Access-time aware cache algorithms

Giovanni Neglia*, Damiano Carra†, Mingdong Feng‡, Vaishnav Janardhan‡,
Pietro Michiardi§ and Dimitra Tsigkari*
*Université Côte d'Azur, Inria, {giovanni.neglia, dimitra.tsigkari}@inria.fr
†University of Verona, damiano.carra@univr.it
‡Akamai Technologies, {mfeng, vjanardh}@akamai.com
§Eurecom, pietro.michiardi@eurecom.fr

***Abstract*—Most of the caching algorithms are oblivious to requests' timescale, but caching systems are capacity constrained and, in practical cases, the hit rate may be limited by the cache's impossibility to serve requests fast enough. In particular, the hard-disk access time can be the key factor capping cache performance. In this paper, we present a new cache replacement policy that takes advantage of a hierarchical caching architecture, and in particular of access-time difference between memory and disk. Our policy is optimal when requests follow the independent reference model, and significantly reduces the hard-disk load, as shown also by our realistic, trace-driven evaluation. Moreover, we show that our policy can be considered in a more general context, since it can be easily adapted to minimize any retrieval cost, as far as costs add over cache misses.**

I. INTRODUCTION

The hit probability is a well-known key metric for caching systems: this is the probability that a generic request for a given content will be served by the cache. Most of the existing literature implicitly assumes that a hit occurs if the content is stored in the cache at the moment of the request. In practice, however, in real caching systems the hit rate is often limited by the speed at which the cache can serve requests. In particular, Hard-Disk Drive (HDD) access times can be the key factor capping cache performance.

As an illustrative example, Figure 1 shows the percentage of CPU and HDD utilization, as reported by the operating system, over two days in the life of a generic caching server. As the amount of requests varies during the day, the resource utilization of the caching server varies as well: during peak hours, HDD utilization can exceed 95%. Such loads may cause the inability to serve a request even if the content is actually cached in the HDD, generating what we call "spurious misses." In case of a pool of cache servers, a solution based on dynamic load balancing may alleviate this problem by offloading the requests to another server. Nevertheless, this solution has its own drawbacks, because the rerouted queries are likely to generate misses at the new cache.

In this paper, we study if and how the RAM can be used to alleviate the HDD load, so that the cache can serve a higher rate of requests before query-rerouting becomes necessary.

The idea to take advantage of the RAM is not groundbreaking. Modern cache servers usually operate as hierarchical caches, where the most recently requested contents are stored also in the RAM: upon arrival of a new request, content is first looked up in the RAM; if not found, the lookup mechanism targets the HDD. Hence, the RAM "shields" the HDD from most of the requests. This RAM cache is often also called the "Hot Object Cache" using Squid web proxy terminology.

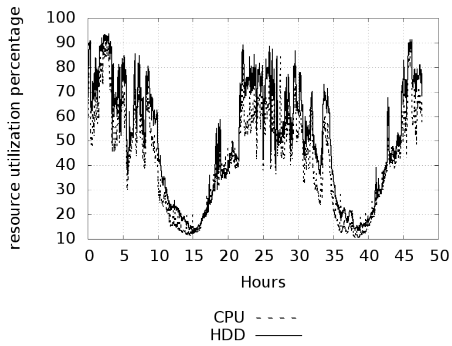

Fig. 1. Graph showing the CPU and HDD utilization percentage of a generic caching server.

The question we ask in this paper is: what is the optimal way to use the RAM? I.e., which content should be duplicated in the RAM to minimize the load on the HDD? We show that, if content popularities are known, the problem can be formulated as a knapsack problem. More importantly, we design a new dynamic replacement policy that, without requiring popularity information to be known, can implicitly solve our minimization problem. Our policy is a variant of q-LRU [1]: in q-LRU, after a cache miss, the content is stored in the cache with probability q and, if space is needed, the least recently used contents are evicted. We call our policy q_i-LRU, because we use a different probability q_i for each content i. The value q_i depends on the content size and takes into account the time needed to retrieve the content from the HDD. Simulation results on real content request traces from the Akamai's Content Delivery Network (CDN) [2] show that our policy achieves more than 80% load reduction on the HDD with an improvement between 10% and 20% in comparison to standard LRU.

While our paper is motivated by the specific problem to reduce the load on the HDD to avoid spurious misses, we observe that similar issues arise in any hierarchical storage system, where we want to use efficiently the fastest storage layers to minimize the overall retrieval time. In this sense, the possible future replacement of HDD by Solid State Drives

(SSD)[1] would not make our study obsolete. Moreover, our results do not depend on the specific function we are trying to minimize, but any retrieval cost represents a valid choice, as long as it is additive over different misses. For example, our policy q_i-LRU could be adapted to minimize the cache miss ratio, the traffic from upstream caches, etc.

The paper is organized as follows. In Section II we formalize the problem and illustrate the underlying assumptions. In Section III we present the policy q_i-LRU and prove its asymptotic optimality. We evaluate its performance under real-world traces in Section IV, and we show preliminary test results in Section V. In Section VI we discuss how q_i-LRU can be adapted to solve a variety of different cost minimization problems by simply changing the expression of the probabilities q_i. Related works are discussed in Section VII.

This article extends the previous conference version [3] in several respects: (i) all the proofs are included in appendices A, A, A, (ii) additional experimental results validate our model in Section IV, and (iii) additional results has been added in Section V, (iv) the applicability of q_i-LRU to the general retrieval cost minimization problem is shown in Section VI, and (v) the related work section has been extended.

II. Model

A. Hard Disk Service Time

Our study relies on some assumptions about the load imposed on the HDD by a set of requests. Consider a single file-read request for content i of size s_i. We call *service time* the time the HDD works just to provide content i to the operating system. Our first assumption is that the service time is a function only of content size s_i. We denote it as $T(s_i)$.[2] The second assumption is that service times are additive, i.e. let A be a set of contents, the total time the HDD works to provide the contents in A is equal to $\sum_{i \in A} T(s_i)$, independently of the specific time instants at which the requests are issued. Note that we are not assuming any specific service discipline for this set of requests: they could be served sequentially (e.g. in a FIFO or LIFO way) or in parallel (e.g. according to a generalized processor sharing).[3] What we require is that concurrent object requests do not interfere by increasing (or reducing) the total HDD service time. Our experiments in Section IV show that this assumption is a reasonable one and the model predicts very well the HDD load.

The analytical results we provide in Section III, which is the main contribution of our work, do not depend on a particular structure of the function $T(s_i)$. Here, we describe a specific form based on past research on HDD I/O throughput [4][5], and on our performance study of disk access time observed in caching servers. We will refer to this specific form later to clarify some properties of the optimal policy. Furthermore, we will use it in our experiments in Section IV.

[1]Note that, SSDs are not going to completely replace HDDs in the near future for large caches, because of their higher cost and the limited number of rewrites they can tolerate.

[2]If the service time is affected by significant random effects, then $T(s_i)$ can be interpreted as the expected service time for a content of size s_i.

[3]The specific service discipline would clearly have an effect on the time needed to retrieve a specific content.

TABLE I
SUMMARY OF THE VARIABLES USED FOR $T(s_i)$

Variable	Meaning	Typical Value
s_i	Size of object i	-
σ	Average seek time	$3.7 \cdot 10^{-3}$ s
ρ	Average rotation time	$3.0 \cdot 10^{-3}$ s
b	Block size	2.0 MB
σ_r	Seek time for read	$3.14 \cdot 10^{-9}$ s/MB
μ	Transfer bandwidth	157 MB/s
ϕ	Controller Overhead	$0.5 \cdot 10^{-3}$ s

Considering the mechanical structure of the HDD, every time a new read needs to be done, we have to wait for the reading arm to move across the cylinders, and for the platter to rotate on its axis. We call these two contributions the *average seek time* and *average rotation time*, and we denote them by σ and ρ respectively. Each file is divided into blocks, whose size b is a configuration parameter. If we read a file whose size is bigger than a block, then we need to wait for the average seek time and the average rotation time for each block.

Once the reading head has reached the beginning of a block, the time it takes to read the data depends on the *transfer speed* μ. Moreover, while reading a file, the reading arm needs to move across tracks and cylinders, so we need to add a contribution due to the *seek time for read*, σ_r, which depends on the size of the file. A last contribution is due to the *controller overhead*, ϕ, that introduces a constant delay.

