Cache Replacement Policy Based on Server Distance

Noriaki Kamiyama*†, Yuusuke Nakano*†, Kohei Shiomoto†
*Department of Information Science, Osaka University, Osaka 565-0871, Japan
Email: {kamiyama.noriaki, nakano.yuusuke}@ist.osaka-u.ac.jp
†NTT Network Technology Labs, Tokyo 180-8585, Japan, Email: shiomoto.kohei@lab.ntt.co.jp

***Abstract*—The transmission bandwidth consumed by delivering rich content is enormous, so it is urgent for Internet service providers to design an efficient delivery system that minimizes the amount of network resources consumed, i.e., minimizing the hop length of delivery flows. Content delivery networks are widely used to reduce the flow hop length and the response time when obtaining content items. To improve the cache efficiency, various methods of replacing caches have been proposed, with the target of improving the cache hit ratio. However, although the effect of delivering content items from caches depends on the distance from the origin server to the users, these methods do not consider the origin server distance. We propose here a cache-replacement policy based on the hop distance to the origin servers. We divide the storage capacity of cache severs into multiple virtual caches and manage content items separately based on the hop distance to origin servers. We also propose an optimal method for designing the capacities of virtual caches that maximizes the total expected reduction of flow hop length. Through numerical evaluation, we show that the proposed method can increase the average expected reduction of flow hop length by about 20% to 80% compared with normal LRU (least recently used).**

I. Introduction

Content delivery networks (CDNs) that use a number of cache servers deployed in multiple networks have been widely used as a method to efficiently transmit content [9][10][12]. Although CDNs are typically operated by CDN providers, e.g., Akamai [1], the number of CDNs operated by large-scale content providers such as Google, and by tier-1 ISPs such as AT&T has been increasing recently [7]. If the storage capacities of cache servers are large enough to store the entire set of content items, the cache hit ratio is maximized, i.e., 100%. In this case, all the content items are delivered from cache servers, and the amount of network resources consumed is minimized. However, in reality, the number of unique content items is huge and continues to increase rapidly, so storing all content items in every cache server is infeasible. If the cache storage capacity is full when caching new content items, known as a cache miss, some content items in the cache are removed to make available space for new content items.

The cache replacement algorithm selecting content items to be removed strongly affects the cache hit ratio, so various methods of cache replacement have been proposed [5][6][8]. Although the cache hit ratio is an important criterion, these methods do not consider the distance to origin servers when selecting content items to be removed. The effect of delivering content items from cache servers depends on the distance to the origin server from the user terminal, i.e., a large reduction effect of flow hop length can be expected for content items whose origin servers are remotely located. Therefore, it is desirable to consider the location of origin servers when replacing content items in caches servers.

In this paper, we propose a cache-replacement policy based on the expected reduction of flow hop length obtained by delivering content items from cache servers. In the proposed method, the storage capacity of a cache server is divided into multiple virtual caches, and content items are separately managed by each virtual cache based on the hop distance to origin servers. We also propose a method of optimally allocating the storage capacity to each virtual cache at each node so that the expected average reduction of flow hop length is maximized. We confirmed through numerical evaluation that the expected average reduction of flow hop length can be improved by up to about 80% by using the proposed cache replacement method. We propose a cache replacement method considering the distance to the origin servers in Section II and show the numerical results in Section III, and finally, we conclude the manuscript in Section IV.

