Change Detection Optimization in Frequently Changing Web Pages

L. B. Meegahapola, P. K. D. R. M. Alwis,
L. B. E. H. Nimalarathna, V. G. Mallawaarachchi,
D. A. Meedeniya
Department of Computer Science & Engineering,
University of Moratuwa, Sri Lanka
{lakmalbuddikalucky.13, alwisroshan.13, eranga.13,
vijini.13, dulanim}@cse.mrt.ac.lk

Sampath Jayarathna
Department of Computer Science,
California State Polytechnic University
Pomona, CA 91768
ukjayarathna@cpp.edu

***Abstract*— Web pages at present have become dynamic and frequently changing, compared to the past where web pages contained static content which did not change often. People have the need to keep track of web pages which are of interest to them, using bookmarks in the web browser and continuously keep track of them in order to get the updates. Tracking changes which occur in these bookmarked web pages and getting updates has become a significant challenge in the current context. Hence there is a need for better and convenient methods to keep track of web pages. Many researches and implementations have been carried out in order to increase the efficiency of change detection of web pages and related algorithms. In this paper, we discuss the architectural aspects of change detection systems and introduce an optimized change detection solution including a web service, browser plugin and an email notification service. Consequently, this research will pave the way for a novel research area to explore.**

Keywords*— *Change detection; change notification; distributed architecture

I. INTRODUCTION

The World Wide Web (WWW) is growing at a quick pace and it has become a challenge to track changes because the content of the Web is changing continuously. In order to detect the changes, people had to retrieve web pages periodically, which was both inefficient and time consuming [1]. However, with the introduction of Change Detection and Notification (CDN) systems, users can easily get notified about changes to web pages without having them to retrieve these documents. At present many CDN systems have emerged to automate the process of change detection. Google Alerts [2] and Follow That Page [3] are two of the most popular change detection services available at present. However, because of the rapid growth in the number of websites being tracked and the vast number of users getting registered every day, current change detection systems have come across many performance issues [4], [5].

Change detection in web pages can be classified into two categories based on the architecture involved in the detection system. Namely they are Server side detection and Client side detection. In change detection systems where server side detection architecture is used, the processing load of the server increases when the server has to crawl a vast number of web pages. Furthermore, the detection frequency for a particular web page may be decreased as well. Scaling of such server based change detection services has become time consuming and expensive. The client side architectural approach utilizes client machines to poll the web pages and track their changes. When we review existing change detection systems based on client side detection, users with spare computing resources will be able detect frequent changes to web pages whereas, the remaining fraction of users may not detect these changes.

The above mentioned two architectures have their own strengths and limitations when considering certain aspects such as computational power, performance, speed of detection and time taken to detect. There is a clear need for an improved solution which optimizes the change detection mechanism and combines the advantages of both architectures while eliminating their weaknesses. This paper explores the limitations of existing change detection systems and proposes an efficient change detection solution.

This paper presents a solution with an optimized change detection system with a hybrid architecture. Section II discusses the related work in the area of CDN systems. Section III describes the system architecture of the proposed solution with its main features and Section IV explains the methodology including the experiments we have carried out. The obtained results discusses in Section V. Finally, Section VI concludes the paper with the importance of the work being carried out.

II. BACKGROUND

A wide range of research studies have been carried out on detecting changes of digital collections [4], [5], [6], [7]. Nadaraj [4], has described an approach for distributed content aggregation and change detection of web content utilizing client resources. In that approach, the consumers obtain the crawling data from a queue and run the crawlers on the working machine. It distributes the work among consumers while increasing the efficiency of the crawling mechanism. It also minimizes the coupling of spiders to particular machines, allowing them to operate freely in distributed networks and can be considered as a scalable content classification approach. Bloom filters have been used to detect the duplicate URLs and

978-1-5090-6491-5/17/$31.00 ©2017 IEEE

content in the site. However, bloom filters can only confirm that the crawled URL was not visited previously.

The work done in [5] has focused on classifying web pages in a digital collection and detecting the changes which have occurred in them. This system keeps track of the differences between two versions of a particular web resource within a digital collection in order to detect the changes which have occurred. As a study, they have mainly considered conference web sites for their data collection. Another interesting work is presented in [6] that detects changes in distributed and collaborative data collection. They have presented a Web Change Detection (WCD) system that detects the changes of the web pages and notifies them. This is primarily used by search engines in order to determine when web pages should be crawled and indexes should be built. They have considered PageRank values using *shash* tool to detect the near duplicates. This tool provides fast change detection with a low maintenance cost. However, if there are many requests at a given time, due to heavy usage, the system may not be able to process those in real time.

