On Randomized Request Redirection in Hierarchical Caching Systems

Y. Thomas Hou[†] Jianping Pan[‡] Sherman X. Shen[‡]

[†]Virginia Tech, The Bradley Department of Electrical and Computer Engineering, Blacksburg, VA, USA

[‡]University of Waterloo, Department of Electrical and Computer Engineering, Waterloo, Ontario, Canada

Abstract— **Adopting Time-to-Live (TTL) based hierarchical caching systems is considered to be a viable approach to support Web content delivery under the weak consistency paradigm. However, with a strictly hierarchical structure in these systems, a single user request may trigger multiple consecutive miss events at cache servers of different levels. This undesirable** *miss synchronization* **can cause a sudden degradation of user-perceived performance in terms of response time. In this paper, we offer a comprehensive examination of this undesirable behavior and propose a** *randomized request redirection* **approach to circumvent the structural restrictions in TTL-based hierarchical caching systems. Performance analysis and evaluation indicate that the proposed approach can effectively rebalance server overhead and network delay, which in turn reduces end user response time.**

Index Terms— **Internet, content delivery, hierarchical caching, weak consistency, miss synchronization**

I. Introduction

Time-to-Live (TTL) based hierarchical caching systems have been proposed, developed and deployed in recent years to support Web content delivery under the weak consistency paradigm [2], [8]. Within these systems, there is a strictly hierarchical structure that consists of multiple cooperative cache servers at different levels in network space [3]. This hierarchy is adopted to aggregate user requests and to share cached objects. On the other hand, a TTL-based expiration scheme is used for cached objects to ensure a certain degree of guarantee in object validity. Under this paradigm, since a cached object is only validated periodically, the cached copy may become *stale* if the original one is updated at its origin server before the next validation. For most Web applications, such discrepancy may still be tolerable in practice to allow users to extract useful (or at least non-harmful) information from the received objects. Moreover, users always have the option to obtain the latest copy of an object by reloading it directly from its origin server.

When an object is retrieved from its origin server, the remaining lifetime, or TTL, of this object is initialized to a value reflecting the maximal tolerance by end users of possible discrepancy. In HTTP/1.1, TTL can either be explicitly specified by origin server with the `Expires` response directive or `max-age` cache control, or it can be heuristically calculated by cache server through the origin server's `Last-Modified` directive [16]. When an object is cached somewhere, the associated TTL is expected to reduce its value continuously and linearly with time. The cached object is considered valid only if it has a positive remaining TTL. If a cached object has a non-positive (or expired) TTL when being requested, the cached copy has to be revalidated by or, if necessary, retrieved from its origin server or a cache server of higher levels in the hierarchy. It has been shown in [1], [3], [8] that due to its relatively lower overhead and less complexity, a TTL-based hierarchical caching system is a viable approach to scale up with explosive growth of the Web.

However, due to the structural restrictions, for end users who are associated with a cache server that is logically farther away from the origin server of an object, a strictly hierarchical caching system may be biased against these users. Specifically, a leaf user is more likely to experience a longer response time per request, a higher cache miss ratio, and a smaller average remaining TTL for cached objects. Also, when a local cache miss happens, it is very likely that regenerated inter-cache requests encounter multiple *miss*es at several higher level servers throughout the hierarchy. This behavior is referred to as *miss synchronization*, and is responsible for a sudden increase in user-perceived response time. These behaviors unnecessarily penalize end users in a hierarchy, and compromise the design goal of a hierarchical caching system.

In this paper, we focus on this structural deficiency with the aim of devising an approach to improve user-perceived performance. The proposed approach introduces request randomization with a geometric probability model to redirect inter-cache requests randomly to a higher level cache server rather than only to the immediate parent cache server. Consequently, this breaks the structural restrictions imposed by a traditional hierarchical caching system.

Our main contributions are two-fold. First, by using request randomization, we can considerably avoid miss synchronization, since cached objects are no longer doomed to expire at the exact same time in consecutive cache servers. Second, by avoiding this unwanted synchronization, we will substantially reduce user-perceived response time, since network delay and server overhead can be reactively rebalanced when a local cache miss occurs. Through a combination of analysis and simulations, we demonstrate that the proposed approach can effectively compensate the structural differentiation in a hierarchical caching system.

