Prevention of Server Connection Timeout Errors and Downtime in Web Application Using Service Workers Caching Technique

[1]Adebayo, O. J., [2]Akanbi, R. Y., [3]Adenaike, I. O., and [4]Orimogunje, O. K.
[1]Department of computer science and engineering, Ladoke Akintola University of Technology, Ogbomosho, Nigeria.
[2]Department of computer science, Lagos State University of Science and Technology, Lagos, Nigeria.
[3]Department of computer science, ODL center, Ladoke Akintola University of Technology, Ogbomosho, Nigeria
[4]Department of computer engineering, The federal Polytechnic, Ede, Osun State, Nigeria.
***Corresponding Author -** Oluwaseun J. Adebayo, *adebayoseun109@gmail.com*

Abstract
Web applications today, are widely used by businesses to reach global audience. Ecommerce and other commercial applications that leverage business performance need to be given highest priority since their performance leads success. Unplanned downtime and server connection timeout error are some of the most prominent forms of failures in web service experienced by the web users, which could be as a result of network disruption or congestion of loads on the web server. This work adopted Service Worker (SW) caching technique to create a fault-tolerant web application that maintained page uptime in the event of network failure and prevented server time-out error. Two demRecords:b Application modules of Electronic Health Records; SW1 module and SW2 module of the same quality and contents were acquired from CLINIC365 domain. The Service Worker caching technique was designed with the four stages of JavaScript procedures which include SW registration, SW installation, SW return request, and SW cache updates. The designed SW caching technique was implemented in SW1 module. The performance of the acquired SW1 and SW2 modules were evaluated and compared using uptime, response time, and throughput. Throughput and response time were collected using GTMETRIX web analytic software, degree of uptimes were collected by establishing a network test with Google console tool network throttle. The performance of fault tolerant SW1 yielded uptime, response time, and throughput of 4 points, 2.6 seconds, and 16.2 Mbps. respectively, while SW2 yielded 0 point, 4.3 seconds, and 12.6 Mbps. This study showed that there was throughput gain with SW1 module, and its uptime was maintained compared to SW2 module. In addition, response time in SW1 was faster compared to that of SW2 which is a great advantage in the prevention of server congestion, and connection time-out errors.

Keywords: Service Worker, Web application, Caching, Web response time

1 Introduction

In modern society, information has now become a tangible resource for making critical decisions especially in time-sensitive environments, World Wide Web (WWW) directory is one major way in which this information can be disseminated. There's no doubt that web surfing has become the most predominant part of the internet usage today whose advantages can never be underestimated. The web started from a simple information sharing system, which has now grown into a rich collection of dynamic and interactive services. This tremendous growth of the web has resulted into high demand for high bandwidth and delay in fetching user request Neha *et al*. [1]. The undisputed advantage of WWW today has made it a common place for WWW users to be, which in turns has almost becoming a disadvantage as network congestion, service disruption, connection timeout, and unplanned server downtimes among others are now becoming more predominant on daily basis, most especially in third-world countries where the available network infrastructures do not have the capacity to sustain or serve the ever-growing numbers of web users. Users sometimes experience unpredictable delay while retrieving web pages from the server, mostly when fetching a very large page with a very poor network strength, which sometimes results in an unexpected server connection timeout error. Hence, the need for techniques or method in reducing the number of interruptions becomes the necessity. Many latencies tolerant techniques have been developed over the years to manage this performance bottlenecks. Most notably are

