An Update-Risk Based Approach to TTL Estimation in Web Caching*

Jeong-Joon Lee†, Kyu-Young Whang†, Byung Suk Lee‡, Ji-Woong Chang†
†Department of Computer Science and
Advanced Information Technology Research Center (AITrc)
Korea Advanced Institute of Science and Technology (KAIST)
373-1, Kusong-Dong, Yusong-Gu, Taejon, Korea
e-mail: {jjlee,kywhang,duribaba}@mozart.kaist.ac.kr

‡Department of Computer Science
University of Vermont
Burlington, Vermont, U.S.A.
email: bslee@cs.uvm.edu

Abstract

Web caching is an important technique for accelerating web applications and reducing the load on the web server and the network through local cache accesses. As in the traditional data caching, web caching poses the well-recognized problem of maintaining cache consistency. Web caching, however, has the leeway of delaying the refreshment of caches when the web server updates the original data, i.e., web caching tries to get better performance allowing tolerable inconsistency. This weak consistency requirement introduced the concept of time-to-live (TTL: the time during which the cached data item is expected to be valid) in the face of future updates. Subsequently, a number of methods have been invented to have the cache server estimate the TTL. However, the two well-known TTL estimation methods—the fixed TTL method and the heuristic method—do not allow intuitive understanding of the estimation processes and lack theoretical reasoning behind them, disallowing administrators from configuring the cache server by their intention. To mend these deficiencies, we propose the update-risk based TTL estimation method. *This method uses a formal, yet intuitive, approach based on probabilistic analysis. In the proposed method, users provide the update risk as the probability that the original data will be updated within the estimated TTL. Then, based on our model, the cache server calculates the value of the TTL using the update risk. The results of our experiments, performed using the logs of a real cache server, show experimentally that the measured update risk closely matches the one used to estimate the TTL. Moreover, the notion of the update risk is clear in its intention and semantics. These confirm the superiority of our method to the conventional ones. We also show the impact of update risk on performance and consistency in order to help administrators select an appropriate value for update risk to obtain performance and consistency desired. In addition, we reilluminate the two aforementioned conventional methods in light of our method.*

*This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc).

1 Introduction

As the web grows into an infrastructure for disseminating information, so does the volume of data exchanged on the web. This explosive growth of the data volume is overloading the web servers and the communication network, causing performance degradation. Web caching offers a solution to this problem by retaining frequently used web pages on the client side [1, 2, 18]. Recent research outputs in this area have been incorporated into such commercial products as Inktomi [10], Imimic [11], CacheFlow [13], InfoLibria [14], and ICS [12].

Like any cached data, cached web pages are copies of the original data (from the web server), and therefore, need to be synchronized with the original data. This consistency requirement is rather weak in the case of web caching as discussed in the references [7] and [15]. That is, it allows the synchronization to be delayed, and thus, allows the clients to access outdated data for some time [7, 8].

Synchronization delay affects consistency and performance of cache. Larger synchronization delay results in lower consistency, but better performance of cache. In other words, cache consistency and performance have a trade-off relationship. Therefore, the cache server administrator should find a reasonable compromising point between consistency and performance of cache.

The TTL methods are widely used for consistency maintenance in the web caching [8]. In these methods, cached data are assumed valid (i.e., up-to-date) for a certain duration, called the *time–to–live (TTL)*, from the point[1] of caching [8]. The TTL is determined using the values of fields such as *Expire* and *Max-age* that accompany a web page retrieved from a web server. The fields, however, are frequently empty [1], and therefore, a number of techniques have been proposed to estimate the TTL from other sources in such a case [4, 6, 8]. Two well-known techniques are the *fixed TTL method* [9] and the *heuristic method* [4, 8]. The former always assigns the same TTL to every data item, whereas the latter determines the TTL as a portion of the interval between the point of caching and the last point in time the original data was modified (as found in the *Last-Modified* field).

The two conventional methods have two major problems. First, they are hardly justifiable because they are not founded upon a formal theory. Second, they hardly attach any intuitive meaning to the TTL they estimate. For instance, the fixed TTL method disregards the discrepancy in the update frequencies of individual web pages, and therefore, fails to reflect the idiosyncratic update patterns of the web server. The heuristic method uses an *arbitrary* portion of the time since the last update and consequently fails to convey any sensible meaning to users. In this paper, we solve these problems by employing a formal method that determines the TTL based on a sound reasoning while conveying an intuitive meaning about the estimated value.

