Elastic Provisioning of Cloud Caches: a Cost-aware TTL Approach

Damiano Carra, Giovanni Neglia, Pietro Michiardi

Elastic Provisioning of Cloud Caches: a Cost-aware TTL Approach

Damiano Carra
University of Verona
Verona, Italy

Giovanni Neglia
Université Côte d'Azur, Inria
Sophia-Antipolis, France

Pietro Michiardi
Eurecom
Sophia-Antipolis, France

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; • **Information systems** → *Cloud based storage*; • **Theory of computation** → *Stochastic approximation*;

KEYWORDS

Dynamic cache, Cost minimization

ACM Reference Format:
Damiano Carra, Giovanni Neglia, and Pietro Michiardi. 2018. Elastic Provisioning of Cloud Caches: a Cost-aware TTL Approach. In *Proceedings of ACM Symposium on Cloud Computing , Carlsbad, CA, USA, October 11–13, 2018 (SoCC '18),* 1 pages.
https://doi.org/10.1145/3267809.3275468

1 INTRODUCTION AND MOTIVATION

In-memory key-value stores used as caches are a fundamental building block both for web services and Content Delivery Networks (CDNs). Cloud operators offer pay-as-you-go elastic services (e.g. Amazon's ElastiCache and Google's Cloud Memorystore) based on open-source software, such as Memcached or Redis. The total operating cost of such services not only includes the cloud storage fee, but also the (more difficult to evaluate) cost due to misses: in fact, the cache miss ratio has a direct impact on the performance perceived by end users, which directly affects the overall revenue of cloud customers.

The analysis of dynamic adaptation of cloud caches has received little attention: the few studies have focused on minimizing storage costs for a given target hit ratio, ignoring that misses may have different costs and disregarding the possibility to tune the hit ratio itself.

In our work, we study the **dynamic** assignment of resources to in-memory data stores used as **caches**. To this aim, we take into account the cost of the storage *and* the cost of the misses, and we adapt the amount of resources to the traffic pattern **minimizing** the total **cost**. We consider an approach based on Time-To-Live (TTL) caches, and we study a model in which the TTL is adapted through *stochastic approximation* iterations and dynamically converges to the best setting.

2 CONTRIBUTION AND RESULTS

TTL-caches for vertical-scalability. In TTL caches, upon a miss, the content is stored locally; a timer with duration T is activated and is reset by the following hits. The content is evicted when the timer expires. The larger T, the more likely the content is to be found in the cache. As a consequence, the larger T, the smaller the number of misses, but the higher the expenditure for storage at the cloud. Our policy adapts dynamically the timer value T to minimize the overall cost, which includes both the storage cost, and the cost due to misses. In particular, a stochastic approximation algorithm updates the timer at the n-th miss, when content $r(n)$ is requested, as follows:

$$T(n) = T(n-1) + \epsilon(n)\left(\hat{\lambda}_{r(n)} m_{r(n)} - c_{r(n)}\right), \qquad (1)$$

where $\hat{\lambda}_r$ is a random unbiased estimate of the arrival rate of content r; $\epsilon(n)$ is the learning rate parameter; c_r and m_r denote respectively how much it costs to store content r per time unit and how much to retrieve it upon a miss. In the companion technical report,[1] we show that *i)* the update rule (1) indeed minimizes the total cost in a stationary setting, *ii)* the TTL policy can be implemented with $O(1)$ computation cost per request, as LRU.

Horizontally scalable cache system. The TTL-based scheme above considers a perfect vertically-scalable system, where memory resources can be smoothly added and removed. Inspired by the TTL-based approach, we design a practical horizontally-scalable system, where cache instances can be added or removed at finite epochs.

The instances are managed by a *load balancer*, which performs the ordinary operations, such as request routing, content retrieval, and content insertion. In addition, the load balancer maintains a virtual cache (VC), with the references of the requested objects. The VC is managed as a TTL cache. The instantaneous size of the VC depends on the timer value T, which is dynamically adapted as in (1) taking into account storage and miss costs. Thus, the size of the VC can be used to determine the number of actual instances to employ in the cluster.

Results. We compare our solution to static baseline configurations using real-world traces collected from Akamai CDN. We observe a total cost reduction ranging from 17% to 40% for highly dynamic traffic scenarios. While cost saving is comparable with that of state-of-the-art solutions based on Miss Ratio Curves (MRC), our solution is more scalable being able to serve about 60% more traffic.

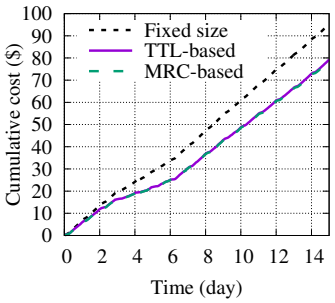

Figure 1: Cumulative cost of different policies.

[1]The companion technical report can be found at: https://arxiv.org/abs/1802.04696

SoCC '18, October 11–13, 2018, Carlsbad, CA, USA

ACM ISBN 978-1-4503-6011-1/18/10.
https://doi.org/10.1145/3267809.3275468