Adaptive TTL-Based Caching for Content Delivery

Soumya Basu
The University of Texas at Austin

Aditya Sundarrajan
University of Massachusetts Amherst

Javad Ghaderi
Columbia University

Sanjay Shakkottai
The University of Texas at Austin

Ramesh Sitaraman
University of Massachusetts Amherst,
Akamai Technologies

ABSTRACT

Content Delivery Networks (CDNs) cache and serve a majority of the user-requested content on the Internet, including web pages, videos, and software downloads. We propose two TTL-based caching algorithms that *automatically* adapt to the heterogeneity, burstiness, and non-stationary nature of real-world content requests. The first algorithm called d-TTL dynamically adapts a TTL parameter using a stochastic approximation approach and achieves a given feasible target hit rate. The second algorithm called f-TTL uses two caches, each with its own TTL. The lower-level cache adaptively filters out non-stationary content, while the higher-level cache stores frequently-accessed stationary content. We implement d-TTL and f-TTL and evaluate both algorithms using an extensive nine-day trace consisting of more than 500 million requests from a production CDN server. We show that both d-TTL and f-TTL converge to their hit rate targets with an error of about 1.3%. We also show that f-TTL requires a significantly smaller cache size than d-TTL to achieve the same hit rate, since it effectively filters out rarely-accessed content.

KEYWORDS

TTL caching, Content Delivery Network, Stochastic Approximation

ACM Reference format:
Soumya Basu, Aditya Sundarrajan, Javad Ghaderi, Sanjay Shakkottai, and Ramesh Sitaraman. 2017. Adaptive TTL-Based Caching for Content Delivery. In *Proceedings of SIGMETRICS '17, June 5–9, 2017, Urbana-Champaign, IL, USA, ,* 2 pages.
DOI: http://dx.doi.org/10.1145/3078505.3078560

1 INTRODUCTION

By caching and delivering content to millions of end users around the world, content delivery networks (CDNs) are an integral part of the Internet infrastructure. The major technical challenge in designing caching algorithms for a modern CDN is *adapting* to heterogeneous, bursty (correlations over time) and non-stationary/transient request statistics. In this effort, applying known heuristics such as Che's approximation fail due to modeling inaccuracies and manually tuning the caching algorithms becomes prohibitively expensive. Thus, our goal is to devise self-tuning TTL-based caching algorithms that can automatically learn and adapt to heterogeneous, bursty and non-stationary traffic and provably achieve any feasible hit rate and cache size.

We propose two TTL-based algorithms: d-TTL (for "dynamic TTL") and f-TTL (for "filtering TTL"). Rather than statically deriving the required TTL values from the request statistics, our algorithms *incrementally adapt* the TTL values after each request, based on the current request patterns. Our algorithms do not have prior knowledge of request statistics; instead we use a stochastic approximation framework and ideas from actor-critic algorithms for parameter adaptation. We implement both d-TTL and f-TTL algorithms and evaluate them using an extensive nine-day trace consisting of more than 500 million requests from a production Akamai CDN server. For a range of object hit rate targets, both d-TTL and f-TTL converge to that target with an error of about 1.3%. For a range of byte hit rate targets, both d-TTL and f-TTL converge to that target with an error that ranges from 0.3% to 2.3%. In particular, f-TTL requires a cache that is 49%(resp., 39%) smaller than d-TTL to achieve the same object hit rate(resp., byte hit rate).

