfhammer, and P. McMinn. 2017. ReDeCheck: An automatic layout failure checking tool for responsively designed web pages. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'17)*. ACM, New York, NY, 360–363. DOI: https://doi.org/10.1145/3092703.3098221

[122] Y. Wang, D. J. DeWitt, and J. Cai. 2003. X-Diff: An effective change-detection algorithm for XML documents. In *Proceedings of the 19th International Conference on Data Engineering*. 519–530. DOI: https://doi.org/10.1109/ICDE.2003.1260818

[123] J. L. Wolf, M. S. Squillante, P. S. Yu, J. Sethuraman, and L. Ozsen. 2002. Optimal crawling strategies for web search engines. In *Proceedings of the 11th International Conference on World Wide Web (WWW'02)*. ACM, New York, NY, 136–147. DOI: https://doi.org/10.1145/511446.511465

[124] WordPress. 2003. WordPress.com: Create a Free Website or Blog. Retrieved from https://wordpress.com/.

[125] D. Yadav, A. K. Sharma, and J. P. Gupta. 2007. Change detection in web pages. In *Proceedings of the 10th International Conference on Information Technology (ICIT'07)*. 265–270. DOI: https://doi.org/10.1109/ICIT.2007.37

[126] B. W. Yohanes, Handoko, and H. K. Wardana. 2011. Focused crawler optimization using genetic algorithm. *TELKOMNIKA* 9, 3 (Dec. 2011), 403–410.

[127] S. Zheng. 2011. Genetic and ant algorithms-based focused crawler design. In *Proceedings of the 2nd International Conference on Innovations in Bio-inspired Computing and Applications*. 374–378. DOI: https://doi.org/10.1109/IBICA.2011.98

[128] R. Zilberman. 2013. Check4Change. Retrieved from https://check4change.com.

Received December 2018; revised June 2019; accepted October 2019