Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT®

Performance Evaluation 55 (2004) 51–68

PERFORMANCE EVALUATION
An International Journal

www.elsevier.com/locate/peva

Analysis and evaluation of expiration-based hierarchical caching systems

Y. Thomas Hou[a,*], Jianping Pan[b,1]

[a] *The Bradley Department of Electrical and Computer Engineering, Virgin Tech, Blacksburg, VA, USA*
[b] *University of Waterloo, Waterloo, Ont., Canada*

Abstract

This paper investigates some fundamental properties and performance issues of the expiration-based caching systems. We focus on the hierarchical caching systems based on the *time-to-live* (TTL) expiration mechanism, and present a basic model for such systems. By analyzing the intrinsic timing behavior in this model, we derive some important performance metrics from the perspectives of caching systems and end users, respectively. We use network simulation results to further substantiate the efficacy of our analysis. Our results show some basic properties and trade-offs for a hierarchical caching system based on the weak consistency mechanism.

Keywords: Web caching; Hierarchy; Weak consistency; Time-to-live; Internet

1. Introduction

An important issue in the design of any caching systems is to maintain some level of *consistency* between cached copies of an object and the object maintained at the origin server (OS). Every time the original object is updated at the OS, copies of that object cached elsewhere become *stale*. The value of cached copies would be greatly reduced if they are not updated accordingly. Caching consistency mechanisms ensure that cached copies of an object are eventually updated to reflect changes to the original object. Depending on how soon the cached copies are updated, cache consistency mechanisms fall into two major categories: *strong consistency* and *weak consistency*.

Under strong consistency, upon an update of an object at the OS, the OS immediately notifies all cache servers (CSs) about this update [6]. Example caching applications that require strong consistency include time-sensitive content delivery (e.g., emergency public announcements). The main problem associated with strong consistency mechanisms (e.g., invalidation [5]) is that they often involve higher overhead and

* Corresponding author. Tel.: +1-540-231-2950; fax: +1-540-231-8292.
E-mail address: thou@vt.edu (Y.T. Hou).
[1] Present address: Fujitsu Laboratories of America, Sunnyvale, CA, USA.

0166-5316/$ – see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0166-5316(03)00099-3

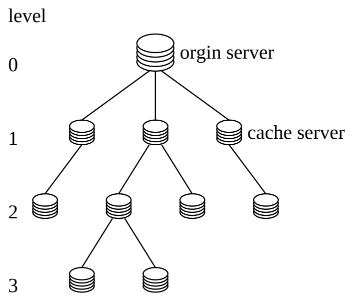


Fig. 1. An example of a hierarchical caching system based on the tree topology.

complexity and are expensive to deploy. Nevertheless, strong consistency is an indispensable approach to deliver mission critical contents on the web.

On the other hand, under weak consistency, it is acceptable to let a user get a somewhat stale object from the CS. The CS only validates the object's freshness with the OS periodically and may lag behind the actual update at the OS. Weak consistency is particularly useful for those web contents that can tolerate a certain degree of discrepancy between the cached content and the content at OS as long as it is understood that such discrepancy does not cause any problem. It is important to keep such discrepancy not to exceed a reasonable period of time. Example applications using weak consistency include online newspapers and magazines, personal homepages, and the majority of web sites—although the original content may be further updated at the OS, it is still useful (or at least not harmful) to retrieve the cached copy at a cache or proxy server.[2] It has been shown that weak cache consistency is a viable and economic approach to deliver content that does not have stringent freshness requirement [5].

To support weak consistency, the concept of *time-to-live* (TTL) is introduced. TTL is an a priori estimate of an object's remaining lifetime and can be used to determine how long a cached object remains useful. Under the TTL approach, each object is initialized with a TTL value, which is supposed to decrease with time if the object is cached elsewhere. An object that has been cached longer than its initial TTL is said to *expire* and the next request for this object will cause the object to be requested (or validated) from the OS or other CSs that have a copy with an unexpired TTL. In practice, the TTL-based strategy is easy to implement (e.g., by using the "Expires" or "Last-Modified" fields in HTTP header [8]).

There are many alternative approaches to construct a caching infrastructure. However, it has been shown [2] that a *hierarchically* organized caching infrastructure is particularly effective to scale up web growth since the Internet topology also tends to be organized hierarchically. In this paper, we consider hierarchical caching systems and investigate their performance and behaviors under the weak consistency paradigm. Although the current HTTP protocols on web caching provide a lot of similar features [4], we intend to conduct our investigation in a more general setting and will not limit ourselves to particular details of the current HTTP implementation.

We start with a basic model, which is a generic hierarchical caching system based on the tree topology. Under the basic model, the root node represents the OS whereas all the other nodes in the tree represent CSs (see Fig. 1). We assume each node (or CS) is deployed in a metropolitan region and the user

[2] A user always has the option to reload the fresh content from the OS if he/she prefers to have the most updated copy of the object.

requests[3] within this particular metro region always go to this regional CS for content service. When the object is not available or its TTL has expired at a CS, the CS will query its immediate parent CS, which may further query its immediate parent CS and so forth, until a "fresh" copy of the object is retrieved or the origin root server is reached. Here, a "fresh" copy refers to a copy of the object with an unexpired TTL (i.e., non-zero TTL). The OS always maintains a current copy of the object and will initialize the TTL of an object upon request. The TTL value for an object at any CS decreases with time. Since TTL is a fundamental parameter that determines the intrinsic behavior of the overall hierarchical caching systems, we analyze the behavior of TTL for a CS at each level of the tree. Based on this analysis, we conduct a performance study for the hierarchical caching systems from the perspectives of both the caching system and end user by deriving performance metrics such as hit/miss rate, response time, and network load. We use simulation results to substantiate the accuracy of our analysis and provide insights on various system design trade-offs.

The remainder of this paper is organized as follows. In Section 2, we present the basic model for the hierarchical caching systems based on the weak consistency mechanism. We also analyze the TTL behavior of the basic model and derive its performance metrics. In Section 3, we present simulation results to substantiate our analytical results for the basic model. Section 4 discusses related work and Section 5 concludes this paper.

2. Modeling and analysis

In this section, we first describe the basic model under our study, and then analyze its performance and behavior.

2.1. Basic model

We assume that the hierarchical caching systems follow a tree structure (see Fig. 1). At level 0, we have one origin (root) server,[4] S_0, which always maintains the latest (updated) copy of an object. The root server is logically connected to some child servers which we refer to as level 1 CSs, each of which are geographically located at different metro regions. Level 1 CS may also connect to some child servers which we call level 2 CSs, and so forth. Finally, a CS that does not have any child CS is called a leaf CS. The maximum number of levels of such a hierarchical tree is called the height of the tree. Fig. 1 shows a simple example of the caching systems based on a tree structure with the height of 3.

We further assume that the aggregated user requests to the CS within a metro region follow a Poisson process.[5] When a user request arrives at the CS, if the object already exists at the CS and its TTL is still greater than 0, the CS will deliver the object to the user. We consider such an event a *user hit*. On the other hand, when the user request arrives at the local CS, if the object does not exist or the TTL timer has

[3] Note that each user may also have the browser cache built on its host and here the user request refers to the request sent to the *proxy CS* by the user after a miss at its own browser cache. That is, we only consider the requests sent to the proxy CS from the user and not consider those request that can be served by the user's own browser cache.

[4] Note that the root server may consist of a cluster of physical servers. But we assume that geographically they all locate at the same site (e.g., an ISPs data center).