In our formal method, we propose the notion of the *update risk* associated with the TTL. For a given cached data item, its update risk is defined as the probability that the data item will become outdated before the TTL expires. On the flip side, (1 − update risk) reflects the *credibility* that the cached data item will remain up-to-date until the TTL expires. This notion of the update risk (or, equivalently, the credibility) is intuitive to users. In our work, we model the number of update occurrences as a Poisson process, which is known to be an effective probablistic model for the number of updates on the web data [5]. Then, we develop a probabilistic model for determining the TTL given an update risk.

We also demonstrate the viability of our method by showing that the update risk measured from a real cache server log using the estimated TTL closely matches the update risk given by the user to estimate the TTL. In addition, we reilluminate the two aforementioned conventional methods in light of our method.

The rest of the paper is organized as follows. In Section 2 we give an overview of the web caching techniques and describe the conventional TTL estimation methods. In Section 3 we explain the concept of the update risk and develop a mechanism for determining the TTL based on the update risk. In Section 4 we describe the experiments performed and present the results. In Section 6 we compare our method with the two conventional methods in terms of the update risk. We conclude the paper in Section 7.

2 Related Work

In this section we first briefly review the architecture of web caching, and then, discuss three existing cache consistency maintenance approaches including the TTL methods. We also give a detailed explanation of the two conventional TTL estimation methods and their intrinsic defects.

2.1 Caching architecture

Based on their positions in the network, web cache servers are classified into the reverse cache server, the transparent cache server, and the proxy cache server as shown in Figure 1[20]. First, the reverse cache server is placed in front of the web server. It reduces the load on the web server by responding to clients' requests on behalf of the server, thereby helping the server scale up to handle heavy workload. Second, the transparent cache server and the proxy cache server are used for multiple clients to share and reuse the data accessed in the same local area network (LAN). These cache servers offer the following benefits obtained by obviating the accesses to remote servers: reduced response time, reduced network traffic, and reduced load on the web server. To use the proxy cache server, a client should explicitly indicate it in its configuration. To use the transparent cache server, a client does not have to because all requests, regardless of the client's configuration, are redirected to the cache server by a network switch. The client cache in Figure 1 is simply an internal cache of the individual browsers. Among these cache servers, our focus is on the client-side cache server. Therefore, from now on our scope is confined to the transparent cache server and the proxy cache server.

2.2 Cache consistency maintenance

We categorize and review existing cache consistency maintenance methods into three approaches in the context of web caching: the pull approach, the push approach, and the TTL approach. These approaches are distinguished by

[1] In this paper, the word "point" denotes a *point* in time as distinguished from an *interval* of time.

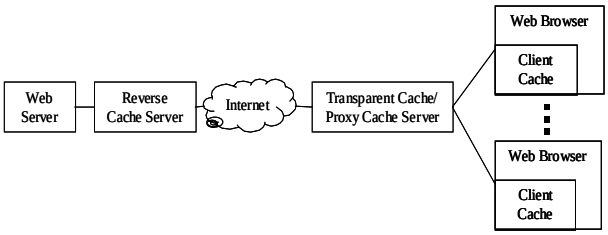

Figure 1. Classification of web cache servers by their positions in the network.

which side between the cache server and the web server takes the initiative of caching. In the pull approach, it is the cache server that connects to the web server and checks the consistency of cached data items. For this purpose, the cache server periodically polls the web server and refreshes its cache if the original data has been updated. In the push approach, by contrast, it is the web server that connects to the cache server and informs it of the updates of the cached data items. The web server sends the identifiers of the updated data items so that the cache server may invalidate them. In the TTL approach, the cache server assumes that the cached data item is valid and does not connect to the server until the TTL expires. The TTL is either provided by the web server or estimated by the cache server. When the TTL expires, the cache server inquires of the web server and re-caches the data item if it has been updated by the server, otherwise re-estimates the TTL. We call this process *revalidation*.

Among the three approaches, the TTL approach is regarded as the most suitable for the web environment [8]. In the pull approach, the polls sent by multiple cache servers increase the workload on the web severs and therefore delay the response. The push approach requires the web server to trace the data items cached by the cache servers. Besides, the web server may inform a cache server of updated data items after the cache server has already discarded them. These overhead impairs the scalability of the server. The push approach, therefore, is a method more appropriate for the reverse cache server. In the TTL approach, however, the cache servers do not connect to the web server during the TTL, thus loading the web server to a lesser extent compared with the pull or push approach. Although the TTL approach leaves the cache server unsynchronized with the web server during the TTL, users' acceptance of weak consistency relieves it from being a problem. In addition, the TTL approach facilitates adding new cache servers, as the pull approach does, and is not necessarily harder to implement than the other two approaches. Implementing the TTL approach involves calculating TTLs and revalidating cached pages but obviates the pollings or interrupts needed for asynchronous I/O's in the pull approach and the tracing and broadcasting of updated pages needed in the push approach For these reasons, the TTL approach is accepted as the most practical and effective approach to maintaining web cache consistency.

