Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

Computer Networks 44 (2004) 235–246

COMPUTER NETWORKS

www.elsevier.com/locate/comnet

Retrieval and freshness thresholds in hierarchical caching systems

Jianping Pan [a], Y. Thomas Hou [a,*], Bo Li [b,*]

[a] The Bradley Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA 24061, USA
[b] Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong

Received 2 November 2002; received in revised form 19 May 2003; accepted 28 August 2003

Responsible Editor: G. Pacifici

Abstract

To scale up with the explosive Web growth, caching systems have been proposed and deployed over the Internet in recent years. Among them, hierarchical caching systems employing expiration-based consistency control mechanisms have become a viable and efficient solution. In this paper, we first analyze the performance of such hierarchical caching systems from the perspectives of both cache servers and end users. Then, we examine retrieval and freshness threshold-based approaches and their impact on system performance and user-perceived QoS. We show that by setting these thresholds appropriately, it is possible that (1) users can impose a consistency QoS requirement on the object that they wish to obtain without too much trade-off in system performance, and (2) performance bias against leaf users due to their unfavorable locations in the hierarchical structure can be mitigated.
© 2003 Elsevier B.V. All rights reserved.

Keywords: Web caching; Weak consistency; Threshold-based prefetching; Internet

1. Introduction

A challenging problem facing the traditional client-server model is that it cannot scale up to cope with the explosive growth in user (client) population. Popular Web sites such as news portal *cnn.com* and hot event site *saltlake2002.com* must handle requests from a huge number of simultaneous users all over the world. This request surge (also known as *flash crowd*) can easily overload any single Web server and its access links. To address this scalability issue, *Web caching* [3] has been proposed and deployed over the Internet in recent years. The primary goal of caching is to reduce the amount of requests sent to and responses received from the origin server by putting copies of popular content closer to end users; this can be achieved in the form of *browser*, *proxy*, or *reverse* caches. Among different caching systems, a hierarchical one consists of multiple cache servers located at different levels in the caching hierarchy. In [5,13], it has been shown that hierarchical caching systems have the greatest potential to aggregate user requests and reduce Internet traffic. In

[*] Corresponding authors. Tel.: +1-540-231-2950; fax: +1-540-231-8292 (Y.T. Hou), tel.: +852-2358-6976; fax: +852-2358-1477 (B. Li).
E-mail addresses: hou@vt.edu (Y.T. Hou), bli@cs.ust.hk (B. Li).

1389-1286/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2003.08.001

this paper, we focus on some performance issues associated with such systems.

An important issue associated with a caching system is the so-called *object validity*. Since the content of a cacheable object may be modified as time goes on, a cached object may only remain valid for a limited amount of time. When an object is modified, all cached copies become *stale* unless they are updated accordingly. There are two validation paradigms. The first one is called *strong consistency*. The origin server of an object keeps track of all cached copies and updates them immediately when the object is modified. This paradigm is useful for time-critical applications that cannot tolerate any discrepancy in content [12,17], e.g., emergence announcements. Although strong consistency is indispensable for certain services, its complexity is high and thus it is expensive to implement.

On the other hand, most Web-based applications (e.g., Web pages) do not have a stringent requirement on content consistency and can be supported by the so-called *weak consistency* paradigm. Under this paradigm, objects are only validated periodically, and there is a possibility that users may obtain a stale object [11]. However, it is understood that such inconsistency (within a certain limit) will not be harmful, and in most cases users can still extract useful information from the received object. [1] Since weak consistency is acceptable for many Web-based applications and, more importantly, inexpensive to implement, it has been quickly developed and deployed over the Internet.

In this paper, we consider the weak consistency paradigm based on the *time-to-live* (TTL) timing mechanism. TTL and its variants are popular and have been widely used in Web caching (e.g., *Expires* and *Last-Modified* in HTTP server response headers [9]). Under the TTL-based weak consistency paradigm, whenever an object is retrieved from its origin server, its TTL is initialized to a value which reflects the maximum tolerance of discrepancy. Once initialized, the TTL of an object will decrease with time. A cached object is considered valid only if its remaining TTL is positive; otherwise, it is considered expired and must be validated again, either with the origin server or some other cache servers.

We examine the expiration-based hierarchical caching systems in a general setting and will not limit ourselves to any specific application or implementation. We focus on performance issues associated with these systems from the perspectives of both cache servers and end users. We then propose two threshold-based approaches that extend the service capability of the basic hierarchical caching system. The first one is the so-called *freshness* threshold, which enables users to set a minimum freshness requirement for the object that they wish to obtain. With this threshold, users can avoid always receiving the so-called *soon-to-expire* objects. We find that by appropriately selecting the freshness threshold parameter, it is possible to deliver this new service feature without significantly compromising the overall system performance. The second approach is the so-called *retrieval* threshold, which is designed to mitigate the location disadvantage associated with leaf users in a hierarchical caching structure. We find that by adjusting this threshold parameter on cache servers at different levels, hierarchical caching systems can offer an unbiased service to all users, regardless of their locations in the caching hierarchy. Both the retrieval and freshness threshold-based approaches are simple to implement, and thus will undoubtedly further help accelerate the deployment of weak-consistency-based hierarchical caching systems.

The remainder of this paper is organized as follows. In Section 2, we show some basic properties of a hierarchical caching system. In Section 3, we propose the retrieval and freshness threshold-based approaches and show how each of them can help improve object freshness and response time for all users in the hierarchical system. Related work is summarized in Section 4. Section 5 concludes this paper.

2. System model

Fig. 1(a) shows a tree-like hierarchical caching system for an object in our study. If a *cache server*

[1] Moreover, users always have the option to obtain the latest object, e.g., by using the "reload" button in their Web browsers.

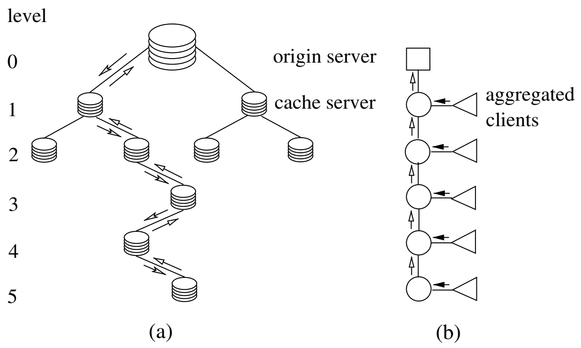

Fig. 1. A hierarchical caching system (a) and its abstract model (b).