protocol, compression, caching, and prefetching among others, of all the listed approaches, caching promises the greatest performance gain, simply by storing copies of popular web objects closer to users rather than accessing them from the origin servers Pallis *et al*. [2]. This increases the response time, reduces the bandwidth utilization, and improves the general performance of web services. Many techniques have been used in the past in implementing a caching technology in web services. Cache Array Routing protocol (CARP), which divides the URL-space among an array of loose coupled caches and lets each cache store only the documents whose URL are hashed to it. Provey and Harrison [3] proposed a distributed Internet cache**.** In their scheme upper-level caches are replaced by directory servers which contain location hints about the documents kept at every cache. Tewari *et al*. [4] also proposed a similar approach to implement a fully distributed Internet caching system where location hints are replicated locally at the institutional caches. Gadde *et al*. [5], used central directory approach where central mapping service ties together a certain number of caches. In Cache mesh system Wang [6], cache servers establish a cache routing table among them, and each cache server becomes the designed cache server for several Websites. User requests are then forwarded to the proper cache server according to the cache routing table. In Summary Cache Fan *et al*. [7], and Cache Digest Rousskov and Wessels [8] caches inter-exchange messages indicating their content and keep local directories to facilitate finding documents in other caches.

A couple of works have been carried out by many researchers in web caching technology for effective web performance. Few of them are;

Pallis *et al*. [9] worked on a clustering-based prefetching scheme on a Web cache environment in order to reduce network resources consumed by Web services as well as the response time consumed by the Web users. They employed a graph-based clustering algorithm to identify clusters of "correlated" Web pages based on the users' access patterns. Evaluations showed that there was reduction in bandwidth utilization, response time (latency) of the web also reduced and increases the throughput. In 2016, Chen *et al*. [10] proposed Cache Optimizer, they adopted hibernate and caching mechanisms which leverages on readily available web logs to create mappings between a workload and database accesses, this was done to help developers optimize the cache configuration to improve the performance of large-scale web applications. Their evaluations were conducted on three open-source applications (Pet clinic, Cloud store, and Open-MRS) and find that Cache Optimizer improves the throughput by 27% - 138% and reduce the application's memory usage. In 2018, Zhao *et al*. [11] proposed empirically assessing opportunities for prefetching and caching in mobile apps, nine research questions were formulated with the focus on the prefetch-ability of HTTP requests and cache-ability of HTTP responses. Their results suggested that prefetching and caching can be useful across a wide range of mobile apps and scenarios, but they are not universally applicable, and their benefits will vary. Certain app categories are more amenable for prefetching and caching. However, there is a non-trivial amount of variation even among different apps within a single category. Xing and Song [12] investigated how to provide mobile content distribution with an end-to-end caching framework that spans the mobile core, mobile edge, and end devices. They worked on a request screening scheme and a request routing scheme. The request screening scheme selects appropriate devices to cache contents and matches other video requests with these local caches. The simulation results showed that, the proposed solutions can effectively offload traffic to device-to-device communications and balance the in-network traffic load to alleviate congestion. Compared to the reference schemes, their solutions achieve significant performance gain by exploiting fine-grained context information and engaging cross-domain collaboration.

Conclusively, all related works on Caching techniques failed to address uptime included in the major aim of this work. This research work adopted Service Worker caching technique, which is a new caching technology designed to replace HTML5 application cache, that was recently deprecated by World Wide Web Consortium (W3C). Service Worker is a JavaScript file that has

the capability to implement a push notification system, background synchronizations and offline experience. The main purpose of Service Worker was to create a progressive web application (PWA). But JavaScript programmers can modify it to sooth several use cases, like network control system among others, as it comes with loads of functionalities. In Section 2, research methodology was discussed. In Section 3, the results and discussion was established, then followed by the Concluding remarks.

2 Research Methodology

Acquisition of web application modules, design and implementations of service worker caching technique for this work was discussed in this section.

2.1 Acquisition of Module

Two EHR web application modules were acquired from clinic 365 domain to evaluate the effectiveness of the Service Worker caching technique. These two modules of EHR run on APACHE web server, the first module was tagged SW1, and the second module was tagged with SW2, both modules contain the same number of patients' records, with four navigations attached to each module. Each navigation displays a table for populating patient's record. The first navigation consisting of 50 records, second navigation contains 100 records, third navigation contains 150 records, and the fourth navigation contains 200 records. Each record comprises of nine variables which includes: surname, other names, card num, phone num, category, patient type, age, gender, and registration date. Three variables were removed from all the records which includes: surname, card num, and phone num, these fields were deactivated for the purpose of patient's identity.