Architecture for a parallel crawling of the web pages using multiple machines and integrating the trivial issues of crawling is presented in [7]. They have provided a three-step algorithm for detecting web page changes. The server has a unique method for distribution of URLs to client machines after determination of their priority index. However, the clients are server nodes themselves. Hence the number of server nodes has to increase when scaling the system. This will yield in high costs as well.

ChangeDetection.com [8] and 'Follow That Page' [3] are popular free online services that provide change detection and notification facilities for online users. Users have to register to these services by providing the page URL and the email address where notifications should be sent. However, if a particular user has a vast number of web pages to track, many performance issues may arise and the user will not be notified regularly. Distill Web Monitor [9] (formerly known as AlertBox) is a Firefox plugin, which monitors web pages to identify changes and sends notifications by email and SMS. However, with this plugin, certain users might not detect the changes as frequently as the remaining fraction of users.

III. SYSTEM ARCHITECTURE

A. Overview

Many researches and implementations have been carried out in order to increase the efficiency of change detection of web pages and related algorithms such as tree comparison techniques [10] and diff algorithms [11]. It is evident from most of the recent research in this area that researchers are more focused on improving the efficiency algorithmically. Lack of focus on architectural aspects of change detection can be clearly seen when exploring the available change detection systems and modern research. To fill this void, we have come up with a new area of research to improve the architectural aspects of change detection and notification systems. We have created a hybrid solution combining server side and client side change detection systems where limitations of both the architectures are reduced. In creating this novel architecturally improved solution we used currently available change detection algorithms and methodologies. The main aim of this architecture is to ensure that the frequent changes are identified and clients are notified. This also makes sure that no intermediate changes are missed by the change detection and notification system.

B. Hybrid Architecture

Fig. 1, shows a high level view of our hybrid solution considered for this research. It comprises of three main components. First component is the Web application with web service. This is where a user can submit a link in which they want to track changes. The second component is the Web browser plugin for client side change detection. In fact we created a Google Chrome app for this purpose. The final component is the Email notification service which alerts users when a change is detected.

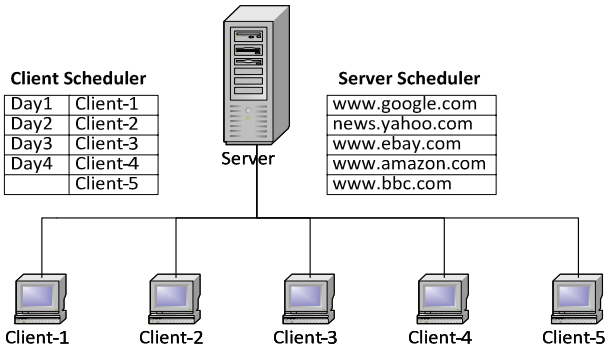


Fig. 1. High level view of the hybrid architecture.

According to the proposed hybrid architecture, the server has a main list of web pages to crawl. As shown in Fig. 1. this list contains web pages which were submitted by clients to track. When a new web page is added, this solution runs a server side algorithm to determine the change frequancy [12] of a particular web page using Poisson model [13]. Here, it is important to estimate the polling rate of a particular web page in order to detect changes. After this process, the web page will be added to the main list of web pages the server has to poll. However, the issue comes when the list increases. In order to keep up with the pace of change detection, the server needs to be scalable. This may result in many negative implications such as high cost and time consumption.

This approach uses client side detection to cater the above mentioned scalability issues. When a web page is added by a user, the server prepares a schedule for all the clients who have requested to track that particular page. When a new client requests to track an existing page, the server includes that client as well for the schedule of that particular web page. After preparing the schedule, the server communicates the polling times and time intervals to the clients. Fig. 1, describes this situation where a Client Scheduler with the clients who have subscribed to get updates for a particular web page is visible. After the scheduling process, the server goes through the list of web pages and tries to detect changes using a machine learning based algorithm [5], where it tries to detect both relevant and

irrelevant changes. If a relevant change is identified, the clients are notified through emails.