(CS) receives a request, and if the requested object is cached locally with a positive TTL, the object is returned immediately with the remaining TTL. Otherwise, the CS generates another request to its *immediate* upstream CS for this object. This process is recursive until a fresh copy (an object with a positive TTL) is obtained. In the worst case, the recursive process terminates at the *origin server* (OS), where the object TTL is initialized to τ. Here, τ represents the maximum amount of time that an object may be inconsistent with the one at the OS. Once initialized, the TTL decreases with time when the object is cached at a CS.

In Fig. 1(b), we take the longest path (or the depth of a tree) and convert requests from all other nodes outside this path as equivalent *Aggregated Client* (AC). If there are multiple longest paths, the choice of a particular path for setting up the simplified abstract model can be made arbitrarily. This simplified abstract model is not meant to be a complete replacement for the original system. Instead, it will solely be used to demonstrate the intrinsic properties of a hierarchical caching system.

Now we focus on the chain-like abstract model in Fig. 1(b). For a CS_d at level d ($1 \leqslant d \leqslant D$ and D is the depth of the tree), we denote the aggregated request rate from AC_d as λ_d. We assume that the aggregated request arrival is a Poisson process. [2]

[2] The request behavior for an individual user can be very complicated. But for the *aggregated* user requests coming from a large user population (e.g., a CS serving users in a metropolitan area), it is reasonable to adopt a Poisson model.

Note that except for leaf CSs, all other CSs also receive requests from their immediate downstream CSs. Under such a system, there are two sets of performance metrics, one measured at AC_d and the other at CS_d. For AC_d, we consider user-perceived response time σ_d^{u}, and for CS_d, we focus on cache miss ratio Γ_d^{s} since it is proportional to the request traffic generated by that CS. These two metrics characterize the performance from the perspectives of end users and cache servers, respectively. Both σ_d^{u} and Γ_d^{s} depend on the current remaining TTL $\tau_d(t)$, which exhibits a *birth–death* renewal process, with its peak value T_d being a random variable defined over $(0, \tau]$. In the rest of this section, we will show some basic properties of hierarchical caching systems.

2.1. Average TTL behavior

A sample path for the TTL behaviors at levels 1 and d is depicted in Fig. 2. Several properties of this TTL evolution path can be observed in this figure.

Property 1. *If an object becomes stale at level* d_1 $(1 \leqslant d_1 \leqslant D)$, *then for any level* d $(d_1 \leqslant d \leqslant D)$, *copies of this object are also stale.*

Note that we have implicitly neglected network delay and server overhead when computing the average TTL, because these delays (which are typically on the order of milliseconds or seconds) are much smaller than τ (which is typically on the order of several hours or even days). If we denote t^* as renewal points for the random process $\tau_d(t)$,

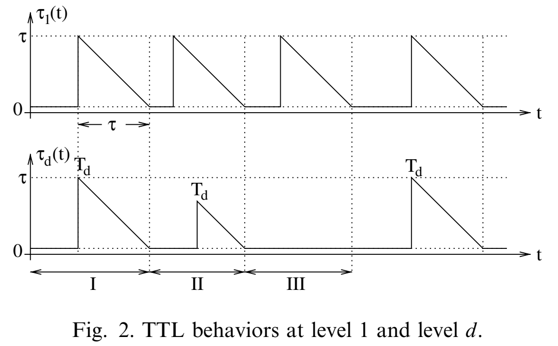

Fig. 2. TTL behaviors at level 1 and level d.

and the time between two successive renewal points as the length of a renewal period, then any two renewal periods are independent and identically distributed. The idle time in each period at CS_d is inversely proportional to the aggregated request rates sent to a subtree rooted at CS_d. We denote the aggregated Poisson request rates at level d as $\Lambda_d = \lambda_d + \lambda_{d+1} + \cdots + \lambda_D$.

Property 2. *There is at most one TTL triangle at level d within any level* 1 *TTL renewal period.*

For each level 1 TTL renewal period, there are three possible scenarios at level d (see Fig. 2):

(I) the level 1 TTL triangle is triggered by a request initially generated within the subtree rooted at CS_d;
(II) the level 1 TTL triangle is triggered by a request initially generated outside the subtree rooted at CS_d, but there is a request generated within the subtree rooted at CS_d during the same level 1 renewal period;
(III) the level 1 TTL triangle is triggered by a request initially generated outside the subtree rooted at CS_d, and there is no request generated within the subtree rooted at CS_d during this level 1 renewal period.

The probability for Scenario I is $p_d^{\mathrm{I}} = \Lambda_d/\Lambda_1$, and the probability for Scenario II is

$$p_d^{\mathrm{II}} = \frac{\Lambda_1 - \Lambda_d}{\Lambda_1} \cdot \int_0^\tau \Lambda_d \cdot \mathrm{e}^{-\Lambda_d \cdot t}\,\mathrm{d}t = \frac{\Lambda_1 - \Lambda_d}{\Lambda_1} \cdot (1 - \mathrm{e}^{-\Lambda_d \cdot \tau}). \tag{1}$$

In Eq. (1), $(\Lambda_1 - \Lambda_d)/\Lambda_1$ is the probability when a request is initially generated outside the subtree rooted at CS_d, and $(1 - \mathrm{e}^{-\Lambda_d \cdot \tau})$ is the probability that there is at least one request generated within the subtree rooted at CS_d during this period. Since these two events are mutually independent, p_d^{II} is the product of these two probabilities. The probability for Scenario III is $p_d^{\mathrm{III}} = 1 - p_d^{\mathrm{I}} - p_d^{\mathrm{II}}$.