2 TTL-BASED CACHING ALGORITHMS

A TTL-based caching algorithm works as follows. When a new object is requested, it is placed in cache and associated with a time-to-live (TTL) value. If no new request is received for that object, the TTL value is decremented in real-time and the object is evicted when the TTL becomes zero. If a cached object is requested, a *cache hit* occurs and the TTL is reset to its original value. When the requested object is not found in cache, a *cache miss* occurs. Consider T different types of content. The *objective* of this work is to (asymptotically) achieve a *target hit rate* $h_t^* \in (0,1)$ and a (feasible) *target size rate* s_t^*, for each type $t \in [T]$. To accurately model CDN traffic, we allow the request traffic to be non-independent and non-stationary; the request traffic can have Markovian dependence over time. The traffic comprises a mix of stationary demands (statistics invariant over the timescale of interest), and non-stationary demands (finitely many requests, or in general requests with an asymptotically vanishing request rate). The complete model is described in [1].

d-TTL Cache. The d-TTL algorithm adapts the TTL value on every request arrival to achieve a target hit rate $h_t^* \ \forall t \in [T]$. d-TTL uses stochastic approximation to dynamically increase the TTL when the current hit rate is below the target, and decrease the TTL when the current hit rate is above the target. Let $\theta_t(l)$ be the TTL value after the l-th request arrival for content type t. Then, if the object experiences a cache miss, d-TTL increases the TTL to, $\theta_t(l+1) = \theta_t(l) + \frac{\eta_0}{l^\alpha}(h_t^*)$, where η_0 is some constant and $\alpha \in (0.5, 1)$.

SIGMETRICS '17, June 5–9, 2017, Urbana-Champaign, IL, USA

DOI: http://dx.doi.org/10.1145/3078505.3078560

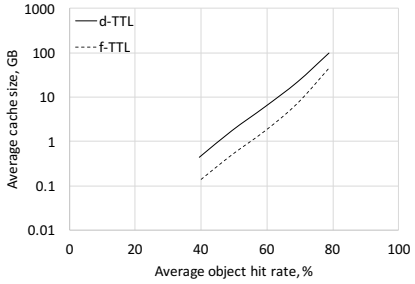

Figure 1: Hit rate curve for object hit rates.

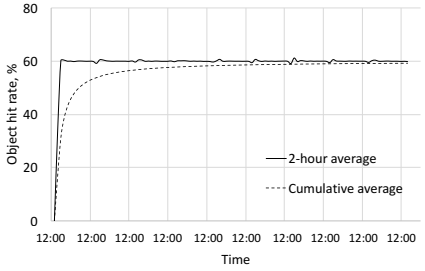

Figure 2: Object hit rate convergence over time for d-TTL; target hit rate=60%.

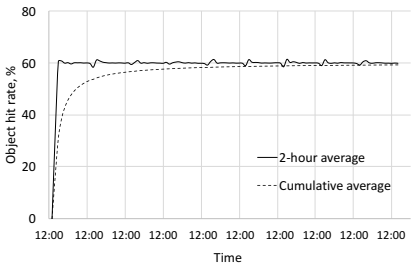

Figure 3: Object hit rate convergence over time for f-TTL; target hit rate=60%.

On the contrary, in the event of cache hit, the TTL decreases to $\theta_t(l+1) = \theta_t(l) - \frac{\eta_0}{l^\alpha}(1 - h_t^*)$.

While d-TTL does a good job of achieving the target hit rate, it does this at the expense of caching rare and unpopular recurring content for an extended period of time, thus causing an increase in cache size without any significant contribution towards the cache hit rate. We present a two-level adaptive TTL algorithm called filtering TTL (f-TTL) that filters out rare and unpopular content to achieve both a target size rate and a target hit rate (at a cache size smaller than d-TTL).

f-TTL Cache. The two-level f-TTL algorithm maintains two caches, a lower-level cache C_s and a higher-level cache C. To facilitate the filtering process, it uses two dynamic TTL values—one ($\theta_t^s(l)$) less than or equal to the other ($\theta_t(l)$). Upon a *cache miss* for object c of type t, object c, potentially unpopular, is cached in C_s with the smaller TTL to ensure quick eviction. Its metadata $\tilde{c}$ (only the object ID and not the actual content) is cached in C_s with the larger TTL to retain *memory* of this request. The two TTLs are then updated to, $\theta_t^s(l+1) = \theta_t^s(l) + \frac{\eta_0}{l}\left(s_t^* - \theta_t^s(l)\right)$ and $\theta_t(l+1) = \theta_t(l) + \frac{\eta_0}{l^\alpha}h_t^*$.

