On Prefetching in Hierarchical Caching Systems

Y. Thomas Hou[†] **Jianping Pan**[‡] **Chonggang Wang**[§] **Bo Li**[§]

[†]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

Abstract— **Hierarchical caching is deployed to scale up the explosive Web growth, and the expiration-based mechanism is adopted as an economic means to support the weak consistency in this context. However, given a hierarchy, the user perceived performance heavily depends on its position. Normally, a user near the hierarchy leaf suffers higher miss rate and longer response time. Such an intrinsic property can discourage users from participating in any hierarchical caching systems. In this paper, we analyze the performance of a proposed approach, *i.e.*, freshness and retrieval threshold based cache prefetching, to mitigate the bias against leaf users. We also use *ns–2* to further substantiate our analysis. By adopting this approach with the appropriate parameters, the fairness among users within a caching hierarchy can be considerably improved.**

I. Introduction

Most Internet applications, including World Wide Web, are built upon the well-known *client-server* model. This service paradigm is challenged by the explosive Web growth in recent years. Popular web sites, *e.g.*, news portal `cnn.com` and hot event `saltlake2002.com`, have to handle a large number of simultaneous accesses. This demand surge can easily overload any single web server and its access links. Another example is the Distributed Denial of Service (DDoS) where a group of hostile clients purposely overload a victim server and cause the service interruption to regular users. All these reveal the scalability weakness embedded in the traditional client-server model.

Web caching [1] now is widely deployed to cope with the Web growth. Caching can be with or near clients (*e.g.*, *browser* or *proxy* cache), or close to servers (*reverse* cache), and it mitigates the scalability issue by aggregating user requests so that the amount and rate of requests sent to the origin server are reduced. It is found [8], [9] that a hierarchical caching system, *i.e.*, multiple cooperative cache servers at different levels, has the greatest ability to aggregate user requests. Therefore, in this paper, we will focus on a caching hierarchy.

A related issue with caching is the object validity. Many cacheable objects are subject to changes during their lifetime. Once an object is modified, all copies cached elsewhere become *stale* unless these copies are updated accordingly. There are two validation paradigms. The first is *strong consistency*, *i.e.*, validity is guaranteed for cached objects upon access. It is used for certain critical applications without any tolerance on discrepancy [2], [15]. Although strong consistency is indispensable for some services, its high overhead and complexity prevent it from being widely deployed in many contexts that have a scale as the Web. Fortunately, many web-based services (*e.g.*, web pages) fall into the second paradigm — *weak consistency*, where a certain degree of discrepancy is allowed. In this paradigm, object is validated periodically and user may obtain a *stale* object [13]. However, as long as the discrepancy is under a given limit, users can still extract useful (or at least non-harmful) information. Since weak consistency is acceptable for many web services and it is very cost effective, it can be rather easily developed and deployed.

J. Pan is currently with Fujitsu Labs. of America, Sunnyvale, CA, USA.

There are many approaches to support the weak consistency, and we choose the Time-To-Live (TTL) mechanism in this paper since it has been widely used [14]. When the object is retrieved from the origin server, TTL is initialized to a value which reflects the maximum tolerance of discrepancy and decreases with time if the object is cached elsewhere. When the cached object is accessed, it is considered valid only if the remaining TTL is still positive; otherwise, the object is validated by (and if necessary, retrieved from) the origin server or an upstream server in the caching hierarchy.

This paper analyzes some intrinsic properties in a hierarchical caching system based on the TTL expiration mechanism. Although these systems and the like have been deployed for some time, the performance and their built-in strength and weakness are mainly demonstrated through the empirical measurement [9]. Based on an analytical model developed in [17], in Section II, we first uncover an intrinsic bias against leaf users in any hierarchy, in terms of higher miss rate and longer response time. The external performance is then explained by the internal behaviors, *i.e.*, smaller average TTL and more intermediate servers for leaf users. In Section III, we consider the effect of cache prefetching to increase the TTL by imposing the freshness and retrieval threshold. Related work are summarized in Section IV. Section V concludes this paper with some pointers on future work.