II. Cache Replacement Based on Distance to Origin Servers

A. Assumptions

- Let N denote the number of nodes in the network; we assume that the network topology is given and that the links do not become a bottleneck in delivering content items. Let p_n denote the ratio of requests generated from users accommodated in node n. Hereafter, we simply write the users accommodated in node n as *users* n.
- M content items with the identical size are provided over the network; let $q_{n,m}$ denote the ratio of requests for content m at node n.
- At all the N nodes, cache servers as well as origin servers are provided. Hereafter, we simply write the cache server and the origin server provided at node n as *cache server* n and *origin server* n. The origin of each content m is stationary and permanently stored at just one node; let $s(m)$ denote the node at which the origin of content m is provided. Content m can always be obtained from origin server $s(m)$, and we do not consider the storage capacity of origin servers. In contrast, the storage capacity of all the caches servers are finite and identical, and each cache server can store B content items at a maximum.
- When a user requests a content item, the cache server is selected by the DNS server of the CDN provider. Here, we simply assume the ideal case in which cache server

978-1-4799-0913-1/14/$31.00

n is selected when user n requests content m. In this case, the hop length of delivery flows is minimized. If a copy of content m exists at cache server n, i.e., a cache hit, content m is delivered from cache server n to the requesting user, and the hop distance of the delivery flow is zero. Otherwise, the request is sent to the origin server $s(m)$, and content m is delivered to user n via node n after caching the copy of content m at cache server n [9][12]. When the storage capacity of cache server n is full, i.e., cache server n already stores B content items, content m is stored after removing one content item selected by the cache replacement algorithm.

- Content items are delivered from the cache or origin server to the user terminal on the minimum-hop routes. Let h_{ij} denote the hop distance from node i to node j on the delivery routes.
- As the baseline method of cache replacement, we assume LRU, which is most widely used in existing CDNs.

B. Cache Replacement Using Virtual Caches

When a user n requests content m, the hop length of delivery flow is zero if content m exists in cache server n, i.e., a cache hit. In contrast, when the requested content item does not exist at node n, i.e., a cache miss, the requested content item is delivered from its origin server. Therefore, we can say that the reduction effect of the hop length of delivery flows obtained by caching content m at node n is $h_{s(m),n}$, the hop distance from node $s(m)$ to node n. In other words, the effect of caching content items depends on the hop distance to the origin servers as well as the popularity of content items. However, the target of all the existing methods of cache replacement is just improving the cache hit ratio, and the distance to the origin servers is not considered. The main idea of the cache-replacement method proposed in this paper is to select content items to be removed based on the distance to the origin servers as well as their popularity.

Now, we introduce $e_{n,m}$, the expected average reduction of flow hop length, as a criterion for evaluating the effect of caching content m at node n; it is defined as

$$e_{n,m} = q_{n,m} h_{s(m),n}. \tag{1}$$

When LRU is used, all content items are treated equally, and a single LRU list is maintained to remove the content item requested least recently. Therefore, content items with a longer hop distance to the origin server are possibly removed to keep other content items with a shorter hop distance to the origin server. To avoid this, we propose to divide each cache server into multiple virtual caches (VCs) and assign each VC to the set of content items with the identical hop distance to the origin server.

Let $H(n)$ denote the maximum hop distance to node n from any origin server, that is, the maximum hop distance to node n from any other nodes. For any k of $1 \le k \le H(n)$, we also define $C_{n,k}$ as the set of content items whose hop distance from the origin server to node n is k, i.e., $h_{s(m),n} = k$. We divide B, the storage capacity of cache server n, into $H(n)$ VCs, $V_{n,1}$, $V_{n,2}$, $\cdots$, $V_{n,H(n)}$, and content items of $C_{n,k}$ are stored only in $V_{n,k}$. Also, let $b(n,k)$ denote the storage capacity assigned to $V_{n,k}$, and $\sum_{k=1}^{H(n)} b(n,k) = B$ is satisfied for any n. At each VC $V_{n,k}$, content items of $C_{n,k}$ are replaced by LRU independently of other VCs. We can regard each VC $V_{n,k}$ as the LRU list for content items with $h_{s(m),n} = k$.

We can expect to increase the total efficiency of cache servers by keeping more content items with longer hop distances in cache servers, and this is achieved by setting larger $b(n,k)$ for larger k. In the next section, we describe an example of content replacement using the proposed cache replacement method. Then, we propose a method of optimally designing $b(n,k)$ to maximize the total sum of the expected average reduction of flow hop length at each node n based on the dynamic programming in Section II-D.