[5] An exact traffic model for each individual user is hard to find. However, it is reasonable to assume that, for a large population in a metro region, the aggregated requests follow a Poisson process.

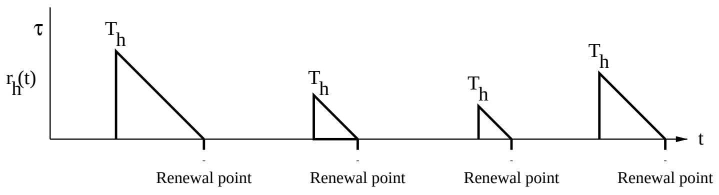


Fig. 2. A sample path of the TTL for a CS at level h.

expired (i.e., decreased to 0), we consider such an event a *user miss*. When a miss happens, the local CS will generate another request and query its immediate parent CS to see if it has the object with a valid TTL. If this parent CS does have this object with an unexpired TTL, we call this event a *system hit* since the request is generated by a child CS rather than *directly* from a user. Upon a system hit, the object will be delivered to the CS and will subsequently be delivered to the user. Otherwise, we have a *system miss* and the parent CS will generate a request and further query its own parent CS and so forth, until the query process reaches the origin root server, in which case we assume that the origin root server always maintains an updated fresh copy of the object. The origin root server will deliver the object with a TTL field initialized to maximum lifetime τ, where $\tau > 0$, and the TTL value decreases linearly as time goes on. Thus, the maximum *age* that an object (delivered to a user) can have under such hierarchical caching systems is bounded by τ. Under the basic model, upon an event of a system hit, not only the user will be delivered a copy of the object with an updated TTL, *all* CSs involved in the query process will also get a copy of this object with an updated TTL.

Note that we distinguish hit rate and miss rate from user and cache system perspectives. Such distinction will help us better understand the details of the system behaviors, as we shall elaborate shortly. To maintain such distinction, at a CS, we need to distinguish user requests and system requests: user requests come from the metro region within the coverage of this local CS while system requests come from its child (including grandchild, etc.) CSs which are caused by user requests from their corresponding remote metro regions.

2.2. Performance analysis

In this section, we investigate the performance of the basic model. We conduct performance evaluation along two dimensions: *caching system performance* and *end user quality of experience*. By caching system's performance, we refer to the behavior and properties of the hierarchical caching structure, such as the aggregated behaviors of TTL, hit/miss rate, average response time, and traffic load at each CS.[6] On the other hand, user's quality of experience refers to user's perceived quality in content delivery, e.g., hit/miss rate, average response time, which only considers requests from end users and does not include auxiliary traffic within the hierarchical caching systems.

2.2.1. Average TTL behaviors

Suppose we are at a CS of level h, $1 \leq h \leq H$, where H is the height of the tree. Denote the (remaining) TTL at the CS as $r_h(t)$. Then $r_h(t)$ is a renewal process [9], with the renewal point starting at time $t = t_*$ when $r_h(t)$ just decreases to 0, i.e., $r_h(t_*) = 0$ and $r_h(t_*^-) > 0$ (see Fig. 2). It should be clear that when

[6] By *aggregated*, we count both external user requests as well as internal requests from child CSs.

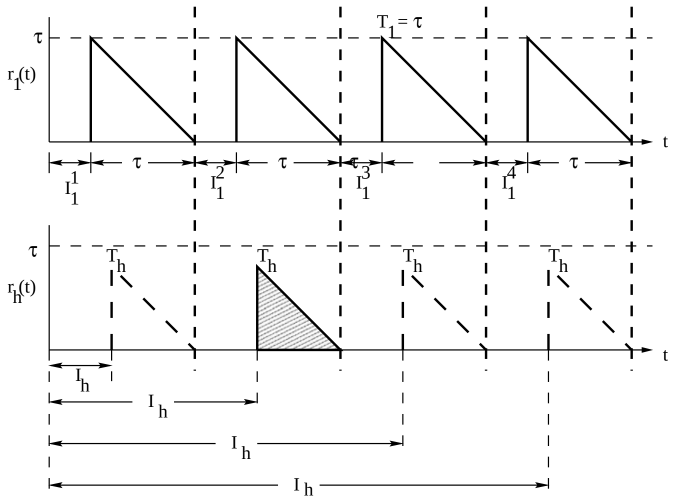


Fig. 3. A sample path of the TTL behavior of a CS at level h in reference to the TTL behavior at level 1.

$r_{h^*}(t) = 0$, then for all $h^* \leq h \leq H$, $r_h(t) = 0$. This is because that under our basic model, an object maintained at a parent CS always has its remaining TTL larger than or equal to the remaining TTL of the same object maintained at its child CSs.

Referring to Fig. 2, denote the *peak* value of $r_h(t)$ during each renewal period as T_h, $1 \leq h \leq H$. Then we have $T_1 = \tau$ and T_h is a random variable defined over $(0, \tau]$ for $2 \leq h \leq H$. We are interested in the average value of T_h, for $2 \leq h \leq H$, denoted as $E(T_h)$, which is an important system parameter.

Due to the nature of the hierarchical tree and TTL-based expiration, there is an important property on TTL that *links* the CSs at all levels. In particular, any TTL renewal point at level h, $2 \leq h \leq H$ (see Fig. 3) coincides (or *synchronizes*) with a renewal point at level 1. However, the converse is not true, i.e., a renewal point at level 1 may not be a renewal point at level h, $2 \leq h \leq H$. This is because that the smaller the h for a CS, the more child servers (and thus the higher user population) it supports, which translates into smaller *idle* period (the time period when $r_h(t)$ remains 0). This observation leads to the fact that the average renewal period at level h, $2 \leq h \leq H$, is larger than the average renewal period at level 1. To be more precise, the average renewal period at each level increases with h, with the smallest at level 1 and largest at level H.

Referring to Fig. 3, for each renewal period at level 1, it is clear that the first request that initiates the TTL triangle within the renewal period follows a Poisson process with the rate Λ_1, which is the *sum* of all Poisson arrival rates at all CSs of the tree with S_1 being its root. This Poisson process (with rate Λ_1) can be considered of an *aggregate* of two Poisson processes: the first with a rate of Λ_h representing the arrivals at the sub-tree with S_h as the root and the second (with a rate of $\Lambda_1 - \Lambda_h$) representing arrivals from the rest of the tree within S_1 excluding the sub-tree S_h. Clearly, the probability that the TTL triangle is initiated by a request from the sub-tree with root S_h is Λ_h/Λ_1 and the probability that the TTL is initiated by a request from the rest of tree (i.e., $S_1 \setminus S_h$) is $(\Lambda_1 - \Lambda_h)/\Lambda_1$.

We now look at the time interval at level h that corresponds to the same renewal period at level 1. There are three cases, and the sum of probabilities of these three cases is 1.

Case 1. With probability Λ_h/Λ_1, the TTL triangle at level 1 is initiated by a request from the sub-tree with root at S_h. In this case, it is obvious that $T_h = \tau$.

