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*

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

Email addresses: andrea.detti@uniroma2.it (Andrea Detti), lorenzo.bracciale@uniroma2.it (Lorenzo Bracciale), pierpaolo.loreti@uniroma2.it (Pierpaolo Loreti), blefari@uniroma2.it (Nicola Blefari Melazzi*)

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.
Key words: caching, invalidation, LRU, Wikipedia

---

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 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%

[1] `http://dev.mysql.com/doc/refman/5.7/en/query-cache-status-and-maintenance.html`

[2] `https://varnish-cache.org/`

of the most popular 10000 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] [10] [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

[3] `https://memcached.org/`

[4] `http://www.squid-cache.org/`

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 at each served request (this is the case of web browser caching) [18].

2.3. Real-world traffic models

Since the Shenker et al. seminal work [3] characterizing the popularity of web content and the related implications on caching systems, traffic demand has been widely studied, to characterize the different types of traffic [19], to assess their impact on caches [20], and to design data-driven caching strategies [21].

However, the modeling of invalidation patterns and of their relationship with request patterns have not received the same attention, at least in the world of caching. In [22] and [23], authors performed experiments and proposed models to estimate the frequency of updating of web pages, with the goal of increasing the performance of web crawlers, by optimizing the polling intervals used to check if a page has been changed or not. However, these papers do not analyze the relationship between requests and invalidation patterns, which instead plays a significant role for LRU performance, according to our analysis in this paper.

To the best of our knowledge, this is the first work that derives statistics of invalidation and correlates them with popularity, in order to assess the LRU performance with invalidation in real world applications, such as Wikipedia.

3. Analytical Model of LRU Caching with Invalidation

Let us consider an LRU cache able to store up to C items taken from an universe set of M distinct items. We consider negligible round trip times between cache and data sources. Under these assumptions, we model the cache operation as follows.

According to the LRU policy, when a request for the item n arrives, the related item is stored or moved to the top of the LRU memory stack. When requests for other objects arrive, the LRU policy tends to move item n towards the bottom of the stack. If the item overtakes the last position C of the stack, it is removed from the cache and we call this event *eviction*.

A cache *hit* occurs if, when a request arrives, the related object is contained in the cache, otherwise a cache *miss* takes place.

In what follows, we derive the cache hit probability for an LRU cache with invalidation, first for the case of Poisson request and invalidation processes, and then for generic renewal processes.

3.1. Characteristic time approximation

The characteristic time approximation, also known as "Che's approximation" [6] [24][5], is a simple yet accurate model for computing the cache hit probability of an LRU cache.

Assuming a Poisson arrival process R_n for the requests of object n, whose inter-arrival frequency is $\lambda_r^{(n)}$, then the cache hit probability h_n for that item can be calculated as:

$$h_n \approx 1 - e^{-\lambda_r^{(n)} t_c} \tag{1}$$

where t_c is the *eviction time*, defined as the time elapsing between the instant of time when item n is inserted at the top of the LRU stack after a request arrival, and the instant of time when the item is removed from the cache as a consequence of an LRU eviction,

[5] In 1977 R. Fagin in [4] provided an in-depth miss rate analysis of an LRU cache. The paper [25] provides the connection between Fagins asymptotic results on the LRU cache and the characteristic time (CT) approximation introduced by Che et al in [6], providing a strong theoretical underpinning for the latter.

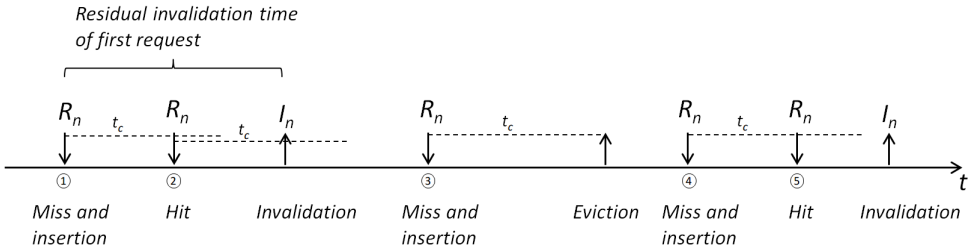


Figure 1: Cache hit and miss for item *n*, the dash line is the eviction time t_c

under the assumption that no other request for that item occurs in the meantime. It follows that a cache hit occurs if the next request arrives before t_c.

Clearly t_c is a random variable that also depends on the specific item n. However, Che's approximation states that t_c can be considered nearly constant for large C. The value of t_c can be obtained by calculating the unique root of the equation:

$$\sum_{n=1}^{M} \left(1 - e^{-\lambda_R^{(n)} t_c}\right) = C \tag{2}$$

This equation can be justified by using different analytical approaches; we use the one proposed in [9]. Due to the PASTA property, the *presence probability* P_n of finding the object n in the cache at a generic time is equal to the probability of finding the object in the cache at a request arrival, i.e. $P_n = h_n$.

Considering B, the random variable modeling the cache occupancy, the sum of all presence probabilities for all the objects (left-hand part) is equal to the average occupation of the cache $E[B]$, which, in case of an LRU cache without invalidation, is equal to its capacity C, being the cache always full.

Despite its simplicity, "Che's approximation" is particularly accurate, even if some arguments are just supported by intuition in the original paper describing it. Later works such as [7] provide a more rigorous argumentation of why this approximation works and capture so well the dynamic of LRU cache.

3.2. Modeling Invalidation

To cope with invalidation, let us introduce another process (I_n) that models the arrival of the invalidation events of the nth object.

Differently from the previous case, we have that an object n can be removed from the cache *either* because of invalidation *or* because of the eviction. It follows that a cache hit occurs if the next request of the process R_n arrives before the eviction time t_c and before the next invalidation time of the process I_n; otherwise a cache miss takes place. For instance, in figure 1 we have a first request arriving to the cache and whose element is consequently inserted at the top of the LRU stack. The second request gets a cache hit because it arrives before both the next invalidation event and the expiration of the eviction time t_c. The third request experiences a cache miss since it arrives after an invalidation event; the fourth request gets a cache miss because it arrives after the expiration of the eviction time.

An invalidation event occurring when an item is stored in the cache implies two different consequences, depending on the invalidation scheme. In case of proactive invalidation, the stale item is immediately removed from the cache. In case of reactive invalidation, the stale item will be removed only at the next eviction or request arrival, and not immediately. A cache hit occurs when a request finds a valid (non-stale) data in the cache, otherwise a cache miss occurs.