2.3 Conventional TTL estimation methods

The two conventional TTL estimation methods—the fixed TTL method and the heuristic method—are purely time-based methods. That is, they consider only time as the criterion for estimating the TTL. We review these two methods briefly here and point out their problems.

The fixed TTL method is used in IIS of MicroSoft [9]. It assigns the same TTL to all data items regardless of the time of update, and thus, is simple and easy to implement. However, the reality is that different data items are updated at different rates, and even the same data item is updated at different rates at different times. Thus, this method lacks any consideration for distinct and time-variant update patterns of data items at all. This deficiency deprives the method of its credibility in setting a reasonable TTL. If the TTL is set too large, the cache server ends up using obsolete data longer than necessary. If the TTL is set too small, it ends up revalidating unnecessarily often, thus wasting the system and network resources.

The heuristic method has been proposed as part of the HTTP/1.1 standard [8]. It estimates the TTL based on the interval between the current caching point (*Now*) and the last modification point (*Last-Modified*) of a data item. More specifically, as shown in Figure 2, it estimates the TTL as (*Now* $-$ *Last-Modified*) $\times M$, where M is an arbitrary proportion constant determined heuristically. An upper limit of 0.1 is recommended as the value of M in HTTP/1.1, and 0.5 by Chankhunthod et al. [4]. Neither upper limit has any theoretical reasoning behind it because it has been chosen with a guess or based on an experience. Although better than the fixed TTL method in reflecting the data update patterns of the web server, this method is still not flexible enough unless the updates occur regularly. Moreover, there is no intuitive meaning we can associate with the value of M.

From the problems observed in these two methods, we realize the need for a new method that enables us to adjust the TTL better to the actual occurrences of updates. We need an approach that is formal enough to provide a theoretical background and intuitive enough to allow a plausible explanation of the estimated TTL as the compromise point between the cache consistency and the performance of cache server.

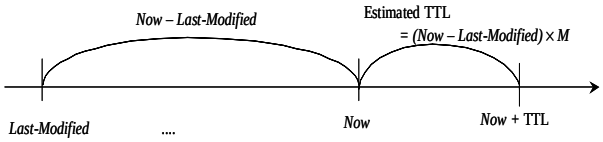

Figure 2. TTL estimation using the heuristic method.

3 Update-Risk Based TTL Method

In this section we give a formal definition of the update risk and then derive a formula for estimating the TTL given the maximum allowed update risk.

3.1 Update risk

Setting the TTL of a cached data item is based on the hypothesis that the data item will remain valid until the TTL expires. However, the hypothesis may prove false. We call its probability the update risk—the risk that the cached data item is updated before the TTL expires—and call such a TTL a *falseTTL*.

Definition 1: The *update risk* ρ_x of a cached data item x for a given TTL t_x is defined as the probability that the original data item of x is updated before the TTL expires. □

Figure 3 shows how the update risk ρ_x of a cached data item x can be obtained from the probability density distribution of updating the data item. By Definition 1, ρ_x is the probability that the data item x is updated between the current caching point *Now* and *Now*+t_x. This probability can be calculated by integrating the probability density distribution in the range.

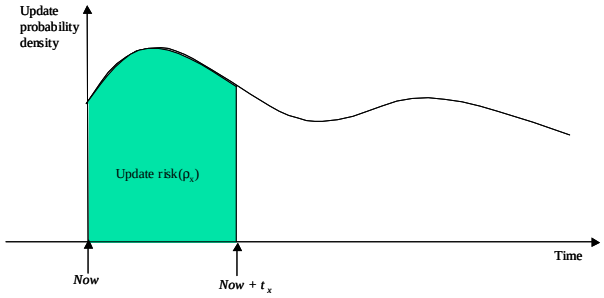

Figure 3. Update risk and update probability distribution.

Definition 1 is made from the perspective of a web server. In contrast, Definition 2 gives an alternate definition from the perspective of a client.