While the server is processing, the clients executes the process of identifying changes based on the schedule. However considering the processing power requirements, clients are only running a light weight change detection algorithm [11]. This is to ensure that the browser plugin does not use a high processing power. In case a change is found, the client plugin notifies the server and server runs the machine learning based algorithm to confirm the change and find both relevant and irrelevant changes. If relevant changes are found, the server will notify all the relevant clients through an email. With the new architecture, the time between two server polls is divided between clients until the time between two client polls reaches a minimum value. This ensures that the changes can be detected even in frequently changing web pages.

C. Change Detection

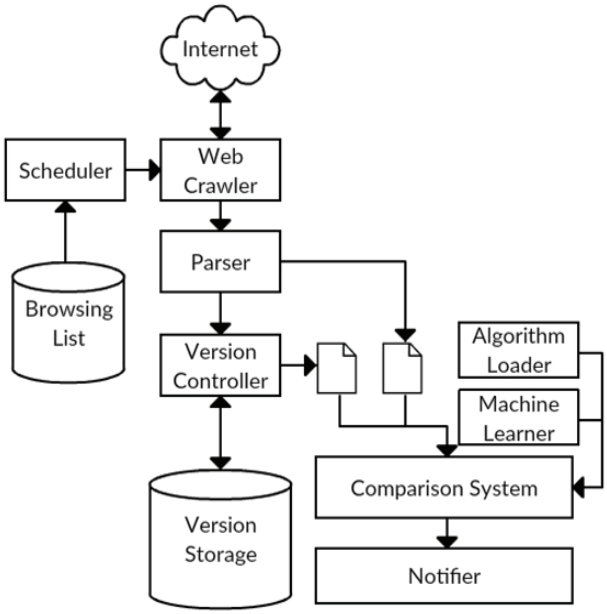


Fig. 2. Server side change detection mechanism.

Fig. 2, presents the work flow diagram of the server side in the proposed hybrid change detection solution. It explains how the components such as *Scheduler*, *Web Crawler*, *Parser* and *Notifier* work together to make the entire process efficient and effective.

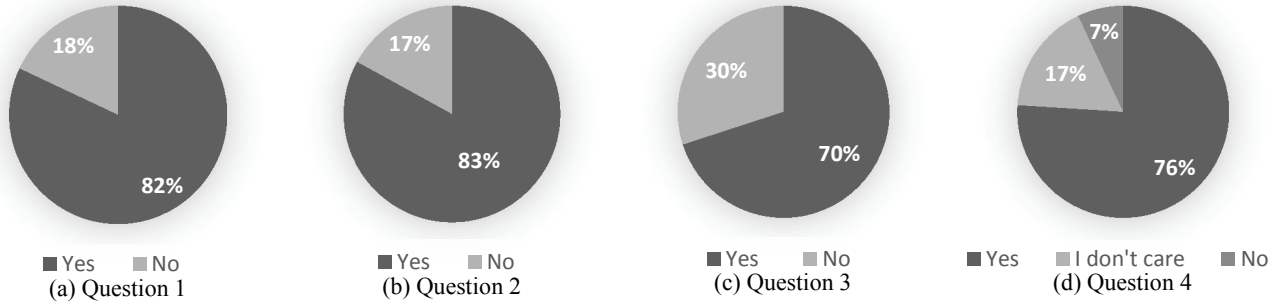
Given below is the terminology used in Fig. 2.

- Browsing List: List of URLs of web pages which were requested by the users to track and detect changes. Each of the URLs is mapped to a set of clients who have requested to detect changes in those web pages.
- Scheduler: Generates two main schedules; the schedule for clients who have requested to track a particular web page and the schedule for the server side detection.
- Web Crawler: Retrieves web content via Internet by crawling through those web pages.
- Version Controller: To detect changes, previous versions of a particular web page need to be there. Version controller is responsible for handling previous versions (handle CRUD operations).
- Version Storage: Collection of previous versions of the web pages in list of URLs.
- Algorithm Loader: Loads the necessary algorithms and modules to detect changes in the web pages. It uses the current version of a web page and the previous version from the Version Storage to detect changes.
- Machine Learner: Consists of the machine learning model which helps to classify the types of changes occurring in web pages.
- Comparison System: Evaluate changes happen in the comparison System or the comparison module.
- Notifier: Email notification system to notify clients who are interested in the changes of a particular web page.