In Scenario I, the TTL triangle at level d has a height of $T_d^{\mathrm{I}} = \tau$ as its peak value, where τ is the maximum TTL initialized at the OS. For Scenario II, the TTL triangle has an average height of

$$T_d^{\mathrm{II}} = \frac{\int_0^\tau (\tau - t) \cdot \Lambda_d \cdot \mathrm{e}^{-\Lambda_d \cdot t}\,\mathrm{d}t}{\int_0^\tau \Lambda_d \cdot \mathrm{e}^{-\Lambda_d t}\,\mathrm{d}t} = \frac{\tau - 1/\Lambda_d + \mathrm{e}^{-\Lambda_d \cdot \tau}/\Lambda_d}{1 - \mathrm{e}^{-\Lambda_d \cdot \tau}}. \tag{2}$$

For Scenario III, $T_d^{\mathrm{III}} = 0$. Therefore, the *average* TTL triangle height at level d is

$$E(T_d) = \frac{p_d^{\mathrm{I}} \cdot T_d^{\mathrm{I}} + p_d^{\mathrm{II}} \cdot T_d^{\mathrm{II}}}{p_d^{\mathrm{I}} + p_d^{\mathrm{II}}} = \frac{\Lambda_d \cdot \tau + (\Lambda_1 - \Lambda_d)(\tau - (1 - \mathrm{e}^{-\Lambda_d \cdot \tau})/\Lambda_d)}{\Lambda_d + (\Lambda_1 - \Lambda_d)(1 - \mathrm{e}^{-\Lambda_d \cdot \tau})}. \tag{3}$$

In Fig. 3, by using Eq. (3), we show some numerical results for $E(T_d)$, the average height of a TTL triangle (or average TTL for short), as a function of the total arrival ratio Λ_d/Λ_1 for CS_d, where $\Lambda_1 = 55$.

We use simulations to further demonstrate the behavior of a hierarchical caching system and to reveal its properties and performance. Our simulator is built upon the Network Simulator (ns-2) framework [14] with our add-on agents (i.e., AC, CS, and OS) and configurable parameters such as λ_{AC} and τ. In Fig. 4, we show the behavior of $E(T_d)$, normalized with respect to τ, from both simulation results (with points) and analysis (lines) for a system where $D = 10$. We use the following

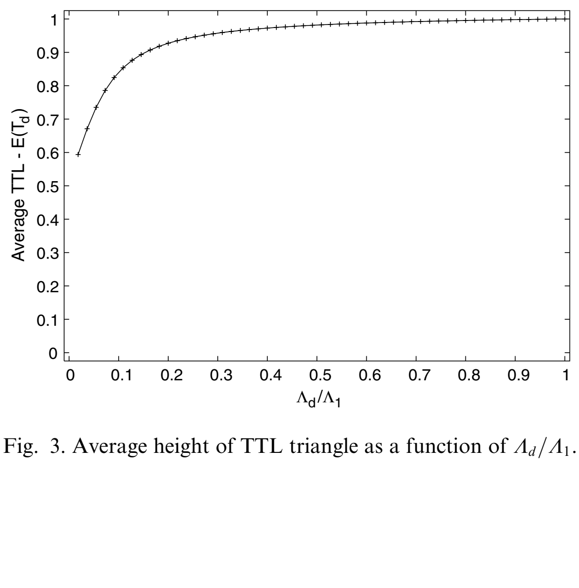

Fig. 3. Average height of TTL triangle as a function of Λ_d/Λ_1.

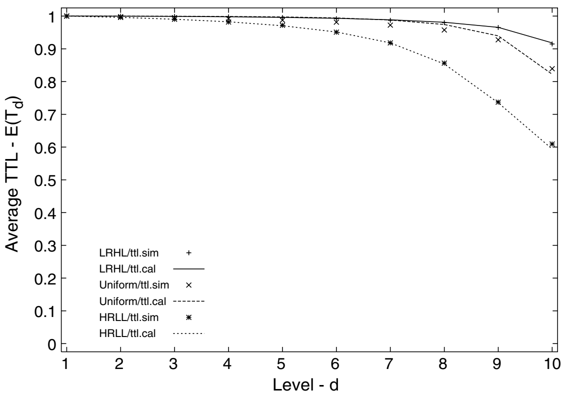

Fig. 4. Average TTL at each level.

three contrasting user request patterns in this simulation:

- Light Root Heavy Leaf (LRHL), i.e., $\lambda_d = d$ for $1 \leqslant d \leqslant D = 10$;
- Uniform, i.e., $\lambda_d = \frac{1}{2}(D+1) = 5.5$ for all $1 \leqslant d \leqslant D = 10$;
- Heavy Root Light Leaf (HRLL), i.e., $\lambda_d = D - d + 1$ for $1 \leqslant d \leqslant D = 10$.

Note that we intentionally set the *total* request arrival rate to be the same under all three traffic patterns in order to show the impact due to different traffic characteristics. The following are several properties extracted from the simulation results.

Property 3. *$E(T_{d_1}) \geqslant E(T_{d_2})$ for any $1 \leqslant d_1 \leqslant d_2 \leqslant D$, i.e., the farther away a CS is from the OS, the smaller average TTL triangle the CS can expect.*

Property 4. *The higher the request rate at a CS, the larger TTL triangle the CS can expect.*

Property 5. $\frac{1}{2}\tau \leqslant E(T_d) \leqslant \tau$ *for* $1 \leqslant d \leqslant D$.

These three properties can also be formally proved based on our earlier analysis for $E(T_d)$ (we omit the proofs here to conserve space). Based on the analytical results for $E(T_d)$ and its properties, in the rest of this section, we show some important performance measures for hierarchical caching systems.

2.2. Cache miss performance

A key metric for a CS is *cache miss rate*, which can be defined as a ratio of cache misses over time or total requests. More precisely, in the first case, the miss rate can be defined as the number of misses per unit time and is denoted as γ_d^s. For γ_d^s, there is 1 miss per renewal period of $I_d + E(T_d)$ at CS_d and $I_d = 1/\Lambda_d$. Therefore,

$$\gamma_d^s = \frac{1}{1/\Lambda_d + E(T_d)}.$$

In the latter case, the miss ratio can be defined as the ratio of the number of misses over the total number of requests and is denoted as Γ_d^s. For Γ_d^s, we consider the following two sub-cases:

1. with probability $p_\Gamma^{(1)} = \lambda_d/\Lambda_d$, the level d TTL triangle is triggered by a request initially generated locally at AC_d. Therefore, for the remaining $E(T_d)$, there are $(\lambda_d + \gamma_{d+1}^s) \cdot E(T_d)$ requests (all hits) generated either locally or from the downstream CS;
2. with probability $p_\Gamma^{(2)} = (\Lambda_d - \lambda_d)/\Lambda_d$, the level d TTL triangle is triggered by a request generated at level d^+, where $d < d^+ \leqslant D$. Therefore, there will be no further requests coming from this particular downstream CS in this TTL triangle period. Thus, for the remaining $E(T_d)$, there are $\lambda_d \cdot E(T_d)$ requests (all hits) that are generated locally at level d.