II. System Model

Figure 1(a) outlines a tree-like hierarchical caching system under the consideration. When a cache server (CS) receives a request, if the object is cached locally with a positive remaining TTL, the object is returned immediately with the remaining TTL; otherwise, the CS generates *another* request to its immediate upstream CS and so forth (until the origin server (OS)) to get a valid copy. This process is recursive until a valid object is obtained. If we take the longest path and convert requests from all nodes other than those in this

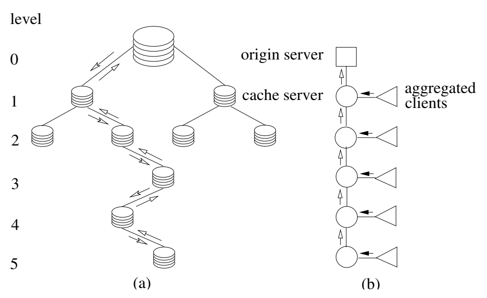

Fig. 1. Hierarchical caching system and its abstract model.

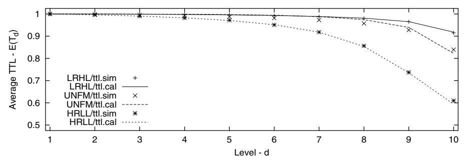

Fig. 2. Average TTL at each level.

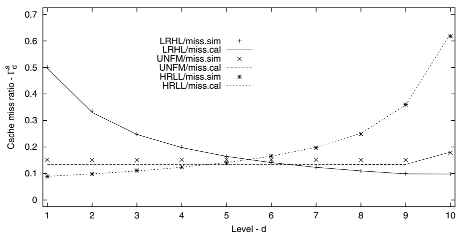

Fig. 3. Cache miss ratio at each level.

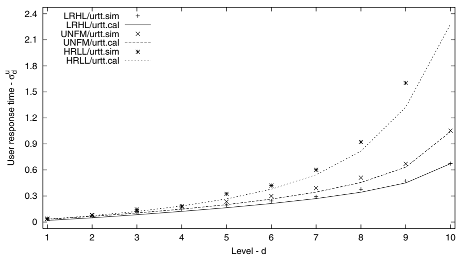

Fig. 4. User response time at each level.

path as the equivalent Aggregated Client (AC), we have the abstract model shown in Fig. 1(b). This branching process is also recursive, and the model developed for the whole tree can be applied to each branch individually. Therefore, in the following, we only consider the chain-like abstract model.

For CS_d at level d ($1 \leq d \leq D$ and D is the chain depth), the aggregated request rate from AC_d is denoted as λ_d. We assume the inter-request time follows an exponential distribution. Although the object access pattern from an individual user is very complex [10], here we only consider the *aggregated* request from a large user population. Therefore, in this system, there are two sets of performance metrics, one for AC_d and the other for CS_d. For AC_d, we primarily focus on the user perceived response time σ^u_d; and for CS_d, the cache miss rate Γ^s_d since it is proportional to the request traffic generated from that CS. These two metrics are in turn based on the remaining TTL, $\tau_d(t)$, at each CS_d.

A. Average TTL behavior

In [17], we obtained a model on the average TTL $\mathtt{E}(T_d)$ which is the expected TTL when an object is retrieved from CS_{d-1}. With $\Lambda_d = \sum_{i=d}^{D} \lambda_i$ and the initial TTL τ,

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

In Fig. 2, we plot $\mathtt{E}(T_d)$ from the *ns-2* simulation results (with points) and the analytical calculation (with lines) for three representative request patterns:

1) Light Root Heavy Leaf (LRHL), $\lambda_d = d$ and $D = 10$;
2) Uniform (UNFM), $\lambda_{d1} = \lambda_{d2} = \frac{D+1}{2}$ for any d_1 and d_2;
3) Heavy Root Light Leaf (HRLL), $\lambda_d = D - d + 1$.

The simulation first confirms our analytical results. It also reveals a property embedded in any caching hierarchy: $\mathtt{E}(T_{d_1}) \geq \mathtt{E}(T_{d_2})$ if $d_1 \leq d_2$, *i.e.*, the further away a CS from OS, the smaller average TTL it has. Next, we observe that although the *total* user request load is the same for all three patterns, due to the different request distributions, the TTL can behave differently. When comparing LRHL to HRLL, it shows another property in a caching hierarchy: the higher the request rate, the larger the average TTL a CS can expect.