IV. Methodology

A. Survey Statistics

Initially we conducted a survey [14] to understand the participants who are interested in detecting web page changes. The sample contained 200 randomly selected users. The results [15] show that out of the sample, 82% had the need to track changes in web sites as given in Fig. 3 (a) and 83% of them kept refreshing some page expecting a change to occur as shown in Fig. 3 (b). Only 70% of the participants have actually used a tool to track the changes, which is depicted in Fig. 3 (c). According to Fig. 3 (d), 76% of the participants liked to have an efficient tool to track changes in web pages.



(a) Question 1 (b) Question 2 (c) Question 3 (d) Question 4

Fig. 3. Answer analysis of the survey questions.

B. Implementation Procedure

When a client detects a change, that client will notify the server and then add the client request to the *User Generated Linked List*. While the server is executing the normal schedule, it runs a micro service to take care of the *User Generated Linked List*. This micro service detects the changes in the web pages in the *User Generated Linked List* using the machine learning based model and notifies the clients if any relevant change is presented. The *Web Crawler* downloads the current version of the web page to the server and the content of the web page parses into a HashMap by the *Parser*. *Version Controller* goes through the *Version Storage* and brings the old version of that particular web page, which is used by the *Comparison System* to detect any changes. *Machine Learner* facilitates to determine relevant changes and if the changes exceed the threshold, the clients will be notified through emails using the *Notifier*.

Given below is the terminology of Algorithm 1.

- time_gap: Optimal time gap between two crawls to detect meaningful changes for a particular link
- crawl_gap: Time gap between two crawls for a particular client for a particular web page.
- delay_to_start: Time that the client will wait after receiving the start command before starting its initial crawling for the new schedule.

According to the pseudocode shown in Algorithm 1, *Scheduler* executes at the beginning of the each iteration. It iterates links in the *Link Storage* and identifies the links with changes in their *Client Lists*. Then *Scheduler* generates schedules for those links. The *time_gap* allocated for each link decides the *crawl_gap* for a particular client. For each client in the *Client List*, the server initiates at different time points (using *delay_to_start*) and maintains the *crawl_gap* from their starting points until the next scheduling.

Algorithm 1 Scheduling Clients

Require: List of clients, Dataset of links
Ensure: Schedule clients to optimize change detection

```
1.  Start
2.      while link in link_storage
3.          if isClientsChanged(link) then
4.              client_list = getClientList(link)
5.              time_gap = getTimeGapFromLink(link)
6.              delay_to_start = 0
7.              crawl_gap = time_gap * size(client_list)
8.              for client in client_list:
9.                  notifyClient(client, link, crawl_gap,
    delay_to_start)
10.             delay_to_start = delay_to_start + time_gap
11. End
```

If the update frequency of a web page is high, it will determine by the Poisson model as described in [13] and the minimum time between two client polls will adjust accordingly to ensure that any changes are not missed by the change detection algorithm. This minimum time depends upon the change frequency and the number of clients available to detect changes of that particular web page. When considering a scenario with a large number of clients, still the server would schedule considering the optimal number of clients necessary to detect changes of the particular web page. Hence, there will not be an overhead for the system when the number of clients for a web page is increased.

C. Experimental Setup

For a selected 20 frequently changing websites, we executed a lightweight algorithm (O(n) time complexity) to detect whether the site was changed. If it has changed, we used a machine learning based algorithm (O(n2) time complexity) to provide more information on the relevancy of the change and to understand the type of changes occurred. The light weight algorithm first creates a tree structure of the web page and then computes a hash value for each node of that tree. It compares hash values of two different versions of the same website and identifies if there is any change. Furthermore, it could detect changed nodes, unchanged nodes, missing nodes and new nodes. Since the purpose of this paper is to describe the architectural aspect of change detection further details are not included.

The experiments were done on a cluster of 11 Virtual Machines (VMs) in Azure private cloud. Each VM had Linux Ubuntu (kernel 3.13.0-36-generic). 10 VMs (acted as clients) were running on 64-bit IntelTM Intel Xeon E312xx (Sandy Bridge) which operates at 2.70GHz. They had one CPU socket, with 2 cores each. L1(d/i) and L2 caches were 32KB and 4096KB respectively. Those had 4GB RAM with one 40GB hard disk. Server VM was running on 64-bit IntelTM Intel Xeon E312xx (Sandy Bridge) which operates at 3.4GHz. It had four CPU sockets, with 2 cores each. L1(d/i) and L2 caches were 64KB and 8192KB respectively and RAM was 16GB.