The rest of this paper is organized as follows. In Section II, we review properties and performance of TTL-based hierarchical caching systems. We then examine the miss synchronization behaviors. In Section III, we propose a *randomized request redirection* approach to alleviate miss synchronization

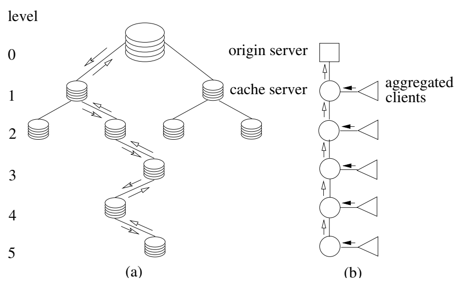

Fig. 1. Modeling of hierarchical caching systems

and to reduce user-perceived response time in hierarchical caching systems. The efficacy and performance of the proposed request randomization approach are substantiated with extensive evaluation results in the same section. Section IV reviews related work. Section V concludes the paper.

II. System Model

A. Hierarchical Caching Systems

Fig. 1(a) illustrates a tree-like hierarchical caching system. When a cache server (CS) receives a request from its local users or downstream CSs, if the requested object is cached locally with a positive remaining TTL, the object is returned immediately. This is considered as a *hit* event. Otherwise, a *miss* event happens, and the CS generates another request to its *immediate* parent CS. This process is recursive until either obtaining a valid copy of the object or reaching the origin server (OS) that always has the latest copy available. When an object is retrieved from the OS, its TTL is initialized to a value (τ) corresponding to the maximal tolerable time-discrepancy by end users, and the TTL value decreases when the object is cached at CSs. If we take the longest simple path from the OS in Fig. 1(a) and convert all user requests to CSs not in this path as aggregated client (AC), we have a chain-like structure in Fig. 1(b). This process is also recursive; the model developed here can be applied to each branch individually. Therefore, we will focus on the chain model.

In [8], we developed a generic analytical model and derived $\mathtt{E}(T_d)$, the average TTL that a CS_d (*i.e.*, a level-d CS) can expect from its parent CS_{d-1}, where $1 \leq d \leq D$ and D is the chain depth. Denote λ_d as the request rate from AC_d to CS_d for a particular object. Although the request pattern from an individual user can be very complicated [4], it is sufficient to approximate the aggregated requests with an exponential inter-arrival time when a very large user population (*e.g.*, a CS serves users from a metropolitan area) is considered. With $\Lambda_d = \lambda_d + \lambda_{d+1} + \cdots + \lambda_D$, we have

$$\mathtt{E}(T_d) = \frac{\Lambda_d \tau + (\Lambda_1 - \Lambda_d)(\tau - \frac{1-e^{-\Lambda_d \tau}}{\Lambda_d})}{\Lambda_d + (\Lambda_1 - \Lambda_d)(1 - e^{-\Lambda_d \tau})} \ . \qquad (1)$$

It is obvious that $\Lambda_{d_1} \geq \Lambda_{d_2}$ if $d_1 < d_2$, since $\Lambda_{d_1} = \lambda_{d_1} + \lambda_{d_1+1} + \cdots \lambda_{d_2-1} + \Lambda_{d_2}$ and $\lambda_d \geq 0$. Moreover, given Λ_1, $\mathtt{E}(T_{d_1}) \geq \mathtt{E}(T_{d_2})$, as $\Lambda_{d_1} \geq \Lambda_{d_2}$. This property shows that in a strict hierarchy, the closer a CS is to the OS, the larger average TTL the CS can expect. Fig. 2 plots the analytical calculation (lines) of (1) and simulation results (points) normalized to τ for the following three request patterns.

