Development of Web Caching Replacement in Internet Service Based on GDSF

Nguyen Xuan Truong, Ho Khanh Lam

Abstract*: *This paper presents a policy to replace the cached web content named GDSF-EXT based on GDSF by adding an extensible cache located in the network device. In processing, these web contents will be retrieved instead of having to search in other devices on the same network layer or at a higher network layer. That helps to reduce the response time of the user's request. The proposed algorithm is compared to GDSF original to evaluate the performance of the network

Keywords*: *Web Cache, Internet Web Caching, Cache Replacement Algorithms, GDSF.

I. INTRODUCTION

In recent years the rapid growth of Web-based services being used by people all over the world via smartphones and computers has led to a considerable exponential increase in the amount of Internet traffic. Web caching is one of the best solutions for improving the performance of Internet Web-based services. Web caching is the temporary storage of Web objects frequently referred to by clients on system caches located closer to the clients, thereby helping to reduce network bandwidth usage and reduce response delay to users which will increase the service quality. It is very useful for the development of high-bandwidth streaming and multimedia services on the Internet and reducing the load on Origin web servers. Web caching is used in a browser of the client machine (client-side caching), the proxy server (proxy caching), and/or the origin server (server-side caching). Client-side caching refers to caches that are built into most web browsers, which caches Internet objects for a single user but from a variety of servers. In server-side caching (also known as reverse caching) Web objects can be stored in a cache that is placed in front of a particular server for reducing the redundant computations and the server load. The proxy cache is found in the proxy server, which serves as an intermediary between users and web servers. The user's request is forwarded to the webserver by the proxy server. When the webserver returns the requested resource to the proxy server, the proxy stores a copy in its proxy cache such that further requests to the same resource by the same user or another user are met at the proxy without contacting the webserver again. Web proxy caching plays a key role in improving Web performance by keeping Web objects that are likely to be visited again in the proxy server close to the user.

This Web proxy caching helps in reducing user-perceived latency, i.e. delay from the time a request is issued until the response is received, reducing network bandwidth utilization, and alleviating loads on the original servers. As Web proxy cache size is limited, a cache replacement policy is needed to handle the cached content. If the cache is full when an object needs to be stored, the replacement policy will determine which object is currently in the cache to be evicted to allow space for the new object. The optimal replacement policy aims to make the best use of available cache space, improve cache hit rates, and reduce loads on the origin server. Therefore, many web cache replacement policies have been proposed based on some important factors of Web objects that can influence the replacement process [1] like recency (recency based policies - LRU), frequency (frequency-based policies - LFU), size (size-based policies -SIZE), cost of fetching *th* object, access latency of object. These factors can be incorporated into the replacement decision. Depending on these factors, the traditional replacement caching policies can be classified into five categories: Recency-based policies; Frequency-based policies; Size-based policies; Randomized policies; Function-based policies [1-8].

LFU-DA and LRU were enhanced using supervised machine learning such as SVM, NB and C4.5 classifier in works [9, 10]. SVM, NB and C4.5 are trained from Web proxy logs files and then intelligently incorporated with LFU-DA to form Intelligent Dynamic Aging (DA) approaches. The proposed intelligent Dynamic Aging approaches considerably improved the performances in terms of HR and BHR of the conventional LFU-DA on a range of real datasets. supervised machine learning techniques are more effectively utilized in the GDS and GDSF to obtain optimal and intelligent Greedy-Dual approaches that can perform better in terms of both bytes HR and BHR.

Function-based policies generally associate each object in the cache with a utility value. The value is calculated based on a specific function incorporating different factors such as time, frequency, size, cost, latency, and different weighting parameters. The object with the smallest value is evicted first. Greedy-Dual-Size (GDS) [11] is the representative policy of the Function-based category. To get better BH and BHR parameters, Greedy-Dual-Size (GDS) seems to be the best BH and BHR [12, 13]. It uses an extension of the SIZE policy. The algorithm combines several factors and assigns a key value for each object stored in the cache. When cache space becomes saturated and a new object is required to be stored in the cache, the object with the lowest key value is removed [5, 6, 8, 9].