Definition 2: The *update risk* ρ_x of a cached data item x for a given TTL t_x is defined as the maximum probability that an arbitrary access to x returns an invalid (i.e., outdated) data item. (In other words, the probability that a client accesses an invalid data item is no more than the update risk of the data item.) □

In Figure 4, t_{ref} denotes the point in time the client accesses a data item x. Then, the probability that x becomes invalid by t_{ref} is the same as the probability that x is updated between *Now* and t_{ref}. This probability is obtained by integrating the update probability density distribution from *Now* to t_{ref} and is the same as the shaded area. Therefore, the probability of x being invalid reaches the maximum value when t_{ref} becomes equal to *Now*+t_x, and the maximum value is equal to the integral of the update probability density distribution from *Now* to *Now*+t_x. The integration returns the update risk of the data item x given the TTL t_x.

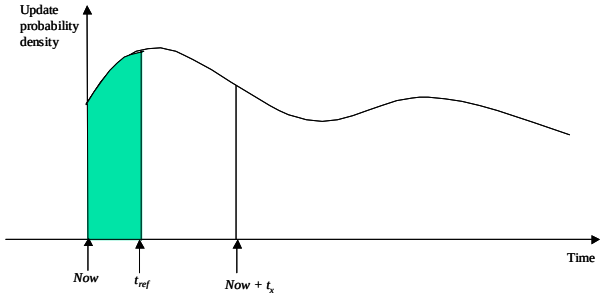

Figure 4. The probability of accessing an invalid data item at the cache access point t_{ref}.

3.2 Update-risk based TTL estimation

As mentioned, the TTL is estimated for a given value of the update risk. Using the probabilistic analysis method, we derive a formula for the estimation here.

Assuming that the updates are independent events, we model the number of updates as a Poisson process. Let $N(T)$ be the number of events occurring in the interval of T, and let $P[C]$ be the probability that the condition C is true. Then, given the Poisson constant μ—the average number of occurrences of an event in unit time (i.e., the average frequency of an event)—the update risk of a data item is expressed as in Theorem 1.

Theorem 1: For a given data item x, if the average update frequency is μ_x and the TTL is t_x, then the update risk of x, denoted by ρ_x, is obtained as:

$$\rho_x = 1 - e^{-\mu_x t_x} \quad (1)$$

PROOF: Since ρ_x is the probability that updates occur at least once in the interval t_x, $\rho_x = P[N(t_x) > 0] = 1 - P[N(t_x) = 0]$. Because $P[N(t_x) = 0] = e^{-\mu_x t_x}$ by the definition of the Poisson process, $\rho_x = 1 - P[N(t_x) = 0] = 1 - e^{-\mu_x t_x}$. □

Corollary 1: For a given data item x, if the average update frequency is μ_x and the update risk is ρ_x, then the TTL of x, denoted by t_x, is obtained as:

$$t_x = -\frac{1}{\mu_x} \log(1 - \rho_x) \tag{2}$$

PROOF: We obtain the Equation (2) trivially from Equation (1). □

Figure 5 shows the update risk ρ_x of a cached data item x as a function of the TTL t_x (Equations (1) and (2)). In Figure 5 we observe that the update risk ρ_x increases faster when the TTL is smaller, and slower when the TTL is larger, eventually saturating toward 1.0. We also see that the TTL t_x of a data item x increases as the update risk ρ_x increases. This means that we need a longer TTL to accommodate a higher update risk.

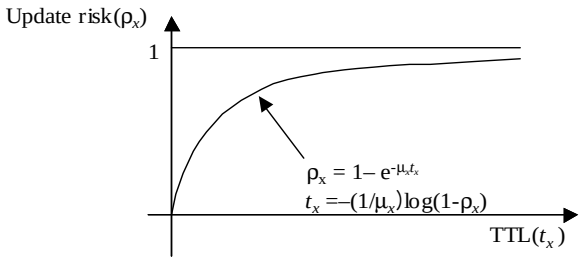

Figure 5. Update risk as a function of TTL.