3.3. Request and Invalidation with exponential distribution

We adapted the "Che model" for the case of invalidation, where invalidation events for item n follow an exponential distribution with rate $\lambda_I^{(n)}$. In this case we can calculate the cache hit probability of item n as:

$$h_n = \left[1 - e^{-\left(\lambda_I^{(n)} + \lambda_R^{(n)}\right) t_c}\right] \frac{\lambda_R^{(n)}}{\lambda_R^{(n)} + \lambda_I^{(n)}} \tag{3}$$

This equation can be derived using probabilistic arguments, which we are going to discuss in section 3.3.3. However, an intuitive explanation is that it represents the probability that there is an event (request or invalidation) arrival before t_c, weighted by the probability that this event is a request.

To calculate the eviction time t_c, we can use exactly the same arguments that led to eq. 2, hence deriving t_c as the unique root the following equation,

$$\sum_{n=1}^{M} P_n = E[B] \tag{4}$$

Differently from the case without invalidation, the presence probability P_n is generally different by the cache hit probability h_n. For its computation we use the approach proposed in [9][10] for which the presence probability P_n can be expressed as the average time $E[S_n]$ spent in the cache by the nth item between two subsequent requests, normalized with respect to the average request inter-time:

$$P_n = \frac{E[S_n]}{1/\lambda_R^{(n)}} \tag{5}$$

The computation of $E[S_n]$, of the average cache occupancy $E[B]$ and, in turn, of t_c (eq. 4) are different for the cases of proactive and reactive invalidation schemes, as discussed in the following subsections.

3.3.1. Eviction time for reactive invalidation

We recall that, for the reactive scheme, objects are removed from the cache only because of the eviction (t_c expiration). Invalidation events make objects not valid anymore, but do not remove them from the cache.

To model this behavior, let us consider two consecutive requests, whose inter-time is t. At the arrival of the first request, the object is placed at the top of the LRU stack and remains in the cache for the whole inter-time period t if $t \leq t_c$; otherwise, it remains in the cache for t_c seconds, then it is removed from the cache until it is reinserted again at the arrival of the second request.

It follows that the average time $E[S_n]$ spent by an object in the cache between two requests can be written as the expected value of the minimum between the request inter-time random variable and the constant t_c:

$$E[S_n] = \int_{t=0}^{t_c} e^{-\lambda_R^{(n)} t} dt = \frac{1 - e^{-\lambda_R^{(n)} t_c}}{\lambda_R^{(n)}} \tag{6}$$

Regarding the average cache occupancy, since invalidation events do not remove the objects from the cache, the cache is always full and thus $E[B] = C$. Consequently, t_c can be derived by eq. 4 re-written as:

$$\sum_{n=1}^{M} \left(1 - e^{-\lambda_R^{(n)} t_c}\right) = C \tag{7}$$

3.3.2. Eviction time for proactive invalidation

In case of proactive invalidation, either eviction and invalidation events remove objects from the cache. This strategy implies the following behavior.

Let us consider two consecutive requests for which: the inter-time is t, the first request arrives at time t_r and the *residual invalidation time* between t_r and the next invalidation time is t_i. At the arrival time t_r the object is placed at the top of the LRU stack and remains in the cache for the whole inter-time period t if $t \leq \min(t_c, t_i)$; otherwise it is removed from the cache after a period equal to $\min(t_c, t_i)$, until it is re-inserted again at the next request arrival.

It follows that the average time $E[S_n]$ can be written as the expected value of the minimum among the request inter-time random variable, the residual invalidation time random variable and the constant t_c. Since the invalidation process is exponentially distributed, the residual time t_i random variable is exponentially distributed as well, with invalidation rate equal to $\lambda_I^{(n)}$, therefore:

$$E[S_n] = \int_0^{t_c} e^{-\left(\lambda_R^{(n)}+\lambda_I^{(n)}\right)t} dt = \frac{1 - e^{-\left(\lambda_R^{(n)}+\lambda_I^{(n)}\right)t_c}}{\lambda_R^{(n)} + \lambda_I^{(n)}} \tag{8}$$

Let us now discuss the computation of the average cache occupancy $E[B]$, which in case of proactive invalidation gets more complicated, since the cache is not always full. For analytic tractability, we resort to an approximation that uses the lower of the two following upper bounds. A first upper bound of $E[B]$ is clearly the cache capacity C. Another upper bound of $E[B]$ is the average occupancy that the cache would have if no eviction occurs or $t_c \to \infty$. Indeed, evictions limit the growing of the cache occupancy. In this latter case the average cache occupancy can be readily written as:

$$\lim_{t_c \to \infty} E[B] = \lim_{t_c \to \infty} \sum_{n=1}^{M} P_n = \sum_{n=1}^{M} \frac{\lambda_R^{(n)}}{\lambda_R^{(n)} + \lambda_I^{(n)}} \tag{9}$$

Consequently, the average cache occupancy can be written as:

$$E[B] \approx \min\left(C, \sum_{n=1}^{M} \frac{\lambda_R^{(n)}}{\lambda_R^{(n)} + \lambda_I^{(n)}}\right) \tag{10}$$

and to derive t_c eq. 4 can be rewritten as:

$$\sum_{n=1}^{M} \left[\frac{\lambda_R^{(n)}}{\lambda_R^{(n)} + \lambda_I^{(n)}} \left(1 - e^{-(\lambda_R^{(n)} + \lambda_R^{(n)}) t_c} \right) \right] = E[B] \tag{11}$$

We observe that the approximation 10 is extremely tight in practical cases of interest, for which the cache size is much lower than the number of objects, e.g., some orders of magnitude lower, and the invalidation rate is lower or in the order of the request rate. Indeed, in these cases the cache is practically always full and eq. 10 returns the cache capacity C, thus providing a negligible error. From some simulation measurements we observed that the maximum approximation error takes place at the discontinuity point, i.e. when $\sum_{n=1}^{M} P_n(t_c \to \infty) = C$. However, this is not a realistic scenario. Indeed, for realistic values of request and invalidation rates, this condition happens when the cache size is in the order of the data set.

3.3.3. Invalidation with generic distributions

In this section, we derive the cache hit probability in case of request and invalidation renewal processes, with generic inter-time distribution, by extending the previous exponential results.