Case 2. With probability

$$\frac{\Lambda_1 - \Lambda_h}{\Lambda_1} \int_0^{\tau} \Lambda_h \, \mathrm{e}^{-\Lambda_h t} \, \mathrm{d}t = \frac{\Lambda_1 - \Lambda_h}{\Lambda_1} (1 - \mathrm{e}^{-\Lambda_h \tau}),$$

there is a request arrival within τ from the sub-tree with root S_h. In this case, the level 1 TTL triangle is not triggered by this request as in Case 1. The average T_h in this case is given by

$$\frac{\int_0^{\tau} (\tau - t) \Lambda_h \, \mathrm{e}^{-\Lambda_h t} \, \mathrm{d}t}{\int_0^{\tau} \Lambda_h \, \mathrm{e}^{-\Lambda_h t} \, \mathrm{d}t} = \frac{\tau - 1/\Lambda_h + (1/\Lambda_h) \, \mathrm{e}^{-\Lambda_h \tau}}{1 - \mathrm{e}^{-\Lambda_h \tau}}. \tag{1}$$

It can be shown that the average T_h in this case (i.e., the right-hand side of (1)) is always greater than $\tau/2$. This can be intuitively explained by the Poisson property of the arrival process.

Case 3. With probability of

$$\frac{\Lambda_1 - \Lambda_h}{\Lambda_1} \int_{\tau}^{\infty} \Lambda_h \, \mathrm{e}^{-\Lambda_h t} \, \mathrm{d}t = \frac{\Lambda_1 - \Lambda_h}{\Lambda_1} \, \mathrm{e}^{-\Lambda_h \tau},$$

there is no request arrival within τ in this interval. In this case, there is no TTL triangle in this interval.

To calculate $E(T_h)$, all we need to do is to take the probabilistically weighted average of T_h under Cases 1 and 2. We have

$$\begin{aligned} E(T_h) &= \frac{(\Lambda_h/\Lambda_1) \cdot \tau + [(\Lambda_1 - \Lambda_h)/\Lambda_1] \cdot (1 - \mathrm{e}^{-\Lambda_h \tau}) \cdot [\tau - 1/\Lambda_h + (1/\Lambda_h) \, \mathrm{e}^{-\Lambda_h \tau}]/(1 - \mathrm{e}^{-\Lambda_h \tau})}{\Lambda_h/\Lambda_1 + [(\Lambda_1 - \Lambda_h)/\Lambda_1] \cdot (1 - \mathrm{e}^{-\Lambda_h \tau})} \\ &= \frac{\Lambda_h \tau + (\Lambda_1 - \Lambda_h)[\tau - 1/\Lambda_h + (1/\Lambda_h) \, \mathrm{e}^{-\Lambda_h \tau}]}{\Lambda_h + (\Lambda_1 - \Lambda_h)(1 - \mathrm{e}^{-\Lambda_h \tau})}. \end{aligned} \tag{2}$$

Property 1. *Under the basic model, the average of T_h at level h, $h = 1, 2, \ldots, H$, has the following property*:

$$\tau \geq E(T_h) > \tfrac{1}{2}\tau, \tag{3}$$

and

$$E(T_1) > E(T_2) > \cdots > E(T_h) > \cdots > E(T_H). \tag{4}$$

The proof for the above properties for $E(T_h)$ is straightforward and is omitted here.

2.2.2. System and user performance metrics

As mentioned earlier, we distinguish the performance metrics along two dimensions: system oriented performance and user perceived performance. Denote Γ_h^{s}, Θ_h^{s}, and σ_h^{s} as the system miss rate, hit rate, and response time at a CS of level h, respectively. The system miss rate, hit rate, and response time take into account all requests, both from the users in the (local) metro region and from child CSs (internal dynamics within the hierarchical caching tree). Similarly, denote Γ_h^{u}, Θ_h^{u}, and σ_h^{u} as the user perceived miss rate, hit rate, and response time for users directly served by a CS of level h, respectively. The user perceived performance parameters consider only requests generated by the users in the local metro region and do not consider those requests forwarded from any child CSs.

Before we calculate the miss rate Γ_h^{s} at level h, we make the following observation (see Fig. 3): each CS can make *at most* one request to its parent CS during any renewal cycle. Denote M_h the number of requests a CS at level h receives from its child CSs during a renewal cycle and c_h the number of child CSs of this server at level h. Then M_h is a random variable defined over $0, 1, \ldots, c_h$ and the probability distribution of M_h is a combinatorial exponential distributions—due to the fact that the user requests at a CS of any level follow a Poisson distribution. Therefore, $E(M_h)$ can be easily calculated explicitly using combinatorics and $E(M_h) \leq c_h$.

To calculate the miss rate, Γ_h^{s}, we condition on whether the first miss (i.e., the request that initiates the TTL triangle) is from a user of the CSs metro region or from a child CS. We have,

$$\Gamma_h^{\mathrm{s}} = \frac{\lambda_h}{\Lambda_h}\left\{\frac{1}{1+[\lambda_h \cdot E(T_h) + E(M_h)]}\right\} + \frac{\Lambda_h - \lambda_h}{\Lambda_h}\left\{\frac{1}{1+[\lambda_h \cdot E(T_h) + E(M_h) - 1]}\right\}. \tag{5}$$

Note that for a constant λ_h and c_h at each level h, the miss rate strictly increases as the level h increases.

The system's hit rate, $\Theta_h^{\mathrm{s}} = 1 - \Gamma_h^{\mathrm{s}}$, is then

$$\Theta_h^{\mathrm{s}} = \frac{\lambda_h}{\Lambda_h}\left\{\frac{\lambda_h \cdot E(T_h) + E(M_h)}{1+[\lambda_h \cdot E(T_h) + E(M_h)]}\right\} + \frac{\Lambda_h - \lambda_h}{\Lambda_h}\left\{\frac{\lambda_h \cdot E(T_h) + E(M_h) - 1}{1+[\lambda_h \cdot E(T_h) + E(M_h) - 1]}\right\}. \tag{6}$$

Denote d_h as the round trip time (rtt) (including processing delay at the CS) between a child CS at level h and its *immediate* parent CS, and assume the delay between an end user and its local CS is negligible. The average system response time, σ_h^{s}, is therefore

$$\sigma_h^{\mathrm{s}} = \Theta_h^{\mathrm{s}} \cdot 0 + \Gamma_h^{\mathrm{s}} \cdot \pi_h, \tag{7}$$

where π_h is the delay until getting a fresh object given that there is a miss at the local CS. From (7), we have

$$\pi_h = \frac{\sigma_h^{\mathrm{s}}}{\Gamma_h^{\mathrm{s}}}. \tag{8}$$

On the other hand,

$$\pi_h = d_n + \left\{\frac{\Lambda_{h-1} - \Lambda_h}{\Lambda_{h-1}} \cdot 0 + \frac{\Lambda_h}{\Lambda_{h-1}} \cdot \pi_{h-1}\right\}. \tag{9}$$

Combining (7)–(9), we have the following recursive relationship for σ_h^{s}:

$$\sigma_h^{\mathrm{s}} = \Gamma_h^{\mathrm{s}} \cdot \left(d_h + \frac{\Lambda_h}{\Lambda_{h-1}} \cdot \pi_{h-1}\right) = \Gamma_h^{\mathrm{s}} \cdot \left(d_h + \frac{\Lambda_h}{\Lambda_{h-1}\Gamma_{h-1}^{\mathrm{s}}} \cdot \sigma_{h-1}^{\mathrm{s}}\right) \tag{10}$$

with $\sigma_1^{\mathrm{s}} = \Gamma_1^{\mathrm{s}} \cdot d_1$.