Overall, the function that estimates the cost of reading a file from the hard disk is given by the following equation (see Table I for a summary of the variables used):

$$T(s_i) = (\sigma + \rho) \left\lceil \frac{s_i}{b} \right\rceil + \left(\frac{1}{\mu} + \sigma_r \right) s_i + \phi. \quad (1)$$

Based on our experience on real-life production systems, the last column of Table I shows the values of the different variables for a 10'000 RPM hard drive.

We have validated Equation (1) through an extensive measurement campaign for two different hard disk drives (10'000 RPM and 7'200 RPM). The results are shown in Figure 2. In the figure, we actually plot the quantity $T(s_i)/s_i$: in Section III, we will illustrate the key role played by this ratio. The estimated value of $T(s_i)/s_i$ has discontinuity points at the multiples of the block size b: in fact, as soon as the size of an object exceeds one of such values, the service time increases by an additional average seek time and an additional average rotation time. The points in the figures represent the output of our measurement campaign for a representative subset of sizes (in particular, for sizes close to the multiples of block size b, where the discontinuities occur). Each point is the average value for a given size over multiple reads. From the experiments, we conclude that the function $T(s_i)$ shown in Equation (1) is able to accurately estimate the cost of reading a file from the HDD. Moreover, in Section IV we compare the HDD load over time, measured as $\sum_{i \in A} T(s_i)$ over intervals of 30 seconds, with the actual load recorded by a real server (the details about the experimental setup and the traces are in Section IV): the results show a very good match between the load derived from the model and the actual load (see Fig. 8).

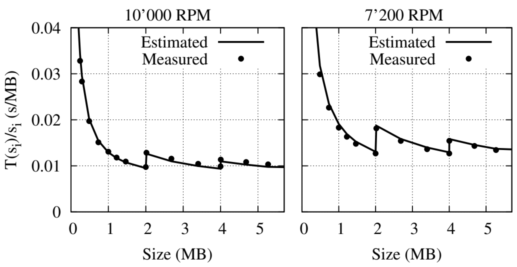

Fig. 2. Graph of the function $T(s_i)/s_i$.

B. Query Request Process

Let $\mathcal{N} = \{1, 2, \dots N\}$ denote the set of contents. For mathematical tractability, as done in most of the works in the literature (see Section VII), we assume that the requests follow the popular Independent Reference Model (IRM), where contents requests are independently drawn according to constant probabilities (see for example [6]). In particular, we consider the time-continuous version of the IRM: requests for content $i \in \mathcal{N}$ arrive according to a Poisson process with rate λ_i and the Poisson processes for different contents are independent. While the optimality results for our policy q_i-LRU are derived under such assumption, significant performance improvements are obtained also considering real request traces (see Section IV).

C. Problem Formulation

In general, the optimal operation of a hierarchical cache system would require to jointly manage the different storage units, and in particular to avoid to duplicate contents across multiple units. On the contrary, in the case of a RAM-HDD system, the problem is usually decoupled: the HDD caching policy is selected in order to maximize the main cache performance metric (e.g. hit ratio/rate), while a subset of the contents stored in the HDD can be duplicated in the RAM to optimize some other performance metric (e.g. the response time). The reason for duplicating contents in the RAM is twofold. First, contents present only in the RAM would be lost if the caching server is rebooted. Second, the global cache hit ratio/rate would not be significantly improved because the RAM accounts for a small percentage of the total storage available at the server. A consequence of such decoupling is that, at any time, the RAM stores a subset of the contents stored in the HDD, denoted by $\mathcal{M}_R$ and $\mathcal{M}_H$ respectively.[4] In our work, we consider the same decoupling principle. As a consequence, our policy is agnostic to the replacement policy implemented at the HDD (LRU, FIFO, Random, ...).

We now look at how the RAM reduces the HDD load. An incoming request can be for a content not present in the HDD (nor in the RAM because we consider $\mathcal{M}_R \subset \mathcal{M}_H$). In this case, the content will be retrieved by some other server in the CDN or by the authoritative content provider, and then stored or not in the HDD depending on the specific HDD cache policy. Note that the choice of the contents to be duplicated in the RAM plays no role here. Read/write operations can occur (e.g. to store the new content in the HDD), but they are not affected by the RAM replacement policy, that is the focus of this paper. We ignore then the corresponding costs. On the contrary, if an incoming request is for a content present in the HDD, the expected HDD service time depends on the set of contents $\mathcal{M}_R$ stored in the RAM. It is indeed equal to

$$\sum_{i \in \mathcal{M}_H \setminus \mathcal{M}_R} \frac{\lambda_i}{\sum_{j \in \mathcal{N}} \lambda_j} T(s_i) = \sum_{i \in \mathcal{M}_H} \frac{\lambda_i}{\sum_{j \in \mathcal{N}} \lambda_j} T(s_i) - \sum_{i \in \mathcal{M}_R} \frac{\lambda_i}{\sum_{j \in \mathcal{N}} \lambda_j} T(s_i), \quad (2)$$

because, under IRM, $\lambda_i / \sum_{j \in \mathcal{N}} \lambda_j$ is the probability that the next request is for content i, and the request will be served by the HDD only if content i is not duplicated in the RAM, i.e. only if $i \notin \mathcal{M}_R$.

Our purpose is to minimize the HDD service time under the constraint on the RAM size. This is equivalent to maximize the second term in Equation (2). By removing the constant $\sum_{j \in \mathcal{N}} \lambda_j$, we obtain then that the optimal possible choice for the subset $\mathcal{M}_R$ in a a RAM of capacity C is the solution of the following maximization problem:

$$\begin{aligned} \underset{\mathcal{M}_R \subset \mathcal{N}}{\text{maximize}} \quad & \sum_{i \in \mathcal{M}_R} \lambda_i T(s_i) \\ \text{subject to} \quad & \sum_{i \in \mathcal{M}_R} s_i \leq C. \end{aligned} \quad (3)$$

This is a knapsack problem, where $\lambda_i T(s_i)$ is the value of content/item i and s_i its weight. The knapsack problem is NP-hard. A natural, and historically the first, relaxation of the knapsack problem is the fractional knapsack problem (also called continuous knapsack problem). In this case, we accept fractional amounts of the contents to be stored in the RAM. Let $h_i \in [0, 1]$ be the fraction of content i to be put in the RAM, the fractional problem corresponding to problem (3) is:

$$\begin{aligned} \underset{h_1, \dots h_N}{\text{maximize}} \quad & \sum_{i=1}^{N} \lambda_i h_i T(s_i) \\ \text{subject to} \quad & \sum_{i=1}^{N} h_i s_i = C. \end{aligned} \quad (4)$$

From an algorithmic point of view, the following greedy algorithm is optimal for the fractional knapsack problem. Assume that all the items are sorted in decreasing order with respect to the profit per unit of size (i.e. $\lambda_i T(s_i)/s_i \geq \lambda_j T(s_j)/s_j$ for $i \leq j$). The algorithm finds the biggest index $\underline{c}$ for which the sum $\sum_{i=1}^{\underline{c}} s_i$ does not exceed the memory capacity. Finally, it stores the first $\underline{c}$ contents in the knapsack (in the RAM) as well as a fractional part of the content $\underline{c} + 1$ so that the RAM is filled up to its capacity. A simple variant of this greedy algorithm guarantees a $\frac{1}{2}$-approximation factor for the original knapsack problem [7, Theorem 2.5.4], but the greedy algorithm itself is a very good approximation

[4]Although it is theoretically possible that a content stored in the RAM and in the HDD may be evicted by the HDD earlier than by the RAM, these events can be neglected in practical settings. For example in the scenario considered in Section IV typical cache eviction times are a few minutes for the RAM and a few days for the HDD for all the cache policies considered.

algorithm for common instances of knapsack problems, as it can be justified by its good expected performance under random inputs [7, Section 14.4].