Upon a *cache hit*, object c—now showing signs of popularity—is cached[1] in the higher-level cache C with the larger TTL, $\theta_t(l)$. Let ϕ be the remaining time for object c. Then, the smaller TTL is updated to $\theta_t^s(l+1) = \theta_t^s(l) + \frac{\eta_0}{l}\left(s_t^* - \theta_t(l) + \phi\right)$ and the larger TTL is decremented to $\theta_t(l+1) = \theta_t(l) - \frac{\eta_0}{l^\alpha}(1 - h_t^*)$.

In f-TTL there is a third possibility. When the requested object c is not in either cache but its metadata $\tilde{c}$ exists, a *virtual hit* occurs. Object c, which is possibly popular is then cached in the higher-level cache with the larger TTL value. The smaller TTL is updated to $\theta_t^s(l+1) = \theta_t^s(l) + \frac{\eta_0}{l}\left(s_t^* - \theta_t(l)\right)$ and the larger TTL increases to $\theta_t(l+1) = \theta_t(l) + \frac{\eta_0}{l^\alpha}h_t^*$.

The extra *memory* in the form of object metadata helps f-TTL cache popular objects for a longer period of time while filtering out rare and unpopular content quickly. Thus, f-TTL utilizes the cache space more efficiently to simultaneously achieve a target hit rate and a target size rate.

The convergence results of both d-TTL and f-TTL algorithms are presented in [1].

[1]Caching an object in the higher-level cache implies evicting the object and/or its metadata from the lower-level cache.

3 EMPIRICAL EVALUATION

We use an extensive data set containing access logs for content requested by users that we collected from a typical production server in Akamai's commercially-deployed CDN, over a period of 9 days. The content requests traces contain 504 million requests (resp., 165TB) for 25 million distinct objects (resp., 15TB). We observe that the content popularity distribution exhibits a "long tail" with nearly 70% of the objects accessed only once. Further, we also see that 80% of the requests are for 1% of the objects. This indicates the presence of a significant amount of non-stationary traffic in the form of "one-hit-wonders" and rarely accessed content.

The performance of a caching algorithm is often measured by its hit rate curve (HRC) that relates the cache size to the hit rate it achieves. We compare the HRCs of d-TTL and f-TTL for object hit rates and show that f-TTL significantly outperforms d-TTL by filtering out the rarely-accessed non-stationary content. The HRCs for object hit rates are shown in Figure 1. Note that the y-axis is presented in log scale for clarity.

From Figure 1 we see that f-TTL always performs better than d-TTL. We find that on average, f-TTL requires a cache that is 49% smaller than d-TTL to achieve the same object hit rate.

For the dynamic TTL algorithms to be useful in practice, they need to converge to the target hit rate with low error. From Figures 2 and 3, we see that the 2 hour averaged object hit rates achieved by both d-TTL and f-TTL have a cumulative error of less than 1.3% while achieving the target object hit rate, on average. We also see that both d-TTL and f-TTL converge to the target hit rate quickly, which illustrates that both d-TTL and f-TTL are able to adapt well to the dynamics of the non-stationary traffic.

A more detailed evaluation of the dynamic TTL algorithms including the analysis of byte hit rates, sensitivity analyses and comparison to the Che's approximation heuristic can be found in [1].

ACKNOWLEDGMENTS

This work is partially supported by the US DoT supported D-STOP Tier 1 University Transportation Center, NSF grant CNS-1652115 and NSF grant CNS-1413998.

REFERENCES

[1] Soumya Basu, Aditya Sundarrajan, Javad Ghaderi, Sanjay Shakkottai, and Ramesh Sitaraman. 2017. Adaptive TTL-Based Caching for Content Delivery. *CoRR* abs/1704.04448 (2017). http://arxiv.org/abs/1704.04448