We model the request and invalidation events as two independent *renewal processes*. The specific nth item, with $n \in \{1, 2, ..., M\}$, is characterized by a stationary request process whose inter-times between subsequent requests are independent and identically distributed (i.i.d.) random variables, and their general CDF, PDF, average frequency and standard deviation are $F_R^{(n)}(t)$, $f_R^{(n)}(t)$, $\lambda_R^{(n)}$, $\sigma_R^{(n)}$, respectively. Similarly, the nth item is also characterized by a stationary invalidation process, whose inter-times between subsequent invalidations are i.i.d. random variables, and their general CDF, PDF, average frequency and standard deviation are $F_I^{(n)}(t)$, $f_I^{(n)}(t)$,$\lambda_I^{(n)}$, $\sigma_I^{(n)}$, respectively.

To compute the cache hit probability, let us consider a request whose arrival time is t_r (e.g. first request of fig. 1). The next request of the same object will be a cache hit if the request inter-time t is lower than both i) the eviction time t_c and ii) the residual invalidation time, from t_r to the next invalidation event. This is for instance the case of the second request of fig. 1, whereas the third and forth request do not verify this

condition. It follows that the cache hit probability can be evaluated as the probability that the request inter-time is less than the minimum between the residual invalidation time and the eviction time t_c, i.e.:

$$h_n = \int_0^{t_c} f_R^{(n)}(t) \left(1 - \overline{F_I}^{(n)}(t)\right) dt \tag{12}$$

where $\overline{F_I}^{(n)}(t)$ is the CDF of the residual invalidation time. Since the invalidation and request processes are independent from each other, a generic request can happen at any time during the invalidation process. Thus the CDF of residual invalidation time can be written as [26]:

$$\overline{F_I}^{(n)}(t) = \frac{\int_0^t 1 - F_I^{(n)}(x)dx}{1/\lambda_I^{(n)}} \tag{13}$$

To calculate the eviction time t_c of reactive and proactive schemes we reuse eq. 4 and the same approach used for the exponential case, as it follows.

3.3.4. Eviction time for reactive invalidation

For reactive invalidation, the average time spent in the cache between two subsequent requests $E[S_n]$ can be written as the expected value of the minimum between the request inter-time and the eviction time t_c, i.e.

$$E[S_n] = \int_0^{t_c} \left(1 - F_R^{(n)}(t)\right) dt \tag{14}$$

It follows that to derive t_c we can solve the following equation:

$$\sum_{n=1}^{M} \frac{\int_0^{t_c} \left(1 - F_R^{(n)}(t)\right) dt}{1/\lambda_R^{(n)}} = C \tag{15}$$

3.3.5. Eviction time for proactive invalidation

For proactive invalidation, the average time $E[S_n]$ can be written as the expected value of the minimum between the request inter-time, the residual invalidation time and the eviction time t_c:

$$E[S_n] = \int_0^{t_c} \left(1 - F_R^{(n)}(t)\right) \left(1 - \overline{F_I}^{(n)}(t)\right) dt \tag{16}$$

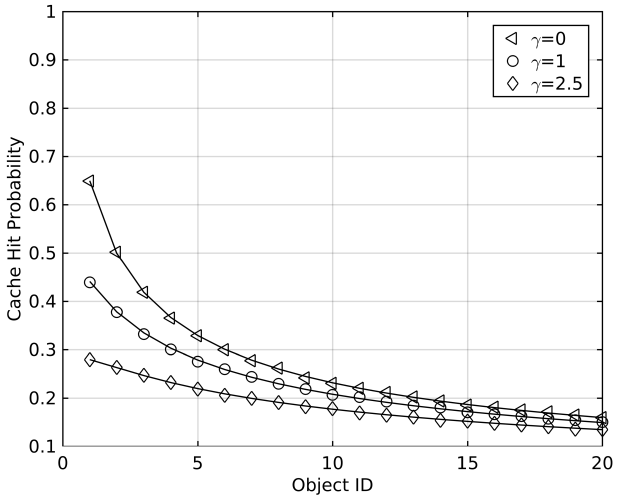


Figure 2: Modeling (lines) and simulation (markers) results of the per-object cache hit probability (h_n) versus the object ID, R_n exp, I_n exp, different values of the normalized invalidation rate γ, proactive invalidation.

For the average cache occupancy $E[B]$ we reuse the approximation of the exponential case, for which it is equal to the minimum between C and the average occupancy in case of $t_c \to \infty$. Therefore,

$$E[B] \approx \min\left(C, \sum_{n=1}^{M} \frac{\int_0^{\infty}\left(1-F_R^{(n)}(t)\right)\left(1-\overline{F_I}^{(n)}(t)\right)dt}{1/\lambda_R^{(n)}}\right) \tag{17}$$

It follows that to derive t_c we can rewrite eq. 4 as follows:

$$\sum_{n=1}^{M} \frac{\int_0^{t_c}\left(1-F_R^{(n)}(t)\right)\left(1-\overline{F_I}^{(n)}(t)\right)dt}{1/\lambda_R^{(n)}} = E[B] \tag{18}$$

4. Numerical Results

In this section, we assess the tightness of our model through simulations and derive some insights about the impact of invalidation on caching performances. Then we present the effect of invalidation in a real-world scenario using the statistics of

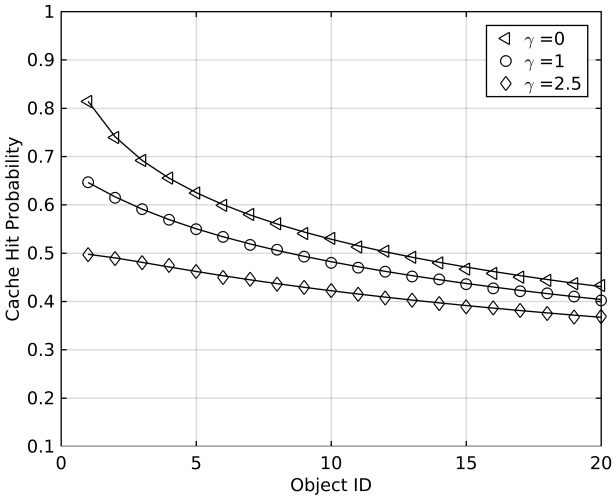


Figure 3: Modeling (lines) and simulation (markers) results of the per-object cache hit probability (h_n) versus the object ID, R_n logn $CV_r = 4$, I_n exp, different values of the normalized invalidation rate γ, proactive invalidation.

the Wikipedia page views and revisions, and finally discuss the case of hierarchical caching.