From a networking point of view, if we interpret h_i as the probability that content i is in the RAM,[5] then we recognize that the constraint in problem (4) corresponds to the usual constraint considered under the cache characteristic time approximation (CTA), first proposed in [8] and later rediscovered in [9]. Under CTA, the effect of the finite cache size is taken into account by imposing the expected cache occupancy for an unbounded TTL-cache [10] to have the form:

$$\sum_{i=1}^{N} h_i s_i = C. \tag{5}$$

The last remark connects our problem to the recent work in [11], where the authors use CTA to find optimal cache policies to solve the following problem:

$$\begin{aligned} \underset{h_1,\ldots h_N}{\text{maximize}} \quad & \sum_{i=1}^{N} U_i(h_i) \\ \text{subject to} \quad & \sum_{i=1}^{N} h_i s_i = C, \end{aligned} \tag{6}$$

where each $U_i(h_i)$ quantifies the utility of a cache hit for content i.[6] Results in [11] do not help us solve our problem (4) because their approach requires the functions $U_i(h_i)$ to be (i) known and (ii) strictly concave in h_i. On the contrary, in our case, content popularities (λ_i) are unknown[7] and, even if they were known, the functions $U_i(h_i)$ would be $\lambda_i h_i T(s_i)$ and then linear in h_i. Besides, deriving the cache policy that solves a given optimization problem, [11] also "reverse-engineers" existing policies (like LRU) to find which optimization problem they are implicitly solving. In Section III, we use a similar approach to study our policy.

After this general analysis of the problem, we are ready to introduce in the next section a new caching policy q_i-LRU that aims to solve problem (4), i.e. to store in the RAM the contents with the largest values $\lambda_i T(s_i)/s_i$ without the knowledge of content popularities λ_i, for $i = 1, \ldots N$.

III. The q_i-LRU policy

We start introducing our policy as a heuristic justified by an analogy with LRU.

Under IRM and the characteristic time approximation, if popularities λ_i are known, minimizing the miss throughput at a cache of capacity C corresponds to solving the following linear problem:

$$\begin{aligned} \underset{h_1,\ldots h_N}{\text{maximize}} \quad & \sum_{i=1}^{N} \lambda_i h_i s_i \\ \text{subject to} \quad & \sum_{i=1}^{N} h_i s_i = C \end{aligned} \tag{7}$$

The optimal solution is analogous to what discussed for problem (4): set hit probabilities to one for the k most popular contents, a hit probability smaller than one for the $(k+1)$-th most popular content, and hit probabilities to zero for all the other contents. The value of k is determined by the RAM size.

Now, it is well known that, from a practical perspective, the traditional LRU policy behaves extremely well, despite content popularity dynamics. LRU is a good heuristic for problem (7): it implicitly selects and stores in the cache the contents with the largest values of λ_i, even when popularities λ_i are actually unknown.

Recall that our purpose is to store the contents with the largest values $\lambda_i T(s_i)/s_i$: then, the analogy between the two problems suggests us to bias LRU in order to store more often the contents with the largest values of $T(s_i)/s_i$. A possible way is the following: upon a cache miss, the newly requested content i is cached with probability q_i, which is an increasing function in $T(s_i)/s_i$. Specifically, we define q_i as follows:

$$q_i = e^{-\beta \frac{s_i}{T(s_i)}}, i \in \mathcal{N}, \tag{8}$$

where $\beta > 0$ is a constant parameter.[8] In practical cases, as discussed in Section IV, we set β such that $q_i \geq q_{\min}$ for every $i \in \mathcal{N}$, so that any content is likely to be stored in the cache after $1/q_{\min}$ queries on average.

Our policy has then the same behaviour of the q-LRU policy, but the probability q is not fixed, it is instead chosen depending on the size of the content as indicated in Equation (8). For this reason, we denote our policy by q_i-LRU.

With reference to Figure 2, the policy q_i-LRU would store with higher probability the smallest contents as well as the contents whose size is slightly larger than a multiple of the block size b. Note that the policy q_i-LRU does not depend on the model described above for the HDD service time, but it requires the ratio $T(s)/s$ to exhibit some variability (otherwise we would have the usual q-LRU).

Until now we have provided some intuitive justification for the policy q_i-LRU. This reasoning reflects how we historically conceived it. The reader may now want more theoretically grounded support to our claim that q_i-LRU is a good heuristic for problem (4). In what follows we show that q_i-LRU is asymptotically optimal when β diverges in two different ways. We first prove in Section III-A that q_i-LRU asymptotically stores in a cache the contents with the largest values $\lambda_i T(s_i)/s_i$, as the optimal greedy algorithm for problem (4) does. This would be sufficient to our purpose, but we find

[5]Since the PASTA property holds under the IRM model, the occupancy probability of content i (i.e. the fraction of time during which content i is in the cache) and its hit probability (i.e. the probability that a request for content i finds the content in the cache) are equal.

[6]The work in [11] actually assumes that all the contents have the same size, but their analysis can be easily extended to heterogenous sizes, as we do in Section III-B.

[7]For this case the authors of [11] suggest to simply replace the unknown request rates with online estimates, but popularity estimation for dynamic contents is still an open research topic (see e.g. [12], [13]) and in Sec. V of [14] we show that it can be tricky even under the stationary IRM.

[8]The reader may wonder why we have chosen this particular relation and not simply q_i proportional to $T(s_i)/s_i$. The choice was originally motivated by the fact that proportionality leads to very small q_i values for some contents. Our analysis below shows that Equation (8) is a sensible choice.

interesting to establish a connection between q_i-LRU and the cache utility maximization problem introduced in [11]. For this reason, in Section III-B, we reverse-engineer the policy q_i-LRU and derive the utility function it is implicitly maximizing as a function of β. We then let again β diverge and show that the utility maximization problem converges to a problem whose optimal solution corresponds to store the contents with the largest values $\lambda_i T(s_i)/s_i$.

A. Asymptotic q_i-LRU hit probabilities

In [1] it is proven that, under the assumptions of the IRM traffic model, the usual q-LRU policy tends to the policy that statically stores in the cache the most popular contents when q converges to 0. We generalize their approach to study the q_i-LRU policy when β diverges (and then q_i converges to 0, for all i). In doing so, we extend their result to the case when contents have heterogeneous sizes and we address some technical details that are missing in the proof in [1].[9]

Let us sort contents in a decreasing order of $\frac{\lambda_i T(s_i)}{s_i}$ assuming, in addition, that $\frac{\lambda_i T(s_i)}{s_i} \neq \frac{\lambda_j T(s_j)}{s_j}$ for every $i \neq j$.

Note that the hit probability h_i associated to the content i for the q_i-LRU policy is given by the following formula (see [1])

$$h_i(\beta, \tau_c) = \frac{q_i(\beta)(1 - e^{-\lambda_i \tau_c})}{e^{-\lambda_i \tau_c} + q_i(\beta)(1 - e^{-\lambda_i \tau_c})}, \tag{9}$$

where τ_c is the eviction time that, under CTA [8], [9], is assumed to be a constant independent of the selected content i.

Now, by exploiting the constraint:

$$C = \sum_i s_i h_i(\beta, \tau_c), \tag{10}$$

it is possible to express τ_c as an increasing function of β and prove that $\lim_{\beta\to\infty} \tau_c(\beta) = \infty$. This result follows [1], but, for the sake of completeness, we present it extensively in Appendix A.

We can now replace $q_i = e^{-\beta \frac{s_i}{T(s_i)}}$ in Equation (9) and express the hit probability as a function of β only, as follows:

$$h_i(\beta) = \frac{1 - e^{-\lambda_i \tau_c(\beta)}}{e^{\tau_c(\beta)\frac{s_i}{T(s_i)}\left(\frac{\beta}{\tau_c(\beta)} - \lambda_i \frac{T(s_i)}{s_i}\right)} + 1 - e^{-\lambda_i \tau_c(\beta)}}. \tag{11}$$

Let us imagine to start filling the cache with contents sorted as defined above. Let $\underline{c}$ denote the last content we can put in the cache before the capacity constraint is violated[10] i.e.

$$\underline{c} = \max\left\{k \Big| \sum_{i=1}^{k} s_i \leq C\right\}.$$

[9]The proof in [1, Appendix A] does not deal carefully with the cases when the accumulation points of $\beta/\tau_c(\beta)$ coincides with $\lambda_k T(s_k)/s_k$ for some value of k (we are using our notation). In these cases some indeterminate limit forms arise and the analysis becomes more complex. Moreover, the proof is very short and its final steps are quite cryptic. We developed our analysis independently from [1], that was not available at the time we submitted the conference version of this paper [3]. The corresponding conference version [15] did not actually prove this result, but it rather proved that there exist two constants k_1 and k_2 with $k_1 \leq k_2$ such that the most popular k_1 contents are stored with probability one and the least popular $N - k_2$ contents with probability 0. The two constants were not estimated and it was unknown what is the asymptotic behaviour of the hit probabilities for the $k_2 - k_1$ contents with intermediate popularity.