2.2 Design of Service Worker Caching

The design of service worker caching technique involved four stages as depicted in Figure 1 of this work, which include registration, installation, request handling, and cache updates.

2.2.1 Registration

To design a SW caching technique for web application, there's a need to kick start the process by first **registering** it in the webpage. This process tells the browser where the service worker JavaScript file lives. The JavaScript **register()** method can be called every time a page loads without concern; the browser then figure out if the service worker is already registered or not, and handle it accordingly. The cache placement then depends on the developer's objective, for example, if the entire application is to be cached, then the placement of the service worker file has to be at the root of the domain. This means that the service worker's scope will be the entire origin. In other words, it means that, the service worker will receive fetch events for everything on the domain. For example, if service worker file is registered at /SW1/sw.js, then the service worker would only see fetch events for pages whose URL starts with /SW1/ (i.e. /SW1/page1/, /SW1/page2/…)

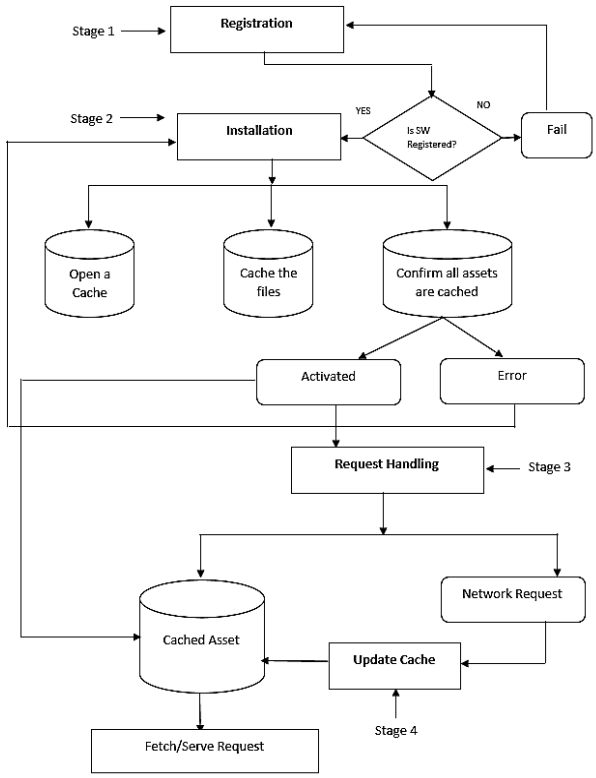

Figure 1: The Design of SW Caching Technique (Cache then Network strategy).

2.2.2 Installation

After a controlled page has kicked off the registration process, attention is then shifted to the point of view of the service worker script, which handles the **install** event. The callback for the **install** event was defined which made the decisions on what type of files are to be cached. **caches.open()** method was called with the desired cache name as depicted in Figure 2 of this work, **cache.addAll()** was then called and passed in the array of files. This is a chain of promises (**caches.open()** and **cache.addAll()**). The **event.waitUntil()** method took a promise and used it to know how long installation will take, and whether it is successful or not. When all the files were successfully cached, then the service worker was then installed. But if **any** of the files fail to download, then the install step will fail. This helps to rely on having all the assets that is defined. Within the **install** callback, the following steps were taken: open a cache; cache the files; and confirm whether all the required assets were all cached or not.

```
var CACHE_NAME = 'my-site-cache-v1';
var urlsToCache = [
 '/',
 '/styles/main.css',
 '/script/main.js'
];
self.addEventListener('install', function(event) {
 // Perform install steps
 event.waitUntil(
  caches.open(CACHE_NAME)
   .then(function(cache) {
    console.log('Opened cache');
    return cache.addAll(urlsToCache);
   })
 );
});
```

Figure 2: Installation of service worker on web application.