4.1. Model evaluation

We consider a cache size with $C = 100$ items and a universe of contents composed by $M = 10000$ ordered objects assuming that, without loss of generality, objects with lower IDs are more popular than objects with higher IDs. The object's popularity is modeled with a Zipf distribution, i.e., the request rate of the nth object is $\lambda_R^{(n)} = \frac{1/n^\alpha}{\sum_n 1/n^\alpha}\lambda_R^{tot}$ where $\alpha = 0.6$ is the Zipf shape factor and λ_R^{tot} is the overall request rate.

To the best of our knowledge, realistic models of the invalidation processes are not fully investigated in literature, thus we make some assumptions on their shape, to proof the validity of our model and to derive some general conclusions. We model the distribution of invalidation rates of the different objects with a Zipf, in which the nth object invalidation rate is $\lambda_I^{(n)} = \frac{1/n^\alpha}{\sum_n 1/n^\alpha}\lambda_I^{tot}$.

Thus, we are considering cases for which there is a relationship between the popularity of an object and its invalidation (i.e. update) rate: the nth object more frequently

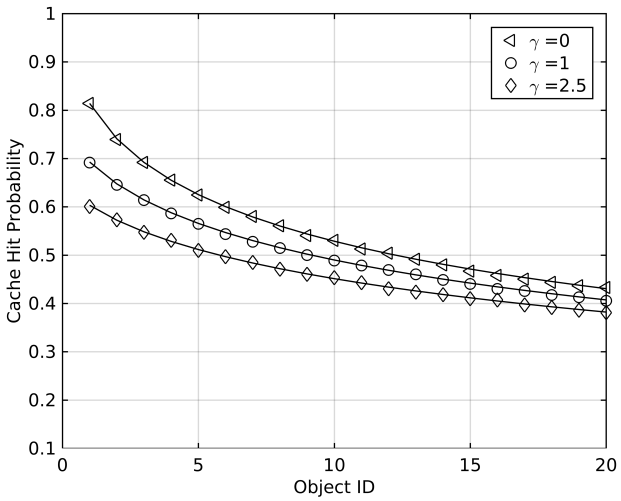


Figure 4: Modeling (lines) and simulation (markers) results of the per-object cache hit probability (h_n) versus the object ID, R_n logn $CV_r = 4$, I_n logn $CV_i = 4$, different values of the normalized invalidation rate γ, proactive invalidation.

requested is also the nth object more frequently invalidated. We also considered a case (fig. 5) where the popularity and the invalidation rates are not correlated, obtained by scrambling the order of the invalidation rates.

Fig. 2 reports the per-object cache hit probability evaluated with the analytical model (eq. 12) and with simulations, assuming that both invalidations and requests are exponentially distributed. We simulated the system for different values of the *normalized invalidation rate* $\gamma = \lambda_I^{tot}/\lambda_R^{tot}$ ($\gamma = 0$ means no invalidation). Only the first 20 objects are shown to avoid cluttering the figure. Solid lines are the modeling results, while markers are simulation results.

Figures 3 and 4 present the same performance, with different request and invalidation distributions. In fig. 3 the request process is modeled with a Lognormal distribution with coefficient of variation (CV) equal to 4, reproducing request streams with temporal locality [27]. In fig. 4 both request and invalidation streams present temporal locality and are modeled with two Lognormal distributions with $CV = 4$.

In all cases, the simulation results (markers) match the analytic ones (solid lines).

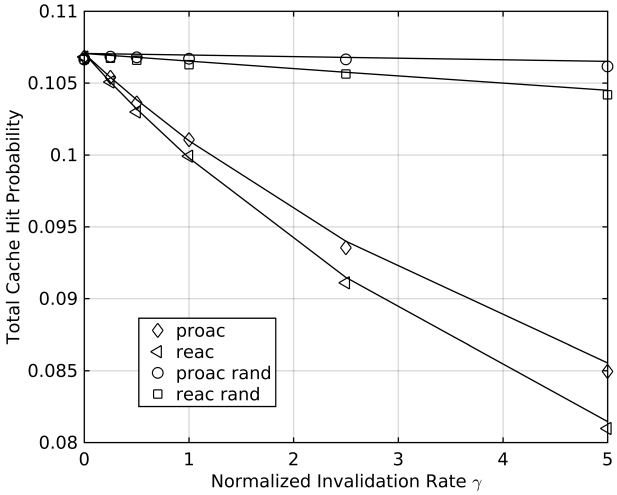


Figure 5: Modeling (lines) and simulation (markers) results of the per-object cache hit probability h_n versus the normalized invalidation rate γ, for different invalidation mechanisms.

This observation holds true for the other figures as well. Also, we notice that the increase of the normalized invalidation rate (γ) leads to a decrease of the per-item cache hit probability, as expected.

Similarly, in fig. 5 we show the total cache hit probability vs. the normalized invalidation rate, for both proactive and reactive invalidation mechanisms. It is shown that the total cache hit probability decreases at the increase of the normalized invalidation rate. In addition, we note that the proactive invalidation scheme provides better performance than the reactive one because it does not waste cache space with stale data.

This is explained in fig. 6, which illustrates the eviction time (t_c) vs. the normalized invalidation rate for both the proactive and the reactive invalidation schemes.

More specifically, it is shown in this figure that when the reactive scheme is used, the eviction time is constant and independent from the invalidation rate, while when the proactive scheme is used, the eviction time increases as a consequence of the greater availability of free spaces caused by the invalidation.

In fig. 5 we also consider a case in which the Zipf invalidation rates are randomly

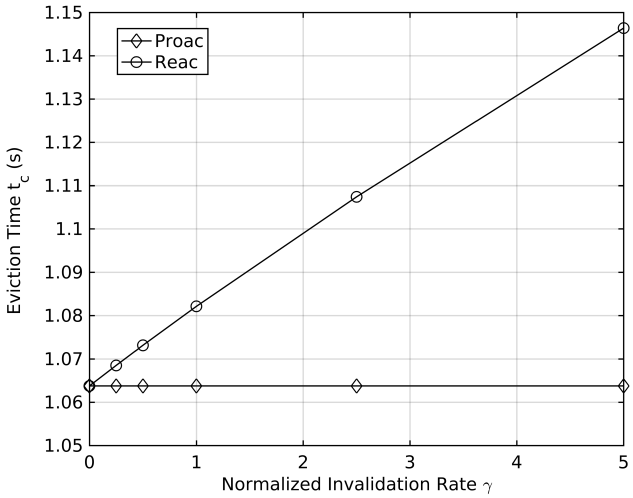


Figure 6: Modeling results of eviction time vs normalized invalidation rate, for proactive and reactive invalidations, R_n logn with $CV_r = 4$, I_n exp

distributed among the item IDs, thus decoupling popularity from invalidation rates. In this scenario (labeled "rand"), the effect of invalidation on caching performance is minor. Therefore, we conclude that invalidation leads to a non-negligible reduction of caching performance only when the invalidation/update events are frequent for popular items.