In the context of this paper, the Poisson constant μ_x denotes the average frequency of *updates*. In practice, as illustrated in Figure 6 we estimate μ_x from the previous K update occurrences (as in the LRU-K [17]) recorded in a cache log file. In this case, we can use an alternate formula shown below for estimating the TTL. Let us first define the *backward K-update distance* of a data item x ($BUD_x(K)$) as a function of K that, given the value of K, returns the interval between the caching point *Now* and the K-th last update point of x. Then, since K updates occurred during $BUD_x(K)$, μ_x is calculated as:

$$\mu_x = \frac{K}{BUD_x(K)} \tag{3}$$

It is worthwhile to consider the appropriate value of K in Equation (3). There is no hard rule for this, but μ_x is meaningful when K falls within the range of temporally local

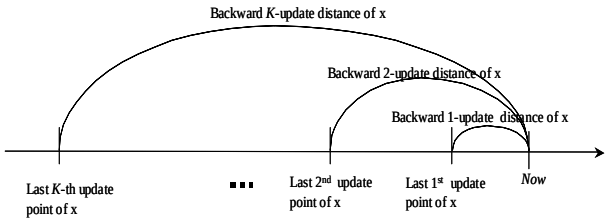

Figure 6. Backward K-update distance of a data item x.

updates. For the rest of the paper, μ_x denotes the value obtained as in Equation (3) unless stated otherwise.

If we use μ_x obtained as in Equation (3), then from Equation (2) we estimate the TTL t_x as:

$$t_x = -\frac{BUD_x(K)}{K} \log(1 - \rho_x) \tag{4}$$

We use Equation (4) to calculate the TTL in our experiments presented in Section 4.

If we assume a uniform interval between two consecutive updates, we can simplify Equation (4) further by ignoring $BUD_x(1)$. When Δ_x denotes the uniform interval between two consecutive updates of a data item x, t_x is simply caculated as follows since $\frac{BUD_x(K) - BUD_x(1)}{K-1} = \Delta_x$, which is independent of K.

$$t_x = -\Delta_x \log(1 - \rho_x) \tag{5}$$

In fact, as K increases, so does the value of Equation (4) asymptotically approaching the value of Equation (5).

4 Experiments

In this section we demonstrate the effectiveness and the practicality of our TTL estimation method through an experiment using the logs generated by a real cache server. For this purpose, we first describe the criterion for assessing our method and the model of the experiments, and then, present the result.

4.1 Assessment criterion

As the metric for judging the effectiveness our TTL estimation method, we use the ratio of the number of the falseTTLs (denoted by #falseTTLs) that occur to the total number of the estimated TTLs (denoted by #TTLs). We call this ratio the *false estimation ratio*.

$$\text{false estimation ratio} = \frac{\#\text{falseTTLs}}{\#\text{TTLs}} \tag{6}$$

In other words, the false estimation ratio is a measure of how often the estimated TTL violates the assumption of cache validity before it expires. Its probabilistic expected value is equivalent to the update risk. Therefore, if the false estimation ratio experimentally obtained and the update risk used to estimate the TTL are close enough, it proves our method effective.

4.2 Experiment model

Figure 7 shows the configuration of the simulation program used in the experiment. We use two kinds of log files (access.log and store.log) downloaded from the IRCache project site [16]. The log files have the data format used by the Squid proxy cache [19]. The access.log file contains information about the reference requests; the store.log file contains information about cache entrances, removals, and updates of the cached data items.

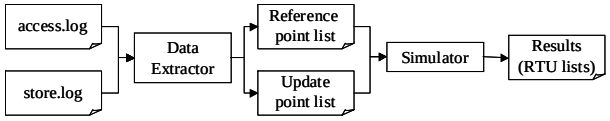

Figure 7. Simulation program configuration.

The data extractor extracts data from the two log files for the simulator to use. From the access.log file, the data extractor extracts the URLs and the reference (time) points of the accessed data items, and then writes them into the *reference point list* file. From the store.log file, it extracts the URLs and the update (time) points of the data items whose access records exist in the access.log, and then writes them into the *update point list* file. The update points are retrieved from the *Last-Modified* field in the updated data items. When using backward K-update distance, we exclude data items with no more than K update records because $(K+1)$ update (time) points are required to calculate a backward K-update distance. As a result, we generate the reference points and update points of about 10,000 data items. Provided with the *reference point list* file and the *update point list* file, the simulator checks, for each URL in the two list files, whether an update of the data item identified by the URL occurs within the TTL estimated using Equation (4). Then, it calculates the false estimation ratio as shown in Equation (6) and writes the resulting false estimation ratio information into a file.

Figure 8 shows the TTL estimation points (t_{est}), the expiration points (t_{exp}), and the next update points (t_u) of a data item as they appear in the cache log. The simulator estimates the next TTL at the first reference point after the expiration of the previous TTL because this is when the cache server assigns a new TTL based on the result of revalidating the cached data item. In case the previous TTL does not exist, it uses the first reference point as the caching point and assigns a new TTL at that point.