C. Example of Content Replacement Using Virtual Caches

Figure 1 shows an example of the network topology and the location of origin servers for content items 1, 2, $\cdots$, 11. For example, the origin servers of content items 4, 5, and 8 are one hop distance from node A, and $C_{A,1}$ consists of these three content items. Therefore, content items 4, 5, and 8 are stored in $V_{A,1}$ at node A, and these content items are updated based on the LRU policy at node A. There are a total of three VCs at node A because $H(A) = 3$, and we also show the list of content items stored in each VC $V_{A,k}$ in Fig. 1. We note that content item 6 is never cached at node A because the origin server of content 6 exists at node A.

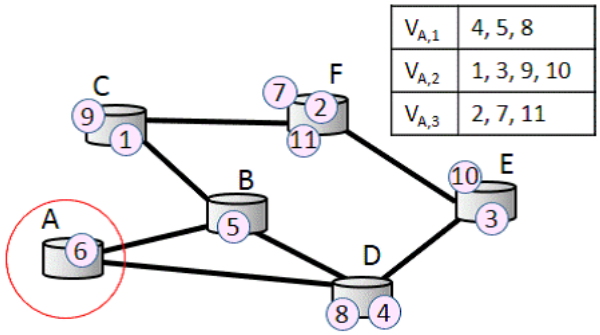

Fig. 1. Example of location of origin servers

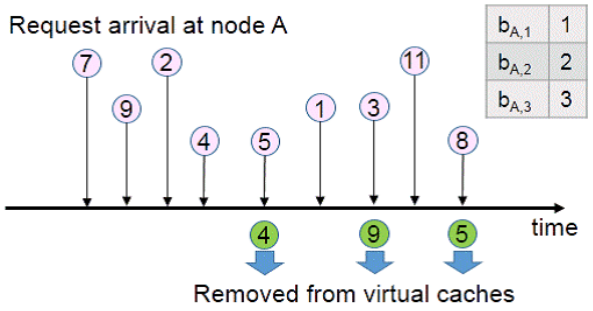

Fig. 2. Example of request arrival and cache replacement

Next, let us consider the case in which requests for content items arrive at node A in the order shown in Fig. 2. In this example, we assume that the capacity allocated to each VC at node A is $b_{A,1} = 1$, $b_{A,2} = 2$, and $b_{A,3} = 3$, and no content items are cached at node A before content item 7 arrives. The initial four requests are for content items 7, 9, 2, and 4, and these four content items are stored in VCs of node A, and no content items are removed because the capacity of these VCs is not full. However, when the fifth request arrives for content item 5, which is stored in $V_{A,1}$, content item 4, which has been stored in $V_{A,1}$ is removed because just a single

content item can be stored in $V_{A,1}$. On the other hand, when the sixth request for content item 1, which will be stored in $V_{A,2}$ arrives, no content item is removed from $V_{A,2}$ because up to two content items can be stored in $V_{A,2}$. As shown in this example, we can differentiate the sojourn time of content items in VCs according to the hop distance to origin servers.

D. Optimally Designing Virtual Caches

Our optimization target for maximizing the effect of cache server n is to maximize the sum of $e_{n,m}$, the expected average reduction of flow hop length obtained by caching content m at node n. The effect of each cache server is totally independent of the cache-replacement policy of other cache servers because we assume that only cache server n is used for users n as mentioned in Section II-A. Therefore, we note that the proposed algorithm to design the VCs can be executed at each node n without considering the VCs at other nodes.