[10]We consider the practical case where $s_1 < C < \sum_{i=1}^{N} s_i$.

We distinguish two cases: the first $\underline{c}$ contents fill exactly the cache (i.e. $\sum_{i=1}^{\underline{c}} s_i = C$), or they leave some spare capacity, but not enough to fit the content $\underline{c}+1$. Next, we prove that q_i-LRU is asymptotically optimal in the second case. The first case requires a more complex machinery that we develop in Appendix A.

Consider then that $\sum_{i=1}^{\underline{c}} s_i < C < \sum_{i=1}^{\underline{c}+1} s_i$. As an intermediate step we are going to prove by contradiction that

Lemma III.1. *If $\sum_{i=1}^{\underline{c}} s_i < C < \sum_{i=1}^{\underline{c}+1} s_i$, it holds:*

$$\lim_{\beta\to\infty} \frac{\beta}{\tau_c(\beta)} = \lambda_{\underline{c}+1} \frac{T(s_{\underline{c}+1})}{s_{\underline{c}+1}}. \tag{12}$$

Proof. Suppose that this is not the case. Then, there exists a sequence β_n that diverges and a number $\epsilon > 0$ such that for all $n \in \mathbb{N}$

$$\text{either } \frac{\beta_n}{\tau_c(\beta_n)} \leq \frac{\lambda_{\underline{c}+1} T(s_{\underline{c}+1})}{s_{\underline{c}+1}} - \epsilon \tag{13}$$

$$\text{or } \frac{\beta_n}{\tau_c(\beta_n)} \geq \frac{\lambda_{\underline{c}+1} T(s_{\underline{c}+1})}{s_{\underline{c}+1}} + \epsilon. \tag{14}$$

If inequality (13) holds, then for all $i \leq \underline{c}+1$,

$$\frac{\beta_n}{\tau_c(\beta_n)} - \frac{\lambda_i T(s_i)}{s_i} \leq \frac{\beta_n}{\tau_c(\beta_n)} - \frac{\lambda_{\underline{c}+1} T(s_{\underline{c}+1})}{s_{\underline{c}+1}} \leq -\epsilon$$

From Equation (11), it follows immediately that

$$\lim_{\beta_n\to\infty} h_i(\beta_n) = 1, \quad \forall i \leq \underline{c}+1,$$

but then it would be

$$\lim_{n\to\infty} \sum_{i=1}^{\underline{c}+1} h_i(\beta_n) s_i = \sum_{i=1}^{\underline{c}+1} s_i > C$$

contradicting the constraint (10). In a similar way, it is possible to show that inequality (14) leads also to a contradiction and then Equation (12) holds. $\square$

Because of the Lemma III.1 and of Equation (11), we can immediately conclude that, when β diverges, $h_i(\beta)$ converges to 1, for $i \leq \underline{c}$, and to 0, for $i > \underline{c}+1$. Because of the constraint (10), it holds that:

$$\lim_{\beta\to\infty} h_{\underline{c}+1}(\beta) = \frac{C - \lim_{\beta\to\infty}\sum_{i\neq\underline{c}+1} h_i s_i}{s_{\underline{c}+1}} = \frac{C - \sum_{i\leq\underline{c}} s_i}{s_{\underline{c}+1}}.$$

The same asymptotic behavior for the hit probabilities holds when $\sum_{i=1}^{\underline{c}} s_i = C$, as it is proven in Appendix A. In particular, when $\sum_{i=1}^{\underline{c}} s_i = C$, $h_{\underline{c}+1}(\beta)$ converges to $(C - \sum_{i=1}^{\underline{c}} s_i)/s_{\underline{c}+1} = 0$. We can then conclude that:

Proposition III.2. *When the parameter β diverges, the hit probabilities for the q_i-LRU policy converge to the solution of the fractional knapsack problem* (4)*, i.e.*

$$\lim_{\beta\to\infty} h_i(\beta) = \begin{cases} 1, & \text{for } i \leq \underline{c}, \\ (C - \sum_{i=1}^{\underline{c}} s_i)/s_{\underline{c}+1}, & \text{for } i = \underline{c}+1, \\ 0, & \text{for } i > \underline{c}+1. \end{cases}$$

Then, the q_i-LRU policy asymptotically minimizes the load on the hard-disk.

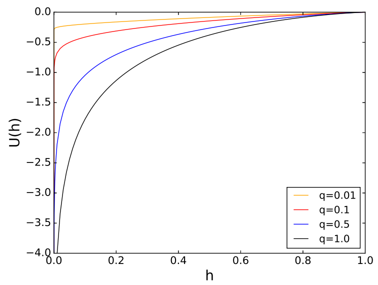

Fig. 3. Utility Function of q-LRU when $\lambda_i s_i = 1$

B. Reverse-Engineering q_i-LRU

In [11], the authors show that existing policies can be thought as implicitly solving the utility maximization problem (6) for a particular choice of the utility functions $U_i(h_i)$. In particular, they show which utility functions correspond to policies like LRU and FIFO. In what follows, we "reverse-engineer" the q_i-LRU policy and we show in a different way that it solves the fractional knapsack problem. More specifically, we use the results for strictly convex utilities in [11] for the limit case of linear utility functions. We proceed similarly to what is done in [11], extending their approach to the case where content sizes are heterogeneous (see Appendix A). We show that the utility function for content i can be expressed as:[11]

$$U_i(h_i) \quad = \quad -\lambda_i s_i \int_0^{1-h_i} \frac{\mathrm{d}x}{\ln\left(1+\frac{1-x}{q_i x}\right)}, \tag{15}$$

that is defined for $h_i \in (0,1]$ and $q_i \neq 0$. Each function $U_i(.)$ is increasing and concave. Moreover, $U_i(h_i) < 0$ for $h_i \in (0,1)$, $U_i(1) = 0$ and $\lim_{h_i \to 0} U_i(h_i) = -\infty$. Figure III-B shows the utility function for different values of q_i and $\lambda_i s_i = 1$.

We are interested now in studying the asymptotic behavior of the utility functions $U_i(h_i)$ when β diverges, and thus q_i converges to zero. We say that $f(x)$ is equivalent to $g(x)$ when x converges to 0 if $\lim_{x\to 0} f(x)/g(x) = 1$, and we write $f(x) \sim g(x)$. The following result holds.

Lemma III.3. *For $h_i \in (0,1]$, when q_i converges to 0,*

$$U_i(h_i) = -\lambda_i s_i \int_0^{1-h_i} \frac{\mathrm{d}x}{\ln\left(1+\frac{1-x}{q_i x}\right)} \sim -\frac{\lambda_i s_i (1-h_i)}{\ln(1/q_i)}.$$

Proof. First, we note that the following inequalities are true for every $\delta > 0$:

$$\int_0^{1-h_i} \frac{\mathrm{d}x}{\ln\left(1+\frac{1-x}{q_i x}\right)} \geq \int_{q_i^\delta}^{1-h_i} \frac{\mathrm{d}x}{\ln\left(1+\frac{1-x}{q_i x}\right)}$$
$$\geq \int_{q_i^\delta}^{1-h_i} \frac{\mathrm{d}x}{\ln\left(1+\frac{1-q_i^\delta}{q_i^{\delta+1}}\right)} \geq \frac{1-h_i-q_i^\delta}{\ln\left(1+\frac{1-q_i^\delta}{q_i^{\delta+1}}\right)},$$

where the second inequality follows from the fact that the integrand is an increasing function of x.[12]

Similarly, it holds

$$\int_0^{1-h_i} \frac{\mathrm{d}x}{\ln\left(1+\frac{1-x}{q_i x}\right)} \leq \frac{1-h_i}{\ln\left(1+\frac{h_i}{q_i(1-h_i)}\right)}. \tag{16}$$

Asymptotically, when q_i converges to zero, the lower bound in (**??**) is equivalent to $\frac{1-h_i}{(1+\delta)\ln(1/q_i)}$, and the upper bound in (16) is equivalent to $\frac{1-h_i}{\ln(1/q_i)}$ for $h_i > 0$. We obtain the following (asymptotic) inequalities when q_i converges to 0

$$\frac{1-h_i}{(1+\delta)\ln(1/q_i)} \leq \int_0^{1-h_i} \frac{\mathrm{d}x}{\ln\left(1+\frac{1-x}{q_i x}\right)} \leq \frac{1-h_i}{\ln(1/q_i)}, \tag{17}$$

for every $\delta > 0$ (when q converges to 0, then $q_i^\delta < 1 - h_i$ asymptotically). Thus, when q_i converges to 0, we get

$$\int_0^{1-h_i} \frac{\mathrm{d}x}{\ln\left(1+\frac{1-x}{q_i x}\right)} \sim \frac{1-h_i}{\ln(1/q_i)}, \tag{18}$$

since, otherwise, we could find an $\varepsilon > 0$ and a sequence $q_{i,n}$ converging to 0 such that for large n

$$\int_0^{1-h_i} \frac{\mathrm{d}x}{\ln\left(1+\frac{1-x}{q_{i,n} x}\right)} \leq (1-\varepsilon)\frac{1-h_i}{\ln(1/q_{i,n})}.$$

But, this would contradict the left-hand inequality in (17) which is valid for every $\delta > 0$.