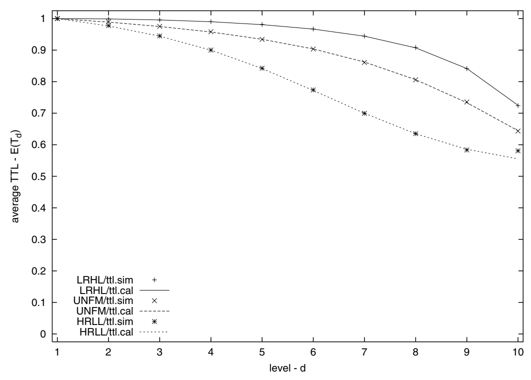

Fig. 2. Average retrieved object TTL at different levels

- *Light-Root-Heavy-Leaf* (LRHL): $\lambda_d = 0.2d - 0.1$ per τ and $1 \leq d \leq D = 10$.
- *Uniform* (UNFM): $\lambda_d = 1$ per τ.
- *Heavy-Root-Light-Leaf* (HRLL): $\lambda_d = 2.1 - 0.2d$ per τ.

We derived the system miss ratio Γ_d^s and user response time σ_d^u in [8] for CS_d and AC_d, respectively. If CS_d caches an object with the remaining TTL $\tau(t^*)$ at t^*, it will not contact CS_{d-1} before $t^* + \tau(t^*)$ for the same object. Using the aggregation property of Poisson processes, we know that there is one upstream request for CS_d in the time period $\mathtt{I}_d + \mathtt{E}(T_d)$ where $\mathtt{I}_d = \frac{1}{\Lambda_d}$ is the idle time for the subtree rooted at CS_d. With $\gamma_d^s = \frac{1}{\mathtt{I}_d + \mathtt{E}(T_d)}$, we have the system miss ratio

$$\Gamma_d^s = \frac{\lambda_d}{\Lambda_d} \cdot \frac{1}{(\lambda_d + \gamma_{d+1}^s)\mathtt{E}(T_d)} + \frac{\Lambda_d - \lambda_d}{\Lambda_d} \cdot \frac{1}{\lambda_d \mathtt{E}(T_d)} \ , \qquad (2)$$

and the user miss ratio

$$\Gamma_d^u = \frac{\lambda_d}{\Lambda_d(1 + \lambda_d)\mathtt{E}(T_d)}$$

if omitting the requests from downstream CSs accordingly.

The user response time at level d is recursively defined as

$$\sigma_d^u = \Gamma_d^u(t_n + t_s + \frac{\Lambda_d \sigma_{d-1}^s}{\Lambda_{d-1}\Gamma_{d-1}^s}) \ , \qquad (3)$$

where t_n is the network delay to CS_{d-1}, t_s is the server overhead, in terms of server delay, at CS_{d-1}, and

$$\sigma_d^s = \Gamma_d^s(t_n + t_s + \frac{\Lambda_d \sigma_{d-1}^s}{\Lambda_{d-1}\Gamma_{d-1}^s})$$

is the system response time for CS_d. When $d = 1$, $\sigma_1^u = \Gamma_1^u(t_n + t_s)$ and $\sigma_1^s = \Gamma_1^s(t_n + t_s)$.

Moreover, we find that user-perceived performance also depends on request patterns. For example, in Fig. 2, LRHL has a higher average TTL than UNFM, which in turn has a much higher average TTL than HRLL, although the overall user request rate is the same at $\Lambda_1 = D$ per τ for these three request patterns. In the following sections, we will use UNFM as a benchmark for performance comparison.