As a result of using the LRU policy to update caches, content items with larger $q_{n,m}$ are more likely to exist in the VCs of node n. Therefore, we can expect that $b(n,k)$ content items of $C_{n,k}$ with the largest values of $q_{n,m}$ are stored in $V_{n,k}$ with high probability on average. Therefore, $U_{n,k}$, the expected hit ratio of $V_{n,k}$, agrees with the total ratio of requests for the most popular $b(n,k)$ content items among $C_{n,k}$, and it is given by

$$U_{n,k} = \sum_{j=1}^{b(n,k)} q_{n,\kappa(n,k,j)} \left(\equiv Q_{n,k,b(n,k)}\right), \tag{2}$$

where $\kappa(n,k,j)$ is the content item with the j-th largest $q_{n,m}$ among content items of $C_{n,k}$, and $Q_{n,k,x}$ is the sum of $q_{n,m}$ of the most popular x content items in $C_{n,k}$. Now, let us define $E_{n,k}$ as the total expected average reduction of flow hop length at $V_{n,k}$, and from (1), it is given by

$$\begin{aligned} E_{n,k} &= \sum_{m \in C_{n,k}} e_{n,m} \\ &= Q_{n,k,b(n,k)} \cdot k. \end{aligned} \tag{3}$$

The target of the proposed VC size design method at node n is to maximize E_n, the total expected average reduction of flow hop length by cache server n, i.e., $E_n = \sum_{k=1}^{H(n)} E_{n,k}$, with the constraint of B, and we define the following integer programming problem for designing $b(n,k)$ at node n:

$$\min E_n = \sum_{k=1}^{H(n)} Q_{n,k,b(n,k)} \cdot k \tag{4}$$

$$s.t. \quad \sum_{k=1}^{H(n)} b(n,k) = B, \tag{5}$$

$$0 \le b(n,k) \le B. \tag{6}$$

In general, to find the optimal solution to the integer programming problem, we need to check all the possible combinations of variables, and we cannot obtain the optimal solution in pseudo-polynomial time. However, if we can recursively break down the original problem into multiple sub-problems of a single variable, we can use dynamic programming to solve the problem [2].

From (1), we confirm that our original optimization problem minimizing E_n is equivalent to the sum of $E_{n,k}$, which is a function of $b(n,k)$, so we can solve the optimal design problem of VCs using dynamic programming. This is because the effect of each VC is determined by just its capacity without being affected by other VCs at the same node.

To explicitly indicate that $E_{n,k}$ is a function of $b(n,k)$ only, we write it as $E_{n,k}(b(n,k))$. Now, we can describe the proposed optimal design method of VCs at each node n using dynamic programming as the following recursive formula:

$$\begin{aligned} \phi_1(a_1) &= \max\Big\{E_{n,1}(b(n,1))\Big\} = E_{n,1}(a_1), \\ &\quad a_1 = 0, 1, \cdots, B, \end{aligned} \tag{7}$$

$$\begin{aligned} \phi_i(a_i) &= \max\Big\{E_{n,i}(b(n,i)) + \phi_{i-1}(a_i - b(n,i))\Big\}, \\ &\quad i = 2, \cdots, H(n), \\ &\quad a_i = 0, 1, \cdots, B. \end{aligned} \tag{8}$$

By solving this recursive formula from $i = 1$ to $i = H(n)$ for each possible value of a_i, we can obtain the optimum capacities of VCs at node n as $b(n,1)$, $b(n,2)$, $\cdots$, $b(n,H(n))$ giving $\phi_{H(n)}(B)$. The number of values $b(n,i)$ can take is B, so the required time to derive the optimal $b(n,i)$ is $O(H(n)B^2)$, and we can obtain a strictly optimal allocation in pseudo-polynomial time.

E. Required Functions of Cache Servers

Here, we describe all the functions that are required for cache servers in order to implement the proposed method of replacing caches using VCs. To optimally design $b(n,k)$, the storage capacity assigned to VC $V_{n,k}$, by using the method mentioned in the previous section, cache server n must estimate $q_{n,m}$, the ratio of requests for content m at node n, for each m of $1 \le m \le M$ as well as $h_{s(m),n}$, the hop distance from the origin server of content m to node n, for each m of $1 \le m \le M$.