The thesis follows immediately from the expression of the utility function (15) and from (18).

□

We consider $q_i = e^{-\beta \frac{s_i}{T(s_i)}}$. Lemma III.3 allows us to conclude that

$$U_i(h_i) \sim -\frac{\lambda_i T(s_i)(1-h_i)}{\beta}, \quad \text{when } \beta \to \infty,$$

and then the utility functions are asymptotically linear. Note that the maximization problem (6) is over the hit probabilities h_i and the solution of the problem will be the same even if the functions $U_i(.)$ are multiplied by a positive constant. We conclude that, when β diverges, the problem (6) can be formulated as follows

$$\begin{aligned} \underset{h_1,\ldots h_n}{\text{maximize}} \quad & \sum_{i=1}^{\mathcal{N}} \lambda_i h_i T(s_i) \\ \text{subject to} \quad & \sum_{i=1}^{\mathcal{N}} h_i s_i = C \end{aligned}$$

which is exactly the formulation of the fractional knapsack problem.

[11] As already observed in [11], the utility function we can derive from an existing policy is not unique. For example an affine transformation $aU() + b$ with $a > 0$ of the function in (15) is also a valid utility function for q_i-LRU.

[12] Note that the inequalities hold both if $q_i^\delta \leq (1-h_i)$and if $q_i^\delta > (1-h_i)$.

IV. Experiments

In this section we evaluate the performance of our q_i-LRU policy. Here we take a numerical perspective, and design a trace-driven simulator that can reproduce the behavior of several caching policies, which we compare against q_i-LRU. We have used both synthetic traces generated according to the IRM and real traces collected at two vantage points of the Akamai network [2]. We proved that q_i-LRU is optimal under the IRM and indeed our experiments not only confirm it but also show significant improvement in comparison to other replacement policies. For this reason, in this section we focus mainly on the results obtained using real traces. In the following, we describe our experimental methodology, show the characteristics of the real traces we use, and present the results of our evaluation.

A. Methodology and Performance indexes

The comparative analysis of different caching policies requires an environment where it is possible to reproduce exactly the same conditions for all the different policies. To do so, we adopt a trace-driven simulation approach, which allows us to control the initial conditions of the system, explore the parameter space and perform a sensitivity analysis, for all eviction policies.

Our simulator reproduces two memory types: the main memory (RAM) and the hard disk (HDD). Each object is stored in the HDD according to the LRU policy. For the RAM we consider 3 different policies: LRU, SIZE and q_i-LRU. They all evict the least recently requested content, if space is needed, but they adopt different criteria to decide if storing a new content after a miss:

- LRU always stores it;
- SIZE stores it if 1) its size is below a given threshold T, or 2) it has been requested at least N times, including once during the previous M hours;
- q_i-LRU stores it with probability q_i, as explained in the previous sections.

So, in addition to comparing q_i-LRU to the traditional LRU policy, we also consider the SIZE policy since small objects are the ones that have a bigger impact on the HDD, in terms of their service time per byte $T(s_i)/s_i$ (see also Figure 2). We therefore prioritize small objects, and we store objects bigger than the threshold T (as the policy LRU-THOLD in [16]) only after they have been requested for at least N times.[13] The SIZE policy can thus be seen as a first attempt to decrease the impact of small objects on the HDD, and ultimately reduce the strain on HDD resources. With the q_i-LRU policy, we aim at the same goal, but modulate the probability to store an object in RAM as a function of its size, and thus service time.

Note that the hit ratio of the whole cache depends only on the size of the HDD and its replacement policy (LRU). The RAM replacement policy does not affect the global hit ratio. In what follows we focus rather on the **total disk service time**: this is the sum of the $T(s_i)$ of all the objects served by the HDD. Smaller disk service times indicate lower pressure on the disk.

[13] [17] shows significant increase of the hit ratio as well as decrease of the number of disk-write operations for $N = 2$. Similar improvements are observed also in [18].

TABLE II
Traces: basic information

	30 days	5 days
Number of requests received	$2.22 \cdot 10^9$	$4.17 \cdot 10^8$
Number of distinct objects	113.15 M	13.27 M
Cumulative size	59.45 TB	2.53 TB
Cumulative size of objects requested at least twice	20.36 TB	1.50 TB

We show the results for a system with 4 GB RAM and 3 TB HDD. We have tried many different values for the RAM size up to 30 GB, and the qualitative results are similar. For the SIZE policy, we have extensively explored the parameter space (threshold T, number of requests N, and number of hours M) finding similar qualitative results. As a representative set of results, we show here the case with $T = 256$ KB, $N = 5$ and $M = 1$ hour. For the q_i-LRU policy, the default value of the constant β is chosen such that $\min_{i \in \mathcal{N}} q_i = 0.1$ (see Equation (8)).

B. Trace characteristics

We consider two traces with different durations and collected from two different vantage points. The first trace has been collected for 30 days in May 2015, while the second trace for 5 days at the beginning of November 2015. Table II shows the basic characteristics of the traces.

Figure 4 shows the number of requests for each object, sorted by rank (in terms of popularity), for both traces. For the 30-day trace, there are 25-30 highly requested objects (almost 25% of the requests are for those few objects), but the cumulative size of these objects is less than 8 MB. Since they are extremely popular objects, any policy we consider stores them in RAM, so they are not responsible for the different performance we observe for the different policies.

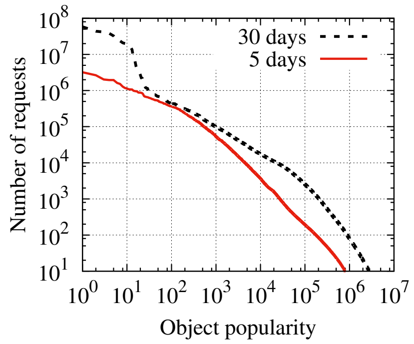

Fig. 4. Number of requests per object (ordered by rank).

Figure 5 shows an alternative version of the information related to the number of requests. In particular, the left hand side of Fig. 5 provides the CDF of the requests versus the percentage of the contents (objects are sorted from the most popular to the least popular). We can see that the 10% (resp. 20%) most popular objects are responsible for 90% (resp.

95%) of the requests. The right hand side of Fig. 5 shows the CDF of the object aggregate service time, i.e., the time needed to retrieve the content from the HDD upon all its requests. From the service time viewpoint, we can see that 20% of the objects are responsible for 90% of HDD load. Given that a fraction of the objects accounts for most of the load on the HDD, one may wonder if the LRU policy is sufficient to *select* the best subset of objects such that the load on the HDD is minimal. We show in Sect. IV-D that this is not the case.

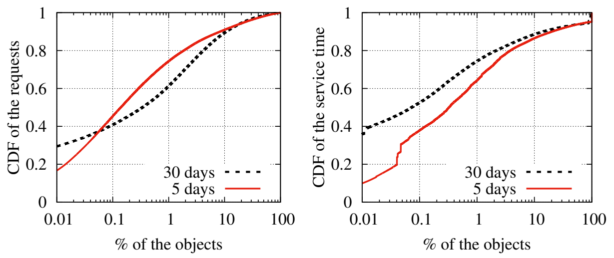

Fig. 5. CDF of the requests for the different objects (left), and the service time for the different objects (right). In both cases, objects are ranked by popularity.

Next, we study the relation between the size and the number of requests of each object. In Figure 6, for each object, we plot a point that corresponds to its size (y-axis) and the number of requests (x-axis). For the 30-day trace, the plot does not include the 30 most popular objects. We notice that the 5-day trace contains only a few objects smaller than 1 kB.