Fig. 7 presents the effect of the temporal locality by assuming Lognormal distributions for both request and invalidation inter-times. More specifically, in this figure we plot the total cache hit probability vs. the coefficient of variation of the requests streams (CV_r) for different coefficients of variation of the invalidations streams (CV_i). The higher the *CV* value, the higher the temporal locality [28]. It is well known that temporal locality in requests brings about a relevant improvement in terms of cache hit probability in an LRU cache. We note that the temporal locality in the invalidation process leads to a similar benefit as well, although much smaller.

This behavior can be explained with an example. Let us consider two invalidation events. If these events occur in the middle of two different request inter-time periods, these events cause two cache misses. If these events are so temporally close as to occur

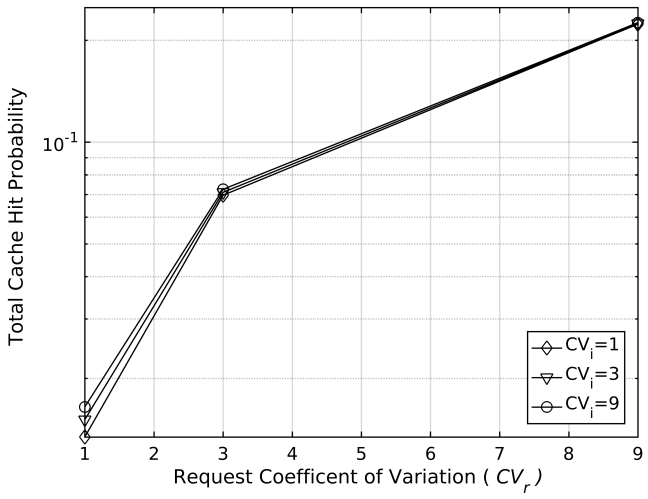


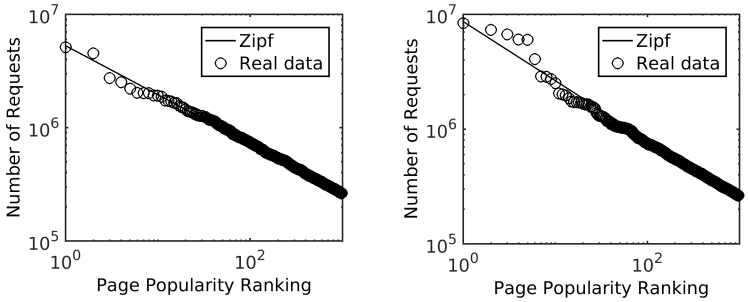
Figure 7: Modeling results of total cache hit probability vs. coefficient of variation, normalized invalidation rate $\gamma = 1$, R_n logn, I_n logn

in the middle of the same request inter-time period, both events could cause at most a single cache miss. Consequently, the more the invalidation events are close to each other (higher locality), the lower is the number of induced miss events and the higher is the cache hit probability. Finally, we point out that this effect of temporal locality can be observed also in figs. 2,3,4; indeed, we are inserting temporal locality, first in the requests (fig.3) and then in the invalidations (fig. 4).

4.2. Analysis with real-world data: the Wikipedia case

To verify the impact of invalidation in a real world application, we apply LRU caching to Wikipedia data, which currently is the fifth most visited website in the world, according to Alexa Rank. We analyzed views and revisions of the top 1000 most popular pages of Wikipedia during two months: September 2015 and September 2016. Each view is considered as a data request and each revision as an invalidation event. We used the Wikipedia REST APIs[6] to obtain the data describing page views and revisions.

[6] `https://wikimedia.org/api/rest_v1/`



(a) Wikipedia views September 2015 (b) Wikipedia views September 2016

Figure 8: Number of Wikipedia page views sorted by page popularity. Fitting with a Zipf distribution whose shape parameters are $s = 0.432$ for 2015, and $s = 0.515$ for 2016

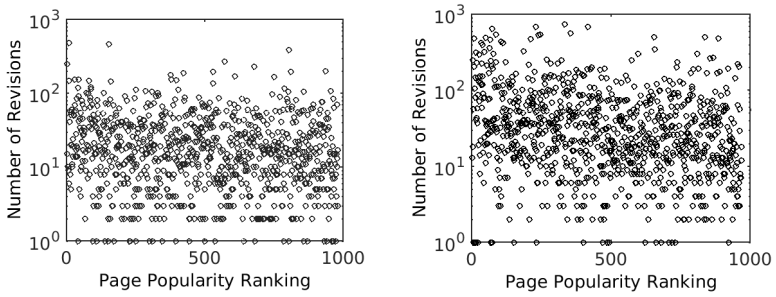
Fig. 8 reports the number of views (or popularity) of the Wikipedia pages sorted by their popularity, i.e. the xth value is the number of views of the xth most viewed page. The resulting distribution follows a Zipf law, where the shape parameters that minimize the RMSE are respectively $s = 0.432$ for September 2015 and $s = 0.515$ for September 2016.

Figure 9 reports the number of revisions for each page in the same time periods, sorting the pages on the x-axis by their popularity, i.e. with the same order of fig. 8. Such a scattered behavior clearly shows the lack of any apparent correlation between the rank a page has in terms of number of views and the rank it has in terms of number of revisions. In other words, there is no relationship between the popularity of an object and its invalidation rate, such as it occurs in the "rand" case of fig. 5.

Figure 10(a) shows the number of revisions in Sept. 2016 but, differently from fig. 9, we used a page revision ranking in x-axis, i.e., the xth value is the number of revisions of the xth most revised page. Differently from the page views, this curve shows a *geometric law* rather than a Zipf one. Specifically, the geometric distribution provides a good level of accuracy if we partition the items in three groups and fit them by using three different values of the parameter p.

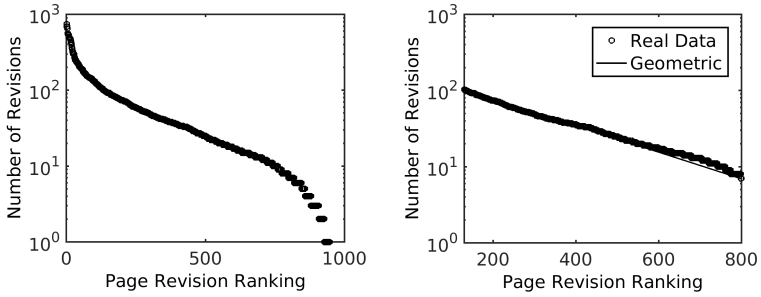
Figure 10(b) provides a detail of objects ranked from 130 to 800 (i.e. the central group of items) fitted with a geometric distribution with parameter $p = 4.013 \times 10^{-3}$.