Cache server n can easily estimate $h_{s(m),n}$ for each m by checking the IP address of its origin server when content m is requested from users n for the first time. Moreover, cache server n can also estimate $q_{n,m}$ for each m by counting the number of requests generated from users n for m during any time interval.

III. Numerical Evaluation

A. Evaluation Conditions

We used network topologies of eight commercial ISP backbone networks in the USA publicly available on the CAIDA web site [3]. Table I summarizes N, the number of nodes, L, the number of links, g, the average node degree, and G, the maximum node degree of each of the eight networks. We generate each user request from a node randomly selected with the probability proportional to the node population r_n, i.e., setting $p_n = r_n / \sum_{i=1}^{N} r_i$ [4]. Content items are delivered from the cache or origin servers to users on the minimum-hop routes.

We set the number of content items as $M = 1,000$. We assume that the ratio of requests for the k-th most popular content item at node n is identical among all N nodes, and we assume it obeys the Zipf distribution with parameter θ. In other words, we set the ratio of requests for content m at node

n as $q_{n,m} = c/m^{\theta}$, where c is the normalization constant to make $\sum_{m=1}^{M} q_{n,m} = 1$. In the entire simulation period, the content count M and the request frequencies of content items $q_{n,m}$ are fixed.

At the beginning of the computer simulation, we randomly place the origin server of each content item among N nodes, and we do not change the location of origin servers during the computer simulation. Because the results of the proposed cache replacement method using the VCs depends on the location of origin servers, we repeat the computer simulation 100 times using a different pattern of origin servers for each parameter set, and we evaluate the average properties among 100 trials.

At all the N nodes, we place a cache server with the identical storage capacity of $B = \lceil \beta M \rceil$, where β is a given parameter taking a real number between zero and unity. When we set β to unity, all the M content items can be stored at each cache server, and the average hop length of delivery flows is zero. In each trial of each parameter set, we start the computer simulation with the state that no content items are stored at any of the N nodes, and we start to measure all the properties after the storage capacity of cache servers is fully utilized, i.e., storing B content items, at $\lceil N/2 \rceil$ or more nodes. We generated 1 million requests after this warm-up period. Unless otherwise stated, we set $\theta = 0.5$ and $\beta = 0.1$.

TABLE I
TOPOLOGICAL PROPERTIES OF EIGHT ISP BACKBONE NETWORKS

ID	Name	N	L	g	G
1	above.net	22	25	2.27	12
2	AGIS	82	92	2.24	4
3	Allegiance Telecom	53	88	3.38	12
4	At Home Network	46	55	2.39	5
5	Genuity	48	53	2.21	5
6	IDT Corp	15	18	2.40	5
7	Qwest	14	26	3.71	7
8	ServInt Internet Services	23	34	2.96	7

B. Effect of Proposed Cache Replacement Method

In this section, we investigate the effectiveness of the proposed method, VC-LRU, by comparing R and E with LRU. Let $\hat{R}$ denote the increase ratio of R obtained by VC-LRU compared with LRU, and it is defined as $\hat{R} = (R_{VC-LRU} - R_{LRU})/R_{LRU}$, where R_{VC-LRU} and R_{LRU} are R in VC-LRU and in LRU, respectively. Similarly, we also define $\hat{E}$ as $\hat{E} = (E_{VC-LRU} - E_{LRU})/E_{LRU}$, where E_{VC-LRU} and E_{LRU} are E in VC-LRU and in LRU, respectively.

Figure 3(a) plots $\hat{R}$ against θ for each of the eight networks. $\hat{R}$ was in the range between -0.1 and zero for almost all the values of θ. Although the degree of degradation of R was small, i.e., less than 10%, we confirmed that the cache hit ratio was slightly degraded by using VC-LRU compared with LRU. When selecting a content item to be removed, VC-LRU considers both the popularity and h_o, the hop distance to the origin servers of content items, whereas LRU considers only the popularity of content items. In other words, in LRU, popular content items are more likely to exist in cache servers compared with VC-LRU. Therefore, the cache hit ratio of VC-LRU was slightly smaller than that of LRU.