B. Cache miss rate

There are two definitions on cache miss ratio/rate. First it can be defined as the ratio of number of misses over number of total requests and denoted as Γ^s_d. Second, it can also be defined as the number of misses per unit time and denoted as γ^s_d. For γ^s_d, there is 1 miss per level d renewal period of $\mathtt{I}_d + \mathtt{E}(T_d)$ where $\mathtt{I}_d = \frac{1}{\Lambda_d}$. Therefore, $\gamma^s_d = \frac{1}{\frac{1}{\Lambda_d} + \mathtt{E}(T_d)}$. For Γ^s_d, we consider the following two cases:

1) With probability $p^1_\Gamma = \frac{\lambda_d}{\Lambda_d}$, an object is retrieved from CS_{d-1} due to a local user request at CS_d. For the remaining $\mathtt{E}(T_d)$, there are $(\lambda_d + \gamma^s_{d+1})\mathtt{E}(T_d)$ *hit* requests generated locally or from a downstream CS.
2) Otherwise, the object is retrieved due to a request from a downstream CS. For the remaining $\mathtt{E}(T_d)$, there are $\lambda_d \cdot \mathtt{E}(T_d)$ *hit* requests only generated locally.

Therefore, $\Gamma^s_d = p^1_\Gamma \frac{1}{(\lambda_d + \gamma^s_{d+1})\mathtt{E}(T_d)} + p^2_\Gamma \frac{1}{\lambda_d \cdot \mathtt{E}(T_d)}$.

We can also derive the user miss rate $\Gamma^u_d = p^1_\Gamma \frac{1}{1 + \lambda_d \cdot \mathtt{E}(T_d)}$ by omitting the downstream requests accordingly.

Figure 3 plots the cache miss ratio Γ^s_d as a function of the CS position (level d) and request patterns. For the UNFM request pattern, it clearly indicates the bias against the leaf CS and its users in term of higher miss rate. HRLL pattern further magnifies this bias since it has less request near the leaf. The figure also suggests that when forming a hierarchical structure, it is important to ensure the leaf CS has sufficient request aggregation (or LRHL-like), otherwise, the leaf users suffer much higher miss rate in HRLL.

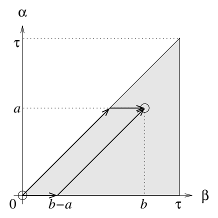

Fig. 5. Relation between retrieval and freshness thresholds.

C. User response time

To calculate the user response time σ_d^u, we first assume the sum of network delay (d_d^n) and server overhead (d_d^s) between CS_d and CS_{d-1} is d_d. Therefore the user response time at level d is $\sigma_d^u = \Gamma_d^u \cdot (d_d + \frac{\Lambda_d}{\Lambda_{d-1}} \cdot \frac{\sigma_{d-1}^s}{\Gamma_{d-1}^s})$, where σ_{d-1}^s is the cache response time at level $d-1$ and $\sigma_d^s = \Gamma_d^s \cdot (d_d + \frac{\Lambda_d}{\Lambda_{d-1}} \cdot \frac{\sigma^s d-1}{\Gamma_{d-1}^s})$. When $d = 1$, $\sigma_1^u = \Gamma_1^u \cdot d_1$ and $\sigma_1^s = \Gamma_1^s \cdot d_1$.

In Fig. 4, there is another bias against the leaf users in term of longer response time. It shows σ_d^u is more likely determined by user's position, since even `LRHL` has the similar bias against the leaf users as the `UNFM` pattern. For `HRLL`, the farther away from OS, the quicker increase in σ_d^u. Even higher request rate can reduce Γ_d^s at leaf, it is still unable to fully compensate σ_d^u due to the intrinsic structure disadvantage in any hierarchical caching systems.

By now, we have identified two bias issues in any hierarchical caching systems. These findings are supported by the analytical calculation and simulation results. We find that the bias is due to a low TTL at the leaf level, and the low TTL in turn is due to less request aggregation and longer distance away from OS for the leaf CS. In the next section, we will explore an approach to mitigate the bias.

III. Cache Prefetching

We first propose two cache prefetching parameters: *retrieval* and *freshness thresholds* for CS and AC, respectively. We then give the analytical and simulation results for two special cases which actually are building blocks for any cases.