Manuscript received on February 22, 2022.
Revised Manuscript received on February 28, 2022.
Manuscript published on March 30, 2022.
* Correspondence Author

Nguyen Xuan Truong*, Hung Yen University of Technology and Education, Hung Yen, Vietnam. Email: truongnx@utehy.edu.vn
Ho Khanh Lam, Hung Yen University of Technology and Education, Hung Yen, Vietnam. Email: hokhanhlam@utehy.edu.vn

GDS removes objects which are no longer requested by users and therefore overcomes the drawbacks of the SIZE policy. But GDS does not take the previous frequency of access for web objects in the account. And Greedy-Dual-Size-Frequency (GDSF) is extended of GDS by containing the access frequency aspect in assigning key value, which is helpful in the web cache replacement algorithm [14]. Nowadays, the hardware becomes more and more perfect, special the memories is bigger. To upgrade the system that already exists based on GDSF, we propose a new method named GDSF-EXT that uses two memories: main memory (main caching in original GDSF) and a second memory for new caching (extended caching). Thus, GDSF-EXT has more cache than the original GDSF, so it could improve the performance of hits with different frequencies. To evaluate the proposed algorithm, a programing writing in C# is developed and shows the ratio of the hit for each step of request. GDSF-EXT shows that are higher performance than GDSF in both theoretical and simulation.

This paper is organized into four parts: system overview, development of our proposal named GDSF-EXT, simulation and calculations to show the performance of algorithms and conclusion.

II. SYSTEM OVERVIEW

A. Submission of the paper

In the Hierarchical Web Caching architecture [15], we choose a hybrid architecture and analyze and evaluate the performance using a queueing model with cache levels: Institution Caches (IC), Regional Caches (RC), Central Caches (CC), Origin Caches (OC) [16].

Typically, ICs connect at the local Point-Of-Presence (POP) nodes, RCs connect at regional nodes, and CCs connect in the national backbone and OCs in the origin network Web Servers. Cache Engine systems could install for ICs, RCs, CCs, and OCs. Thus, the cache systems must implement the algorithm to replace the Web Cache distributed layer by layer and between layers.

The average response time for HTTP access in an ISP's layered Web caching architecture has been suggested in our previous publication [16]:

$$E[R_{WC}] = E[R_{3H}] + (Miss_3)(E[R_{2H}] + (Miss_2)(E[R_{1H}] + (Miss_1)(E[R_{0H}]))) \quad (1)$$

Where $E[R_{3H}], E[R_{2H}], E[R_{1H}], E[R_{0H}]$ - Average access response time of respective cache levels: IC, RC, CC, and OC; $Miss_3, Miss_2, Miss_1$ - Cache miss ratio at respective cache levels: IC, RC, CC, and OC.

We propose the process of responding to HTTP requests of the ISP network with the hybrid Web caching architecture shown in Figure 1. Herein, the IC response process is shown by the process at the RC, CC, and similar to OC levels. Given that the IC has n Proxy Caches (from Proxy Cache 0 to Proxy Cache n-1), where Proxy Cache 0 is closest to the HTTP request and Proxy Cache n-1 (Root-level Cache proxy) is farthest away from the HTTP request. The priority of object search for an HTTP request at the IC is to look at the peer level (from Proxy Cache 0 to Proxy Cache n-1). Only when missed at the IC will the HTTP request be forwarded to the RC. A similar way is applied for RC and CC, CC and OC.

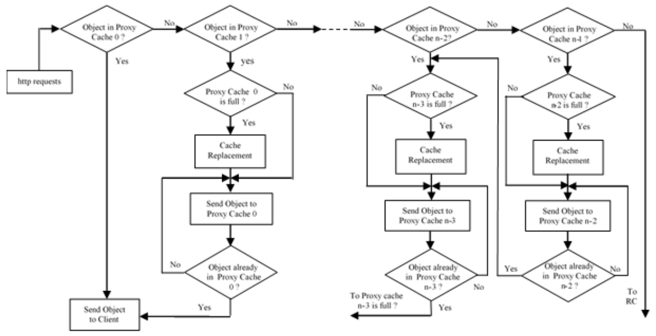

http requests
Object in Proxy Cache 0 ?
No
Yes
Object in Proxy Cache 1 ?
No
yes
Proxy Cache 0 is full ?
No
Yes
Cache Replacement
Send Object to Proxy Cache 0
Object already in Proxy Cache 0 ?
No
Yes
Send Object to Client
No
Object in Proxy Cache n-2?
No
Yes
Proxy Cache n-3 is full ?
No
Yes
Cache Replacement
Send Object to Proxy Cache n-3
Object already in Proxy Cache n-3 ?
No
Yes
To Proxy cache n-3 is full ?
Object in Proxy Cache n-1 ?
No
Yes
Proxy Cache n-2 is full ?
No
Yes
Cache Replacement
Send Object to Proxy Cache n-2
Object already in Proxy Cache n-2 ?
Yes
No
To RC

