Modeling and Analysis of An Expiration-Based Hierarchical Caching System

Y. Thomas Hou [*] Jianping Pan [†] Bo Li [‡] Xueyan Tang [‡] Shivendra Panwar [§]

[*] Virginia Tech, The Bradley Department of Electrical and Computer Engineering, Blacksburg, VA, USA
[†] University of Waterloo, Waterloo, Ontario, Canada
[‡] Hong Kong University of Science and Technology, Kowloon, Hong Kong
[§] Polytechnic University, Brooklyn, NY, USA

***Abstract*— Caching is an important means to scale up the growth of the Internet. Weak consistency is a major approach used in Web caching and has been deployed in various forms. This paper investigates some properties and performance issues of an expiration-based caching system. We focus on a hierarchical caching system based on the *Time-To-Live* (TTL) expiration mechanism and present a basic model for such a system. By analyzing the intrinsic TTL timing behavior in the basic model, we derive several important performance metrics from the perspective of the caching system and end users, respectively. Our results offer some basic understanding of a hierarchical caching system based on the weak consistency paradigm.**

I. Introduction

As Van Jacobson once put it several years ago, "With 25 years of Internet experience, we've learned exactly *one* way to deal with exponential growth: *caching*" [6]. This statement undoubtedly indicates the significance of caching as a primary means to scale up the Internet. There are significant advantages of deploying cache systems over the Internet [7]: (1) for end users, a nearby cache server can often satisfy requests faster than a faraway origin server; (2) for network providers, the cache servers can reduce the amount of traffic over the network; and (3) for service providers, the cache servers can distribute the load that the origin servers have to handle and increases reliability in offering Internet services.

An important issue in the design of a caching system is to maintain some level of *consistency* between cached copies of an object and the object maintained at the origin server. Every time the original object is updated at the origin server, copies of that object cached elsewhere becomes *stale*. 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 origin server, the origin server immediately notifies all cache servers about this update. 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 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 allowed to let a user get a somewhat stale object from the cache server. The cache server only validates the object's freshness with the origin server periodically and may lag behind the actual update at the origin server. Weak consistency is particularly useful for those web contents that can tolerate a certain degree of discrepancy between cached content and the content at origin server as long as it is understood that such discrepancy in time does not cause any harm. 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 origin server, it is still useful (or at least not harmful) to retrieve the cached copy at a cache or proxy server.[1] It has been shown that weak cache consistency is a viable and economic approach to deliver content that does not have a strict 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 life time 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, after which it is supposed to decrease with time. 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 origin server or some cache server that has 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 [7]).

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 light of this, in

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

this paper, we consider a hierarchical caching system. We conduct an investigation of its performance and behavior under the weak consistency paradigm, which employs the TTL-based expiration mechanism. Although the current HTTP protocols on Web caching provide a lot of similar features [4], we intend to conduct our investigation of weak consistency in a more general setting and will not limit ourselves to the details of the HTTP implementation.

We start with a basic model, which is a generic hierarchical caching system based on tree topology. Under the basic model, the root node represents the origin server whereas all the other nodes in the tree represent cache servers (see Fig. 1). We assume each node (or cache server) is deployed in a metropolitan region and the user requests[2] within this particular metro region always goes to this regional cache server for content service. When the object is not available or its TTL has expired at a cache server, the cache server will query its immediate parent cache server, which may further query its immediate parent cache server and so forth, until a "fresh" copy of the object is retrieved or the origin root server is reached. Here, a "fresh" copy is defined to be a copy of the object with an unexpired TTL (*i.e.*, positive TTL). The origin root server always maintains an updated copy of the object and will initialize the TTL of an object upon request. The TTL value for an object at any cache server (except the origin) decreases with time. Since TTL is a fundamental parameter that determines the intrinsic behavior of the overall hierarchical caching system, we analyze the behavior of TTL for a cache server at each level of the tree. Based on this analysis, we conduct a performance study for the hierarchical caching system from the perspectives of both the caching system and end users by deriving performance metrics such as hit rate, miss rate, response time, and network load.

The remainder of this paper is organized as follows. In Section II, we present the basic model for the hierarchical caching system based on weak consistency. In Section III, we analyze the TTL behavior of the basic model and derive its performance metrics. Section IV discusses related work and Section V concludes this paper.