A. Prefetching Parameters

We introduce a new parameter β_d, *retrieval threshold*, for CS_d, and we adopt a *reactive* approach to utilize it, *i.e.*, the internal request is only generated when the object is accessed and the remaining TTL is below β. Additionally, AC_d can have the *freshness threshold* α_d, which reflects the minimum remaining TTL that a user can accept. It is obvious that $0 \leq \alpha_d \leq \beta_d \leq \tau$ as the shaded area in Fig. 5. When $\beta_d = \tau$, it prohibits the object caching. If the remaining TTL is in between α_d and β_d upon receiving a request, we adopt a *conservative* approach, *i.e.*, CS returns the object immediately, meanwhile, generates another request to the upstream CS.

For a general pair of (α, β), Fig. 5 also shows a working path from the basic model where $\alpha = \beta = 0$. We can first increase β to $b - a$ while keeping $\alpha = 0$, then increase α and β simultaneously to a and b. Alternatively, 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, we can just focus on the cases $0 \leq \alpha = \beta \leq \tau$ and $\alpha = 0 \leq \beta \leq \tau$.

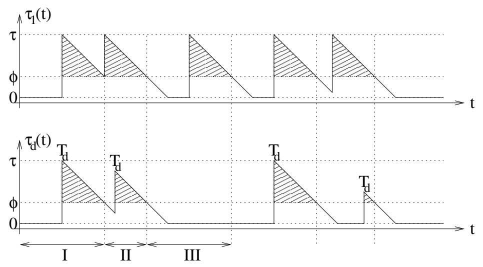

Fig. 6. TTL behavior with freshness threshold.

B. Freshness threshold

We first consider the situation when a user imposes the freshness requirement $\alpha_d > 0$. Assuming users have similar freshness requirement on an object no matter where they are, without loss of generality, we have $\alpha_d = \alpha$ for $1 \leq d \leq D$.

1) Average TTL behavior: If we choose $\alpha = \beta = \phi$ and observe the TTL behavior at levels 1 and d, we have the scenario in Fig. 6. We can redefine the level 1 renewal period between two consecutive level 1 renewal points when $\tau_1(t_-^*) > \phi$ and $\tau_1(t^*) = \phi$. Again, if we condition the TTL behavior in level d during a given level 1 renewal period, we have three scenarios. Actually, if we shift the t axis by ϕ, we can apply the same technique that we used in [17].

I) With probability $p_{\mathtt{ft}}^{\mathrm{I}} = \frac{\Lambda_d}{\Lambda_1}$, the level 1 TTL *triangle* is triggered by the request initially generated *within* the subtree rooted at CS_d, and $t_{\mathtt{ft}}^{\mathrm{I}} = (\tau - \phi) + \phi$;

II) With probability $p_{\mathtt{ft}}^{\mathrm{II}} = \frac{\Lambda_1 - \Lambda_d}{\Lambda_1} \cdot \int_0^{\tau-\phi} \Lambda_d \cdot e^{-\Lambda_d \cdot t} dt$, the level 1 TTL *triangle* is triggered by the request initially generated *outside* the subtree rooted at CS_d, but there is a request initially generated within the subtree rooted at CS_d in the same level 1 renewal period, so that $t_{\mathtt{ft}}^{\mathrm{II}} = \frac{\int_0^{\tau-\phi} (\tau-\phi)\Lambda_d \cdot e^{-\Lambda_d \cdot t} dt}{\int_0^{\tau-\phi} \Lambda_d \cdot e^{-\Lambda_d \cdot t} dt}$; and

III) There is no request initially generated within the subtree rooted at CS_d in this renewal period.