Figure 1. Find object (Web page) in IC for Client HTTP request

Set HR (Hit Rate) cache hit rate to *H* and the average response of each Proxy Cache is *R*. Then for the IC cache level, there are corresponding values: $H_{IC0}, R_{IC0}, H_{IC1}, R_{IC1}, .., H_{ICn-1}, R_{ICn-1}$ .Similar to RC and CC $H_{RC0}, R_{RC0}, H_{RC1}, R_{RC1}, .., H_{RCm-1}, R_{RCm-1}$, $H_{CC0}, R_{CC0}, H_{CC1}, R_{CC1}, .., H_{CCk-1}, R_{CCk-1}$.

For OC, the probability of the OC miss is always 1, (H_{OC} =1, $Miss_0$ =0). So we do not consider OC in the following formulas. The average response of each cache level is:

$$E[R_{IC}] = E[R_{3H}] = R_{IC0} + (1 - H_{IC0})(R_{IC1} + (1 - H_{IC1})(R_{IC2} + (1 - H_{IC2})(R_{IC3} + ...))) \quad (2)$$

$$E[R_{RC}] = E[R_{2H}] = R_{RC0} + (1 - H_{RC0})(R_{RC1} + (1 - H_{RC1})(R_{RC2} + (1 - H_{RC2})(R_{RC3} + ...))) \quad (3)$$

$$E[R_{CC}] = E[R_{1H}] = R_{CC0} + (1 - H_{CC0})(R_{CC1} + (1 - H_{CC1})(R_{CC2} + (1 - H_{CC2})(R_{CC3} + ...))) \quad (4)$$

$$E[R_{OC}] = E[R_{0H}] = R_{OC0} + (1 - H_{OC0})(R_{OC1} + (1 - H_{OC1})(R_{OC2} + (1 - H_{OC2})(R_{OC3} + ...))) \quad (5)$$

Combination Eqs. (1-5), we have

$$Miss_3 = \prod_{i=0}^{n-1}(1 - H_{ICi}); Miss_2 = \prod_{i=0}^{m-1}(1 - H_{RCi}); Miss_1 = \prod_{i=0}^{k-1}(1 - H_{CCi}) \quad (6)$$

The larger the number of Web caches (Proxy Cache), the higher the cache hit ratio (HR) of each Web cache, the smaller the miss rate ($Miss_i$) at each cache level. This depends on factors such as the size of Web objects, the capacity of the Web Cache systems, the cache replacement algorithm, the structure of the entire cache level (IC, RC, CC, and OC) on Internet Web Caching architecture.

In fact, depending on the population index of each region, the development of fast mobile communication services, and the young population, according to Zipf [17]: Web proxy cache systems set up there should have an investment in capacity and capacity to meet demand. So, even though they have the same network level, the Web proxy cache systems will differ in capacity and capacity because the number of popular websites is different, and the sizes of Web objects are also different. In addition, some Web sites may at one point go unnoticed by users and are susceptible to being superseded by a cache replacement policy, but later on, they can be referred to by an explosion of references. or need to be accessing history for important groups of users. At that time, these websites have to search the Internet on other Web cache systems, which are unlikely to exist.

III. DEVELOPMENT OF GDSF-EXT ALGORITHM

GDS algorithm used the cost of fetching an object, the size of an object, and an aging factor to compute the key-value following Eq. (7).

K(i) = L+ C(i)/S(i) (7)

where *C(i)* is the cost of fetching object *i* from the server into the cache; *S(i)* is the size of object *i*; *L* is an aging factor. *L* starts at 0 and is updated to the key value of the last replaced object. The key value *K(i)* of object *i* is updated using the new *L* value since object *i* is accessed again. Thus, larger key values are assigned to objects that have been visited recently. If the cost is set to 1, it becomes GDS (1), and when the cost is set to *P* = 2+size /536 [11], it becomes GDS(P).