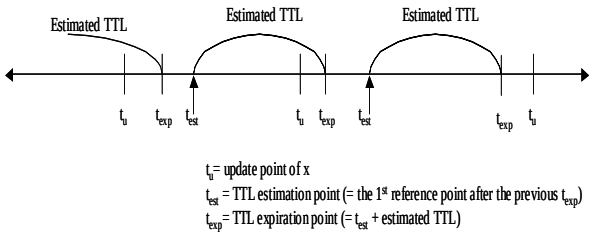

Figure 8. TTL estimation points, TTL expiration points, and the next update points based on a cache log.

4.3 Experiment result

Figure 9 compares the average false estimation ratio measured and the average update risk specified by the user of the 10,000 data items extracted from the two log files. We use 1, 2, 3, and 4 as the value of K and do not consider any higher values. The reason for this is that, as shown in Equation (5), the TTL t_x shows a tendency of becoming independent of K as K increases as long as it does not break the temporal locality of update occurrences.

In Figure 9, we see that the average false estimation ratio increases almost linearly with the update risk. The slope is close enough to 1 proving the quality of our TTL estimation method.

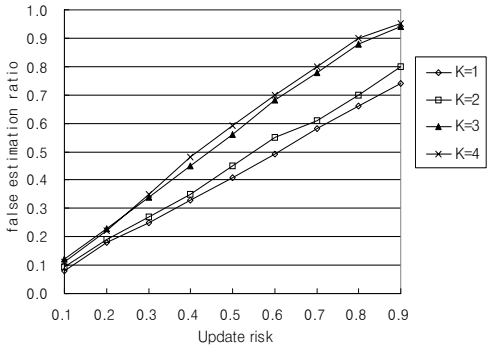

Figure 9. False estimation ratio vs. update risk.

Figure 10 is a plot of the similarity between the false estimation ratio and the update risk, where the *similarity* is defined as:

$$\text{similarity} = \frac{\text{false estimation ratio}}{\text{update risk}} \qquad (7)$$

The figure shows that the similarity ranges between 0.8 and 1.2 for K ranging between 1 and 4. It is closest to 1 when $K = 2$, suggesting that 2 is the optimum value of K. This result is consistent with the observation made in the reference [17].

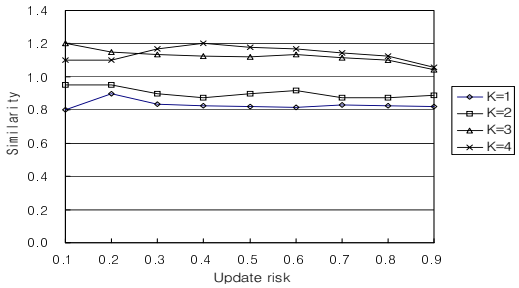

Figure 10. The similarity values for K =1,2,3, and 4.

5 Impact of update risk on consistency and performance of cache

In this section, to provide a help for finding the compromising point between consistency and performance of cache, we empirically show the impact of update risk on consistency and performance of cache.

In the same experiment as described in Section 4, we add an assumption that the cache server can store all the referenced data in order to show only the impact of update risk excluding other factors such as the replacement algorithm. Then, we add new measures: the consistency ratio and the hit ratio. We define the *consistency ratio* as the ratio of the number of valid references to the total number of references. A *valid reference* is defined as a reference to the data synchronized with the original data(we note that the reference occurring within a TTL may not be a valid reference.). The *hit ratio* is the ratio of the number of references to the data in cache to the total number of references(we note that the number of references in cache includes invalid references).

$$\text{consistency ratio} = \frac{\text{number of valid references}}{\text{total number of the references}}$$

$$\text{hit ratio} = \frac{\text{number of references to data in cache}}{\text{total number of the references}}$$

As shown in Figure 11, we see the consistency ratio and the hit ratio have a trade-off relationship. Therefore, the cache server administrator need to find a compromising point according to network congestion. When the network is congested heavily, we may enlarge update risk in order to increase the hit ratio and to reduce congestion and response time sacrificing the consistency ratio. On the other hand, when the network is not congested, we may reduce update risk to increase the consistency ratio within the tolerable bounds of response time.