2.2.3 Request Handling

After a service worker was installed and the user navigates to a different page or refreshes, the service worker began to receive fetch events, an example of which was depicted in figure 3. From the code depicted in figure 3, the fetch event have been defined within **event.respondWith()**, and a promise is passed in from **caches.match()** method. This method looked at the request and finds any cached results from any of the caches created by service worker. If there's a matching response, the cached value will be returned, else, the result of a call to fetch will be returned, which will make a network request and return the data if anything can be retrieved from the network. The subsequent requests can be cumulatively cached, by handling the response of the fetch request and then adding it to the cache.

2.2.4 Updating Cached Assets

To prevent fetching a stale information from the page, the service worker JavaScript file needs to be instructed to constantly check the cached assets and compare it to the server assets, and make sure that they are perfectly matched. This process is done during the developmental stage. Whenever SW1 module is navigated through, the browser

tried to re-download the script file that defined the service worker in the background. If there is even a byte's difference in the service worker file compared to what it currently has, it will consider it *new*. The new service worker will be started, and the **install** event will be triggered. At this point the old service worker is still controlling the current pages so the new service worker will enter a waiting state. When the currently open pages in SW1 module are closed, the old service worker will be deactivated and the new service worker will take over. Once the new service worker takes control, its **activate** event will be triggered. One common task that will occur in the **activate** callback is the cache management so as to wipe out any old caches in the install step. **activate** event will be triggered. One common task that will occur in the **activate** callback is the cache management to wipe out any old caches in the install step.

```
self.addEventListener('fetch', function(event) {
  event.respondWith(
    caches.match(event.request)
      .then(function(response) {
        // Cache hit - return response
        if (response) {
          return response;
        }
        return fetch(event.request).then(
          function(response) {
            // Check if we received a valid response
            if(!response || response.status !== 200 || response.type !== 'basic') {
              return response;
            }
// Clone the response. A response is a stream and because we want the browser to
// consume the response as well as the cache consuming the response, we need to
// clone it so we have two streams.
            var responseToCache = response.clone();
            caches.open(CACHE_NAME)
              .then(function(cache) {
                cache.put(event.request, responseToCache);
              });
            return response;
          }
        );
      })
    );
});
```

Figure 3: Fetching a response from service worker (Request Handling).

2.3 System Evaluation

The performance of service worker caching technique was evaluated in this research work by comparing the SW1 EHR module that is cached with SW2 EHR module that is not cached.

GTMETRIX analytic software and Google console tool were used to collect performance statistics. Response time, uptime, and throughput are the metrics used for the comparisons.

For the web's content, four navigations were examined across each module as discussed in section 2.2 of this research work. These two EHR modules were carbon-copy of one another, i.e. the same page-size, the same number records per page, and their documents were fetched with the same network strength. The only difference between these EHR modules is that SW1 module was cached with service worker and the SW2 module was not.

(i) Response time and throughput of both EHR modules were measured using GTMETRIX analytic software

(ii) Degree of uptime/downtime of both EHR modules was measured using Google console tool network throttle