We now calculate the user perceived hit rate (Θ_h^{u}), miss rate (Γ_h^{u}), and response time (σ_h^{u}), for users directly served by a CS of level h. These performance metrics will be slightly different from those corresponding to the system performance. This is because we need to filter out the effect of the requests from child CSs (which represent internal dynamics of the hierarchical caching system). Again, by conditioning on whether the first request comes from local users or child CSs, we have

$$\Theta_h^{\mathrm{u}} = \frac{\lambda_h}{\Lambda_h}\left\{\frac{\lambda_h \cdot E(T_h)}{1+\lambda_h \cdot E(T_h)}\right\} + \frac{\Lambda_h - \lambda_h}{\Lambda_h} \cdot 1. \tag{11}$$

As $\Gamma_h^{\mathrm{u}} = 1 - \Theta_h^{\mathrm{u}}$, we have, for the users' miss rate,

$$\Gamma_h^{\mathrm{u}} = \frac{\lambda_h}{\Lambda_h}\left\{\frac{1}{1+\lambda_h \cdot E(T_h)}\right\}. \tag{12}$$

The response time that a user experiences is

$$\sigma_h^{\mathrm{u}} = \Theta_h^{\mathrm{u}} \cdot 0 + \Gamma_h^{\mathrm{u}} \cdot \pi_h = \Gamma_h^{\mathrm{u}} \cdot \left(d_h + \frac{\Lambda_h}{\Lambda_{h-1}\Gamma_{h-1}^{\mathrm{s}}} \cdot \sigma_{h-1}^{\mathrm{s}}\right) \tag{13}$$

with $\sigma_1^{\mathrm{u}} = \Gamma_1^{\mathrm{u}} \cdot d_1$.

2.2.3. *Network traffic load*

One of the major benefits of a caching system is to reduce the overall network traffic load and thus to achieve *scalability* as the Internet grows. Here, we calculate the network load associated with the hierarchical caching systems. One way to measure network traffic load for the hierarchical caching systems is to perform an accounting on how much traffic each CS generates to its immediate parent CS. Note that a CS will initiate a request to its parent CS only when a request (either from local users in the metro region or from child CSs) incurs a miss. Denote the average request rate that a CS S_h at level h sends to its parent CS as ρ_h^{s}. By definition, we have

$$\rho_h^{\mathrm{s}} = \frac{1}{E(I_h + T_h)}, \tag{14}$$

where I_h is the idle period during a renewal cycle for a CS at level h and $E(I_h) = 1/\Lambda_h$.

3. Simulation investigation

In this section, we use simulation results to demonstrate the intrinsic properties of hierarchical caching systems.

3.1. *Simulation settings*

Our simulation is built on the network simulator *ns-2* platform [7]. We define three new objects, namely, OS, CS, and *aggregated clients* (AC) as follows. An OS is a reply-only object which always returns the requested object with its TTL value initialized to τ. An AC is a request-only object with the aggregated request rate λ_{AC}. For an CS object, if it has the requested object with a positive (unexpired) TTL, it will return such object to the local AC or its child CS; otherwise, the CS will generate another request to its immediate parent CS. This process is recursive until the request reaches the OS. For each CS and AC objects at each level of the hierarchy, we attach a logging facility in the simulation to record requests and replies events, which will be used for off-line data processing.

Fig. 4 shows the three topologies of caching systems in our simulation study. Under the "flat" caching topology (see Fig. 4(a)), each CS can only make requests directly to the OS. Therefore, under the flat structure, the level number only represents the logical distance between a CS and the OS. On the other hand, the "chain" (Fig. 4(b)) and "tree" (Fig. 4(c)) topologies represent two basic hierarchical scenarios.

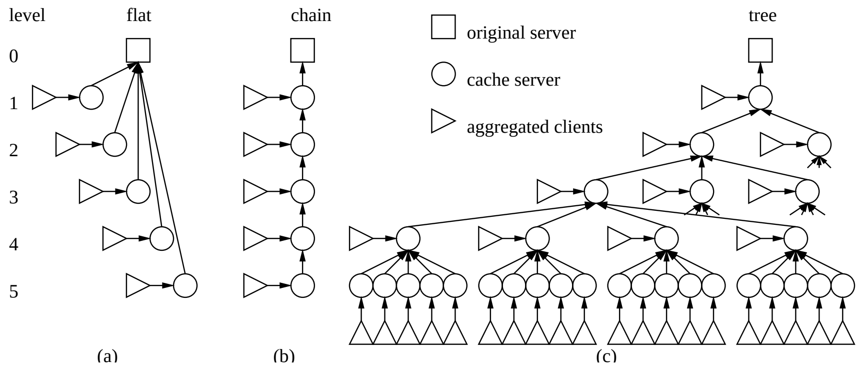


Fig. 4. Topologies of caching systems used in the simulation study: (a) flat, (b) chain, and (c) tree.

Here, upon a miss of a request (i.e., the CS does not have the object with a positive TTL), a CS will always generate another request to its *immediate* parent CS.

Also note that the chain topology is a special case for the tree topology with a child-span of 1. For the results presented here, we use a binary *complete* tree (with a span of two for all non-leaf CSs). We set the maximum number of levels $H = 10$ for all these three topologies. We use the rtt between two consecutive levels as a measure of logical distance. For illustrative purpose, in our simulation, we set the rtt between two consecutive levels to 2 units. As an example, a CS corresponding to level 4 will have 8 units of rtt between itself and the OS.

We will show the simulation results when $\tau = 1$ unit of TTL time[7] and $\lambda_h = 1$ per TTL unit time for all h (i.e., the same user request rate at each level), unless otherwise stated explicitly in Section 3.2.4. To obtain the average value from simulation samples, we repeat our simulations multiple times to minimize the deviation and to converge toward the mean value, each time with a random initial seed. Also, to eliminate any transit effect from the initial system warm-up period, we run each simulation over a sufficient period of time.

3.2. Simulation results and discussions

In this section, we present simulation results and discussions for the hierarchical caching systems. We organize our presentation as follows. First, we examine the TTL behavior (i.e., $E(T_h)$) at each CS, which is the most important parameter in characterizing the dynamics of the hierarchical caching systems. Then, we present the simulation results for the performance metrics from both CSs and user's perspectives. This is followed by a study of traffic load generated by the CSs of each level. Finally, we show how the user request patterns can affect the performance outlook of caching systems.

3.2.1. Average TTL behavior

To get a clear picture on how TTL behaves, we first present a set of simulation snapshots showing the TTL behaviors at different levels under the chain topology. Fig. 5 shows the sample TTL evolution (in

[7] It is worth pointing out that the TTL time (usually in several hours to few days) is much larger than the rtt time (in hundreds of milliseconds to few seconds). Thus, we can omit the rtt when calculating the remaining TTL. But we will consider rtt when calculating the response time.