Combining the above two sub-cases, we have

$$\Gamma_d^s = p_\Gamma^{(1)} \frac{1}{(\lambda_d + \gamma_{d+1}^s)E(T_d)} + p_\Gamma^{(2)} \frac{1}{\lambda_d \cdot E(T_d)}.$$

We can also derive the miss ratio for local requests, denoted as Γ_d^u. All we need to do is to omit the components for downstream requests in Γ_d^s, i.e.,

$$\Gamma_d^u = p_\Gamma^{(1)} \frac{1q}{1 + \lambda_d \cdot E(T_d)}.$$

Fig. 5 plots the cache miss ratio Γ_d^s as a function of CS position (level d) and request patterns. For the uniform request pattern, the cache miss ratio is almost constant at different levels except for a slight increase for leaf CSs. But in the HRLL traffic

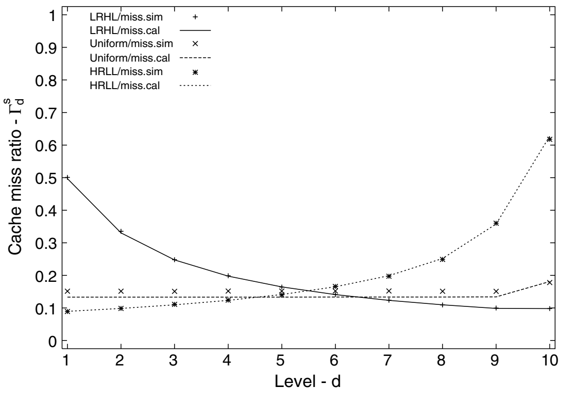

Fig. 5. Cache miss ratio at each level.

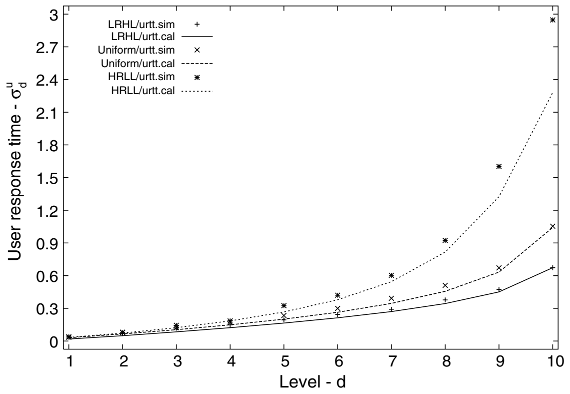

Fig. 6. User response time at each level.

pattern, the cache miss ratio increases quickly as a CS is farther away from the OS. However, for the LRHL request pattern, the increased request rate for leaf CSs can actually compensate for their structural disadvantage. Indeed, the increased request rate reduces the cache miss ratio for leaf CSs. Apparently, the cache miss ratio is directly related to the traffic generated by a CS. Therefore, when designing a hierarchical caching system, it is important to ensure that leaf CSs have sufficient user requests.

2.3. User response time

From the perspective of end users, the ultimate performance measure for a caching system is user's response time, which we denote as σ^{u}_d. Assume that the sum of network delay and server overhead between CS_d and CS_{d-1} is d_d. Then, user response time at level d can be calculated recursively as

$$\sigma^{\mathrm{u}}_d = \Gamma^{\mathrm{u}}_d \cdot \left(d_d + \frac{\Lambda_d}{\Lambda_{d-1}} \cdot \frac{\sigma^{\mathrm{s}}_{d-1}}{\Gamma^{\mathrm{s}}_{d-1}} \right),$$

where $\sigma^{\mathrm{s}}_{d-1}$ is cache response time at level $d-1$, i.e.,

$$\sigma^{\mathrm{s}}_d = \Gamma^{\mathrm{s}}_d \cdot \left(d_d + \frac{\Lambda_d}{\Lambda_{d-1}} \cdot \frac{\sigma^{\mathrm{s}}_{d-1}}{\Gamma^{\mathrm{s}}_{d-1}} \right),$$

and for $d = 1$, $\sigma^{\mathrm{u}}_1 = \Gamma^{\mathrm{u}}_1 \cdot d_1$ and $\sigma^{\mathrm{s}}_1 = \Gamma^{\mathrm{s}}_1 \cdot d_1$.

We again use simulation results to demonstrate the behavior of user response time. In the simulations, d_d is normalized to one delay time unit. In Fig. 6, we plot the user response time as a function of CS position and request pattern. Clearly, there is a performance bias against users that are farther away from the OS, regardless of the particular traffic pattern in the caching system.

Based on these basic properties for a hierarchical caching system, in the next section, we explore approaches in the context of prefetching, in order to extend service features and improve the performance of caching systems.

3. Threshold-based cache prefetching

As we have demonstrated in the previous section, the central theme of a TTL-based weak consistency paradigm is driven by the behavior of object TTL at CSs. From user's perspective, the TTL of a retrieved object gives an upper bound of the period within which the object may still be considered "valid". In the basic model, users do not have any control over object freshness. Therefore, it would be desirable to offer users some control over how fresh they wish the object to be. Another aspect associated with TTL is user response time. The farther away a CS is from the OS, the smaller the average height of the TTL triangle it can expect, which leads to larger average user response time.

In this section, we show that by appropriately imposing different thresholds on TTL, it is possible to offer additional control to users with regard to their freshness requirement and to improve leaf users' response time.

3.1. Threshold parameters

To mitigate performance bias against leaf users (in terms of user response time), we introduce *retrieval threshold*, which is denoted as β_d for a CS at level d. This parameter sets the minimum remaining TTL of the object that can be retrieved by a CS. There are two possible ways to maintain this threshold. One is *proactive*, meaning that whenever the remaining TTL of an object is below β_d at a CS_d, the CS will immediately initiate an internal request to its parent CS_{d-1} and so forth, until an object with the remaining TTL larger than β_d is retrieved. The other approach is *reactive*, meaning that a CS_d will only send a request to its parent CS_{d-1} for an object when there is a request to CS_d and the TTL of the cached object is less than β_d. Since the first approach is overly aggressive in keeping the remaining TTL above β_d, and would generate excessive overhead requests, we will only consider the second approach, which will be elaborated later in this section.