We implemented a java application to run on client VMs to model the client side architecture (CSA). Then clients crawl a particular website and detect changes. The crawler runs repeatedly to detect changes for a list of 10 web pages in order to model the server side architecture (SSA). The proposed hybrid architecture (HA) mentioned in Section III consists of clients equipped with the lightweight change detection algorithm and a server equipped with a complex algorithm that used machine learning to classify changes. Furthermore, the server has a scheduling algorithm to schedule clients to crawl a particular web page by taking turns and identifying whether changes have occurred or not.

We assigned the same web page used in CSA to 5 client VMs of HA. Then we compared the performance of a single client VM of HA with a client VM of CSA. We added 5 more VMs (5 more clients) and let them crawl the same web page, modelling the arrival of 5 new clients in HA. Ultimately, it would mean that 10 clients are subscribed to detect changes in that particular web page. This approach is used to compare the impact of the number of clients subscribed for a particular web page. We detected changes in a list of 10 web pages using both HA and SSA and compared the performance of the server. Next, we added 10 more new websites to track in HA and compared the arrival of new websites to HA to compare

the impact on performance when new web pages were introduced. In each test, we collected Java Flight Recorder [16] dumps to analyze CPU, memory and network usage.

V. Results and Discussion

We have measured the average resource utilization of both client and server architectures with the proposed hybrid architecture (HA). As shown in Table I, the client side architecture requires more resources than the proposed hybrid architecture, because each client has to crawl and detect changes on its own. In the proposed HA, when the number of clients subscribed for a particular web page doubled from 10 to 20, the detection frequency of a client got almost halved without affecting the frequency of the overall change detection as shown in Fig. 4 (a), (b) and (c). Ultimately, when more clients are added, the overall change detection frequency is increased, because the time between two server polls are divided among more clients to poll that particular website.

Resource utilization of the server has not deviated much when the architecture is changed from SSA to HA, which can be seen according to the results in Table II and Fig. 5 (a), (b) and (c). This is due to the server crawls through its website list repeatedly regardless of the number of web sites. Since clients crawl for changes in parallel to the server in HA, the frequency of detecting changes of a particular web page is still much higher than SSA. Another advantage in HA is that the server does not have to scale up in order to detect changes faster and reduce the time taken to process a list of websites. In real-world scenarios, since the number of web pages to be tracked can go beyond 100,000, this proposed method would be crucial.

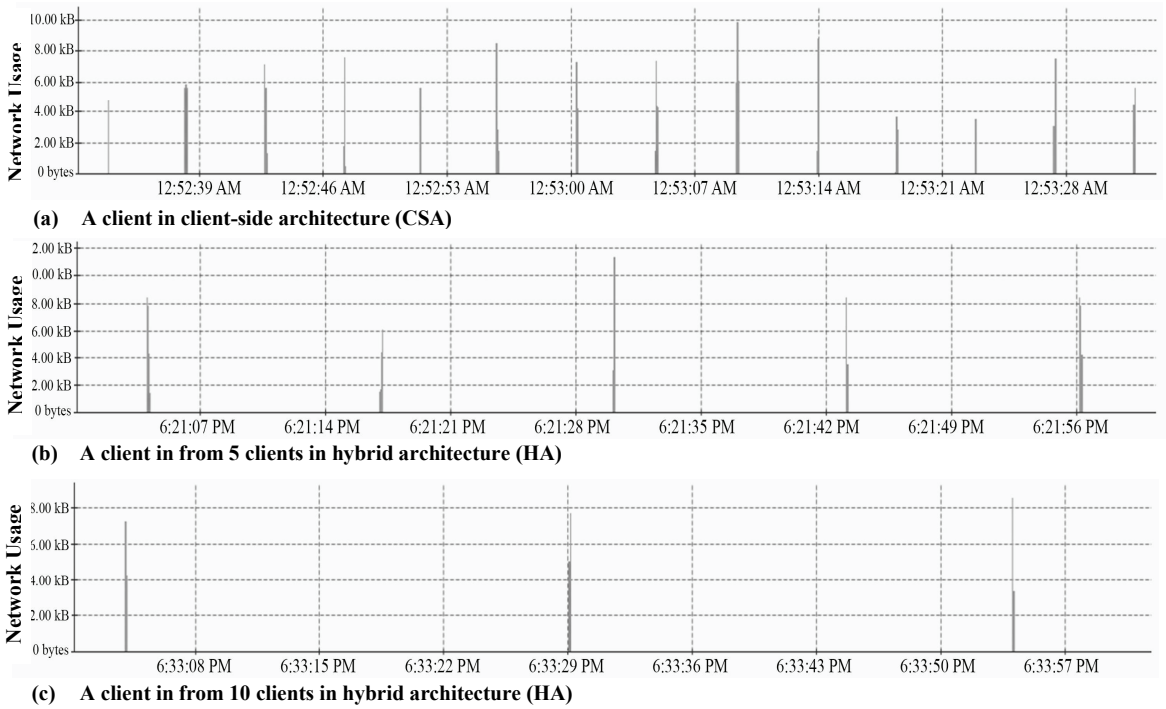
Although the results provided are for a set of 20 web pages and 11 virtual machines, the proposed solution can be easily scaled to cater a large number of web pages and devices, as it is supported by the design of the scheduler and the hybrid architecture. Furthermore, results clearly show that even when no client is online (worst-case scenario), it would provide good results as the server side detection system. However, when many clients use the system, above worst case scenario has a low probability to occur. Hence, when the system is used by many users, the system would work effectively all the time compared to any currently available system.