While Greedy-Dual-Size-Frequency (GDSF) is extended of GDS by containing the access frequency aspect in assigning key value. GDSF has considered the access frequency as in Eq. (8).

$$K(i) = L+F(i)*(C(i)/S(i)) \tag{8}$$

where *F(i)* is the frequency of the visits of object *i*. Initially, when object *i* is requested by the user, *F(i)* is initialized to 1. If object *i* is in the cache, its frequency is increased by one. Similar to GDS, we then have GDSF(1) and GDSF(P). GDSF has the best BHR. However, GDSF does not predict future accesses. GDS and GDSF are used effectively in Root-level proxies.

Our algorithm proposes called GDSF-EXT to overcome this drawback by including in each Web cache system a LEWC extended local web cache (Local Extended Web Cache) to temporarily store discarded Web objects when executing GDSF. For Proxy Cache levels GDSF algorithm gives a good HR and BHR ratio. However, the fact that the variety of Web objects, especially the contents of multimedia services, does not make these algorithms highly efficient. Because at one time a Web object i is considered to have the least access frequency with the smallest K(i) value, it is discarded, but at other times it has a high access frequency. Or conversely the object at a time object i is considered to have large K(i) so it is cached, but then it has the smallest K(i) and is discarded.

The GDSF-EXT implementation process is as follows (Fig. 2): when you slide the first object in the Web cache, you must search the LEWC extended cache to see if any previous objects have been replaced that match the HTTP request. If so, it's hit the Web cache. Only when not in LEWC should the HTTP request be forwarded to the next Web cache at the same level. Figure 2 shows the implementation of the GDSF-EXT replacement algorithm for the Web Cache case at IC levels: IC0 and IC1.

When performing a replacement, first finding the least-referenced region of a size that fits the replacement (not necessarily the last), the replaced object is written to LEWC, and a new Web object from neighboring web cache is moved into the alternate zone. If not found in the less-referenced region, then search outside this area from the beginning (by SIZE), which means that the replacement object is large. The replaced object will also be written to the LEWC. In case there is no area of sufficient size, two neighboring regions must be selected instead. The proposed algorithm's HR needs to include web objects that are replaced but written to LEWC and then passed back to the Web cache for HTTP request responses.