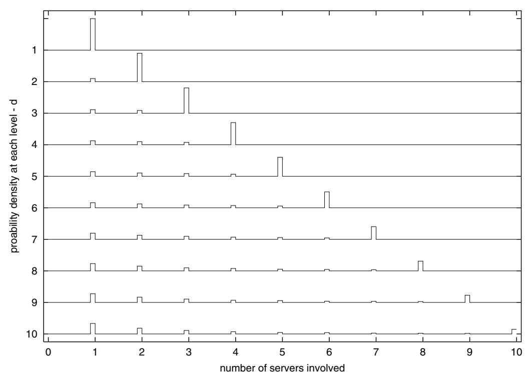

Fig. 3. Probability of the number of servers involved for a local *miss*

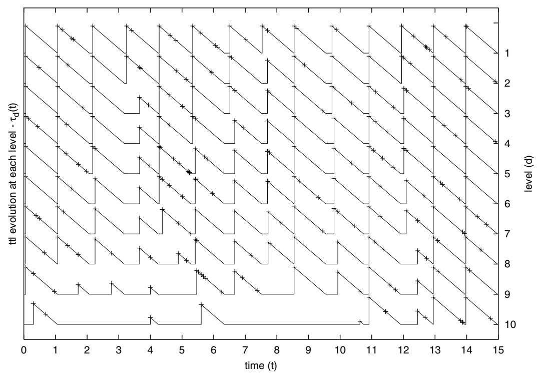

Fig. 4. TTL evolution at different levels during a time window

B. Cache Miss Synchronization

Although (3) gives the average user response time, we are more interested in the case when a user request incurs a miss at the local CS. In many Web applications, the user-perceived quality of online experience is determined by the worst-case, instead of average performance. To link t_n and t_s for the end user-experienced response time t when a local miss occurs, we use a linear *utility* function

$$t = w_n t_n + w_s t_s \ , \tag{4}$$

where w_n and $w_s = 1 - w_n$ are the weights for network delay and server overhead, respectively. Therefore, we need to examine how many CSs are involved in order to obtain a valid object when a local miss occurs, which determines the associated server overhead and network delay. Fig. 3 illustrates the probability of the number of servers involved for a *miss*ed user request at different levels. For a level-d request, the possible outcomes are $\{1, 2, \cdots, d\}$, since with a strict hierarchy, the request incurs at most d cache *miss*es before reaching the OS.

In Fig. 3, we observe an undesirable synchronization behavior. When a user request incurs a cache miss at its local CS, it is very likely that the regenerated requests trigger several *miss*es at multiple upstream CSs. With a very high probability, the request is finally fulfilled by the OS. During this process, the request has gone through each CS and incurred additional processing at CSs along its path toward the OS.

There are two factors that cause the observed miss synchronization. First, in a strictly hierarchical structure, each CS is *only* allowed to contact its immediate parent CS. Second, the parent and child CSs share the *same* TTL timeout value. Therefore, if the TTL becomes 0 at CS_d, all CS_{d^+} for $d^+ \geq d$ have a 0 TTL as well. Wherever there is an AC_{d^*} request within the subtree rooted at CS_d, all $CS_{d^*_-}$ for $d^* \geq d^*_- \geq d$ guarantee a cache miss. Therefore, this particular user request suddenly experiences an unexpected long response time.

This behavior is also illustrated in Fig. 4 for the TTL evolution at CSs of different levels during a time window in the simulation. In this figure, the connected lines represent the remaining TTL value, and the disconnected crosses represent a user or inter-cache request. Due to the aforementioned two restrictions in a strictly hierarchical structure, a TTL *triangle* at CS_d is always covered by a triangle at CS_{d-1}; moreover, these two triangles coincide at the same time when TTL is reduced to 0. No matter when a new request comes, both CS_d and CS_{d-1} will encounter a cache miss, which means that the request has to travel further and inquire more CSs.

One approach to alleviating such miss synchronization and its consequence is to introduce the request randomization technique. In addition, if we can trade slightly increased network delay for considerably reduced server overhead, we can overcome the undesired synchronization associated with a strict hierarchy. That is, instead of only being allowed to contact its immediate parent CS, a CS will have the option to choose one of its upstream CSs randomly. We will elaborate the details of this approach in the next section.

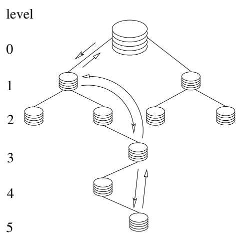

Fig. 5. Randomized Request in hierarchical caching systems

III. Randomized Request

Fig. 5 illustrates the proposed request randomization approach. For instance, when a level 5 user request incurs a miss at its local CS, the CS can randomly choose either the OS or one of the level $1, 2, 3, 4$ CSs along its path toward the OS. Suppose that it picks the level 3 CS. In turn, the level 3 CS can choose either the OS or one of the level $1, 2$ CSs, if another cache miss occurs. In this example, the level 1 CS is chosen. This randomized process is recursive; it terminates after either obtaining a valid object or reaching the OS.

