Computer Networks 134 (2018) 55–65

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Modeling LRU cache with invalidation

Andrea Detti[a,b], Lorenzo Bracciale[a], Pierpaolo Loreti[a], Nicola Blefari Melazzi[a,b,*]

[a] Electronic Engineering Department, University of Rome Tor Vergata, Rome, Italy
[b] Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT), Italy

ARTICLE INFO

Article history:
Received 27 March 2017
Revised 14 December 2017
Accepted 17 January 2018
Available online 31 January 2018

Keywords:
Caching
Invalidation
LRU
Wikipedia

ABSTRACT

Least Recently Used (LRU) is a very popular caching replacement policy. It is very easy to implement and offers good performance, especially when data requests are temporally correlated, as in the case of web traffic.

When the data content can change during time, as in the case of dynamic websites or within databases, there is the need to prevent the cache to serve stale data. This is usually done by triggering an invalidation event in the cache, to purge all the previously cached data concerning the invalidated data item. The invalidation process tends to worsen the caching performance, since stored items can be invalidated after a short time, thus wasting storage space.

Several models in the literature allow quantifying the cache hit probability of an LRU cache, but, to the best of our knowledge, the presence of invalidation events has not been taken into account so far.

In this paper, we present an analytical performance evaluation of LRU caches that takes into account data requests and invalidation events, both modeled as independent renewal processes. Simulation results show the accuracy of our model. Moreover, we apply our model to evaluate the LRU performance in the case of a real application, Wikipedia. Finally, we evaluate by means of simulations the effect of invalidation in hierarchical caching.

Our work allows us to conclude that the presence of invalidation events does not severely impact the LRU performance in single caches. As a matter of fact, invalidation effects can be ignored there, unless the invalidation rate is comparable with the request rate and the per-object invalidation rate and request rate are highly correlated. However, in the case of hierarchical caching, even a limited effect of invalidation on first-level caches is sufficient to noticeably affect the performance of second level/downstream caches.

1. Introduction

Caching is a well-known technique, used in web and database applications to reduce the data transport latency, processing load and network traffic and reduce/eliminate the occurrence of congestion / bottlenecks. A caching system is typically placed between the user(s) and the data source(s) and stores a copy of the response data of some requests, so that subsequent identical requests are served directly by that system instead of from the origin server.

A caching system has a finite storage size. Therefore, some requested data items may be found in the cache (cache hit), while others are not (cache miss). In the case of a cache miss, the caching system fetches the requested item from the origin server (or more formally, it fetches the server response), possibly stores a copy of it for future use, and then serves the user. The main performance measure of a caching system is the cache hit probability, which is the probability that a generic request can be served with a cached item (cache hit), instead of being forwarded to the origin server (cache miss).

A caching scheme determines which data items should be stored in the cache, e.g., in order to maximize the cache hit rate. More specifically, a *replacement policy* is a caching strategy that decides whether a requested item should enter the cache in case of a cache miss, as well as which item should then be evicted from the cache (i.e., which cached item will be replaced by the requested item), if the cache storage space is exhausted. Such decisions are based on the user behavior (i.e. the requests pattern), which is used to understand which item is addressed more frequently.

Least Recently Use (LRU) is probably the most popular caching scheme, mainly due to its simple implementation, its low and constant cache update overhead, and its relatively good performance. The LRU policy is implemented in software as a finite-size stack

* Corresponding author at: Electronic Engineering Department, University of Rome Tor Vergata, Rome, Italy. Via del Politecnico 1, Rome, Italy.
E-mail addresses: andrea.detti@uniroma2.it (A. Detti), lorenzo.bracciale@uniroma2.it (L. Bracciale), pierpaolo.loreti@uniroma2.it (P. Loreti), blefari@uniroma2.it (N.B. Melazzi).

https://doi.org/10.1016/j.comnet.2018.01.029
1389-1286/

of cached items. For each request, if the requested item is in the stack, then it is moved at the top of the stack; otherwise the requested item is inserted at the top of the stack and the last item of the stack is removed, to comply with the storage limit.

Besides its simplicity, LRU also provides very good performance in terms of cache hit probability. This is due to the fact that LRU exploits the temporal correlations among requests, which are often found in web and database traffic patterns. The temporal locality refers to the tendency of recently requested items to be addressed again, which makes the most recently requested items good candidates for caching.

However, any caching system must cope with a fundamental consistency problem: how to prevent a cache from serving *stale* data items, i.e. items whose version is older than the one available at the source. Indeed, the content that is stored in the origin server is often dynamically updated. Therefore, it's necessary to check and enforce the consistency between the cached copy of the data stored in the caching system and the original data stored in the content server (data source).

There are two types of data consistency: weak and strong. Weak consistency mechanisms include the association of a time-to-live (TTL) or an expiration time (in HTTP 1.1) to the cached data. When this timer expires, the consistency of the cached data has to be checked by contacting the origin server. Weak consistency schemes cannot guarantee data consistency, since there is always the possibility that the data items have been updated at the origin server between two consistency checks (i.e. while the TTL or expiration timer was still running). Thus, this strategy can be used only for applications that can tolerate data inconsistency, to some extent.