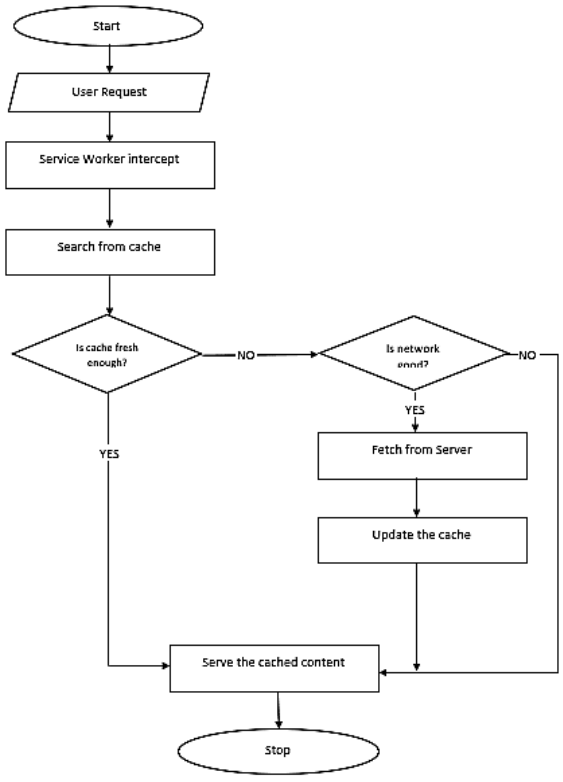

Figure 4: SW System Flowchart.

3 Results and Discussion

3.1 Evaluation Results on SW1 Web Application Module

The evaluation result on SW1 started with network test, which was achieved by disrupting the network connecting the SW1 module to the internet, this experiment was conducted on HP Intel(R) Core(TM) i3-10110U CPU running at @ 2.10GHz to 2.59 GHz processor speed on Windows 10 operating system, the average internet speed during the evaluation process was 680kbps. The evaluation was carried out by setting the network throttling option in Google console tool of chrome browser to offline, the purpose of this exercise was to establish whether SW1 will experience a downtime when network is set to offline. The result showed that SW1 module, which was designed using service worker's caching technique, maintained a constant uptime in the event of network failure, and has its uptime rating with 4 points across all the four navigations, one point representing a successful uptime achievement for each navigation. the experiment was repeated 5 times across each navigation, and the average score was taken across each variable. Response time and throughput were collected through GTMETRICS analytics tool, this experiment was also repeated 5 times across each navigation, and the average response time and throughput were taken across all navigations. The overall evaluation showed that using SW1 module, uptime, response time, and throughput were found to be 4 points, 2.6 seconds, and 16.2 Mbps respectively. Table 1 displayed the network test for SW1 module, Table 2 showed the results from SW1 displaying the response time, and throughput, while figure 5 showed the weighted rates and statistical metrics of SW1 module.

Table 1: SW1 Network test

S/N	Navigation	Page Size(KB)	Records	Network Test	Status
1	Nav1	241	50	pass	1
2	Nav2	287	100	pass	1
3	Nav3	328	150	pass	1
4	Nav4	368	200	pass	1

Table 2: SW1 Latency and Throughput (Performance test)

S/N	Navigation	Records	Page Size(KB)	Response time (s)	Throughput (Mbps)	Performance (%)
1	Nav1	50	241	0.86	2.30	98
2	Nav2	100	287	0.56	4.20	100
3	Nav3	150	328	0.57	4.71	100
4	Nav4	200	368	0.61	4.94	100
	Total	**500**	**1224**	**2.60**	**16.2**	**398**

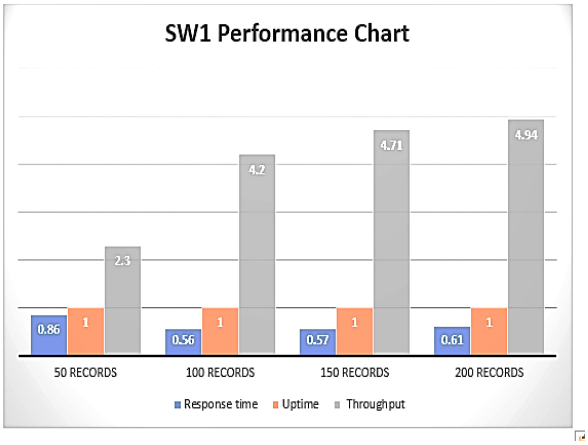

Figure 5: SW1 module's Performance Chart.