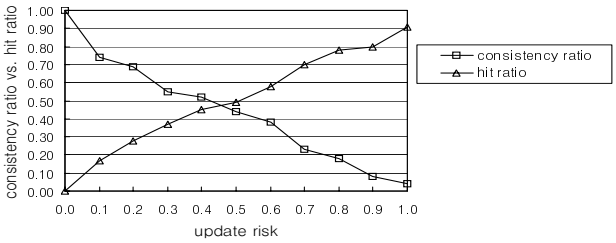

Figure 11. The consistency ratio and the hit ratio as update risk changes.

As shown in Figure 11, when update risk is 0 (TTL is set to 0), the consistency ratio is 1 and the hit ratio is 0 since nothing can be cached. On the other hand, when the update risk is 1 (TTL is set to an infinite value), the hit ratio does not grow to 1 since the first references of each data cannot be read from cache; the consistency ratio does not degrade to 0 since references occurring between the point of caching and the point of update is valid.

6 Comparison with Conventional Methods

In this section, we compare our method with the two conventional methods—the fixed TTL method and the heuristic method—described in Section 2.3 in terms of the update risk.

The fixed TTL method assigns the same TTL to all data items regardless of their update patterns. Putting it in terms of the update risk, the TTL is not influenced by the update risk at all, as shown in Figure 12. As a result, each cached data item is subject to a different update risk, which renders the risk so unpredictable as to impair the validity of cached data as a whole.

The heuristic method takes a certain proportion (M) of the backward 1-update distance (i.e., *Now* $-$ *Last-Modified*) as the value of TTL. Figure 13 shows how TTL changes with respect to the value of M. Since the backward 1-update distance is a known constant at the point of caching, TTL is a linear function of M, with the the backward 1-update distance given as the slope.

By now we know that the TTL is a function of the update risk in our method, and is a function of M in the heuristic method. Thus, once we know the relationship between the update risk and M, we can deduce the relationship between the two TTL estimation methods. Theorem 2 serves the purpose.

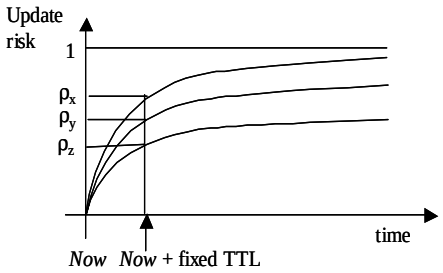

Figure 12. Update risk ρ vs. TTL in the fixed TTL method.

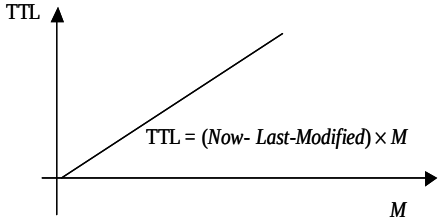

Figure 13. TTL as a linear function of M in the heuritsitc method.

Theorem 2: If in the update-risk based method we use the backward 1-update distance, which is the distance used in the heuristic method, then the heuristic method with the proportion constant M is equivalent to the update-risk based method with an update risk of $1 - e^{-M}$.

PROOF: From Equation (3), the update frequency μ_x of a data item x in the update-risk based TTL estimation method using the backward 1-update distance is estimated as:

$$\mu_x = \frac{1}{BUD_x(1)} \quad (8)$$

$$= \frac{1}{\textit{Now} - \textit{Last-Modified}(x)} \quad (9)$$

where *Last-Modified*(x) denotes the last update point of x in reference to the current caching point *Now*. Then, from Equation (9), the TTL t_x of the heuristic method is obtained as:

$$t_x = (\textit{Now} - \textit{Last-Modified}(x)) \times M \quad (10)$$

$$= \frac{1}{\mu_x} \times M \quad (11)$$

Hence, $M = \mu_x \times t_x$. By substituting M into $\mu_x t_x$ in Equation (1), we obtain the update risk ρ_x as:

$$\rho_x = 1 - e^{-M} \quad (12)$$

□

7 Conclusions

We have presented a new method for estimating the TTL under the weak cache consistency requirement of client-side cache servers such as the transparent cache server and the proxy cache server. Our method is based on the notion of update risk, defined as the probability of the original data item being updated during the TTL. The TTL of a cached data item is estimated given the maximum allowed update risk of the data item. For this purpose, we have developed a formal method under the assumption that the process of update occurrences is modeled as a Poisson process. The Poisson constant, which is interpreted as the average frequency of update occurrences, can be estimated as the inverse of the average interval between two consecutive updates for the past K update occurrences, where K is recommended to be within the temporal locality zone of updates.