To offer users some control over how fresh the object that they hope to obtain will be, we introduce *freshness threshold*, denoted as α_d for an AC at level d. The freshness threshold is set by users; it reflects user application requirements. It is a measure of how long the user wishes the object to remain fresh after the user obtains the object. Comparing freshness threshold with the retrieval threshold, it is easy to see that we must have $0 \leqslant \alpha_d \leqslant \beta_d \leqslant \tau$, which is shown in the shaded area in Fig. 7. It is also worth noting that when $\beta_d = \tau$, the effect of caching disappears.

In the case when the object's remaining TTL is between α_d and β_d, the CS has two alternatives to serve a user request. With the first approach, the CS may deliver the object to the user immediately since the remaining TTL is greater than α_d, while generating another request to its upstream CS in order to retrieve an object satisfying its retrieval threshold β_d. With the second approach, the CS may temporarily withhold the current user request while initiating another request to its upstream CS for a copy with a remaining TTL greater than β_d.

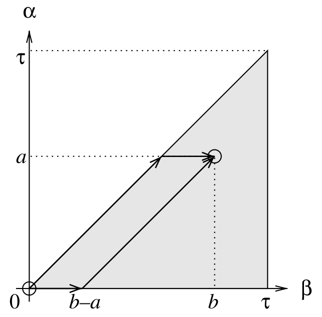

Fig. 7. Relation between retrieval and freshness thresholds.

Upon receipt of this updated object, the CS will also deliver it to the user. Clearly, the withholding strategy could increase user response time (it will also provide better-than-expected object freshness). Here, we will focus on the strategy that does not withhold user request, since users are more sensitive to a longer response time when waiting for an object.

To link the basic model discussed in Section 2 where $\alpha = \beta = 0$ to the threshold-based cache prefetching models where $\alpha = a > 0$ and $\beta = b > 0$, in Fig. 7, we show two possible evolution paths. For the first path, we can first increase β to $b - a$ while keeping $\alpha = 0$. Then we can increase α and β simultaneously to a and b. On the second path, we can first increase (α, β) from $(0, 0)$ to (a, a), and then increase β from a to b while holding $\alpha = a$. Therefore, the general case where $\alpha = a > 0$ and $\beta = b > 0$ can be derived by appropriately shifting α or β from the basic model ($\alpha = \beta = 0$), and it is only necessary to study two building-block cases where $0 \leqslant \alpha = \beta \leqslant \tau$ and $\alpha = 0 \leqslant \beta \leqslant \tau$, respectively.

3.2. Freshness threshold

We first consider the case in which a user has a freshness threshold $\alpha_d > 0$. For simplicity, we assume that users share the same freshness requirement for an object no matter where they are. Therefore, we have $\alpha_d = \alpha$ for all $1 \leqslant d \leqslant D$.

3.2.1. Average TTL behavior

To follow one path in Fig. 7, we start with $\alpha = \beta = \phi$ and observe the TTL behaviors at level

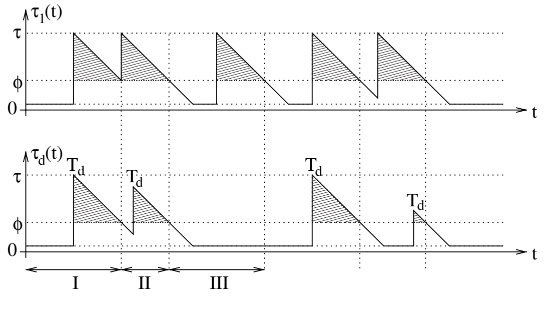

Fig. 8. TTL behavior with freshness threshold.

1 and d, as we did in Fig. 4. Fig. 8 shows the TTL triangle [3] in this case. Comparing Fig. 8 to Fig. 4, we observe that if we redefine the renewal period in Fig. 8, most of the TTL analysis that we have done earlier for Fig. 4 can be applied here. Specifically, in Fig. 8, we define the level 1 renewal point $\tau_1(t^*)$ as $\tau_1(t^*_-) > \phi$ and $\tau_1(t^*) = \phi$. The analysis for the average height of a TTL triangle at level d would then follow the same token as that in Section 2.1, *except* that we need to take into account the vertical shift of ϕ.

To elaborate, we condition the TTL behavior at level d during a given level 1 renewal period on the following three scenarios (also see Section 2.1):

(I) with probability $p^{\mathrm{I}}_{\mathrm{ft}} = \Lambda_d/\Lambda_1$, the level 1 TTL triangle is triggered by a request initially generated within the subtree rooted at CS$_d$, and $T^{\mathrm{I}}_{\mathrm{ft}} = (\tau - \phi) + \phi = \tau$;

(II) with probability

$$p^{\mathrm{II}}_{\mathrm{ft}} = \frac{\Lambda_1 - \Lambda_d}{\Lambda_1} \cdot \int_0^{\tau-\phi} \Lambda_d \cdot \mathrm{e}^{-\Lambda_d \cdot t}\,\mathrm{d}t,$$

the level 1 TTL triangle is triggered by a request generated outside the subtree rooted at CS$_d$, but there is also a request generated within the subtree rooted at CS$_d$ within the same level 1 renewal period. Then we have

$$T^{\mathrm{II}}_{\mathrm{ft}} = \phi + \frac{\int_0^{\tau-\phi} (\tau - \phi)\Lambda_d \cdot \mathrm{e}^{-\Lambda_d \cdot t}\,\mathrm{d}t}{\int_0^{\tau-\phi} \Lambda_d \cdot \mathrm{e}^{-\Lambda_d \cdot t}\,\mathrm{d}t};$$

(III) the level 1 TTL triangle is triggered by a request generated outside the subtree rooted at CS$_d$, and there is no request generated within the level d CS subtree during this level 1 renewal period.