Therefore, the average TTL behavior at level d is $\mathtt{E}_{\mathtt{ft}}(T_d) = \frac{p_{\mathtt{ft}}^{\mathrm{I}} \cdot t_{\mathtt{ft}}^{\mathrm{I}} + p_{\mathtt{ft}}^{\mathrm{II}} \cdot t_{\mathtt{ft}}^{\mathrm{II}}}{p_{\mathtt{ft}}^{\mathrm{I}} + p_{\mathtt{ft}}^{\mathrm{II}}}$. After the algebraic manipulation, we have

$$\mathtt{E}_{\mathtt{ft}}(T_d) = \phi + \mathtt{E}_{\tau:=\tau-\phi}(T_d).$$

That is, by replacing τ in Eq. (1) with $\tau - \phi$, or reducing the *effective* TTL of an object at the origin server to $\tau - \phi$, we can get the *effective* average TTL, *i.e.*, $\mathtt{E}_{\mathtt{ft}}^*(T_d) = \mathtt{E}_{\mathtt{ft}}(T_d) - \phi$, at each level. By using the same technique, it is relatively straightforward to obtain $\Gamma_{\mathtt{ft},d}^s$, $\sigma_{\mathtt{ft},d}^u$, and other metrics.

Figure 7 further substantiates our analysis. We present results in `HRLL` as it is the worst scenario in Fig. 2. Here, $\alpha_d = \beta_d = \phi_d \in \{0, 0.1\tau, 0.3\tau, 0.7\tau, \tau\}$ for all $1 \leq d \leq D$, *i.e.*, ϕ is exponentially interleaved with upper and lower bounds. By increasing α, or AC's asking for fresher objects, it is equivalent to increase the request aggregation (in Fig. 2) and push the average TTL curve upwards to the bound.

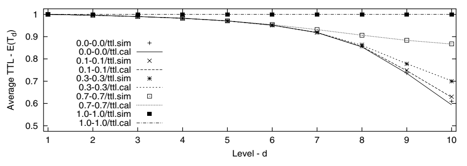

Fig. 7. Average TTL at each level with freshness threshold.

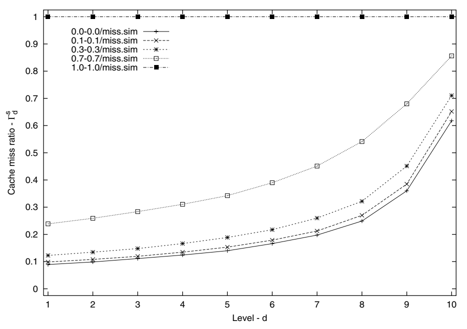

Fig. 8. Cache miss ratio at each level with freshness threshold.

2) Cache miss rate: When $\alpha = \tau$, it is the TTL upper bound where $\mathtt{E}_{\mathtt{ft},\phi=\tau}(T_d) = \tau$. It looks *similar* to a flat caching system where each CS sends request directly to OS. But actually it is even worse since every upstream CS is contacted and deemed a *miss*, as Fig. 8 shows. This reveals a system tradeoff on the object freshness and network traffic: the higher freshness to support, the higher cache miss rate to expect and more network traffic to generate. When α is already large (*e.g.*, $\alpha = 0.7\tau$), further increase in α only has marginal effect on $\mathtt{E}_{\mathtt{ft}}(T_d)$ since it already approaches its upper bound, but a small α increase can bring a rapid increase in $\Gamma^s_{\mathtt{ft},d}$.

3) User response time: Another tradeoff from user's perspective is revealed in Fig. 9. When $\alpha = 0.1\tau$, there is only a slight increase in $\sigma^u_{\mathtt{ft},D}$ due to the less request aggregation. But for AC's at other levels, $\sigma^u_{\mathtt{ft},d}$ $(1 \leq d < D)$ are actually reduced due to the cache prefetching triggered by lower levels. However, when α is large (*e.g.*, 0.7τ in Fig. 9), $\sigma^u_{\mathtt{ft},d}$ $(1 \leq d \leq D)$ increases quickly as level d grows. Especially when $\alpha = \tau$, the user response time increases linearly independent of any request aggregation since there is no *hit* except at OS. Taking both the object freshness and response time into account, clearly α should be chosen relatively small to utilize the benefit in a hierarchical caching system.

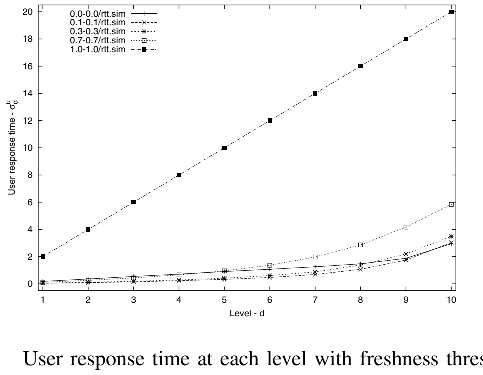

Fig. 9. User response time at each level with freshness threshold.