Now we evaluate the effects of invalidation in LRU caches using these real world



(a) Sept. 2015 (b) Sept. 2016

Figure 9: Number of revisions for the top 1000 most popular Wikipedia pages. Pages sorted by number of views.



(a) Wikipedia revisions September 2016 (b) Fitting with a geometric distribution

Figure 10: Wikipedia revision of 1000 most popular pages (sept 2016), sorted by number of revisions. All pages are shown in 10(a), while in 10(b) we consider the pages ranked between 130 and 800 and fit them with a geometric distribution.

data. In particular, we consider a universe of objects whose size is $M = 10000$, a cache size of $C = 100$ items, object popularity distributed with a Zipf whose shape parameter is $s = 0.515$, invalidation rates distributed geometrically with $p = 4.013 \cdot 10^{-3}$; the popularity of a page and its invalidation rate are not correlated. We consider a proactive invalidation scheme and we model the inter-time between subsequent invalidation events with an exponentially distributed random variable, following [22] and [23]. As for the request inter-times, we used a Lognormal distribution with coefficient of variation (CV_r) equal to 4.

We first consider the case of a single server side LRU cache, receiving all request

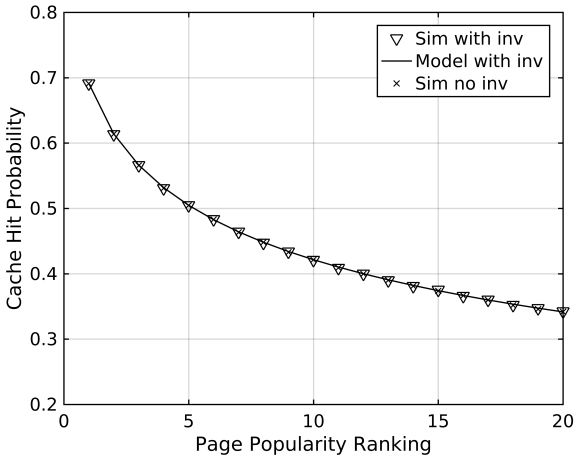


Figure 11: Modeling (lines) and simulation (markers) of the cache hit probability for the first 20 most popular Wikipedia pages, with and without proactive invalidation.

and invalidation events. The total requests rate is $187.77s^{-1}$ and the total invalidation rate is 0.021 s^{-1}, which correspond to the average rates of the Wikipedia views and revisions in Sept. 2016. Figure 11 shows the cache hit probability versus the object id using these rates. We report both the real case with invalidation and an ideal case without invalidation. As we can see, the impact of invalidation in such server side LRU cache is negligible, since the volume of invalidations is much smaller than the volume of requests coming from all over the world.

In figure 12, we show how the average cache hit probability changes when the request rate is reduced, hence the normalized invalidation rate γ is increased. The lowest $\gamma = 1.12 \cdot 10^{-4} (\approx 0.021/187.77)$ represents the previous case of a cache that receives all the Wikipedia requests and invalidations. The figure shows that we have to decrease a lot the requests rate, i.e., increase a lot γ, to produce a significant impact of the invalidation mechanism on the caching performance.

4.3. Hierarchical Caching

Many network scenarios imply the presence of more than one cache in the end-to-end path, where requests not served by a cache (i.e., cache misses) become the input of

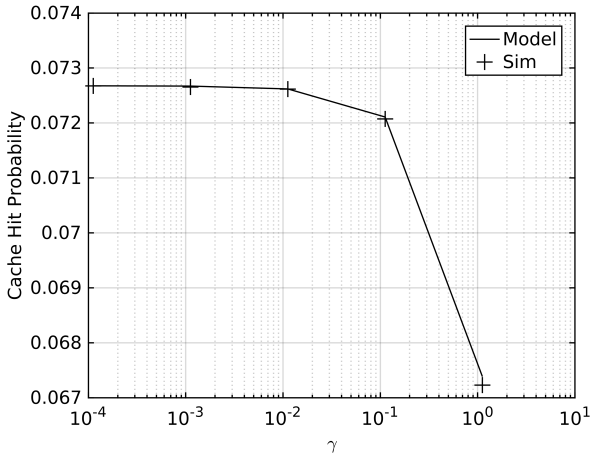


Figure 12: Total cache hit probability vs. the normalized invalidation rate γ

the next cache of the path. For instance, this is the case of browser and edge caches, or the case of cache networks such in Content Delivery Networks or Information Centric Networks.

In this section, we analyze the impact of invalidation in a basic cache network topology, namely a two level cache hierarchy formed by eleven caches: ten first level caches connected to a single second level cache. Even if this is a simple scenario, it allows deriving some general insights. The ten first-level caches are loaded with independent lognormal request streams, whose popularity follows a zipf distribution with $\alpha = 0.6$. Per-object invalidation rate is equal to the per-object request rate observed at a first level cache. Invalidation uses a proactive scheme. The analysis is performed by means of simulations, but our model is still exploited to understand the impact of invalidation also in the case of hierarchical caching.

Let us initially analyze the case without invalidation. It has been previously observed that the effectiveness of a cache in a hierarchy decreases when going up in the hierarchy [9][29]. Indeed, an LRU cache behaves like a filter of requests of popular items, since many of them experience a cache hit and thus do not reach to the next level cache. Consequently, the next level cache will see a request stream whose re-

lated object popularity is more flat and this implies a lower LRU cache hit probability, similarly to what occurs if we loaded the cache with a Zipf with a lower α parameter .

We quantify these observations by using the rank-frequency plot in fig. 13 (Object ID = rank) and the cache hit probability measurements reported in fig.14. Fig.13 shows that the request frequency seen by a first level cache perfectly fits a zipf distribution, i.e., it resembles a straight line, using a log-log scale. The request frequency seen by the second level cache (without invalidation) is more flat, due to the reduction of the number of requests of popular objects reaching the cache. Such a reduction of the popularity skewness has a dramatic impact on the cache hit probability of the second level cache, which is about 4.5 times smaller than the one provided by a first-level cache, as shown in fig.14.