II. The Basic Model

In a TTL-based caching system, each cached object is associated with a TTL value, which was first initialized by the cache server or origin server where the object was retrieved. The TTL value decreases with time and expires when it reaches 0. A cached object is considered *fresh* when its TTL is greater than zero; otherwise, it is considered *stale*.

We assume the hierarchical caching system follow a tree structure (see Fig. 1). Typically, any cache server (including the origin server) is deployed within a metro region. At level

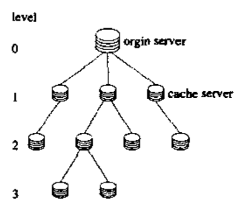

Fig. 1. An example of a hierarchical caching system based on tree topology.

0, we have one origin (root) server[3], S_0, which always maintain the latest (updated) copy of an object. The root server is logically connected to some child servers which we refer to as level 1 cache servers, each of which are geographically located in different metro regions. Level 1 cache server may also connect to some child servers which we call level 2 cache servers, and so forth. Finally, a cache server that does not have any child cache server is called a leaf cache server. The maximum number of levels of a hierarchical tree is also called the height of the tree. Figure 1 shows a simple example of our caching system based on a tree structure with height of 3.

We assume that the aggregate user requests to the cache server within a metro region follow a Poisson process.[4] When a user request arrives at the cache server, if the object already exists at the cache server and its TTL is still greater than 0, the cache server 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 cache server, if the object does not exist or the TTL timer has expired (*i.e.*, decreased to 0), we consider such an event a *user miss*. When a miss happens, the local cache server will generate a request and query its immediate parent cache server to see if it has the object with a valid TTL. If this parent cache server does have this object with an unexpired TTL, we call this event a *system hit* since the request is generated by a child cache server rather than *directly* from a user. Upon a system hit, the object will be delivered to the cache server and will subsequently be delivered to the user. Otherwise, we have a *system miss* and the parent cache server will generate a request and further query its own parent cache server and so forth, until the query process (in the worst case) 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 system is bounded by τ. Under the basic model, upon the event of a system hit, not only the user will be delivered a copy of the object with an updated TTL, *all* the cache servers involved in the query process will also get a copy of this object with an updated TTL.

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

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

[4]An exact traffic model for individual user is still an open research topic. However, it is reasonable to assume that, for a large population in a metro region, the aggregate requests follow a Poisson process.

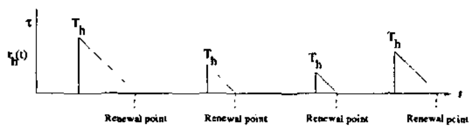

Fig. 2. A sample path of the TTL for a cache server at level h.

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 behavior, as we shall soon find out. To maintain such distinction, at a cache server, we need to distinguish user requests and system requests: user requests come from the metro region within the coverage of this local cache server while system requests come from its child (including grandchild and so forth) cache servers which are caused by users from a *remote* metro region.

III. 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 behavior of TTL, miss rate, hit rate, average response time, and traffic load at each cache server.[5] On the other hand, user's quality of experience refers to user's perceived quality in content delivery, *e.g.*, hit rate, miss rate, average response time, which only counts requests from end users and does not include auxiliary traffic within the hierarchical caching system.

A. Average TTL Behavior

We start our analysis with the most general form of a caching system. Suppose we are at a cache server of level h, $1 \leq h \leq H$, where H is the height of the tree. Denote the (remaining) TTL at the cache server as $r_h(t)$. Then $r_h(t)$ is a renewal process [8], 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 $r_{h^*}(t) = 0$, 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 cache server always has its remaining TTL larger than or equal to the remaining TTL of the same object maintained at its child cache servers.

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 a fundamental system parameter in our performance study for the basic model.

Due to the nature of the hierarchical tree and TTL-based expiration, there is an important property on TTL that *links* the

[5]By *aggregated*, we mean both external user requests as well as internal requests from child cache servers.