When a valid object is retrieved, only the CSs that participate in this query process will be updated, since other CSs are bypassed and unaware of this cache update. In this example, only the chosen level 1, 3, and 5 CSs are updated. After this update, when the object at CS_4 expires, it is possible that CS_5 still has a positive remaining TTL for the same object, since it has recently received a fresher copy from CS_3 directly.

A. Probability Model

Here, we adopt a *geometric* probability model for the randomized request. Suppose that there is a cache miss at CS_d, CS_d can direct the request to the OS at level 0 with probability $p_0^r = p$, or to CS_1 with probability $p_1^r = r \cdot p$, or to CS_2 with probability $p_2^r = r^2 \cdot p$, and so on. The probability to direct a request to its immediate parent CS_{d-1} is $p_{d-1}^r = r^{d-1} \cdot p$. Given $\sum_{i=0}^{d-1} p_i^r = 1$, we have $p = (\sum_{i=0}^{d-1} r^i)^{-1}$ when $r \neq 0$. Clearly, the choice of r can affect the query behavior.

- When $r \to \infty$, it becomes the basic model with a strictly hierarchical structure within which any CS_d can only contact its immediate parent CS_{d-1}.
- When $r = 0$, it becomes a flat structure without any hierarchy, *i.e.*, any CS_d can only contact the OS directly.
- When $r = 1$, requests are uniformly distributed to one of upstream CSs or the OS.
- When $r > 1$, an *optimistic* strategy is in effect, since CS_d believes that an object with the valid remaining TTL is nearby; it therefore will most likely direct the request to an immediate, or one of the next-to-immediate, CSs.
- When $0 < r < 1$, a *pessimistic* strategy is in effect, since CS_d believes that an object with the valid remaining TTL is only available near the OS; it therefore is likely to direct the request to a CS that is farther away from itself, *i.e.*, near the OS.

However, if CS_d overestimates or underestimates the availability of a nearby valid object in either the optimistic or pessimistic case, an adverse effect on overall performance may occur. With overestimation, there is likely another cache miss at nearby CSs, which means higher server overhead. On the other hand, underestimation would mandate further travel, with a larger network delay. Therefore, it is important to examine a broad range of r for a given request pattern in order to find a good balance between the network delay and server overhead experienced by a particular user request.

B. Performance Evaluation

1) Server overhead: We choose $r \in \{\frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1, 2, 4, 8\}$ and present the simulation results with the UNFM request pattern. Fig. 6 shows the probability density of server overhead at different levels with the randomized request ($r = 2$) when a miss occurs. Comparing Fig. 6 to Fig. 3, we find that for most levels, only 1 or 2 CSs are sufficient to fulfill a request, which greatly reduces server overhead. In contrast, in Fig. 3, there is a high probability that a request has to contact every upstream CS (until the OS) with *doomed miss*es.

Next, we show the effect of r on randomization behavior. Fig. 7 plots that the lower (or more *pessimistic*) that r is, the less number of CSs are involved. This is because a lower r implies that a request *jump*s further and quicker toward the OS on each attempt. The basic model consistently has the

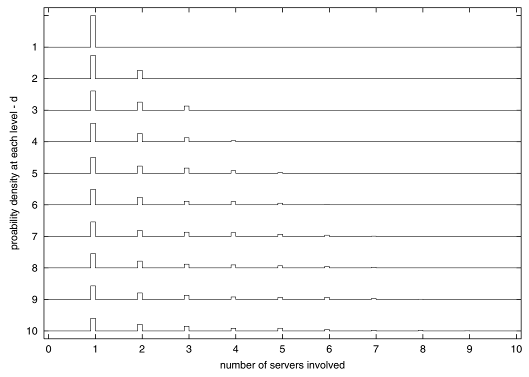

Fig. 6. Probability of the number of servers involved for a local *miss* ($r = 2$)