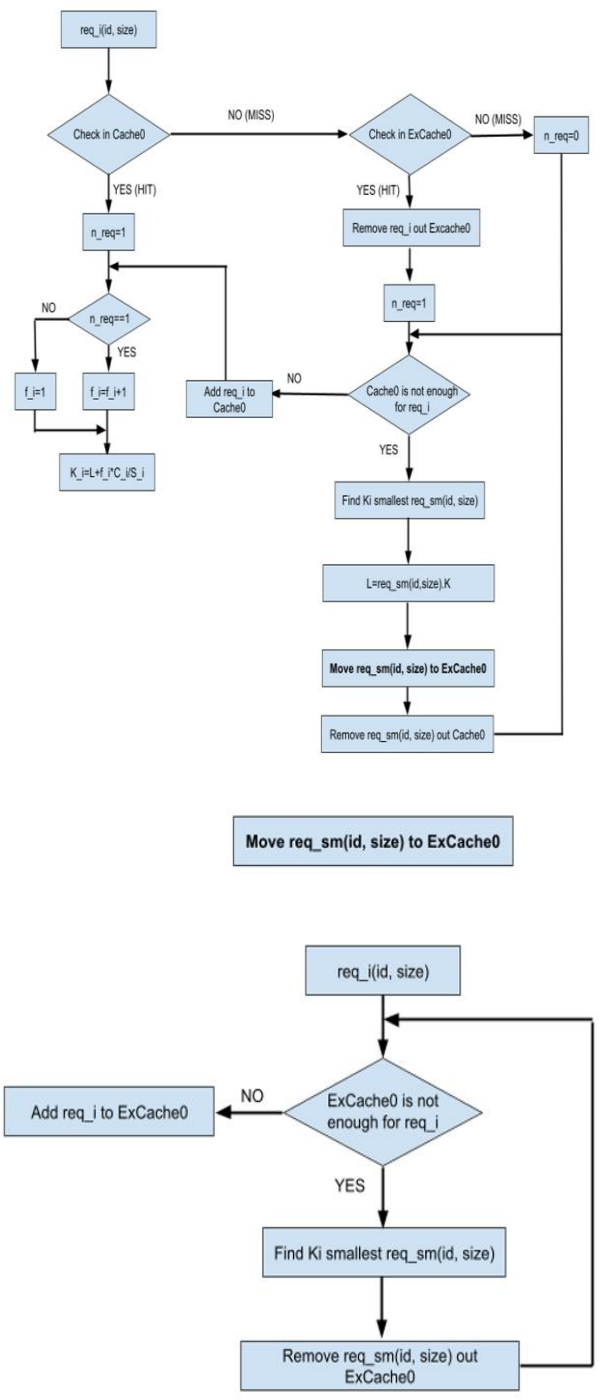

req_i(id, size)
Check in Cache0
NO (MISS)
Check in ExCache0
NO (MISS)
n_req=0
YES (HIT)
n_req=1
YES (HIT)
Remove req_i out Excache0
n_req=1
NO
n_req==1
YES
f_i=1
f_i=f_i+1
Add req_i to Cache0
NO
Cache0 is not enough for req_i
YES
K_i=L+f_i*C_i/S_i
Find Ki smallest req_sm(id, size)
L=req_sm(id,size).K
Move req_sm(id, size) to ExCache0
Remove req_sm(id, size) out Cache0
Move req_sm(id, size) to ExCache0
req_i(id, size)
Add req_i to ExCache0
NO
ExCache0 is not enough for req_i
YES
Find Ki smallest req_sm(id, size)
Remove req_sm(id, size) out ExCache0

Figure 2. GDSF-EXT algorithm

Based on Fig. 2, the cache hit ratio of the GDSF-EXT and GDSF algorithms are calculated as:

$$HR_{GDSF-EXT} = \frac{HN_{in}C + HN_{in}LEWC}{TN} = \frac{(Number\ of\ hit\ obj\ in\ web\ cache) + (Number\ of\ hit\ objects\ in\ LEWC)}{Total\ number\ of\ requested\ objects} \tag{9}$$

$$HR_{GDSF} = \frac{HN_{in}C}{TN} = \frac{Number\ of\ hit\ objects\ in\ web\ cache}{Total\ number\ of\ requested\ objects} \tag{10}$$

From Eqs. (9) and (10), it is clear that $HR_{GDSF-EXT} > HR_{GDSF}$ when the same value of $HN_{in}C$.

In addition, the web objects are retrieved directly from LEWC in response to HTTP requests. The system will make small the response delay, and the replacement algorithm is fast. The process of replacing caches in Web Caches is similar at all cache levels. The disadvantage of GDSF-EXT is that it requires additional cache expansion to store the replaced Web objects. However, the problem of memory capacity is easily solved with current memory technology.

To demonstrate that the GDSF-EXT algorithm has high performance for many web caches at each cache level in the internet web caching architecture of the ISP network which will be necessary to simulate and test a large number of HTTP requests and calculation Eq (1).

Cache replacement algorithm GDSF-EXT

Input: the sequence of web objects requested: $w_1, w_2, \ldots, w_s$
Cache content queue: Top: web object with the maximum value $K(I_k)$, bottom: web object with the minimum value $K(I_1)$
Ext-cache: store web object evicted from cache: $R(w_i), \ldots$
Output: the processing of each object by the cache replacement algorithm, and then return the requested object sequence.