The same figures also show what happens when we insert the invalidation process. Fig.14 shows that, as expected, the cache hit probability of a first-level cache decreases but, surprisingly, the cache hit probability of the second level cache increases. The reason of such a behavior is the result of two contrasting effects taking place at the second level cache:

1. the presence of invalidation events tend to *decrease* the cache hit probability and this reduction is proportional to the ratio γ between the invalidation rate and request rate (fig. 12);
2. since the cache hit probability of the first-level caches is lower, the filtering effect that they have on request streams is lower and the popularity observed at the second level tends to be more similar to the one observed at the first level. Therefore, the popularity is more skewed and this tends to *increase* the cache hit probability.

In the considered case, the second effect prevails over the first one and we have a cache hit improvement for the second level cache. Overall this implies that, albeit not reported, the cache hit probability provided by the whole system in presence and absence of invalidation are very close.

Finally, fig.15 shows how the ratio between invalidation and request rate (γ) changes for the different caches. In our simulations, each first level cache is loaded with a request rate that is equal to the invalidation rate. It follows that the value of γ seen by a

first level cache is equal to 1. We have ten first-level caches thus, considering the cache hierarchy as a closed system, the total γ is equal to 0.1, since the invalidation rate is independent by the number of caches.

What is more interesting is the shape of γ seen by the second level cache. We observe that if the cache hit probability of first-level caches was equal to zero, each request would arrive to the second level cache and, consequently, the value of γ would be equal to 0.1. The figure shows that by decreasing the object popularity (greater Object ID) the value of γ tends to such limit value of 0.1, since the related cache hit probability at the first level tends to zero. Conversely, for popular objects, the cache hit probability at first level is higher, thus the request rate arriving to the second level is lower and the γ of these objects is rather higher than 0.1.

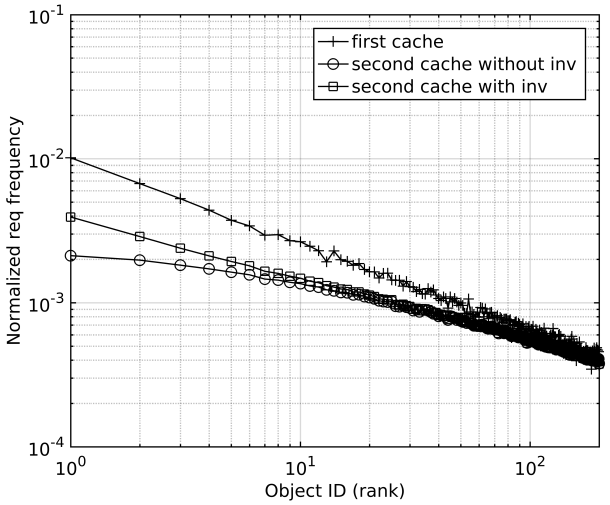


Figure 13: Rank-frequency plot of a two level cache hierarchy, with and without proactive invalidations, first level cache R_n logn with $CV_r = 4$, I_n exp, $\gamma = 1$

5. Conclusions

In this paper, we extended the literature models of LRU caches to include in the analysis the data invalidation case. Our model describes the cache hit probability for

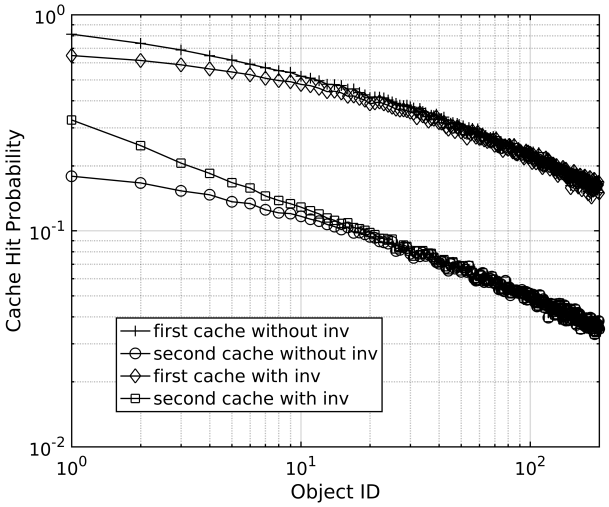


Figure 14: Cache hit probability plot of of a two level cache hierarchy, with and without proactive invalidations, first level cache R_n logn with $CV_r = 4$, I_n exp, $\gamma = 1$

request and invalidation renewal processes having generic distribution, and for reactive and proactive invalidation schemes.

The main result obtained with our model is that the invalidation worsens the cache performance only when the invalidation rate is close or higher than the request rate, at least in single, first-level, caches. This is not a common situation, as it does not occur often in practice; indeed the number of expected requests is usually much higher than the invalidation ones. Consequently, we conclude that LRU caching is a valuable caching strategy also in presence of invalidation events.

Another result worthy of note is that temporal locality both in the request and in the invalidation patterns improves the caching performance, although remarkable improvements only show up for temporal locality in the request pattern. Furthermore, proactive schemes provide limited benefit versus the reactive ones, at least in the observed absence of correlation between the popularity of an object and its invalidation rate.

Finally, we show that, in the case of hierarchical caching, even a limited effect of invalidation on first-level caches is sufficient to perceptibly affect the performance of

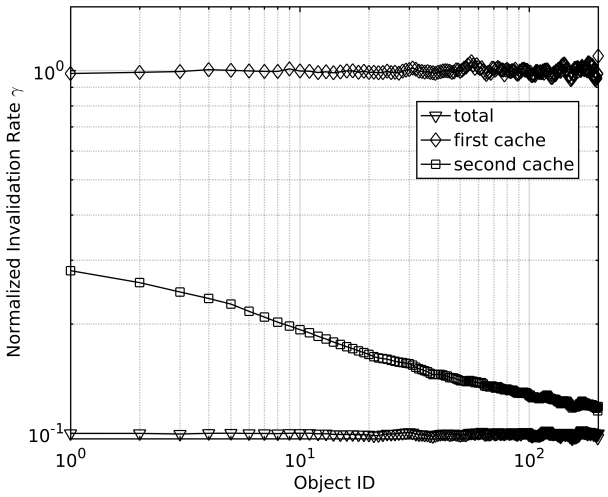


Figure 15: Ratio between invalidation and request rates, with proactive invalidations, first level cache R_n logn with $CV_r = 4$, I_n exp, $\gamma = 1$

second level/downstream caches.

Acknowledgement

This work was partly funded by the EU H2020 Bonvoyage and EU-JP H2020 ICN2020 projects.