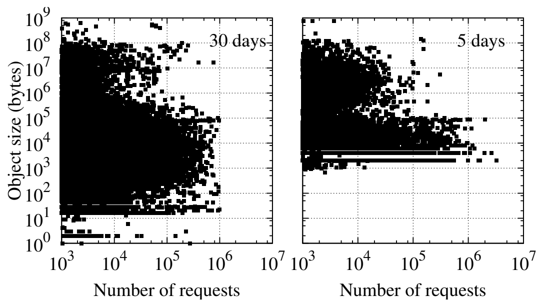

Fig. 6. Size vs Number of requests. For ease of representation, we consider the objects with at least 1000 requests (for the 30-day trace, we do not include the 30 most popular objects).

This is also shown in Figure 7, where we plot the empirical Cumulative Distribution Function (CDF) for the size of the requested objects (without aggregating requests for the same object). The 30-day trace contains a lot of requests for small objects, while the 5-day trace contains requests for larger objects (e.g., see the 90-th percentile). In the 30-day trace we have then a larger variability of the ratio $T(s)/s$ (see Figure 2) and we expect q_i-LRU to be able to differentiate more among the different contents and then achieve more significant improvement, as it is confirmed by our results below.

C. Simulator validation

The evaluation of our scheme is based on trace-driven simulation so we can have full control of the experimental settings. One may ask if this approach is sufficiently accurate in reproducing the actual systems. We have already shown that the HDD model used in our simulator is very accurate (see Sect. II-A and in particular Fig. 2). We now show how the performance indexes captured by our simulator are equivalent to the ones recorded by a production machine.

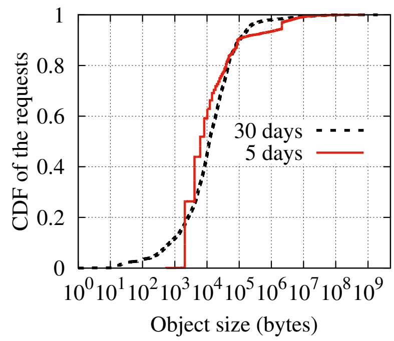

Fig. 7. Given an object size, the CDF shows the cumulative fraction of the requests up to that object size (for the 30-day trace, we do not include the 30 most popular objects).

Along with the 5-day trace, we have a *machine performance trace* where, every 30 seconds, two main performance indexes are recorded by the machine that has received the requests: the machine disk load and the amount of data served. The machine where these indexes have been collected used a LRU policy. We then have instructed our simulator to produce, given the 5-day trace as input, a performance trace as output to be compared with the machine performance trace: every 30 seconds, the simulator writes (i) the sum of the $T(s_i)$ of the objects served from the HDD, which can be used as an indication of the disk load, and (ii) the bytes served (RAM and HDD) – both indexes are computed in each 30-second interval, we do not take averages from the beginning.

The comparison of the two performance traces, generated by the simulator and by the machine, when we consider the bytes served, is straightforward, since the byte served are given by the request arrival pattern that are recorded on the request trace, and they are necessarily the same. The comparison is instead extremely interesting when we consider the load on the HDD since (i) it further confirms the model of the HDD we used and (ii) it validates the design of the simulator, where we have focused on the basic behavior of the cache, without modeling the complex operations of the Operating System (OS). In other words, even if the cache run on a machine managed by an OS, the impact of the OS management is not significant.

Figure 8 shows that indeed our simulator is able to reproduce the same disk utilization over time as recorded on the real machine. Note that we recorded the sum of the $T(s_i)$, so, in order to be able to compare with the output of the real machine, we need to normalize the values: in particular, we use the highest value observed in the output. We performed this normalization for the output of the real machine as well. In this way, the range of both outputs is between 0 and 1. The figure shows a small portion of the trace, but both traces overlap for the whole duration.

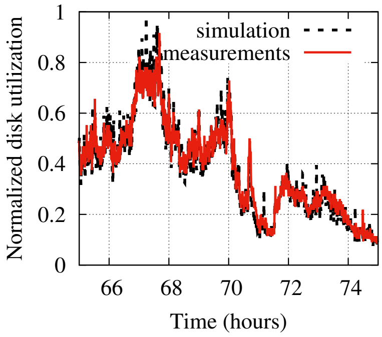

Fig. 8. Machine normalized disk load: comparison between the output recorded by the real machine and the output produced by our simulator.

D. Comparative analysis of the eviction policies

Tables III and IV summarize the aggregate results for the two traces we consider in our study. For the hit ratio, we see that the q_i-LRU policy can serve more requests from the RAM. On the other hand, the overall number of bytes served by RAM is smaller: this means that the RAM is biased towards storing small, very popular objects, as expected. The last column shows the gain, in percentage, in disk service time between each policy and LRU, which we take as a de-facto reference (e.g., -10% for policy "x" means that its disk service time is 10% smaller than for LRU). This is the main performance metric we are interested in. For the 30-day trace, the q_i-LRU policy improves by 23% the disk service time, over the LRU policy. For the 5-day trace, the improvement of q_i-LRU over LRU is smaller, topping at a bit more than 7%. The reason behind this result relates to the object size distribution in the trace: as shown in Figure 7, the trace contains objects starting from 1 kB, while, for the 30-day trace, 20% of the requests are for objects smaller than 1 kB. The impact of these objects on the overall $T(s_i)$ is significant.

TABLE III
RESULTS FOR THE 30-DAY TRACE WITH 4 GB RAM.

		% reqs	bytes served	service time	Δ (%) w.r.t. LRU
LRU	RAM	73.06	509 TB	4907 h	-
	HDD	26.94	157 TB	1663 h	-
SIZE	RAM	76.38	512 TB	5055 h	+ 3.02%
	HDD	23.62	154 TB	1515 h	-8.90%
q_i-LRU	RAM	84.27	489 TB	5294 h	+7.89%
	HDD	15.73	177 TB	1276 h	-23.27%

TABLE IV
RESULTS FOR THE 5-DAY TRACE WITH 4 GB RAM.

		% reqs	bytes served	service time	Δ (%) w.r.t. LRU
LRU	RAM	79.61	159 TB	1058 h	-
	HDD	20.39	23 TB	219 h	-
SIZE	RAM	80.31	160 TB	1064 h	+ 0.57%
	HDD	19.69	22 TB	213 h	-2.74%
q_i-LRU	RAM	84.72	149 TB	1074 h	+1.51%
	HDD	15.28	33 TB	203 h	-7.31%

Next, we take a closer look at our policy, q_i-LRU, in comparison to the reference LRU policy. We now consider the contribution to the overall hit ratio of each object, to understand their importance to cache performance. For the 30-day trace, we sorted the objects according to their rank (in terms of popularity) and their size, and plot the difference between LRU hit ratio and q_i-LRU hit ratio. Figure 9 shows that both policies store the same 1000 most popular objects; then, the q_i-LRU policy gains in hit ratio for medium-popular objects. Switching now to object size, both policies store the same set of small objects, while q_i-LRU gains hit ratio with the medium-size objects.

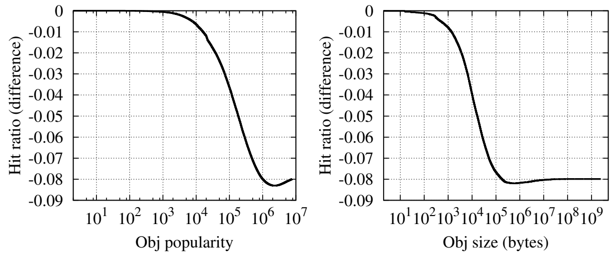

Fig. 9. Difference between hit ratios when objects are ordered by popularity (left) and by size (right) for the 30-day trace.

Figure 10 considers the contribution to the disk service time of each object (ordered by rank or by size) and shows the difference between q_i-LRU and LRU. Clearly, medium popular objects and medium size objects contribute the most to the savings in the service time that our policy achieves.

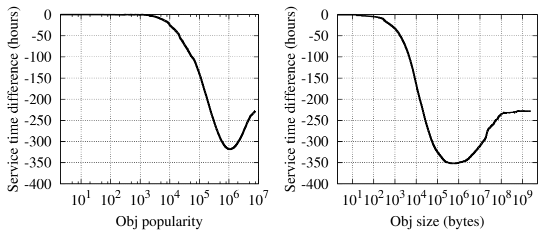

Fig. 10. Difference between service time (served by the RAM) when objects are ordered by rank (left) and by size (right) for the 30-day trace.