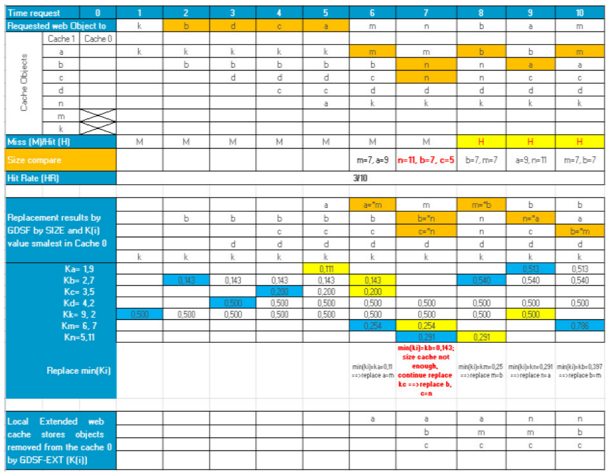
```
1.  Initialize $L \leftarrow 0$;
2.  **For** each object $w_i$ in $w_1, w_2, \ldots, w_s$ $F(w_i)$ is initialized to
    1 if object $w_i$ is requested by the user is not in the cache
3.      **If** $w_i$ is already in the cache then
4.          $K(w_i) = L+F(w_i)*(C(w_i)/S(w_i))$;
5.      **End if**;
6.      **If** $w_i$ is not in cache then
7.          **If** $w_i$ is not in Ext-cache then
8.              **While** there is not enough room in the cache
    for $w_i$ ;
    **If** $S(I_i) >= S(w_i)$ and $K(I_i)$ is the minimum value of all
    the objects in the cache then evict $I_i$ from cache and
    store $I_i$ into Ext-cache
9.              **End while**;
10.             Bring $w_i$ into cache;
11.             $K(w_i) = L+F(w_i)*(C(w_i)/S(w_i))$;
12.         **If** $w_i$ is in Ext-cache then
13.             **While** there is not enough room in the cache
    for $w_i$ ;
    **If** $S(I_l) >= R(w_i)$ and $K(I_i)$ is the minimum value of all
    the objects in the cache then evict $R_l$ from the cache and
    write $I_i$ from Ext-cache into the cache
14.             **End while**;
15.             Bring $I_i$ into cache;
16.         $K(I_i) = L+F(I_i)*(C(I_i)/S(I_i))$;
17. **End For**;
18. Return to requested web object sequence;
```

IV. SIMULATION AND CALCULATIONS

A. Illustrating The Implementation GDSF-EXT Algorithm

Illustrating the implementation of the GDSF-EXT algorithm is shown in Figure 3 between two Proxy Cache systems 0 and 1 (or Web Cache). Assuming that in Proxy Cache 0 has size =25 and free. Similarly in Proxy Cache 1 with Web objects: a=9, b=7, c=5, d=2, n=11, m=7, k=2. Thus, the HR hit rate is 3/10 based on 10 hits. If DGSF is used, these objects b, a, and m must be discarded, they are not recoverable for Proxy Cache 0 when re-referenced. Therefore, the HR = 0/10 is achieved with GDSF.

The following table simulates the array of objects accessed in cache 0: k, b, d, c, a, m, n, d, a, m



Time request		0	1	2	3	4	5	6	7	8	9	10
Requested web Object to			k	b	d	c	a	m	n	b	a	m
Cache Objects	Cache 1	Cache 0										
	a		k	k	k	k	k	m	m	b	b	m
	b			b	b	b	b	b	n	n	a	a
	c				d	d	d	c	n	n	c	c
	d					c	c	d	d	d	d	d
	n						a	k	k	k	k	k
	m											
	k											
Miss (M)/Hit (H)			M	M	M	M	M	M	M	H	H	H
Size compare								m=7, a=9	n=11, b=7, c=5	b=7, m=7	a=9, n=11	m=7, b=7
Hit Rate (HR)								3/10				
Replacement results by GDSF by SIZE and K(i) value smalest in Cache 0							a	a=*m	m	m=*b	b	b
				b	b	b	b	b	b=*n	n	n=*a	a
						c	c	c	c=*n	n	c	b=*m
					d	d	d	d	d	d	d	d
			k	k	k	k	k	k	k	k	k	k
Ka= 1,9							0,111				0,513	0,513
Kb= 2,7				0,143	0,143	0,143	0,143	0,143		0,540	0,540	0,540
Kc= 3,5						0,200	0,200	0,200				
Kd= 4,2					0,500	0,500	0,500	0,500	0,500	0,500	0,500	0,500
Kk= 9, 2			0,500	0,500	0,500	0,500	0,500	0,500	0,500	0,500	0,500	
Km= 6, 7								0,254	0,254			0,786
Kn=5,11									0,291	0,291		
Replace min(Ki)								min(ki)=ka=0,11 ==>replace a=m	min(ki)=kb=0,143; size cache not enough, continue replace kc ==>replace b, c=n	min(ki)=km=0,25 ==>replace m=b	min(ki)=kn=0,291 ==>replace n=a	min(ki)=kb=0,397 ==>replace b=m
Local Extended web cache stores objects removed from the cache 0 by GDSF-EXT (K(i))								a	a	a	n	n
									b	m	m	b
									c	c	c	c