TABLE I. Average resource utilazation of a client in client side architecture vs. hybrid architecture

Average usage	Any Number of Clients (CSA)	5 Clients (HA)	10 Clients(HA)
CPU	3.84%	1.64%	0.966%
Memory	45MiB	39.4Mib	31MiB
Network	177,734 bytes	95,354 bytes	43,827 bytes

TABLE II. Average resource utilization of the server in server side architecture vs. hybrid architecture

Average usage	10 Web Pages (SSA)	10 Web Pages (HA)	20 Web Pages (HA)
CPU	31.1%	34.2%	31.1%
Memory	234MiB	234MiB	196MiB
Network	418,646 bytes	420,435 bytes	422,439 bytes



(a) A client in client-side architecture (CSA)

(b) A client in from 5 clients in hybrid architecture (HA)

(c) A client in from 10 clients in hybrid architecture (HA)

Fig. 4. Performance Analysis of Clients.

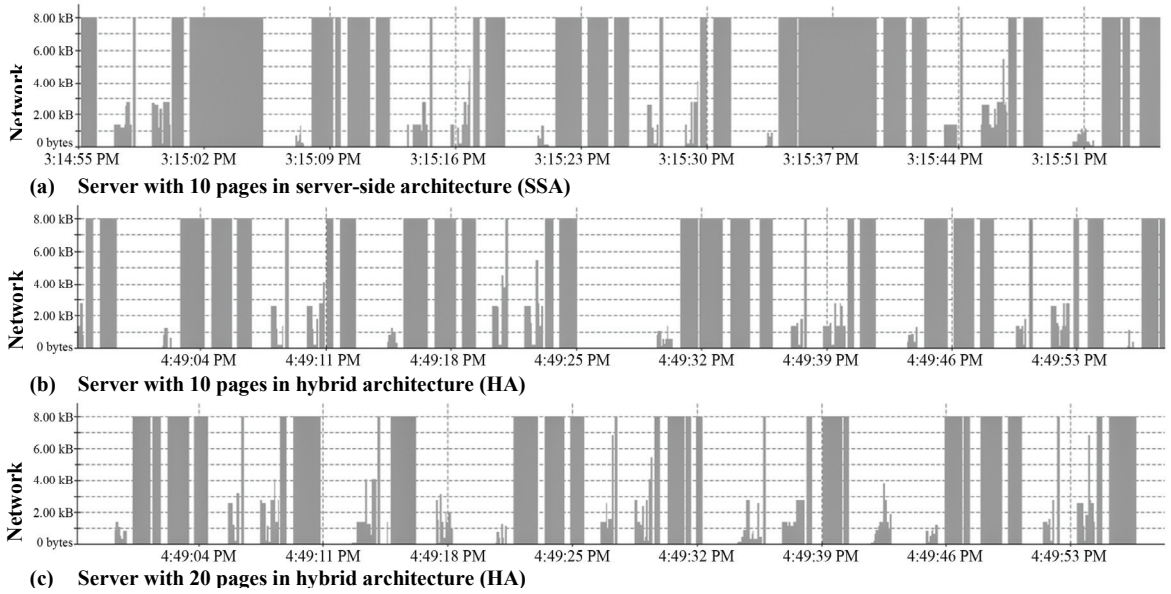


Fig. 5. Performance analysis of the Server.