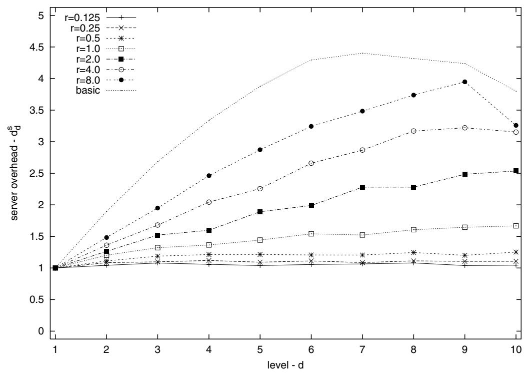

Fig. 7. Server overhead with randomized request for a local *miss*

highest number of involved CSs for all levels, since a CS is only allowed to contact its immediate parent CS. It is worth pointing out that the lower level CSs (instead of the leaf CSs) have the highest server overhead due to relatively lower request aggregation and average TTL.

2) Network delay: Fig. 8 shows the network delay corresponding to the different rs in Fig. 7. Clearly, a lower server overhead typically comes with a higher network delay. As shown in Fig. 8, network delay almost increases linearly with d when r is 0.125. The higher the r is, the slower network delay increases with d. When $r > 1$, network delay increases even faster at lower CSs (except leaf CSs) due to relatively lower request aggregation and smaller average TTL. If $r \to \infty$, network delay is proportional with the server overhead in Fig. 7, since a request has to contact each upstream CS sequentially. Obviously, there is a tradeoff between the server overhead and network delay for a given request pattern.

3) Response time: Fig. 9 further illustrates the weighted overall response time for $w_s \in \{0.9, 0.8, \cdots, 0.1\}$ when $r = 2$. When the per-server overhead is higher than the per-hop network delay, *i.e.*, $w_s \geq 0.5 \geq w_n$, the randomized request approach has consistently better overall performance than that of a strictly hierarchical structure. Therefore, it is appropriate to trade slightly increased network delay for considerably reduced server overhead, especially in the current Web context,

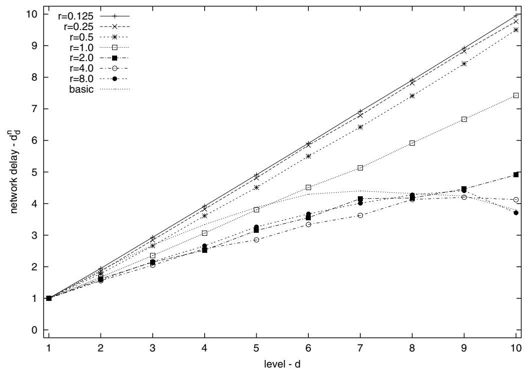

Fig. 8. Network delay with randomized request for a local *miss*

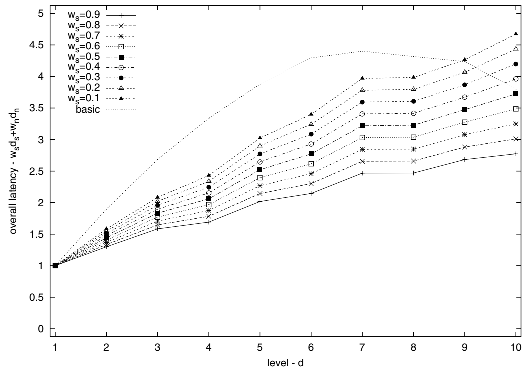

Fig. 9. Response time with randomized request for a local *miss* ($r = 2$)

within which many popular Web servers and their cache servers are already highly overloaded [12], [13]. When taking both network delay and server overhead into consideration, we find that an r that is slightly greater than 1 (*e.g.*, $r = 2$) is a good choice to balance these two sources of user-perceived response time. For $r = 2$, request randomization is optimistic in the availability of a nearby valid object; at the same time, it avoids too much overestimation.