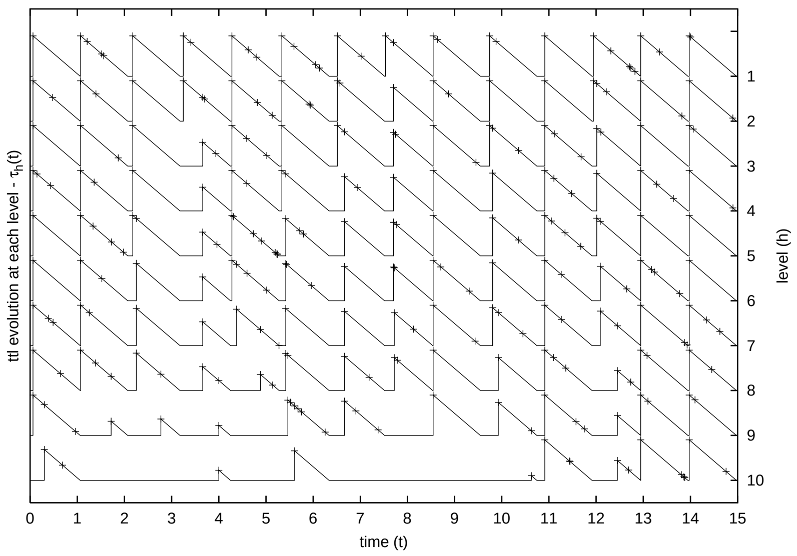


Fig. 5. A sample path of TTL evolution at level h, $1 \leq h \leq 10$ under the chain topology.

connected lines) and received requests (cross points) at each level during a time window in the simulation. As expected, each TTL triangle falls within its parent CSs TTL triangle, with each renewal point when TTL decreases to 0 being synchronized to a renewal point of its parent CS. The smaller the h is (i.e., the closer a CS to the OS in the hierarchy), the denser the TTL triangle within the time window. This is because the request aggregation and object sharing increase as a CS is closer to the OS.

We now examine the TTL behavior quantitatively and compare it with our closed form result in Section 2.2. In Fig. 6, we plot $E(T_h)$ as a function of h, using values from calculation (Eq. (2)) and simulation, respectively, for the three topologies. The connected lines represent the calculated results from our analysis while the disconnected points shows the results extracted from the simulation results. Clearly, our analysis for $E(T_h)$, $1 \leq h \leq H$ matches the actual simulations for all cases. With the flat structure, each CS always gets the object with the maximum TTL ($\tau = 1$) (at the expense of larger response time and traffic load). Under the chain or tree topologies, the average TTL $E(T_h)$ at a CS is always lower than that in the flat structure as expected. A smaller TTL implies that, when an object is delivered to the a user or child CS, the object is of less freshness since it might already be cached for a while although still valid in the weak consistency paradigm. Also, we observe that the TTL behaviors for both the chain and tree topologies strictly follow Property 1, i.e., $1 \geq E(T_1) > E(T_2) > \cdots > E(T_{10}) > 1/2$.

3.2.2. *System and user performance metrics*

We now move on to our present analysis and simulation results for the performance metrics from both CSs and user’s perspectives. In particular, we show (in Fig. 7) the hit rate[8] and response time (delay) from a CSs (i.e., system) and user’s perspectives.

In Fig. 7(a), we plot the hit rate at CSs of each level from both analysis (Eq. (6)) and simulation results for the flat, chain, and tree topologies. Clearly, our analysis matches simulation results very well. For the

[8] Since miss rate is the complement of the hit rate, we omit to present its simulation results to conserve space in the paper.

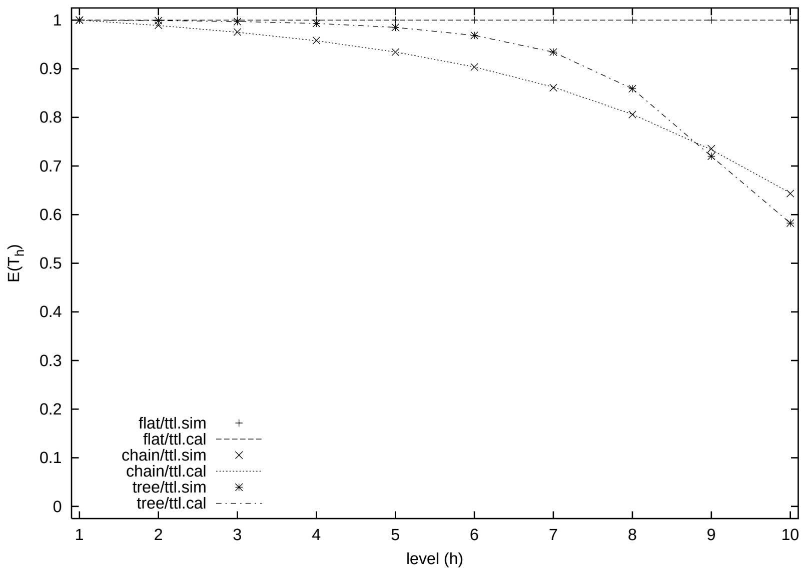
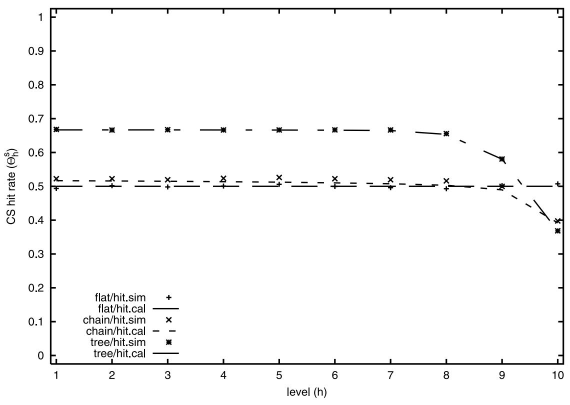


Fig. 6. $E(T_h)$ behaviors for the flat, chain, and tree topologies.

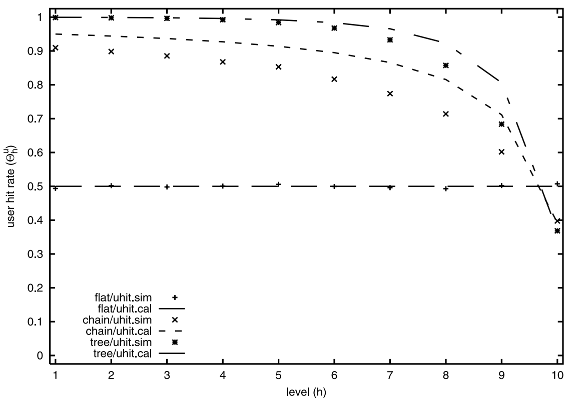
flat structure, the hit rate is the same for CSs at all levels (i.e., 0.5)—due to the fact that each CS interacts with the OS directly and independently from other CSs. On the other hand, the hit rate under chain or tree topology exhibits the non-increasing behavior. Furthermore, the CS hit rate for the tree is higher than that for the chain at almost all levels except the leaf CSs, and the hit rate for the chain is in turn higher than that for the flat topology. This demonstrates the effect of request aggregation and object sharing under the hierarchical caching systems. That is, a CS (except the leaf CS) under the tree topology handles a higher volume of requests than a CS at the same level under the chain or flat topologies.

At the leaf CS, mainly due to a smaller TTL and also with less request aggregation and object sharing, the hit rate under chain or tree topology is even lower than the flat structure. That shows that the system favors CSs closer to the OS and penalizes CSs faraway from the OS, which is the intrinsic limitation of any hierarchical caching systems. One of our future work is to seek techniques to alleviate such bias against leaf CSs in an hierarchical system.