VI. CONCULSION

Change Detection and Notification (CDN) systems have made a significant impact on the area of information retrieval by automating the process of change detection. However, still there are opportunities for improvement on the current implementations of CDN systems when considering performance and speed of detection. This paper has proposed a hybrid architecture that supports the optimization of current change detection systems. Survey responses have shown that people prefer more efficient means to keep track of the web pages. An experimental study was carried out to compare the performance of the proposed hybrid architecture. The results indicate that there is a significant improvement in the performance of change detection. This study has paved the way for a new area of research in the field of change detection using the proposed hybrid architecture.

REFERENCES

[1] S. Chakravarthy and S. Hara, "Automating Change Detection and Notification of Web Pages (Invited Paper)," in 17th International Workshop on Database and Expert Systems Applications, Krakow, Poland, 2006, pp. 465-469.

[2] "Google Alerts - Monitor the Web for interesting new content," [Online]. [Accessed 8 February 2017].

[3] "Follow That Page - web monitor: we send you an email when your favorite page has changed," [Online]. Available: https://www.followthatpage.com. [Accessed 8 February 2017].

[4] S. Nadaraj, "Distributed Content Aggregation& Content Change Detection using Bloom Filters," International Journal of Computer Science and Information Technologies,, vol. 7, no. 2, pp. 745-748, 2016.

[5] S. Jayarathna and F. Poursardar, "Change Detection and Classification of Digital Collections," in 2016 IEEE International Conference on Big Data, Washington D.C., USA, 2016, pp. 1750-1759.

[6] M. Prieto, M. A. lvarez, V. Carneiro and F. Cacheda, "Distributed and Collaborative Web Change Detection System," Computer Science and Information Systems, vol. 12, no. 1, pp. 91-114, 2015.

[7] D. Yadav, A. Sharma, J. Gupta, N. Garg and A. Mahajan, "Architecture for Parallel Crawling and Algorithm for Change Detection in Web Pages," in 10th International Conference on Information Technology, Orissa, India, 2007.

[8] "Change Detection - Know when any web page changes," [Online]. Available: https://www.changedetection.com/. [Accessed 4 Feburary 2017].

[9] "Distill Web Monitor," [Online]. Available: https://addons.mozilla.org/en-us/firefox/addon/alertbox/. [Accessed 4 February 2017].

[10] S. D. Jain and H. Khandagale, "A Web Page Change Detection System For Selected Zone Using Tree Comparison Technique," International Journal of Computer Applications Technology and Research, vol. 3, no. 4, pp. 254 - 262, 2014.

[11] Y. Wang, D. J. DeWitt and J. Y. Cai, "X-Diff: an effective change detection algorithm for XML documents," in 19th International Conference on Data Engineering, Bangalore, India, 2003, pp. 519-530.

[12] D. Ford, C. Grimes and E. Tassone, "Keeping a Search Engine Index Fresh: Risk and optimality in estimating refresh rates for web pages," in Proceedings of the 40th Symposium on the Interface: Computing Science and Statistics, Durham, NC, USA, 2008, pp. 1-14.

[13] C. Grimes and S. O'Brien, "Microscale Evolution of Web Pages," in 17th International World Wide Web Conference, Beijing, China, 2008, pp. 1149-1150.

[14] "Survey on Change Detection in Webpages," [Online]. Available: https://vijinim.typeform.com/to/dlnPdY. [Accessed 14 February 2017].

[15] "General report - Survey on Change Detection in Webpages," [Online]. Available: https://vijinim.typeform.com/report/dlnPdY/kcXx. [Accessed 5 March 2017].

[16] "Java Flight Recorder Runtime Guide," [Online]. Available: https://docs.oracle.com/javacomponents/jmc-5-4/jfr-runtime-guide/run.htm#JFRUH164. [Accessed 27 February 2017].