These results have been obtained using the two traces from Akamai network. In order to explore the effect of popularity skewness on the q_i-LRU performance, we resort to IRM synthetic traces. In particular we generate objects with sizes drawn from a Pareto distribution with shape equal to 0.4 (roughly fitting the empirical distribution found in the 30-day traces). The catalogue is 10 million objects, and we have 2 billion requests (2×10^9). The objects are requested according to their popularities, which are independently distributed according to a Zipf distribution with different typical values of the parameter $\alpha = 0.6 \dots 1.2$ (see [19]).

We considered a 3TB HDD and different values for the RAM (10 GB, 20 GB and 30 GB), but the results are similar. We observe that the global hit rate of the cache is the same under LRU and under q_i-LRU for any size of HDD and RAM, because the contents stored in both cases in the HDD are exactly the same. Table V summarizes the performance of the RAM cache for the 10 GB case.

TABLE V
RESULTS WITH DIFFERENT SKEWNESS USING ZIPF DISTRIBUTION FOR THE POPULARITY OF THE OBJECTS.

Alpha	LRU: % reqs served by RAM	q_i-LRU: % reqs served by RAM	% service time saved from HDD
0.6	2.52	57.51	14.38
0.8	17.22	65.88	20.39
1.0	54.85	84.19	29.27
1.2	87.19	96.30	37.57

Smaller values of α correspond to more homogeneous popularities (heavier distribution tails). In this situation LRU fails to store the most popular contents achieving a very low hit rate and consequently a high load on the HDD. q_i-LRU performs much better in terms of the hit rate (more than 10 times larger than what LRU achieves for $\alpha = 0.6$), and it reduces correspondingly the HDD service time, even if the relative improvement is only 14% because the reference point is the large HDD load for LRU. As the distribution tail becomes lighter (i.e., α increases) the RAM serves more contents for both policies. While the hit rate gap reduces, the relative service time saving increases, because now savings are compared with a smaller reference point.

E. Sensitivity analysis

Next, we study the behavior of q_i-LRU as a function of the parameter β, but we plot the results for the parameter $q_{\min} = \min_{i \in \mathcal{N}} q_i$, that is easier to interpret, being the minimum probability according to which a content is stored in the RAM.

Figure 11 provides two different views. On the left-hand side, it shows the percentage of HDD service time offloaded to the RAM by q_i-LRU, both under the 30-day trace and a synthetic IRM trace generated using the same empirical distributions for object size and popularity as in the 30-day trace. As expected, under IRM, the improvement from q_i-LRU increases as $q_{\min}$ decreases, i.e. as β increases. Interestingly, the HDD benefits even more under the 30-day trace, with more than 80% of the service offloaded to the RAM. This is due to the temporal locality effect (see e.g. [20]), i.e. to the fact that requests typically occur in bursts and then the RAM is more likely to be able to serve the content for a new request than it would be under the IRM model. We observe also that the performance of q_i-LRU are not very sensitive to the parameter $q_{\min}$ (and then to β), a feature very desirable for practical purposes. The right-hand side of Figure 11 shows the relative improvement of q_i-LRU in comparison to LRU (calculated as difference of the HDD service time under LRU and under q_i-LRU, divided by the HDD service time under LRU). While q_i-LRU performs better and better as $q_{\min}$ decreases with the IRM request pattern, the gain reduces when $q_{\min}$ approaches 0 (β diverges) with the 30-day trace. This is due also to temporal locality: when the probabilities q_i are very small, many contents with limited lifetime have no chance to be stored in the RAM by q_i-LRU and they need to be served by the HDD. Despite this effect, q_i-LRU policy still outperforms LRU over a large set of parameter values and obtain improvements larger than 20% for $0.02 < q_{\min} < 0.4$.

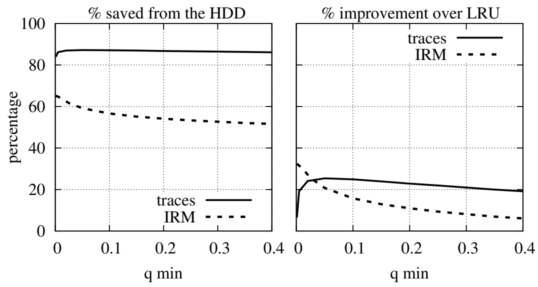

Fig. 11. Sensitivity analysis to the value of $q_{\min}$.

V. AKAMAI: PRELIMINARY RESULTS

In this section we evaluate the performance of our q_i-LRU policy in deployed infrastructure. The evaluation of a new scheme in such a scenario is not simple, since the deployed infrastructure is much more complex than an isolated machine fed with a trace, and the performance comparison with or without the q_i-LRU policy is not straightforward.

A. Experimental settings

Akamai network consists of hundreds of thousands machines for scalability reasons. The Akamai Mapping system directs the user request to a specific machine, based on factors including locality, load changes, machine failures etc [2]. The traffic on any two machines are not exactly the same at any time. For this reason, the comparison between two machines, one with the q_i-LRU policy enabled, and the other with the default Akamai policy, is not simple.

In our case, we decide to consider a set of machines in two different periods, first with a reference caching policy, and then with the q_i-LRU policy enabled.

As for performance comparison, the data that is possible to collect from a production machine do not include the more granular metrics that we used to evaluate our solution in Sect. IV. In particular, when a request is served by the RAM, the system does not record what might have been the HDD service time, i.e., the time that it would take if the request were served by the HDD. Instead, the system records the load on the HDD, the requests served, the total bytes served by the RAM and by the HDD.

The preliminary results we show consider a set of servers in USA. Due to the complexity of introducing a new policy in a deployed infrastructure, we defer to an extended version of this work the definition of a more accurate measurement campaign, to further substantiate the intuition we obtain with our preliminary deployment results.

B. Results

Figure 12 shows two aggregated performance indexes recorded at machines, when the q_i-LRU policy is not enabled and when it is enabled. To avoid effects due to weekly patterns, the observation period was one week, and here we show the first significant four days.

The left-hand side of the figure shows the normalized disk utilization - the normalization factor is the highest value

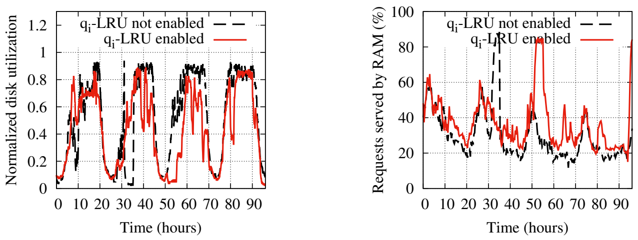

Fig. 12. Results from deployed infrastructure: Normalized disk utilization (left) and percentage of requests served by the RAM (right).

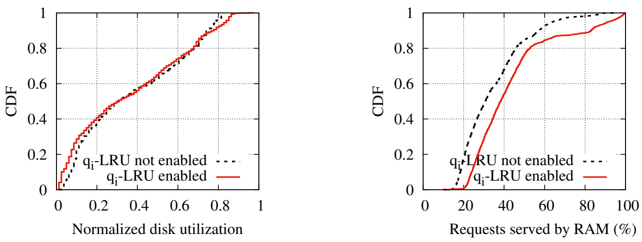

Fig. 13. Results from deployed infrastructure: CDF of the normalized disk load (left) and the percentage of requests served by the RAM (right), when q_i-LRU policy is enabled and not enabled.

observed during the whole observation period. As we noted in Sect IV-C, the service time and the disk utilization are highly correlated, therefore we can take such a measure as an indication of the service time. The figure shows that the disk utilization is equivalent when the q_i-LRU policy is enabled or not. This is also confirmed in Figure 13 (left-hand side) that shows the corresponding CDF. The result is due to the fact that such a metric is used by the Akamai Mapping system to decide when rebalancing the load. In other words, our policy does have an impact on the disk load, but the Mapping system compensates the diminished load by rebalancing the requests.

The benefits of the q_i-LRU policy, therefore, can be seen if we consider the requests served by the RAM. The right-hand side of Figure 12 compares time evolution of the percentage of requests served by RAM when the q_i-LRU policy is not enabled and when it is enabled. The right-hand side of Figure 13 shows the corresponding CDF. On average, when the q_i-LRU policy is enabled, machines are able to serve 10% more of the requests from the RAM, which is a desirable effect we have observed also in the simulation results. The two peaks occurring after 35 and 53 hours depend on the specific traffic patterns that take place when the disk load is not high, and therefore they are not representative of the average behaviour.