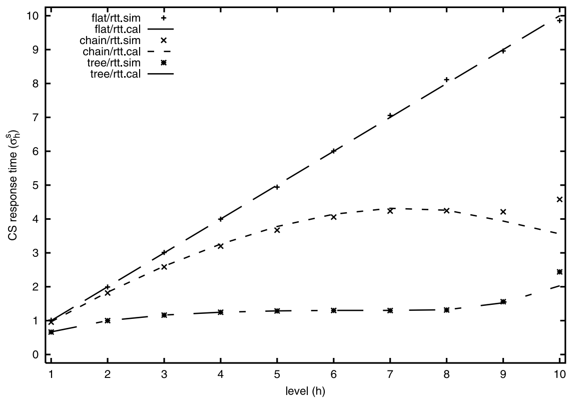
In Fig. 7(b), we plot the hit rate experienced by a user at each level through both analysis (Eq. (11)) and simulation results for the flat, chain, and tree topologies. We observe the similar hit rate behaviors from user's perspective. For the flat topology, the user perceived performance is always the same as the system metrics since there is no inter-CS requests. Comparing Fig. 7(b) to (a), we find that, under the chain or tree topology, a CS's hit rate is always less than the user's hit rate. This is because once there is a miss at a CS for a particular user request, this request may trigger *multiple* misses at CSs along the upstream path toward the OS, which leads to a higher CS miss rate (or lower CS hit rate). We name this as "miss synchronization", and another part of our future work will explore techniques to avoid such synchronization which can cause traffic surge among the path back to the OS. We also find that, for a leaf CS, users' hit rate (as well as the response time to be discussed in the following) is the same as that for the leaf CS under all topologies. This can be easily explained by the fact that a leaf CS can only have requests from users (i.e., no child CS below it).



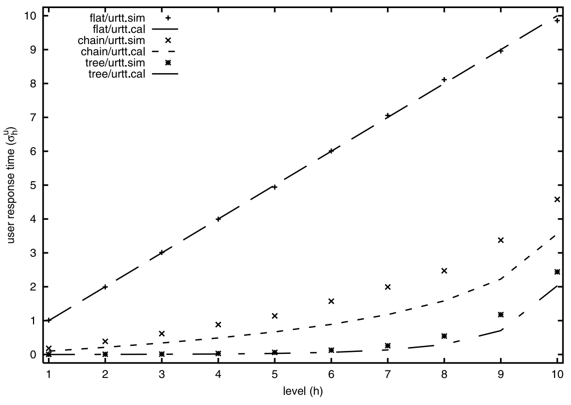
(a) Cache hit rate.



(b) User hit rate.



(c) Cache response time.

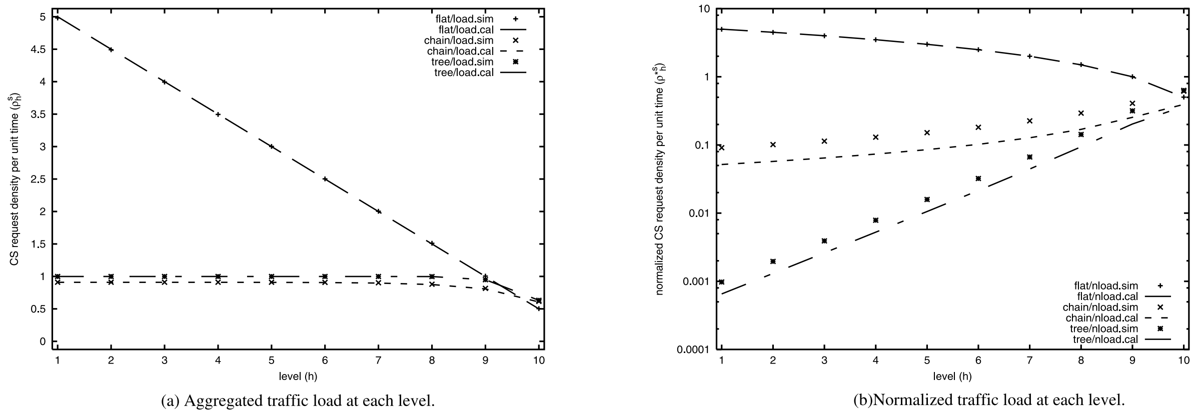


(d) User response time.

Fig. 7. Cache and user hit rate and response time under the flat, chain, and tree topologies.

Fig. 7(c) and (d) shows the response time for a request from a CS and user's perspectives at each level, respectively. The response time shown in the figure is in unit of time with the rtt between two consecutive levels of CS being 2 units (see Section 3.1 for our simulation settings). Therefore, the average response time is proportional to (more precisely, twice) the average number of levels (or CSs) that a request needs to travel (query) in order to get a hit.

In Fig. 7(c), under the flat topology, the response time increases linearly as the level increases, which is expected. Since the hit rate is 0.5 based on our simulation settings (i.e., $\tau = 1$ and $\lambda_h = 1$ for all levels), the distance (measured in time) between the leaf CS (at level 10) and the OS is 20 units, the average response time for a leaf CS is, therefore, 10 units. The response time under the tree topology is smaller than that under the chain topology, and the response time under the chain topology is much smaller than that under the flat topology, especially for the leaf CSs and users. In particular, under the tree topology, a request only travels less than 1 hop upward (or 2 unit of *rtt*) on average to get a hit. This is a substantial improvement than under the flat topology, where the response time increases linearly with the level number h. This demonstrates that, under the hierarchical caching systems, a request can be fulfilled by a nearby CS along its upstream path (toward to OS). Fig. 7(d) shows the response time



(a) Aggregated traffic load at each level.
(b)Normalized traffic load at each level.

Fig. 8. Traffic load at each level under the flat, chain, and tree topologies.

from user's perspective at each level for all three topologies. We have similar observations as in Fig. 7(c). Furthermore, user perceived response time is even lower than that for the system for the chain or tree topology since in most cases, when a user request comes to its local CS, the object is very likely valid in cache due to a previous user or system request.

3.2.3. *Network traffic load*

Another important performance behavior for caching system is the traffic load, in particular, the request and response traffic traveling through the network and processed by CSs. From the perspective of network providers, such traffic measure is an important input for network capacity planning and traffic engineering. On the other hand, from the perspective of service providers (or content delivery providers), traffic load distribution among CSs is directly related to user perceived latency as well as an indication of whether any additional load balancing is necessary.

Fig. 8(a) shows the aggregated request traffic volume at each level, which is the sum of request traffic traversing the same level for the three topologies. Again due to the request aggregation and object sharing, the hierarchical caching systems (i.e., chain or tree) have much lower traffic load at most levels (except the leaf level) than the flat structure. In particular, under the flat topology, the closer to the OS, the higher network traffic load, which poses a potential congestion bottleneck at or near the OS. In contrast, for the hierarchical caching systems under the chain or tree topology, network traffic is evenly distributed at all levels, which fulfills the objective of load balancing for both the network providers and service providers. For the leaf level, similar to the reason with the cache hit rate in Fig. 7(a), hierarchical systems can have slightly higher traffic demand than the flat one due to a lower TTL and less request aggregation and object sharing.

We need to point out that the *lump sum* network traffic at the same level in Fig. 8(a) does not fully demonstrate the superior advantage of the highly aggregated hierarchical caching systems such as the tree topology. This is because the user population supported under the tree topology is much larger than that under the chain or flat topology. To illustrate this point, in Fig. 8(b), we plot the *normalized traffic load*, defined as the ratio of request traffic summed over the CSs at the same level normalized with respect to the total number of user requests received at the same level. For clarity, we use the $\log_{10}$ scale for the vertical axis in Fig. 8(b) due to the small numerical scale of the result for the tree topology. In Fig. 8(b), we find that, for the tree topology, the normalized traffic load per request at each level is several orders lower than that under the flat topology for upstream CSs (i.e., CSs close to the OS), due to its ability to aggregate requests. At the leaf level, the advantage of tree topology disappears since there is no more request aggregation and object sharing at a leaf CS.