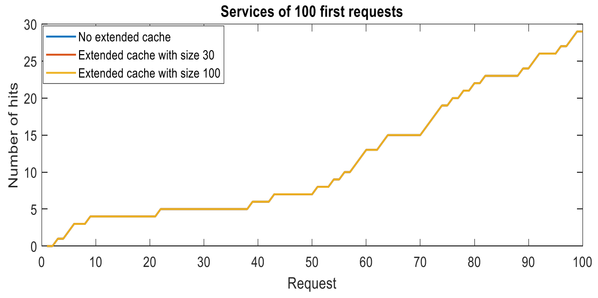
Figure 3. Illustrating the implementation of the GDSF-EXT algorithm

B. Setting and simulation

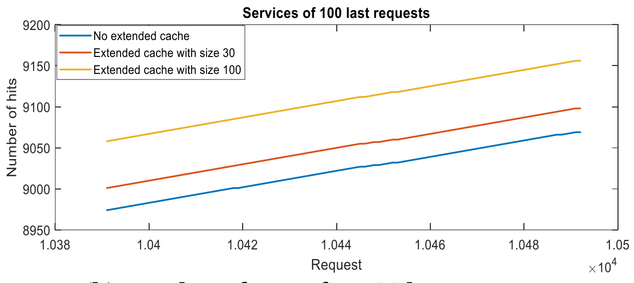
In this simulation, we used the request from Table. 1. The main caching has size 1000. And GDSF-EXT is considered in two cases: extended cache size 30, extended cache size 100 and extended cache size 500.

Table. 1. All requests for the simulation

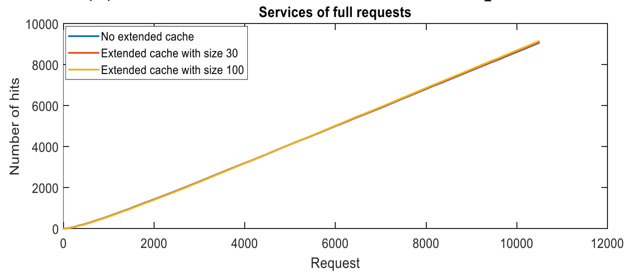
Time request	ID	Size
1	26	1
2	3	2
3	3	2
4	341	2
5	3	2
6	3	2
7	28	8
8	0	1
9	0	1
10	103	1
11	4	1
12	7	1
13	335	4
14	165	2
15	688	1
16	18	1
17	10	1
18	46	10
19	22	1
20	237	1
...	....	....
10492	853	1



(a) Number of HITs for 100 first requests



(b) Number of HITs for 100 last requests



(c) Number of HITs for all requests

Fig. 3. Simulation result of original GDSF and GDSF-EXT with the difference extended caching

Table 2. Hit of original GDSF and GDSF-EXT With the Difference Extended Caching

Algorithm	Original	Extended cache with size 30	Extended cache with size 100
Percent of hit	86.44%	86.71%	87.27%

Based on the result in Fig. 3, we can see all algorithms have a similar result within 100 first requests (Fig. 3a) because the main caching is not full at this time. It is a really big change in 100 last requests (Fig. 3b) when the number of hits meets 9156, 9098, and 9096 related to original GDSF, GDSF-EXT with extended cache with size 30, and GDSF-EXT with extended cache with size 100. Thus, both theoretically and simulations show that the GDSF-EXT has higher performance than the original GDSF algorithm by using a new extended caching

V. CONCLUSION

This paper presents a new algorithm named GDSF-EXT based on GDSF for the web caching replacement by using an extended cache as a second memory. Therefore, the frequency of requests is considered better than GDSF. From the equation of theoretically and code C# for simulation, all results show that GDSF-EXT has a higher performance of HIT than GDSF. Then our proposal could be applied to the web caching replacement area to make better consummation. In future work, we will use a queueing model and a random-time Petri network to perform simulations and performance analysis for a full system