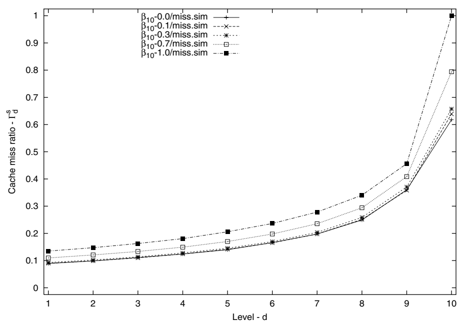

Fig. 10. Cache miss ratio at each level with retrieval threshold.

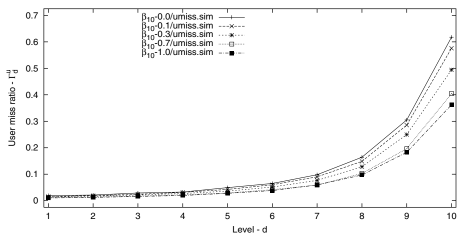

Fig. 11. User miss ratio at each level with retrieval threshold.

C. Retrieval threshold

Now we consider the situation when the CS has a retrieval threshold $\beta > 0$ for the internal request it generates.

1) Average TTL behavior: If we only consider the average TTL, $\mathtt{E}_{\mathtt{rt}}(T_d)$, at CS_d with β_d alone, when a request with $\alpha_d = 0$ initially generated from AC_d arrives, it is identical to the case when $\alpha = \beta$. Eventually CS_d has to get a copy of the requested object with the remaining TTL larger than β_d, although the user request can be fulfilled earlier with a copy with less freshness. Therefore, we omit plotting $\mathtt{E}_{\mathtt{rt}}(T_d)$ here since $\mathtt{E}_{\mathtt{rt}}(T_d) = \mathtt{E}_{\mathtt{ft}}(T_d)$ when $\beta_d = \phi$.

2) Cache miss rate: As different CS's can choose different β_d according to its position (d) in the hierarchy, without loss of generality, in Fig. 10, $\beta_D \in \{0, 0.1\tau, 0.3\tau, 0.7\tau, \tau\}$ (*i.e.*, only the leaf CS_D adopts cache prefetching, and all other parameters are the same as in Fig. 3). $\beta_D = 0$ is equivalent to the basic model without cache prefetching. β_D is also exponentially interleaved. When $\beta_D < 0.3\tau$, it only increases the cache miss ratio near leaf slightly when comparing to $\beta_D = 0$ since we do count prefetching traffic as a cache *miss*. Therefore, we also plot the user miss ratio in Fig. 11. When $\beta_D > 0 = \alpha_D$, since an object is more likely available upon request at CS_D with a valid remaining TTL due to a previous prefetching, the user miss ratio $\Gamma^u_{\mathtt{rt},D}$ is indeed reduced. It is worth mentioning that a small β_D can have considerable improvement in $\Gamma^u_{\mathtt{rt},D}$, while the overhead in $\Gamma^s_{\mathtt{rt},d}$ is still negligible. Furthermore, the larger β_D, the lower $\Gamma^u_{\mathtt{rt},D}$ and higher $\Gamma^s_{\mathtt{rt},d}$, since the cache system works more diligently.

3) User response time: Figure 12 shows $\sigma^u_{\mathtt{rt},d}$ is reduced accordingly with β_D, since $\Gamma^u_{\mathtt{rt},d}$ illustrated in Section III-C.2

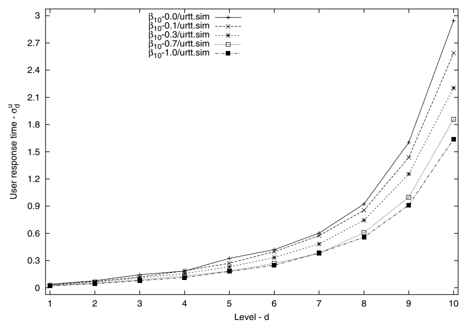

Fig. 12. User response time at each level with retrieval threshold.

is reduced due to the cache prefetching.