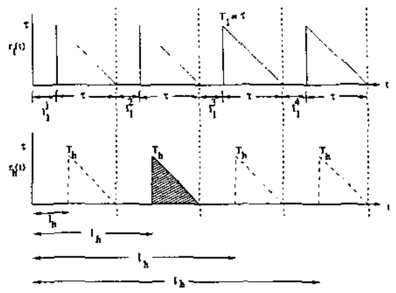

Fig. 3. A sample path of the TTL behavior of a cache server at level h in reference to the TTL behavior at level 1.

cache servers at all levels. In particular, any TTL renewal point at level h, $2 \leq h \leq H$ (see Fig. 2) coincides (or *synchronizes*) with some 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 cache server, the more child servers (and thus user population) it will support, 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 rate Λ_1, which is the *sum* of all Poisson arrival rates at all cache servers of the tree with $\mathcal{S}_1$ being its root. This Poisson process (with rate)Λ_1 can be considered as an *aggregate* of two Poisson processes: the first with a rate of Λ_h representing the arrivals at the sub-tree with $\mathcal{S}_h$ as the root and the second (with a rate of $\Lambda_1 - \Lambda_h$) representing arrivals from the rest of the tree within $\mathcal{S}_1$ excluding the sub-tree $\mathcal{S}_h$. Clearly, the probability that the TTL triangle is initiated by a request from the sub-tree with root$\mathcal{S}_h$ Λ_h/Λ_1 and the probability that the TTL is initiated by a request from the rest of tree (*i.e.*, $\mathcal{S}_1 \backslash \mathcal{S}_h$) 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 $\mathcal{S}_h$. In this case, $T_h = \tau$.

Case 2: With probability $\frac{\Lambda_1 - \Lambda_h}{\Lambda_1} \cdot \int_0^\tau \Lambda_h e^{-\Lambda_h t} dt = \frac{\Lambda_1 - \Lambda_h}{\Lambda_1} \cdot (1 - e^{-\Lambda_h \tau})$, there is a request arrival within τ from the sub-tree with root $\mathcal{S}_h$. The average T_h in this case is given by

$$\frac{\int_0^\tau (\tau - t)\Lambda_h e^{-\Lambda_h t} dt}{\int_0^\tau \Lambda_h e^{-\Lambda_h t} dt} = \frac{\tau - \frac{1}{\Lambda_h} + \frac{1}{\Lambda_h} e^{-\Lambda_h \tau}}{1 - e^{-\Lambda_h \tau}}. \quad (1)$$

It can be shown that the average T_h in this case (*i.e.*, the right 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 $\frac{\Lambda_1-\Lambda_i}{\Lambda_1} \cdot \int_\tau^\infty \Lambda_h e^{-\Lambda_h t} dt = \frac{\Lambda_1-\Lambda_h}{\Lambda_1} \cdot e^{-\Lambda_h \tau}$, 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, *i.e.*,

$$\begin{aligned} E(T_h) &= \frac{\frac{\Lambda_h}{\Lambda_1} \cdot \tau + \frac{\Lambda_1-\Lambda_h}{\Lambda_1} \cdot (1-e^{-\Lambda_h \tau}) \cdot \frac{\tau - \frac{1}{\Lambda_h} + \frac{1}{\Lambda_h} e^{-\Lambda_h \tau}}{1-e^{-\Lambda_h \tau}}}{\frac{\Lambda_h}{\Lambda_1} + \frac{\Lambda_1-\Lambda_h}{\Lambda_1} \cdot (1-e^{-\Lambda_h \tau})} \\ &= \frac{\Lambda_h \tau + (\Lambda_1 - \Lambda_h)(\tau - \frac{1}{\Lambda_h} + \frac{1}{\Lambda_h} e^{-\Lambda_h \tau})}{\Lambda_h + (\Lambda_1 - \Lambda_h)(1-e^{-\Lambda_h \tau})} . \end{aligned} \quad (2)$$

Property 1: Under the basic model, the average of T_h at level h, $h = 1, 2, \cdots, H$, has the following properties: (1) $\tau \geq E(T_h) > \tau/2$ and (2) $E(T_1) > E(T_2) > \cdots > E(T_h) > \cdots > E(T_H)$.
We omit the proof here due to paper length limit.

The average value of T_h, *i.e.*, $E(T_h)$ is a fundamental system parameter characterizing the behavior of the hierarchical caching system. In the following, we calculate performance parameters such as hit rate, miss rate, response time, from both the system and the users' perspectives. All of these performance parameters hinge upon the value of $E(T_h)$.