Other applications, though, such as on-line trading systems, cannot tolerate such inconsistencies. In this case, the use of strong consistency mechanisms, also known as invalidation mechanisms, is required. There are two types of invalidation mechanisms: proactive and reactive. In *proactive* invalidation, when an item is updated, the data source sends to the relevant caches an invalidation request, directing them to remove such cached item. This form of invalidation is very common in database systems (e.g. MySQL).

In *reactive* invalidation, which is commonly used in web systems, if the cache contains the content upon a request arrival, then the cache sends a conditional request (If-None-Match) to the origin server, which then replies either with an HTTP 304 response NOT MODIFIED, if the cached item matches with the corresponding data item stored in the content server (cache hit), or with the full data response, if the cached item is stale (cache miss).

Then, we have a cache hit only if the requested item is found in the cache and it is not stale. Nevertheless, surprisingly enough, the impact of invalidation mechanisms on the LRU cache hit rate has not been studied in the literature. The goal of this article is to address this issue.

Accordingly, the contribution of this paper is to extend the existing models of LRU caches, with the aim of evaluating the cache hit probability in presence of both request and invalidation events, modeled as renewal processes. Then we use the extended model to derive insights on the impact of invalidation patterns on cache performance and compare proactive and reactive strategies. We also evaluate the performance of LRU caching systems with frequently invalidated data in real world scenarios, using a dataset extracted from Wikipedia traffic. Finally, we evaluate the effect of invalidation in hierarchical caching.

2. Related work

Several caching replacement policies have been proposed in the literature, from simple FIFO, LRU, and LFU schemes to the recent Time To Live (TTL) based cache [1], SG-LRU cache [2] and many other ones. Among them, LRU is perhaps the most popular in real-world systems, given its implementation simplicity and very good performance in case of traffic with *temporal locality* [3]. For instance, MySQL, the world's most popular open source database, has a built-in feature called Query Cache that uses an LRU cache to store query results.[1] Reverse proxies, such as Varnish[2] (used by 5.2% of the most popular 10,000 sites in the web), memory object caching systems, such as Memcached[3] (used by Wikipedia, Flickr, LiveJournal, Craigslist), and several client-side caching proxies, such as the popular and historical Squid Proxy,[4] use LRU as the default solution for their memory replacement policies.

2.1. Performance evaluation of LRU

LRU caches have been studied for a long time, with models and approximations devised to calculate the cache hit probability [4,5]. Several years later, in 2002, Che et al. provided a very practical approach for LRU performance modeling called "Che's approximation" [6]. The model exploits several approximations to derive very simple formulas for computing the cache hit probability, given a certain popularity statistics of the contents and Poisson request inter-times. Despite its simplicity, Che's approximation achieves a very high accuracy, as recognized by many authors, even if a complete mathematical analysis of such model has been provided only 10 years after the original paper, by Fricker et al. in [7]. However, recently, it has been noted in [8] that the "Che's approximation" is essentially a re-phrasing of the Fagin asymptotic formula [4]. Thus, we also refer to it as "Characteristic Time Approximation".

Che's approximation paved the way for many research works that extend the original model to a broader set of cases, for instance to cope with different inter-time distributions and cache chains [9–11]. To the best of our knowledge, this is the first work that presents the effects of invalidation in LRU caching systems, considering proactive and reactive invalidation schemes, as described below.

2.2. Maintaining cache consistency with data invalidation

A main issue in cache systems is to guarantee data consistency, i.e. to prevent caches to serve stale data to clients. In particular, solutions can be classified in two main categories, providing a *weak* and a *strong* consistency of the data [12,13].

In case of weak consistency strategies, client queries might still be served with inconsistent (stale) data items, which can be stale up to a period of time or with a certain probability [14]. Weak consistency mechanisms are easily to implement, being usually based on a validity period included in a content header; this is for instance the case of Information Centric Networks (ICN) [15], which have recently renewed the interest in caching systems.

One such approach is the TTL cache strategy, where an item is invalidated after an expiration time, calculated from the start of cache placement [16,17].

Even though in some scenarios it may be acceptable to use stale data, there are other cases, such as databases or specific web applications (e.g., on line trading), in which strong consistency is necessary, i.e., the cache should never provide stale data. This can be done either with a *proactive approach*, in which the data source pushes a notification to the cache, signaling a data changes and triggering the cache to clean the changed data item (as it occurs in MySQL), or with a *reactive approach*, where it is up to the cache to contact the server for checking the consistency of the stored data

[1] http://dev.mysql.com/doc/refman/5.7/en/query-cache-status-and-maintenance.html.
[2] https://varnish-cache.org/.
[3] https://memcached.org/.
[4] http://www.squid-cache.org/.