By combining the above three cases, we can calculate the average height of the TTL triangle at level d (denoted as $E_{\mathrm{ft}}(T_d)$ and ft for freshness threshold) as

$$E_{\mathrm{ft}}(T_d) = \frac{p^{\mathrm{I}}_{\mathrm{ft}} \cdot T^{\mathrm{I}}_{\mathrm{ft}} + p^{\mathrm{II}}_{\mathrm{ft}} \cdot T^{\mathrm{II}}_{\mathrm{ft}}}{p^{\mathrm{I}}_{\mathrm{ft}} + p^{\mathrm{II}}_{\mathrm{ft}}} = \frac{\Lambda_d \tau + (\Lambda_1 - \Lambda_d)[\tau - \phi \mathrm{e}^{-\Lambda_d(\tau-\phi)} - \frac{1}{\Lambda_d} + \frac{1}{\Lambda_d}\mathrm{e}^{-\Lambda_d \tau}]}{\Lambda_d + (\Lambda_1 - \Lambda_d)[1 - \mathrm{e}^{-\Lambda_d(\tau-\phi)}]}.$$

After some algebraic manipulation, we have

$$E_{\mathrm{ft}}(T_d) = \phi + E_{\tau \overset{\mathrm{def}}{=} \tau - \phi}(T_d). \tag{4}$$

That is, $E_{\mathrm{ft}}(T_d)$ can be obtained by replacing τ in Eq. (3) with $\tau - \phi$, plus a positive shift of ϕ. By using the same token, it is straightforward to obtain $\Gamma^{\mathrm{s}}_{\mathrm{ft},d}$, $\sigma^{\mathrm{u}}_{\mathrm{ft},d}$, and other performance metrics.

We use numerical results to substantiate the above analysis. Fig. 9 shows the average height of the TTL triangle at each level when user requests follow the HRLL traffic pattern in the caching system. Recall that HRLL is the worst case traffic pattern among the three that we have discussed (see Fig. 4). In Fig. 9. We used $\alpha_d = \beta_d = \phi$ and $\phi \in \{0, 0.1\tau, 0.3\tau, 0.7\tau, \tau\}$. In particular, the case

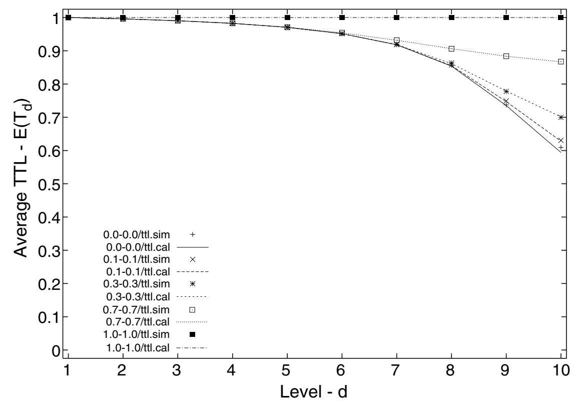

Fig. 9. Average height of TTL triangle at each level with freshness threshold and HRLL request pattern.

[3] Although the TTL evolution is no longer a triangle-like curve for the case with threshold-based prefetching, for the sake of presentation simplicity, we still refer it as *triangle*.

when $\phi = 0$ corresponds to the basic case where there is no freshness threshold. By increasing ϕ, the average height of the TTL triangle is pushed upward. In the extreme case when $\phi = \tau$, each request must go to the OS, which corresponds to the case without caching in the system.

3.2.2. Cache miss ratio

In Fig. 10, we show cache miss ratio with the same traffic pattern and parameters used in Fig. 9. In the case of $\alpha_d = \tau$, $1 \leqslant d \leqslant D$, we have $E_{\text{ft},\phi=\tau}(T_d) = \tau$. This is the same case as the one under a *flat* caching system where each CS sends requests directly to the OS. [4] Clearly, we have a trade-off between object freshness and network traffic: the fresher the object delivered to the user, the higher cache miss ratio the system will experience, with a resulting increase in network traffic. When α is already large (e.g., $\alpha = 0.7\tau$), a slight increase in α could bring a large increase in $\Gamma^{\text{s}}_{\text{ft},d}$ (see Fig. 10).

3.2.3. User response time

We now address user response time when there is a freshness threshold. Intuitively, one would expect that user response time would increase substantially if users impose additional freshness requirements on the objects. It turns out that this is not the case. Fig. 11 shows the user response time at each level d, with the same traffic pattern and parameters used in Fig. 9. The case of $\phi = 0$ corresponds to no freshness threshold (as in the case we discussed in an earlier section). Over a wide range of ϕ, $0 < \phi < 0.7$, the increase of user response time is barely noticeable. Only when ϕ is substantially large (i.e., $\phi \geqslant 0.7$) do we see a significant increase in user response time for leaf level users. This fact demonstrates that the freshness threshold is a viable service option that can be offered to users without much performance compromise in user response time.

[4] To be more precise, the actual performance here would be even worse than a flat system since each upstream CS is queried in this update process, bringing additional processing overhead to these servers.

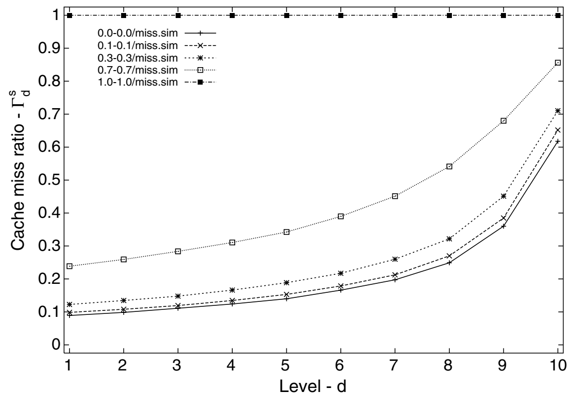

Fig. 10. Cache miss ratio at each level with freshness threshold and HRLL request pattern.

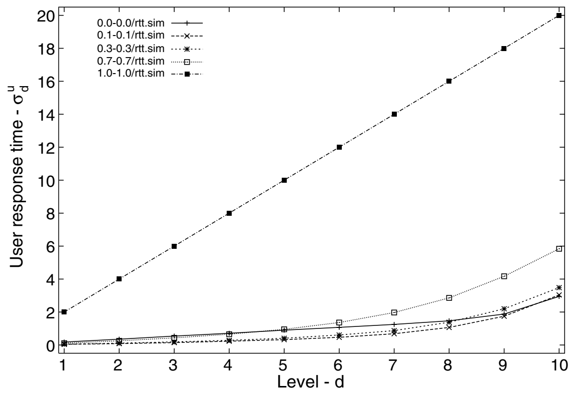

Fig. 11. User response time at each level with freshness threshold and HRLL request pattern.