However, the tradeoff between the shorter $\sigma^u_{\mathtt{rt},d}$ (also lower $\Gamma^u_{\mathtt{rt}d}$) and more inter-CS traffic (also higher $\Gamma^s_{\mathtt{rt},d}$) needs to be rebalanced as β_d grows. As Figs. 10 and 12 indicate, when $\beta_D > 0.7\tau$, a further increase in β_D only has minor improvement in terms of $\Gamma^u_{\mathtt{rt},d}$ and $\sigma^u_{\mathtt{rt},d}$, but it can impose a rapid increase in $\Gamma^s_{\mathtt{rt},d}$ and inter-CS traffic for the level with large β_D. In the case of $\beta_D = \tau$, since such requests can only be fulfilled at OS, traffic surges at level D as many prefetching requests generated. But only few prefetches can bring improvement in user perceived performance. This implies that there is no need for CS to work too hard, or $\beta_d - \alpha_d$ should be less than a small positive threshold.

IV. Related Work

Web caching has attracted many research efforts in recent years [16]. But most work in the literature focus 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 [4]. There are several related work on cache consistency and hierarchical caching [8], [9], [6], [13], but they are mainly in the context of protocol design, empirical measurement, and experimental simulation. In this paper and [17], we take an analytical approach to better understand the intrinsic property of the average TTL behavior in a hierarchical caching system with the TTL based weak consistency.

Cohen *et. al.* [7] also considered a two layer hierarchical system with the TTL expiration scheme, but they more focused on the miss rate and user request patterns. Their work motivates us to further explore the TTL behaviors at different levels, especially in a hierarchical system. In our work, we also derive cache miss rate and user response time based on the obtained TTL model. Furthermore, we evaluate the cache prefetching approach to improve the fairness among users at different locations of the hierarchy. Prefetching has been proposed in the literature (see [11], [12]). Different from these work, we choose to illustrate the tradeoffs between network and user performance metrics analytically, and we demonstrate that an appropriate balance among these two metrics is achievable for a given user request pattern.

V. Conclusions

In this paper, based on an analytical model for hierarchical caching systems with the TTL expiration mechanism, we identified the bias against leaf users in terms of higher cache miss rate and longer response time, due to the fact of less request aggregation and more intermediate servers for leaf users. By introducing the cache prefetching based on the freshness and retrieval thresholds, we increased the request rate with auxiliary inter-cache traffic. This approach can improve the fairness among users, at the cost of additional network traffic. Using simulation results, we show that the tradeoff is appropriate under a suitable set of parameters.

The combination of the cache prefetching approach and other techniques, *e.g.*, request randomization, certainly deserves further investigation. For example, when TTL is below a certain threshold, next request can be directed further away. Also the threshold approach can be made TTL-aware and user-friendly in order to achieve further adaptability and efficiency in a hierarchical caching system.

References

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

[2] C. Liu and P. Cao. Maintaining strong cache consistency for the World-wide Web. *IEEE Trans. on Computers*, 47(4):445–457, 1998.

[3] The Network Simulator – ns-2. `http://www.isi.edu/nsnam/ns`.

[4] M. Arlitt and C. Williamson. Internet Web servers: Workload characterization and implications. *IEEE/ACM Transactions on Networking*, 5(5):631–644, 1997.

[5] A. Dingle and T. Partl. Web cache coherence. *Computer Networks and ISDN Systems*, 28(7-11):907–920, 1996.

[6] A. Rousskov and V. Soloviev. A performance study of the Squid proxy on HTTP/1.0. *World-wide Web Journal*, 2(1-2):47–67, 1999.

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

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

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

[10] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. *Proc. IEEE INFOCOM'99*, 1999.

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

[12] E. Cohen and H. Kaplan. Prefetching the means for document transfer: A new approach for reducing Web latency. *Proc. IEEE INFOCOM'2000*, 2000.

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

[14] A. Dingle. Cache consistency in the HTTP 1.1 draft. *First Workshop on Web Caching*, 1996.

[15] H. Yu, L. Breslau, S. Shenker. A scalable web cache consistency architecture. *Proc. ACM SIGCOMM'99*, 1999.

[16] J. Wang. A survey of web caching schemes for the Internet. *ACM Computer Communications Review*, 25(9):36–46, 1999.

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