B. Performance Metrics

As mentioned earlier, we distinguish performance metrics along two dimensions: system 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 cache server of level h, respectively. The system miss rate, hit rate, and response time take into account of all requests, both from the users in the (local) metro region and from child cache servers (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 at a cache server 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 cache servers.

Before we calculate the miss rate Γ_h^s at level h, we make the following observation: each cache server can make at most one request to its parent cache server during any renewal cycle. Denote M_h the number of request a cache server at level h receives from its child cache servers during a renewal cycle and c_h the number of child cache servers of this server at level h. Then M_h is a random variable defined over $0, 1, \cdots, c_h$ and the probability distribution of M_h is a combinatorial of exponential distributions — due to the fact that the user requests at a cache server of any level follows 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 cache server's metro region or from a child cache server. We have,

$$\begin{aligned} \Gamma_h^s &= \frac{\lambda_h}{\Lambda_h} \left\{ \frac{1}{1 + [\lambda_h \cdot E(T_h) + E(M_h)]} \right\} \\ &\quad + \frac{\Lambda_h - \lambda_h}{\Lambda_h} \left\{ \frac{1}{1 + [\lambda_h \cdot E(T_h) + E(M_h) - 1]} \right\} . \end{aligned} \quad (3)$$

Note that for a constant λ_h and c_h at each level h, the miss rate increases as the level h increases.
The system's hit rate, $\Theta_h^s = 1 - \Gamma_h^s$, is then

$$\begin{aligned} \Theta_h^s &= \frac{\lambda_h}{\Lambda_h} \left\{ \frac{\lambda_h \cdot E(T_h) + E(M_h)}{1 + \lambda_h \cdot E(T_h) + E(M_h)} \right\} \\ &\quad + \frac{\Lambda_h - \lambda_h}{\Lambda_h} \left\{ \frac{\lambda_h \cdot E(T_h) + [E(M_h) - 1]}{1 + \lambda_h \cdot E(T_h) + [E(M_h) - 1]} \right\} . \end{aligned} \quad (4)$$

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

$$\sigma_h^s = \Theta_h^s \cdot 0 + \Gamma_h^s \cdot \pi_h , \quad (5)$$

where π_h is delay until getting a fresh object given that there is a miss at the local cache server. From (5), we have

$$\pi_h = \frac{\sigma_h^s}{\Gamma_h^s} , \quad (6)$$

On the other hand

$$\pi_h = d_h + \left\{ \frac{\Lambda_{h-1} - \Lambda_h}{\Lambda_{h-1}} \cdot 0 + \frac{\Lambda_h}{\Lambda_{h-1}} \cdot \pi_{h-1} \right\} . \quad (7)$$

Combining (5), (6) and (7), we have the following recursive relationship for σ_h^s.

$$\sigma_h^s = \Gamma_h^s \cdot (d_h + \frac{\Lambda_h}{\Lambda_{h-1}} \cdot \pi_{h-1}) = \Gamma_h^s \cdot (d_h + \frac{\Lambda_h}{\Lambda_{h-1} \Gamma_{h-1}^s} \cdot \sigma_{h-1}^s) , \quad (8)$$

with $\sigma_1^s = \Gamma_1^s \cdot d_1$.

We now calculate the user perceived hit rate (Θ_h^u), miss rate (Γ_h^u), and response time (σ_h^u), at a cache server 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 cache servers (which represent internal dynamics of the hierarchical caching system). Again, by conditioning on whether the first request comes from local users or child cache servers, we have,

$$\Theta_h^u = \frac{\lambda_h}{\Lambda_h} \left\{ \frac{\lambda_h \cdot E(T_h)}{1 + \lambda_h \cdot E(T_h)} \right\} + \frac{\Lambda_h - \lambda_h}{\Lambda_h} \cdot 1 . \quad (9)$$

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

$$\Gamma_h^u = \frac{\lambda_h}{\Lambda_h} \left\{ \frac{1}{1 + \lambda_h \cdot E(T_h)} \right\} . \quad (10)$$

The response time a user experiences is

$$\sigma_h^u = \Theta_h^u \cdot 0 + \Gamma_h^u \cdot \pi_h = \Gamma_h^u \cdot (d_h + \frac{\Lambda_h}{\Lambda_{h-1}\Gamma_{h-1}^s} \cdot \sigma_{h-1}^s) \, , \quad (11)$$

with $\sigma_1^u = \Gamma_1^u \cdot d_1$.