3.3. Retrieval threshold

Now we consider the situation when a CS_d has a retrieval threshold $\beta_d > 0$, while $\alpha_d = 0$ for AC_d.

3.3.1. Average TTL behavior

We first consider the average TTL with a retrieval threshold (rt), $E_{\text{rt}}(T_d)$, at CS_d with β_d. When a request with $\alpha_d = 0$ from AC_d arrives, it is not hard to see that this is the same case as when $\alpha = \beta$ in Section 3.2.1. Eventually, CS_d has to get a copy of the requested object with a remaining TTL larger than β_d, although the user request can be fulfilled earlier with a copy of less freshness. Therefore, we omit plotting $E_{\text{rt}}(T_d)$ at CS_d here, since $E_{\text{rt}}(T_d) = E_{\text{ft}}(T_d)$ when $\beta_d = \phi$. However, since in Section 3.1

we adopt the strategy without request withholding and proactive retrieval, the cache miss rate/ratio, and all user-oriented performance metrics (including user perceived object TTL, miss rate/ratio, and response time), will be different from the case only with a freshness threshold.

3.3.2. Cache miss ratio

Here again we only present the results with the HRLL request pattern, since this is the worst case scenario for leaf users. We assume that different CSs can choose different β_d according to their location (d) in a hierarchy. Without loss of generality, we plot in Fig. 12 the cache system miss ratio for $\beta_D \in \{0, 0.1\tau, 0.3\tau, 0.7\tau, \tau\}$ (i.e., only the leaf CS_D adopts a retrieval threshold), and all other parameters are the same as those used in Fig. 5. The case when $\beta_D = 0$ is equivalent to the basic model without employing a retrieval threshold. As expected, when the retrieval threshold β_D increases, the cache system miss rate also increases, since we do count prefetching as a result of cache *miss*. Furthermore, for the certain range of $0 < \beta_D < 0.7$, the increase in cache miss ratio is not very significant.

In Fig. 13, we plot the *user* miss ratio for the same simulation. Here, we find that when the retrieval threshold increases, there is improvement (i.e., reduction) in the user miss ratio. Such improvement is particularly profound for leaf users. The larger the retrieval threshold β_D is, the smaller the user miss ratio becomes.

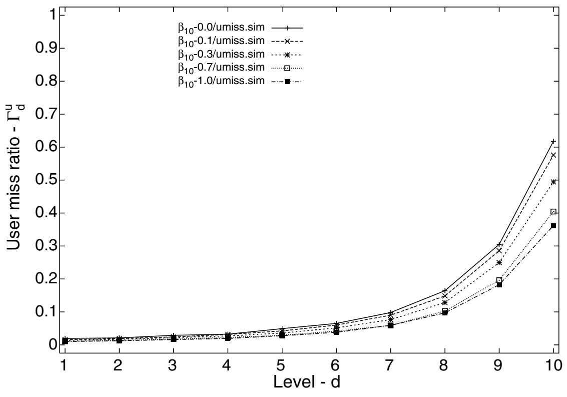

Fig. 13. User miss ratio at each level with retrieval threshold and HRLL request pattern.

3.3.3. User response time

We now discuss the improvement of user response time as a consequence of retrieval threshold. Fig. 14 shows that $\sigma^{\mathrm{u}}_{\mathrm{rt},d}$ is reduced accordingly with the increase of β_D. Again, the reduction of user response time is significant for leaf users. However, when $\beta_D > 0.7\tau$, a further increase in β_D only results in a minor reduction in user response time. In the extreme case when $\beta_D = \tau$, even though all user requests must be sent to the OS (resulting in a large volume of network traffic), the improvement of user response time becomes negligible when compared to the case that $\beta_D = 0.7\tau$. This fact suggests that the retrieval threshold needs to be carefully tuned toward the proper operating point to achieve the best trade-off in terms of

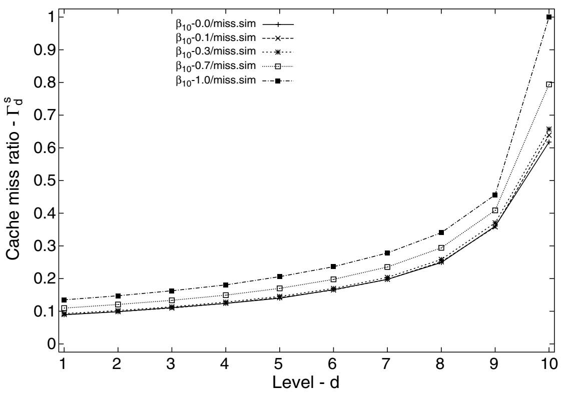

Fig. 12. Cache miss ratio at each level with retrieval threshold and HRLL request pattern.

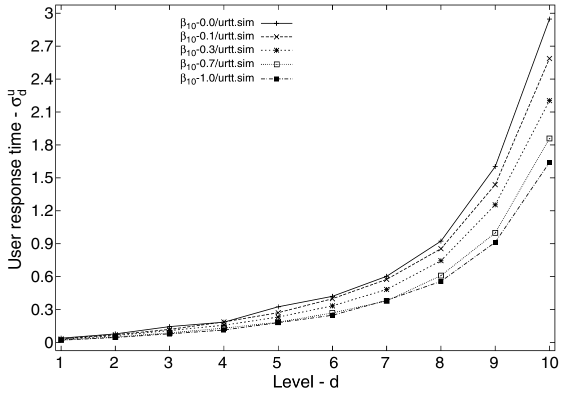

Fig. 14. User response time at each level with retrieval threshold and HRLL request pattern.

performance improvement in user response time and additional overhead in network traffic.

4. Related work

Web caching has attracted many research efforts in recent years [16]. Most work focuses on the user (or Web browser), proxy, and server characterization by collecting trace logs, fitting to known statistical distributions, and regenerating requests in simulation or a controlled testbed [1,2]. There are several related work on cache consistency and hierarchical caching [5,11,13,15], but they are mainly in the context of protocol design, empirical simulation, and experimental measurement. In this paper, we take an analytical approach to better understand the intrinsic properties in hierarchical caching systems built upon the TTL-based expiration scheme.