3.2.4. User request pattern

So far our simulation results are based on the uniform request pattern at each level (see Section 3.1), where user request rate $\lambda_h = 1$ for all $h = 1, 2, \ldots, 10$. In this set of simulation results, we further explore how non-uniform user request patterns at each level can affect the system's and user's performance outlook. In particular, we consider two contrary scenarios:

- *Heavy root-light leaf* (*HR-LL*). This represents the case where the closer it is toward the OS, the more user request rate a CS receives from its local metro region. In our simulation study, we choose $\lambda_1 = 1.9$, $\lambda_2 = 1.7, \ldots, \lambda_{10} = 0.1$.
- *Light root-heavy leaf* (*LR-HL*). This represents the opposite of the HR-LL scenario. Here, the farther away it is from the OS, the larger the user request rate a CS receives from its local metro region. In our simulation study, we choose $\lambda_1 = 0.1$, $\lambda_2 = 0.3, \ldots, \lambda_{10} = 1.9$.

As we can find that in both scenarios, the total system request $\sum_{\mathrm{AC}} \lambda_{\mathrm{AC}}$ is the same as the one in the previous sections (referred to as *uniform* scenario). We want to compare their results with the previous sections to show how the request distribution can affect the system and user performance outlook.

For illustration purpose, we will only present simulation results and numerical calculation for the chain topology. Fig. 9 shows the TTL behaviors for the HR-LL, uniform, and LR-HL user request patterns. We observe that the TTL behavior under these traffic patterns still strictly follow Property 1, i.e., $1 \geq E(T_1) > E(T_2) > \cdots > E(T_{10}) > 1/2$. However, even when the total request is the same, due to the difference in user request patterns, $E(T_h)$ for the HR-LL case is smaller than that for the uniform case, and $E(T_h)$ for the uniform case is in turn smaller than that for the LR-HL case. This is intuitive since under

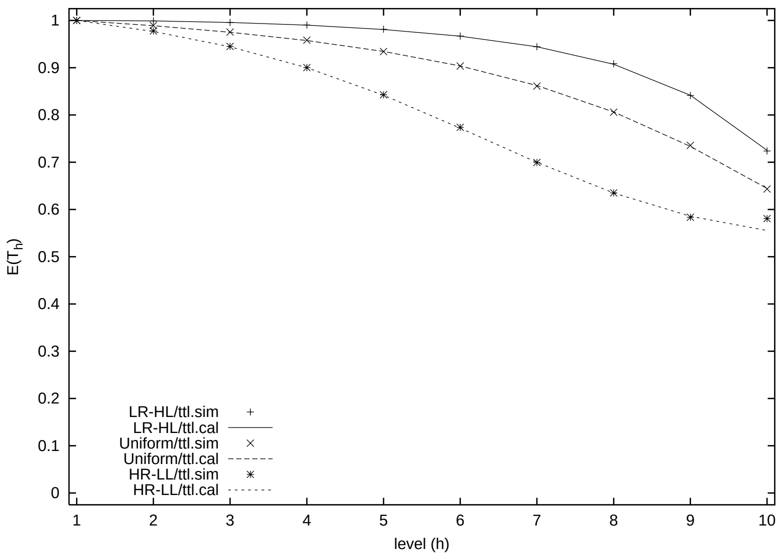
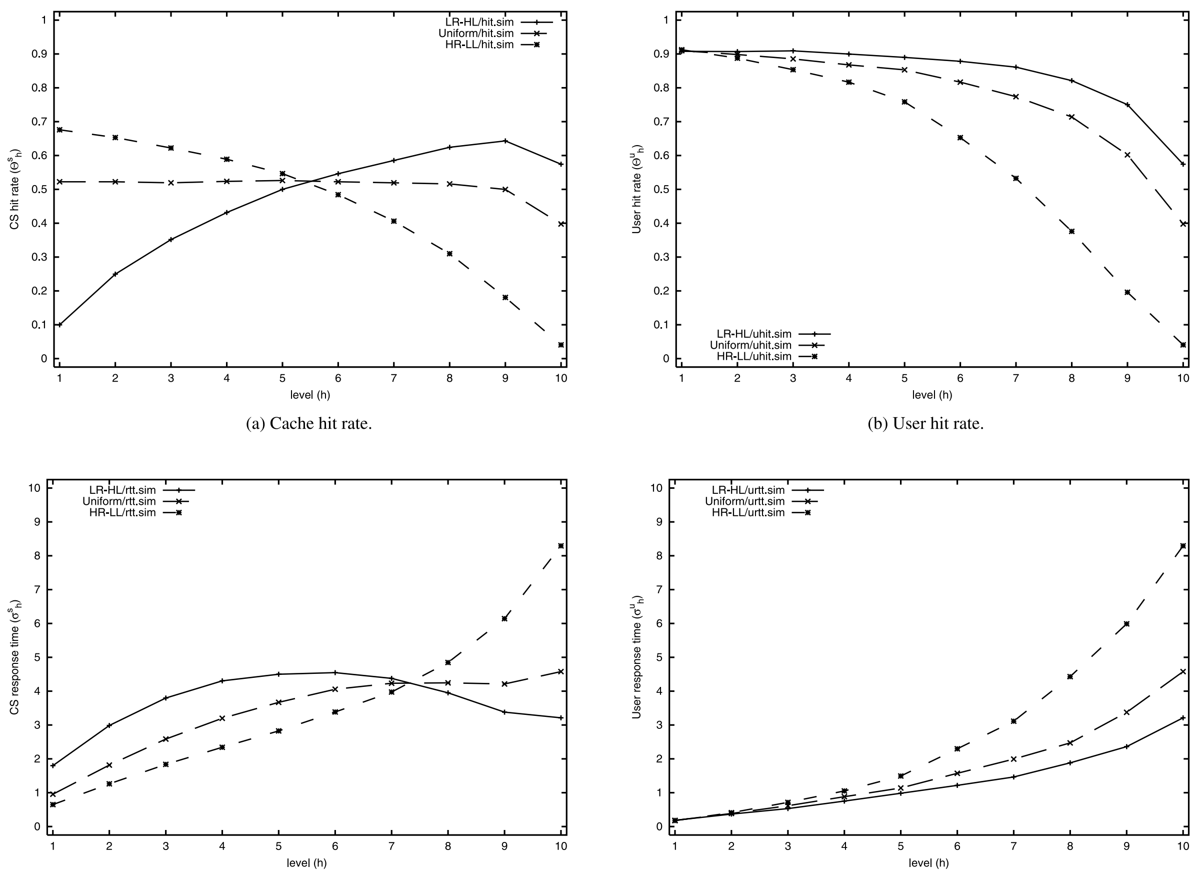


Fig. 9. TTL behaviors under different request patterns for the chain topology.



(a) Cache hit rate.

(b) User hit rate.

(c) Cache response time.

(d) User response time.

Fig. 10. System and user performance under different request patterns for the chain topology.

the LR-HL case, more user requests at the leaf CS help to compensate the loss of request aggregation and object sharing due to its farther distance from the OS. Even when the request rate changes gradually, the aggregated effect on TTL at different levels is still considerable. As simulation results closely match our analytical results, we will only present simulation results for the rest of this section.