REFERENCES

1. S. Podlipnig and L. Böszörmenyi, "A survey of Web cache replacement strategies," *J ACM Comput. Surv.*, vol. 35, no. 4, pp. 374–398, 2003.
2. C. S. MukeshDawar, "A Review on Web CachingTechniques," *J International Journal of Advanced Research in Computer Science Software Engineering*, vol. 2277.
3. D. R. CH, "Study of The Web Caching Algorithms for Performance Improvement of The Response Speed," *J Indian Journal of Computer Science Engineering*, vol. 3, no. 2, 2012.
4. K. Arora and D. R. CH, "Web cache page replacement by using LRU and LFU algorithms with hit ratio: a case unification," *J Int. J. Comput. Sci. Inf. Technol*, vol. 5, no. 3, pp. 3232-3235, 2014.
5. W. Ali, S. M. H. Shamsuddin, and A. S. Ismail, "A Survey of Web Caching and Prefetching," 2011.
6. W. Kin-Yeung, "Web cache replacement policies: a pragmatic approach," *IEEE Network*, vol. 20, no. 1, pp. 28-34, 2006.
7. S. Hosseini-Khayat, "Investigation of generalized caching," Washington University, 1998.
8. O. A. Mohammed and S. A. Talab, "Novel Web Cache Replacement Algorithm," *International Journal of Computer Science and Information Technology & Security (IJCSITS)*, vol. 8, no. 3.
9. W. Ali, S. M. Shamsuddin, and A. S. Ismail, "Intelligent Web proxy caching approaches based on machine learning techniques," *Decision Support Systems*, vol. 53, no. 3, pp. 565-579, 2012/06/01/ 2012.
10. W. Ali, S. Sulaiman, and N. B. H. Ahmad, "Performance improvement of least-recently-used policy in web proxy cache replacement using supervised machine learning," in *SOCO 2014*, 2014.
11. P. Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms," in *USENIX Symposium on Internet Technologies and Systems*, 1997.
12. T. Ma, J. Qu, W. Shen, Y. Tian, A. Al-Dhelaan, and M. Al-Rodhaan, "Weighted Greedy Dual Size Frequency Based Caching Replacement Algorithm," *IEEE Access*, vol. 6, pp. 7214-7223, 2018.
13. L. Keqiu and S. Hong, "An Improved GreedyDual Cache Document Replacement Algorithm," in *IEEE/WIC/ACM International Conference on Web Intelligence (WI'04)*, 2004, pp. 457-460.
14. L. Cherkasova, "Improving WWW Proxies Performance with Greedy-Dual- Size-Frequency Caching Policy," 1998.
15. P. Rodriguez, "Web caching architectures: hierarchical and distributed caching," 1999.
16. H. KL and T. NX, "Performance Analysis of Hybrid Web Caching Architecture," *American Journal of Networks and Communications*, vol. 4, no. 3, pp. 37-43, 2015.
17. G. K. Zipf, "Relative Frequency as a Determinant of Phonetic Change," *Harvard Studies in Classical Philology*, vol. 40, pp. 1-95, 1929.

AUTHORS PROFILE

Mr. Nguyen Xuan Truong is Vice Head of Department of Training, Hung Yen University Technology and Education, Vietnam. He has completed M.Sc (Information Technology) at Hanoi University of Science and Technology. Currently, he is pursuing his Ph.D. candidate in Mathematical foundations for informatics at Military Institute of Science and Technology, Vietnam. His interest is web caching replacement algorithms for internet service and optimization networks.

Dr. Ho Khanh Lam has more than 17 years of practical experience in the field of computer technology and data transmission networks and the internet. He is a lecturer at Hung Yen University of Technology and Education at the faculty of information technology since 2011. He got Ph.D. degree in 1999 with the topic: improving the reliability of communication networks. He has successfully guided 02 doctoral students to successfully defend their doctoral thesis and published many scientific works in the fields of optimization and performance evaluation of computer network systems, parallel computing, supercomputers. He has successfully guided 02 doctoral students to successfully defend their doctoral thesis and published many scientific works in the fields of optimization and performance evaluation of computer network systems, parallel computing supercomputers...