3.2 Evaluation Results on SW2 Web Application Module

The evaluation result on SW2 started with network test, which was achieved by disrupting the network connecting the SW2 module to the internet, this experiment was conducted on HP Intel(R) Core(TM) i3-10110U CPU running at @ 2.10GHz to 2.59 GHz processor speed on Windows 10 operating system, the average internet speed during the evaluation process was 680kbps. The evaluation was carried out by setting the network throttling option in Google console tool of chrome browser to offline, the purpose of this exercise was to establish whether SW2 will experience a downtime when network is set to offline. The result showed that SW2 module, which was designed without the service worker's caching technique, maintained downtime in the event of network failure, and has its uptime rating with 0point across all the four navigations, one point

representing a successful uptime achievement for each navigation, the experiment was repeated 5 times across each navigation, and the average score was taken across each variable. Response time and throughput were collected through GTMETRICS analytics tool, this experiment was also repeated 5 times across each navigation, and the average response time and throughput were taken across all navigations. The overall evaluation showed that using SW2 module, the uptime, response time, and throughput were found to be 0 point, 4.3 seconds, and 12.6 Mbps respectively. The SW2 module also showed a precision of 0% response time gain factor below SW1 module. Table 3 displays the network test for SW2 module, Table 4 showed the results from SW2 displaying the response time and throughput, and figure 6 showed the weighted rates and statistical metrics of SW2 module.

Table 3: SW2 Network Test

S/N	Navigation	Page Size(KB)	Records	Network Test	Status
1	Nav1	241	50	fail	0
2	Nav2	287	100	fail	0
3	Nav3	328	150	fail	0
4	Nav4	368	200	fail	0

Table 4: SW2 Latency and Throughput (Performance test)

S/N	Navigation	Records	Page Size(KB)	Response time (s)	Throughput (Mbps)	Performance (%)
1	Nav1	50	241	1.10	1.80	93
2	Nav2	100	287	0.84	2.80	95
3	Nav3	150	328	0.75	3.58	96
4	Nav4	200	368	0.68	4.43	96
	Total	**500**	**1224**	**4.30**	**12.6**	**380**

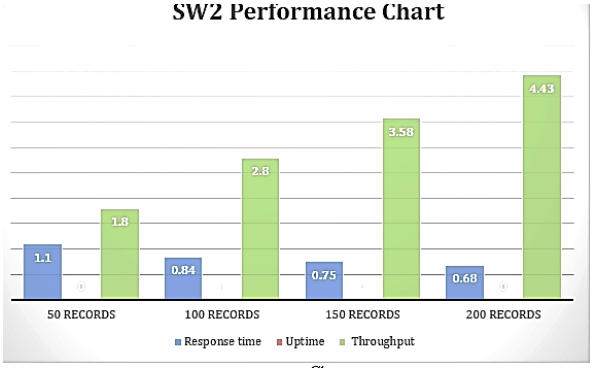

Figure 6: SW2 module's Performance Chart.

3.3 Discussion of Results of Evaluation on SW1 and SW2 Modules

SW1 module is assumed to be better than SW2 module considering the response time, uptime, and throughput. This means that 500 records of 1.3MB webpage can be fetched using SW1 with a response time of 2.6 seconds and gain a throughput of 16.2 Mbps, likewise the same 500 records of 1.3MB webpage can be fetched using SW2 with a response time of 4.3 seconds and gain a throughput of 12.6 Mbps, meaning that, user spend less time in fetching a page with SW1, and much work done is acquired in lesser time compared to SW2 module. Also, the degree of uptime with SW1 module was stable with 4points compared to SW2 whose degree of uptime was not maintained having recorded 0 point across all the four navigations. This implies that downtime can be avoided with SW1, user browsing with SW1 designed module will enjoy availability of page even in the event of network disruptions.