In Fig. 10(a), we plot the CS hit rate at each level for the HR-LL, uniform, and LR-HL cases. We find that the hit rate can have drastically different behaviors under different user request patterns. In particular, we observe that the higher user request rate at a CS, the higher the CS hit rate will be. For HR-LL, due to the fewer user requests, less request aggregation and object sharing, the cache hit rate decreases very quickly for CSs close to the leaf. However, for LR-HL, the less request aggregation and object sharing are effectively compensated by the increasing user request; except for the leaf CS, cache hit rate increases as h grows. On the other hand, Fig. 10(b) shows the user hit rate at each level under the HR-LL, uniform, and LR-HL request patterns. Obviously, we find that the user hit rate performance favors the LR-HL traffic pattern than that for the HR-LL traffic pattern since for users at all levels, LR-HL always has the highest hit rate consistently.

Table 1
A comparison of performance versus topologies and traffic patterns

Topology	Scenario	TTL	User hit rate	User response time
Flat	Uniform	1.000	0.500	5.500
Chain	LR-HL	0.935	0.840	1.295
	Uniform	0.881	0.772	1.719
	HR-LL	0.794	0.603	2.797
Tree	Binary complete	0.904	0.878	0.466

Fig. 10(c) shows the response time at CS for different levels under these three request patterns (also see Fig. 10(a)). Although the response time for these cases have different performance for CSs at different levels, the CS response time under the LR-HL traffic pattern is more desirable than that under the HR-LL traffic pattern due to its concave behavior. For HR-LL, due to less request and farther away from the OS, when h grows, cache response time increases very quickly. However, for LR-HL, initially cache response time increases when h grows since it becomes farther away from the OS. But as h further increases, the cache response time will actually decrease, due to the increase in object sharing under HL traffic pattern. Fig. 10(d) shows the user perceived response time under these three request patterns. Again, we find that the performance under the LR-HL traffic pattern is consistently better than that under the HR-LL traffic pattern. Overall, we find that the user request distribution among the CSs can have great impact on system and user behaviors and performance metrics. Fig. 10 suggests that when forming a hierarchical system, network and service providers should maintain the LR-HL-like user request pattern to increase the system hit rate and to reduce the user response time. It also suggests that by increasing the request rate with higher user population or additional auxiliary requests introduced by prefetching, etc., the topology disadvantage for leaf CSs in a hierarchical caching systems can be somehow alleviated.

3.2.5. Summary of simulation results

From the above results for the TTL behavior, hit rate, response time, traffic load, and user request pattern, we observed some important properties and trade-offs for the flat and hierarchical caching systems. Table 1 shows some numerical results for comparison purpose. For a flat topology, although it can obtain an object with the largest TTL initialized at the OS, it usually has higher miss rate (or lower hit rate) and larger response time, and generates more and uneven traffic load (especially in term of the per request load) than a hierarchical system (chain or tree). Under the hierarchical caching system, we observe that there is a bias against leaf CS, which is due to the loss of request aggregation and object sharing at leaf CS. Through our simulation results, we have also conclusively demonstrated that, for content distribution employing the weak consistency paradigm, a hierarchical caching system is a scalable solution. When adopting a hierarchical caching system, it is also important for the network and service providers to arrange the user request pattern in a proper manner to maximize the performance potential and avoid some bias due to topology configuration.

4. Related work

Although caching has been used in distributed computer and database systems for many years, its potential in solving Internet scalability has only been discovered and explored in recent years. There are some prior research efforts on web caching based on weak consistency. For example, in [1,5], the authors demonstrate the efficacy of web caching under weak consistency by using timer-based protocols. Chankhunthod et al. [2] demonstrated that a caching system using the hierarchical architecture can be very effective to scale up the web growth. Yu et al. [10] showed a scalable invalidation approach based on hierarchy and application-level multicast routing. However, none of these prior efforts has investigated the TTL expiration-based weak consistency problem for a hierarchical caching system in a formal setting as we have done in this paper.

The most relevant work to ours is that by Cohen and Kaplan [3], where the authors focus on the effects of aging on the miss rate of the cache. Among other things, the authors in [3] considered a simple tree with a height of 2 and compared its miss rate with other configurations under different request arrival patterns. Motivated by the work in [3], this paper aims to have a deeper understanding of expiration-based hierarchical caching systems with some theoretical underpinning. By casting such hierarchical caching systems with a simple but generic model, we are able to obtain better understanding of the time domain behavior of weak consistency and a comprehensive set of performance metrics from both system's and user's perspectives.

5. Conclusions

This paper presents a fundamental study of hierarchical caching systems based on the weak consistency paradigm. Based on a simple model for hierarchical caching systems using the concept of TTL expiration mechanism, we analyze the intrinsic timing behavior of such systems and derive important performance metrics from both the system's and user's perspectives. Our simulation results further demonstrate the efficacy of our analysis and reveal insights on various design trade-offs. Our results are general and can be applied to distributed systems employing expiration-based weak consistency.

Acknowledgements

The authors wish to thank Xueyan Tang of Hong Kong University of Science and Technology for pointing out a simpler derivation for the average TTL.

References

[1] V. Cate, Alex—a global file system, in: Proceedings of the 1992 USENIX File System Workshop, Ann Arbor, MI, May 1992, pp. 1–12.
[2] A. Chankhunthod, P. Danzig, C. Neerdaels, M.F. Schwartz, K.J. Worrell, A hierarchical Internet object cache, in: Proceedings of the USENIX 1996 Technical Conference, San Diego, CA, January 1996, pp. 153–163.
[3] E. Cohen, H. Kaplan, Aging through cascaded caches: performance issues in the distribution of Web content, in: Proceedings of the ACM SIGCOMM Conference, San Diego, CA, August 2001, pp. 41–53.
[4] A. Dingle, Cache consistency in the HTTP 1.1 proposed standard, in: Proceedings of the First Workshop on Web Caching. http://www.w3cache.icm.edu.pl/workshop/program.html.
[5] J. Gwertzman, M. Seltzer, World-wide web cache consistency, in: Proceedings of the 1996 USENIX Technical Conference, San Diego, CA, January 1996, pp. 141–151.
[6] C. Liu, P. Cao, Maintaining strong cache consistency in the world-wide web, in: Proceedings of the 17th IEEE International Conference on Distributed Computing Systems (ICDCS'97), May 1997, pp. 12–21.

[7] The Network Simulator—ns-2. http://www.isi.edu/nsnam/ns/.
[8] M. Rabinovich, O. Spatscheck, Web Caching and Replication, Addison-Wesley, Reading, MA, 2002.
[9] S.M. Ross, Introduction to Probability Models, 4th ed., Academic Press, New York, 1989, Chapter 7.
[10] H. Yu, L. Breslau, S. Shenker, A scalable Web cache consistency architecture, in: Proceedings of the ACM SIGCOMM Conference, Cambridge, MA, August 31–September 3, 1999.

Y. Thomas Hou obtained his B.E. degree from the City College of New York in 1991, M.S. degree from Columbia University in 1993, and Ph.D. degree from Polytechnic University, Brooklyn, New York, in 1998, all in Electrical Engineering. From 1997 to 2002, Dr. Hou was a research scientist and project leader at Fujitsu Laboratories of America, IP Networking Research Department, Sunnyvale, CA (Silicon Valley). He is currently an Assistant Professor at Virginia Tech, The Bradley Department of Electrical and Computer Engineering, Blacksburg, VA. Dr. Hou's 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. Dr. Hou 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.

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, Ont., Canada. Since September 2001, he has been a Member of Research Staff with the IP Networking Research Department at Fujitsu Laboratories of America, Sunnyvale, CA. His research interests include transport protocols and application services for multimedia, high-speed and mobile networks. He is a member of ACM and IEEE.