Cohen et al. [8] considered a two layer hierarchical system with the TTL scheme, but they are more focused on the miss rate and user request pattern. Their work motivates us to further explore the TTL behaviors at different levels, especially within a hierarchical structure. In our work, we also analytically derive cache miss ratio and user response time based on the obtained TTL model. Che et al. [6] focused on the impact of access frequency of multiple objects on cache replacement algorithms and modeled it as a tandem of low-pass filters. On the other hand, the focus in this paper is on the temporal properties, not the spatial constraints. Our goal is to show how to set up different types of thresholds to control the TTL behavior, deliver richer service features to end users, and improve leaf user's response time. Prefetching has been proposed in the literature (see [4,7,10]). In contrast to previous work, we explore how different threshold settings can deliver various performance-cost trade-offs for a hierarchical caching system.

5. Conclusions

In this paper, we first developed an analytical model for hierarchical caching systems with the TTL-based expiration mechanism. Then we assessed the performance of such systems from the perspectives of both cache servers and end users. We also revealed the performance issues associated with the structure of the caching system. By introducing the cache prefetching based on freshness and retrieval thresholds, we showed that a hierarchical caching system could be enhanced to offer more control options to users and to mitigate performance issues related to caching structure.

References

[1] G. Abdulla, E. Fox, M. Abrams, Shared user behavior on the World-Wide Web, in: Proc. WebNet'97, 1997.

[2] M. Arlitt, C. Williamson, Internet Web servers: workload characterization and implications, IEEE/ACM Trans. Network. 5 (5) (1997) 631–644.

[3] G. Barish, K. Obraczka, World Wide Web caching: trends and techniques, IEEE Commun. Mag. 38 (5) (2000) 178–184.

[4] P. Cao, E. Felten, A. Karlin, K. Li, A study of integrated prefetching and caching strategies, in: Proc. ACM SIGMETRICS'95, 1995.

[5] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, K. Worrell, A hierarchical Internet object cache, in: Proc. USENIX'96, 1996, pp. 153–163.

[6] H. Che, Z. Wang, Y. Tung, Analysis and design of hierarchical Web caching systems, in: Proc. IEEE INFOCOM'01, 2001, pp. 1416–1424.

[7] E. Cohen, H. Kaplan, Prefetching the means for document transfer: a new approach for reducing Web latency, in: Proc. IEEE INFOCOM'2000, 2000.

[8] E. Cohen, H. Kaplan, Aging through cascaded caches: performance issues in the distribution of Web content, in: Proc. ACM SIGCOMM'01, 2001, pp. 41–53.

[9] A. Dingle, Cache consistency in the HTTP 1.1 draft, in: ICM Workshop on Web Caching, 1996.

[10] D. Duchamp, Prefetching hyperlinks, in: Proc. 2nd USENIX Symp. on Internet Tech. and Syst., 1999.

[11] J. Gwertzman, M. Seltzer, World-Wide Web caching consistency, in: Proc. USENIX'96, 1996, pp. 141–151.

[12] C. Liu, P. Cao, Maintaining strong cache consistency for the World-Wide Web, IEEE Trans. Comp. 47 (4) (1998) 445–457.

[13] A. Mahanti, C. Williamson, D. Eager, Traffic analysis of a Web proxy caching hierarchy, IEEE Network 14 (3) (2000) 16–23.

[14] Network Simulator—ns-2. Available from <http://www.isi.edu/nsnam/ns>.

[15] A. Rousskov, V. Soloviev, A performance study of the Squid proxy on HTTP/1.0, World-Wide Web J. 2 (1/2) (1999) 47–67.

[16] J. Wang, A survey of Web caching schemes for the Internet, ACM Comp. Commun. Rev. 25 (9) (1999) 36–46.
[17] H. Yu, L. Breslau, S. Shenker, A scalable Web cache consistency architecture, in: Proc. ACM SIGCOMM'99, 1999.

Jianping Pan obtained his B.S. and Ph.D. degrees in Computer Science from Southeast University, Nanjing, China in 1994 and 1998, respectively. From 1999 to 2001, he was a Postdoctoral Fellow and Research Associate with the Center for Wireless Communications at University of Waterloo, Waterloo, Ontario, Canada. Since September 2001, he has been a Member of Research Staff at Fujitsu Laboratories of America, Sunnyvale, California, USA. He was a Visiting Researcher at Virginia Tech during summer 2003. His research interests include transport protocols and application services for multimedia, high-speed and mobile networks. He is a member of ACM and IEEE.

Y. Thomas Hou obtained his B.E. degree from the City College of New York in 1991, the M.S. degree from Columbia University in 1993, and the Ph.D. degree from Polytechnic University, Brooklyn, New York, in 1998, all in Electrical Engineering. From 1997 to 2002, he was a research scientist and project leader at Fujitsu Laboratories of America, IP Networking Research Department, Sunnyvale, California (Silicon Valley). He is currently an Assistant Professor at Virginia Tech, The Bradley Department of Electrical and Computer Engineering, Blacksburg, Virginia. His research interests include wireless video sensor networks, multimedia delivery over wireless networks, scalable architectures, protocols, and mechanisms for differentiated services Internet, and service overlay networking. He has published extensively in the above areas and is a co-recipient of the 2002 IEEE International Conference on Network Protocols (ICNP) Best Paper Award and the 2001 IEEE Transactions on Circuits and Systems for Video Technology Best Paper Award. He is a member of IEEE and ACM.

Bo Li received the B.S. and M.S. degrees in Computer Science from Tsinghua University, Beijing, P.R. China, in 1987 and 1989, respectively, and the Ph.D. degree in Computer Engineering from the University of Massachusetts at Amherst in 1993. Between 1994 and 1996, he worked on high performance routers and ATM switches in IBM Networking System Division, Research Triangle Park, North Carolina. Since January 1996, he has been with Computer Science Department, the Hong Kong University of Science and Technology, where he is now an Associated Professor. He also holds an Adjunct Researcher position at Microsoft Research Asia (MSRA), Beijing, China. His current research interests include wireless and mobile networks supporting multimedia, video multicast, and all optical networks with WDM. He has published over 140 technical papers in referred journals and conference proceedings. He has been an Editor or a Guest Editor for 14 journals, and involved in the organization of over 30 conferences. He currently serves as the Co-Chair of the Technical Program Committee of IEEE Infocom'2004. He is a member of ACM and a senior member of IEEE.