C. Network Traffic Load

So far we have calculated the hit rate, miss rate, and response time from both the system's and user's perspectives. There is one more important system performance metric that we want to include. This is the traffic load associated with the hierarchical caching system. As we discussed earlier, 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 continues to grow. Here, we calculate the network load associated with the hierarchical caching system. Based on this result, it can be shown that the hierarchical caching system has superior scalability property comparing to a non-hierarchical (or "flat") caching system.

One way to measure network traffic load for the hierarchical caching system is to perform an accounting on how much traffic each cache server generates to its immediate parent cache server. Note that a cache server will initiate a request to its parent cache server only when a request (either from local users in the metro region or from child cache servers) incurs a miss. Denote the average request rate that a cache server S_h at level h sends to its parent cache server as ρ_h^s. By definition, we have

$$\rho_h^s = \frac{1}{E(I_h + T_h)} \, , \quad (12)$$

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

IV. Related Work

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] demonstrate that a caching system using the hierarchical architecture can be very effective to scale up the Web growth. In [9], Yu *et al.* show a scalable invalidation approach based on hierarchy and application-level multicast routing. However, none of these prior efforts investigate 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 *age* on the miss rate of the cache. Among other things, the authors in [3] consider a simple tree with a height of 2 and compare 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 a expiration-based hierarchical caching system with formal theoretical underpinning. By casting such hierarchical caching system 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'sand user's perspectives.

V. Conclusions

Caching is an important means to scale up the Internet growth and weak consistency is amajor approach used in Web caching. This paper presents a fundamental study of a hierarchical caching system based on the weak consistency paradigm, which builds upon the concept of TTL expiration. Although the current HTTP implementations of Web caching provide a lot of similar features, our investigation in the weak consistency paradigm is carried out in a general setting and is not limited to any details of the current HTTP implementations. The main contribution of this paper is to offer some fundamental understanding on weak consistency based hierarchical caching systems. Based on a basic model for a hierarchical caching system using the concept of TTL expiration mechanism to achieve weak consistency, we analyze the intrinsic timing behavior of such a system and derive important performance metrics from both the system's and user's perspectives. Our results are general and can be applied to other cache systems employing expiration-based consistency.

We have done extensive simulation work to further demonstrate the efficacy of our analysis and reveal insights on various trade-off between performance and cost. Due to the page limit, we will leave these simulation results in a future paper.

References

[1] V. Cate, "Alex—A global file system," in *Proc. 1992 USENIX File System Workshop,* pp. 1–12, Ann Arbor, MI, May 1992.

[2] A. Chankhunthod, P. Danzig, C. Neerdaels, M.F. Schwartz, and K.J. Worrell, "A hierarchical Internet object cache," in *Proc. USENIX 1996 Technical Conference,* pp. 153–163, San Diego, CA, Jan. 1996.

[3] E. Cohen and H. Kaplan, "Aging through cascaded caches: performance issues in the distribution of Web content," in *Proc. ACM SIGCOMM Conference,* pp. 41–53, San Diego, CA, August 2001.

[4] A. Dingle, "Cache consistency in the HTTP 1.1 proposed standard," in *Proc. ICM Workshop on Web Caching,* Warsaw, Poland, Sept. 1996. Available at `http://w3cache.icm.edu.pl/workshop/program.html`.

[5] J. Gwertzman and M. Seltzer, "World-Wide Web cache consistency," in *Proc. 1996 USENIX Technical Conference,* pp. 141–151, San Diego, CA, Jan. 1996.

[6] V. Jacobson, "How to kill the Internet," A presentation at *ACM SIGCOMM'95 Middleware Workshop,* Cambridge, MA, Aug. 28, 1995. Available at `http://www.root.org/ip-development/`.

[7] M. Rabinovich and O. Spatscheck, *Web Caching and Replication,* Addison-Wesley, 2002.

[8] S.M. Ross, *Introduction to Probability Models, Fourth Edition,* (Chapter 7), Academic Press, Inc., 1989.

[9] H. Yu, L. Breslau, and S. Shenker, "A scalable Web cache consistency architecture," in *Proc. ACM SIGCOMM Conference,* Cambridge, MA, Aug. 31–Sept. 3, 1999.