Figure 3(b) plots $\hat{E}$ against θ for each of the eight networks. As θ decreased, $\hat{E}$ increased, and the superiority of VC-LRU against LRU improved. As θ decreased, and the difference in popularity among content items diminished, the hop distance to the origin server becomes a more dominant factor in determining E than the content popularity. As a result, we can increase E, the overall expected average reduction of flow hop length, by using VC-LRU compared with LRU. When θ was less than 0.5, for example, VC-LRU improved E by about 20% to 80% compared with LRU. In summary, although the cache hit ratio was slightly degraded, the overall expected reduction effect of flow hop length was largely improved by using the proposed VC-LRU method.

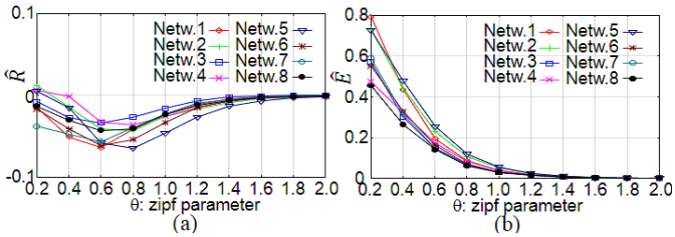

Fig. 3. Increase ratio of average cache hit ratio and expected average reduction of flow hop length against Zipf parameter θ in each of eight networks

IV. CONCLUSION

In this paper, we proposed a cache-replacement policy based on the hop distance to origin servers. In the proposed method, the storage capacity of cache severs is divided into multiple virtual caches, and content items are managed separately based on the hop distance to origin servers. We also proposed an optimal design method of the capacities of virtual caches that maximizes the total expected reduction of flow hop length. Through a numerical evaluation, we showed that the proposed method can increase the average expected reduction of flow hop length by about 20% to 80% compared with normal LRU while slightly degrading the cache hit ratio. In future, we will investigate the effect of the proposed cache replacement method when the content popularity dynamically varies.

REFERENCES

[1] Akamai Technologies, http://www.akamai.com
[2] R. Bellman, Dynamic Programming, Dover Pubns, 2003.
[3] CAIDA, http://www.caida.org/tools/visualization/mapnet/Data/
[4] K. Cho, K. Fukuda, H. Esaki, and A. Kato, The impact and implications of the growth in residential user-to-user traffic, ACM SIGCOMM 2006, pp.207-218, Sept. 2006.
[5] T. Johnson and D. Shasha, 2Q: a low overhead high performance buffer management replacement algorithm, VLDB 1994.
[6] M. Karlsson and M. Mahalingam, Do We Need Replica Placement Algorithms in Content Delivery Networks?, WCW 2002.
[7] C. Labovitz, S. Iekel-Johnson, J. Oberheide, and F. Jahanian, Internet Inter-Domain Traffic, ACM SIGCOMM 2010.
[8] N. Megiddo and D. S. Modha, ARC: a self-tuning, low overhead replacement cache, USENIX FAST 2003.
[9] E. Nygren, R. Sitaraman, and J. Sun, The Akamai Network: A Platform for High-Performance Internet Applications, ACM SIGOPS 2010.
[10] J. Ott, M. Sanchez, J. Rula, and F. Bustamante, Content Delivery and the Natural Evolution of DNS, ACM IMC 2012.
[11] J. Ratkiewicz, et al., Characterizing and modeling the dynamics of online popularity, Physical Review Letters, 105, 158701, 2010.
[12] A. Su, D. Choffnes, A. Kuzmanovic, and F. Bustamante, Drafting Behind Akamai: Inferring Network Conditions Based on CDN Redirections, ACM Trans. Networking, 17(6), pp.1752-1765, 2009.