IV. Related Work

There have been many attempts to introduce randomization techniques into Web caching systems. Realizing a centralized cache server can impose a performance bottleneck and a single-point failure, Malpani *et al.* [5] proposed to let a client randomly choose its own *master* proxy from a group of cooperative cache servers, and to allow these servers to communicate over the IP multicast. This *per-client* randomization approach can also be extended to the *per-object* approach, *i.e.*, choosing the proxy server based on object content or identifiers (*e.g.*, URL). There are other related efforts. Psounis *et al.* [15] proposed a randomized cache replacement scheme; Feder *et al.* [14] proposed a randomized request reordering scheme for cache servers. Many inter-cache protocols and their performance [6], [7], [9] have been developed and analyzed for cooperative cache servers. These efforts focus on introducing randomization in the horizontal dimension (*i.e.*, choosing the cache server that users should contact, and arranging communication among sibling cache servers). These studies are complementary to our work, which focuses on introducing randomization in the vertical dimension in a hierarchical caching system, *i.e.*, among parent and child cache servers. There are other cooperative caching schemes based on distributed hashing functions [10], [11] which allow minimal object reassignment when cache servers change. No matter whether it is in the horizontal or vertical dimension, cooperative caching systems can improve the performance for hit events. However, these systems unavoidably suffer additional overhead when a cache miss occurs. The approaches developed in this paper can be applied in these contexts to help improve system performance even when miss events occur.

V. Conclusions

This paper examined the performance issues and underlying causes associated with the synchronization of miss events in hierarchical caching systems. We proposed a randomized request redirection approach that logically breaks the restrictions in a strictly hierarchical system. Our analysis and evaluation results demonstrated that the proposed approach can help mitigate the structural discrimination in such hierarchical systems, and can improve end user-perceived performance.

References

[1] G. Barish and K. Obraczka. World Wide Web caching: trends and techniques. *IEEE Communications Magazine*, 38(5):178–184, 2000.

[2] J. Gwertzman and M. Seltzer. World-wide Web caching consistency. *Proc. USENIX'96*, pp. 141–151, 1996.

[3] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell. A hierarchical Internet object cache. *Proc. USENIX'96*, 153–163, 1996.

[4] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: evidence and implications. *Proc. IEEE Infocom'99*, pp. 126–134, 1999.

[5] R. Malpani, J. Lorch, and D. Berger. Making world wide web caching servers cooperate. *Proc. 4th Int'l WWW Conference*, 1996.

[6] L. Fan, P. Cao, J. Almeida, and A.Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.

[7] D. Wessels, and K. Claffy. ICP and the Squid web cache. *IEEE J. Selected Areas in Communications*, 16(3):345–357, 1998.

[8] Y.T. Hou, J. Pan, B. Li, X. Tang, and S.S. Panwar, Modeling and analysis of an expiration-based hierarchical caching system. *Proc. IEEE Globecom'02*, 2002.

[9] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. Levy. On the scale and performance of cooperative Web proxy caching. *Proc. 17th ACM Symp. O.S. Principles*, pp. 16–31, 1999.

[10] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi. Web caching with consistent hashing. *Proc. 8th Int'l WWW Conference*, 1999.

[11] V. Valloppillil and K.W. Ross. Cache array routing protocol v1.0. *IETF Internet Draft*, 1998.

[12] T. Voigt and P. Gunningberg. Handling multiple bottlenecks in Web servers using adaptive inbound controls. *Proc. IFIP Protocol for High Speed Networks*, pp. 50–68, 2002.

[13] G. Banga and P. Druschel. Measuring the capacity of a Web server under realistic loads. *World Wide Web Journal*, 2(1-2):69–83, 1999.

[14] T. Feder, R. Motwani, R. Panigrahy, and A. Zhu. Web caching with request reordering. *Proc. ACM Symp. on Disc. Alg.*, pp. 104–105, 2002.

[15] K. Psounis and B. Prabhakar. A randomized Web cache replacement scheme. *Proc. IEEE Infocom'01*, pp. 1407-1415, 2001.

[16] E. Cohen and H. Kaplan. Aging through cascaded caches: Performance issues in the distribution of web content. *Proc. ACM Sigcomm'01*, pp. 41–53, 2001.