To verify the effectiveness and practicality of our method, we have performed experiments using real web cache log files. We have compared the update risk and false estimation ratio with the result of obtaining a close match between the two—confirming the merit of our method. We have also shown the impact of update risk on the consistency ratio and the hit ratio, which can be used to help the administrator determine an appropriate value of update risk.

We then have discussed how the two conventional TTL methods compare in the framework of our method. The fixed TTL method assigns the same TTL without regard to the actual update risk of each data item. The heuristic method is a special case of our method in which the update risk is equal to $1-e^{-M}$, where M is an arbitrary proportion of the distance between the caching point and the immediately previous update point.

We are currently investigating the feasibility of applying the proposed method to dynamic web data generated on demand. Recently, cache products for dynamic web data such as dynamai, Xcache, and spider cache have been commercialized [3]. Caching techniques for dynamic data are more complicated since dynamicity of data makes it difficult to detect updates on web data. I think, however, TTL estimation for dynamic web data will be more important since none of dynamic data has TTL and therefore the cache server must always estimate it.

Compared with the two conventional methods—the fixed TTL method and the heuristic method—our method proves to generate the TTL that reflects the reality of cache accesses better. Besides, our method gives the TTL a meaning that is well defined and intuitively comprehensible. In summary, we believe that our method provides a clear understanding and a formal basis for the TTL–based approach to maintaining web cache consistency.

References

[1] Arlitt, M. F. and Willaiamson, C. L., "Web Server Workload Characterization: The Search for Invariants," In *Proc. Int'l Conf. on Measurement and Modeling of Computer Systems*, ACM SIGMETRICS, Philadelphia, pp. 126–136, May 1996.

[2] Barish, G. and Obraczka, K., "World Wide Web Caching: Trends and Techniques," *IEEE Communications*, Vol. 38, No. 5, pp. 178–184, May 2000.

[3] Internet Caching Resource Center, http://www.caching.com/hitsandmisses/, 2001.

[4] Chankhunthod, A., Danzig, P. B., and Neerdaels, C., "A Hierarchical Internet Object Cache," In *Proc. USENIX Technical Conf.*, USENIX Association, San Diego, Calif., pp. 153–164, Jan. 1996.

[5] Cho, J., and Garcia-Molina, H., "Synchronizing a database to Improve Freshness," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, Dallas, pp. 117–128, May 2000.

[6] Colajanni, M. and Yu, P. S., "Adaptive TTL Schemes for Load Balancing of Distributed Web Servers," In *Performance Evaluation Review*, ACM SIGMETRICS, Vol. 25, No. 2, pp. 36–42, Sept. 1997.

[7] Gwertzman, J. and Seltzer, M., "World-Wide Web Cache Consistency," In *Proc. USENIX Technical Conf.*, USENIX Association, San Diego, Calif., pp. 141-152, Jan. 1996.

[8] Fielding, R., Gettys, J., Mogul, J. C., Frystyk, H., and Berners-Lee, T., Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, http://nic.ddn.mil/ftp/rfc/rfc2616.txt, June 1999.

[9] Internet Information Services 5.0 Technical Overview, http://www.microsoft.com/windows2000/docs/, 2001.

[10] Inktomi Essential, http://www.inktomi.com, 2001.

[11] The Leading Edge Web Caching Software, http://www.imimic.com, 2001.

[12] Novell Internet Cacing System, http://www.novell.com/products/ics, 2001.

[13] The Content Smart Company, http://www.cacheflow.com, 2001.

[14] Enabling Rich Media Applications with Award Winning Infrastructure, http://www.info-libria.com, 2001.

[15] Liu, C., and Cao, P., "Maintaining Strong Cache Consistency in the World-Wide-Web," *IEEE Trans. on Computers*, Vol. 47, No. 4, pp. 445–457, Apr. 1998.

[16] IRCache Home, http://www.ircache.net/, 2001.

[17] O'Neil, E. J., O'Neil, P. E., and Weikum, G., "The LRU-K Page Replacement Algorithm For Database Disk Buffering," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, Washington, D.C., pp. 297–306, May 1993.

[18] Shim, J., Scheuermann, P., and Vingralek, R., "Proxy Cache Design: Algorithms, Implementation and Performance," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 11, No. 4, pp. 549–562, July/Aug. 1999.

[19] Pearson, O., *Squid: A User's Guide*, http://www.squid-cache.org, 2001.

[20] Williams, B., "Transparent Web caching solutions," In *Proc. Int'l Workshop on WWW Caching*, Manchester, England, June 1998.