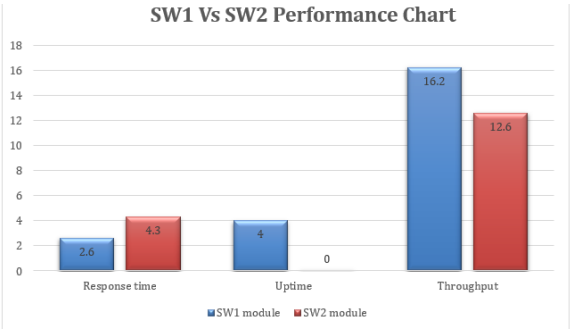

Figure 7: SW1 module against SW2 module Performance Chart.

Conclusions

A performance evaluation of two different web application modules SW1 and SW2 were carried out to analyse the efficiency of service worker caching technique in preventing connection timeout errors and downtime in web application. The SW1 and SW2 modules were evaluated, and compared using uptime, response time, and throughput. The performance evaluation showed that SW1 module that has the implementation of service worker caching technique performed better than SW2 module that is without service worker's implementation. The study also showed that the 500 records of patient's information collected from CLINIC365 domain when rendered on the browser using SW1 and SW2 modules, the results are as follow: degree of uptime of 4 points and 0 point respectively, response time of 2.6 seconds and 4.3 seconds respectively, and throughput of 16.2 Mbps and 12.6 Mbps respectively. SW1 module was able to maintain uptime in the event of network failure while SW2 experienced constant downtime. This study also showed that there was throughput gain with SW1, and its response time was faster compared to that of SW2 module, which is a great advantage in the prevention of server congestion, and connection time-out errors.

Conclusively, it was gathered that service worker caching was able to improve the overall web performances of clinic 365 module through: uptime maintenance, latency reduction and throughput gain.

References

[1] Neha K, Esha D, Neha S, Proposed Framework for the Reduction of Web Congestion using Classification, *International Journal of Engineering and Advanced Technology*, 3(2), 2013, 123-128.

[2] Pallis G, Athena V, Jaroslav P, A Clustering-Based Prefetching Scheme on a Web Cache Environment, *Computers and Electrical Engineering*, 34, 2008, 309-323.

[3] Povey D, Harrison J, A distributed Internet cache, Proceedings of the 20th Australian Computer Science Conference, Sydney, Australia, Feb. 1997.

[4] Tewari R, Dahlin M, Vin H, Kay J, Beyond hierarchies: Design considerations for distributed caching on the Internet, Technical Report TR98-04, Department of Computer Science, University of Texas at Austin, February 1998.

[5] Gadde S, Rabinovich M, Chase J, Reduce, reuse, and recycle: an approach to building large Internet caches, *Proceedings of the HotOS'97 Workshop*, May 1997 [http://www.cs.duke.edu/ari/cisi/crisprecycle/crisp-recycle.htm].

[6] Wang Z, Cache mesh: a distributed cache system for World Wide Web, Web Cache Workshop, 1997.

[7] Fan L, Cao P, Almeida J, Broder AZ, Summary cache: A scalable wide-area Web cache sharing protocol, *Proceedings of Sigcomm*, 1998.

[8] Rousskov A, Wessels D, Cache Digest, *Proceedings of 3rd International WWW Caching Workshop*, June 1998.

[9] Pallis G, Athena V, Jaroslav P, A Clustering-Based Prefetching Scheme on a Web Cache Environment, *Computers and Electrical Engineering*, 34, 2008, 309-323.

[10] Chen TH, Shang W, Hassan AE, Nasser M, Cache Optimizer: Helping Developers Configure Caching Frameworks for Hibernate-Based Database-Centric Web Applications, Conference: The 2016 24th ACM SIGSOFT International Symposium

[11] Zhao Y, Wat P, Laser MS, Medvidović N, Empirically Assessing Opportunities for Prefetching and Caching in Mobile Apps., *International Conference on Automated Software Engineering At: Montpellier, France*, 2018. DOI:10.1145/3238147.3238215.

[12] Xing H, Song W, Collaborative Content Distribution with an End-to-End Caching Framework, *IEEE Access*, 2020, PP (99):1-1 DOI:10.1109/ACCESS.2020.2981665.