References

[1] N. C. Fofack, P. Nain, G. Neglia, D. Towsley, Analysis of ttl-based cache networks, in: Performance Evaluation Methodologies and Tools (VALUETOOLS), 2012 6th International Conference on, IEEE, 2012, pp. 1–10.

[2] G. Hasslinger, K. Ntougias, F. Hasslinger, O. Hohlfeld, Performance evaluation for new web caching strategies combining lru with score based object selection, in: Teletraffic Congress (ITC 28), 2016 28th International, Vol. 1, IEEE, 2016, pp. 322–330.

[3] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, Web caching and zipf-like distributions: Evidence and implications, in: INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, Vol. 1, IEEE, 1999, pp. 126–134.

[4] R. Fagin, Asymptotic miss ratios over independent references, Journal of Computer and System Sciences 14 (2) (1977) 222 – 250. doi:https://doi.org/10.1016/S0022-0000(77)80014-7.
URL http://www.sciencedirect.com/science/article/pii/S0022000077800147

[5] P. Flajolet, D. Gardy, L. Thimonier, Birthday paradox, coupon collectors, caching algorithms and self-organizing search, Discrete Applied Mathematics 39 (3) (1992) 207 – 229. doi:https://doi.org/10.1016/0166-218X(92)90177-C.
URL http://www.sciencedirect.com/science/article/pii/0166218X9290177C

[6] H. Che, Y. Tung, Z. Wang, Hierarchical web caching systems: Modeling, design and experimental results, IEEE Journal on Selected Areas in Communications 20 (7) (2002) 1305–1314.

[7] C. Fricker, P. Robert, J. Roberts, A versatile and accurate approximation for lru cache performance, in: Proceedings of the 24th International Teletraffic Congress, ITC '12, International Teletraffic Congress, 2012, pp. 8:1–8:8.
URL http://dl.acm.org/citation.cfm?id=2414276.2414286

[8] C. Berthet, Approximation of LRU caches miss rate: Application to power-law popularities, CoRR abs/1705.10738.
URL http://arxiv.org/abs/1705.10738

[9] N. B. Melazzi, G. Bianchi, A. Caponi, A. Detti, A general, tractable and accurate model for a cascade of lru caches, IEEE Communications Letters 18 (5) (2014) 877–880.

[10] G. Bianchi, A. Detti, A. Caponi, N. Blefari Melazzi, Check before storing: what is the performance price of content integrity verification in lru caching?, ACM SIGCOMM Computer Communication Review 43 (3) (2013) 59–67.

[11] M. Garetto, E. Leonardi, V. Martina, A unified approach to the performance analysis of caching systems, ACM Transactions on Modeling and Performance Evaluation of Computing Systems 1 (3) (2016) 12.

[12] E. Nahum, A. Iyengar, R. Tewari, A. Shaikh, Web caching, consistency, and content distribution, in: The Practical Handbook of Internet Computing, Chapman and Hall/CRC, 2004.

[13] J. Cao, Y. Zhang, G. Cao, L. Xie, Data consistency for cooperative caching in mobile environments, Computer 40 (4).

[14] W. Li, E. Chan, D. Chen, S. Lu, Maintaining probabilistic consistency for frequently offline devices in mobile ad hoc networks, in: Distributed Computing Systems, 2009. ICDCS'09. 29th IEEE International Conference on, IEEE, 2009, pp. 215–222.

[15] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, R. L. Braynard, Networking named content, in: Proceedings of the 5th international conference on Emerging networking experiments and technologies, ACM, 2009, pp. 1–12.

[16] O. Bahat, A. M. Makowski, Measuring consistency in ttl-based caches, Perform. Eval. 62 (1-4) (2005) 439–455. doi:10.1016/j.peva.2005.07.015.
URL http://dx.doi.org/10.1016/j.peva.2005.07.015

[17] S. Alouf, N. C. Fofack, N. Nedkov, Performance models for hierarchy of caches: Application to modern dns caches, Performance Evaluation 97 (Supplement C) (2016) 57 – 82, performance Evaluation Methodologies and Tools: Selected Papers from VALUETOOLS 2013. doi:https://doi.org/10.1016/j.peva.2016.01.001.
URL http://www.sciencedirect.com/science/article/pii/S016653161600002X

[18] K. Fawaz, H. Artail, Dcim: Distributed cache invalidation method for maintaining cache consistency in wireless mobile networks, IEEE Transactions on Mobile Computing 12 (4) (2013) 680–693.

[19] J. Li, S. Ma, Characterization and modeling of video popularity, International Journal of Communication Systems 27 (11) (2014) 2604–2615. doi:10.1002/dac.2493.
URL http://dx.doi.org/10.1002/dac.2493

[20] M. Z. Shafiq, A. R. Khakpour, A. X. Liu, Characterizing caching workload of a large commercial content delivery network, in: Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on, IEEE, 2016, pp. 1–9.

[21] S. Li, J. Xu, M. van der Schaar, W. Li, Popularity-driven content caching, in: Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on, IEEE, 2016, pp. 1–9.

[22] J. Cho, H. Garcia-Molina, The evolution of the web and implications for an incremental crawler, Tech. rep., Stanford (1999).

[23] J. Cho, H. Garcia-Molina, Estimating frequency of change, ACM Transactions on Internet Technology (TOIT) 3 (3) (2003) 256–290.

[24] C. Fricker, P. Robert, J. Roberts, A versatile and accurate approximation for lru cache performance, in: Proceedings of the 24th International Teletraffic Congress, International Teletraffic Congress, 2012, p. 8.

[25] M. Dehghan, W. Chu, P. Nain, D. Towsley, Sharing lru cache resources among content providers: A utility-based approach, arXiv preprint arXiv:1702.01823.

[26] L. Kleinrock, Queueing Systems, Vol. I: Theory, Wiley Interscience, 1975.

[27] V. Almeida, A. Bestavros, M. Crovella, A. de Oliveira, Characterizing reference locality in the www, in: Fourth International Conference on Parallel and Distributed Information Systems, 1996, pp. 92–103. doi:10.1109/PDIS.1996.568672.

[28] R. Fonseca, V. Almeida, M. Crovella, B. Abrahao, On the intrinsic locality properties of web reference streams, in: INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies, Vol. 1, IEEE, 2003, pp. 448–458.

[29] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. Ng, V. Sekar, S. Shenker, Less pain, most of the gain: Incrementally deployable icn, in: ACM SIGCOMM Computer Communication Review, Vol. 43, ACM, 2013, pp. 147–158.