In summary, the preliminary results indicates that our q_i-LRU policy is indeed able to alleviate the stress on the disk by exploiting in a more efficient way the RAM.

VI. Extension to other performance metrics

We designed our policy q_i-LRU to solve the following specific problem: minimize the expected HDD load to reduce the number of spurious misses. To this purpose, we have considered that a miss for content i generates a cost c_i for the HDD equal to the time the HDD needs to retrieve content i, $c_i = T(s_i)$. We observe that our theoretical results in Sections III do not depend on the specific structure of the function $T(s_i)$. It follows that if we choose

$$q_i = e^{-\beta \frac{s_i}{c_i}}, \quad \forall i \in \mathcal{N}, \tag{19}$$

the policy q_i-LRU is solving—in the sense explained in the Section III—the following general problem:

$$\begin{aligned} \underset{\mathcal{M}\subset\mathcal{N}}{\text{maximize}} \quad & \sum_{i\in\mathcal{M}} \lambda_i c_i \\ \text{subject to} \quad & \sum_{i\in\mathcal{M}} s_i \le C. \end{aligned} \tag{20}$$

The policy stores in the cache the set of contents $\mathcal{M}^*$, solution of problem (20). By reverting the reasoning in Section II-C, we can conclude that this set also minimizes the expression

$$\sum_{i\in\mathcal{N}\setminus\mathcal{M}} \frac{\lambda_i}{\sum_{j\in\mathcal{N}}\lambda_j} c_i = \sum_{i\in\mathcal{M}} \frac{\lambda_i}{\sum_{j\in\mathcal{N}}\lambda_j} c_i - \sum_{i\in\mathcal{M}} \frac{\lambda_i}{\sum_{j\in\mathcal{N}}\lambda_j} c_i,$$

i.e. the expected cost generated by a miss.

Hence, the policy q_i-LRU is able to minimize any retrieval cost as far as i) the cost is additive over different misses, ii) the cost c_i of a miss is known by the cache, so that it is possible to compute the probabilities q_i, according to Equation (19).

We provide a few examples of meaningful performance metrics q_i-LRU could optimize. If $c_i = 1$, the goal is to minimize the cache miss ratio. If $c_i = s_i$, the goal is to minimize the traffic from upstream servers/caches. In these cases, the computation of the probabilities q_i does not pose any problem. It is also possible to minimize the expected retrieval time if the cost of an object is indeed its retrieval time from the server. In this case, c_i may not be immediately available to the cache, but the cache can maintain some estimates for the retrieval times of the most requested objects or use some approximate function for such costs (e.g. on the basis of the url). Similar considerations hold for other metrics like ISP/AS operational costs, or damage to flash memories in hierarchical caches, whose minimization is the aim of the caching policies proposed respectively in [21], [22]) and in [23].

VII. Related Work

Cache replacement policies have been the subject of many studies, both theoretical and experimental. We focus here on the more analytical studies, which are closer to our contribution. Moreover, our policy is explicitly designed to mitigate the burden on the HDD, a goal not considered in most previous experimental works, despite its practical importance.

Most of the theoretical work in the past has focused on the characterization of the performance of LRU, RANDOM, and FIFO [9][24][15][25]. All these works do not assume different levels of caches, where one level replicates the content stored in the other level to decrease the overall response delay. Moreover, they do not aim to design optimal caching policies.

Some papers have proposed heuristic cache policies with different optimization goals, like minimizing the ISP/AS operational costs [21], [22] or the damage to flash memories in hierarchical caches [23]. Their solutions are tailored to the specific problem considered and do not apply to reducing the HDD load. The q_i-LRU policy, instead, can be applied to different problems as shown in Section VI.

Closer to our application is [26], that considers a 2-level hierarchy, with the content stored in the SSD and DRAM. The authors design a policy which decreases the response time by pre-fetching the content from SSD to DRAM. To this aim, they focus on a specific type of content, videos divided into chunks, for which the requests are strongly correlated, and a request for a chunk can be used to foresee future requests for other chunks of the same content. In our work, instead, we provide a model for the q_i-LRU policy which does not assume any correlation on the requests arrivals, but prioritize the content that imposes a high burden on the HDD.

The problem of minimizing the time-average retrieval cost has been studied under the name of *File Caching* problem [27], when the sequence of content requests is unpredictable. In this case no algorithm can provide absolute worst-case guarantees and it is then standard to perform a competitive analysis of cache policies [28], [29], [30]. Our work considers instead that the request sequence exhibits some regularity and in particular contents have different popularities.

The idea to probabilistically differentiate content management according to the ratio c_i/s_i had already been considered in [31], where, upon a hit, content i is moved to the front of the queue with some probability $\tilde{q}_i$. The authors of [32] prove that, under Zipf's law for popularities, the asymptotic hit ratio is optimized when the probabilities $\tilde{q}_i$ are chosen to be inversely proportional to document sizes. More recently, the use of size-aware policies to optimize the hit ratio has also been advocated by [33].

The most related work to ours is the cache optimization framework in [11], that we have widely discussed through the paper. We stress again here the two main differences: we do not assume content popularities to be known (nor to be explicitly estimated) and the utility functions are linear.

In [14] a subset of the authors study the general framework of caching policies maximizing linear utilities. That paper builds on a few elements presented here: i) finding the optimal set of contents is a knapsack problem and ii) the idea to use a biased version of q-LRU. The paper focuses on time-variant policies, that can converge with probability one to the optimal set of contents. It proposes DynqLRU, a dynamic version of q_i-LRU, and discusses how such policy can be adapted to a scenario where popularities may vary over time.

VIII. Conclusion

Caches represent a crucial component of the Internet architecture: decreasing the response time is one of the primary objectives of the providers operating such caches. This objective can be pursued by exploiting the RAM of the cache server, while keeping most of the contents in the HDD.

In this paper, we presented a new cache replacement policy that takes advantage of the access-time difference in the RAM and in the HDD to reduce the load on the HDD, so that to improve the overall cache efficiency for a capacity constrained storage systems. Our policy, called q_i-LRU, is a variant of q-LRU, where we assign a different probability q_i to each content based on its size.

We proved that q_i-LRU is asymptotically optimal, and we provided an extensive trace-driven evaluation that showed between 10% and 20% reduction on the load of the HDD with respect to the LRU policy. Moreover, the preliminary results from Akamai production environment shows that our policy is able to increase the percentage of requests served by the RAM (for a given disk load).

Finally, the policy q_i-LRU can be adapted to solve any retrieval cost minimization problem, when the retrieval costs are additive over different misses.

This work was partially supported by the Italian National Group for Scientific Computation (GNCS-INDAM).

Appendix A
Proof of $\lim_{\beta\to\infty} \tau_C(\beta) = \infty$

We define the function f as follows:

$$f(\tau_C, \beta) = \sum_{i=1}^{N} s_i h_i = \sum_{i=1}^{N} \frac{s_i(e^{\lambda_i \tau_C} - 1)}{e^{\beta \frac{s_i}{T(s_i)}} + e^{\lambda_i \tau_C} - 1}. \tag{21}$$

As we discussed in Section III-A, CTA implies that $f(\tau_C, \beta) = C$.

We will prove that $\lim_{\beta\to\infty} \tau_C = +\infty$. We differentiate the formula (21) with respect to β and τ_C and we obtain

$$\begin{aligned}
\frac{\partial f}{\partial \tau_C} &= \sum_i \frac{s_i \lambda_i e^{\lambda_i \tau_C + \beta \frac{s_i}{T(s_i)}}}{(e^{\beta \frac{s_i}{T(s_i)}} + e^{\lambda_i \tau_C} - 1)^2} \\
\frac{\partial f}{\partial \beta} &= \sum_i \frac{-s_i^2 e^{\beta \frac{s_i}{T(s_i)}} (e^{\lambda_i \tau_C} - 1)}{T(s_i)(e^{\beta \frac{s_i}{T(s_i)}} + e^{\lambda_i \tau_C} - 1)^2}.
\end{aligned}$$

The first partial derivative is strictly positive while the second is negative for all the values $\beta > 0$ and $\tau_C > 0$ and, therefore, by the implicit function theorem, τ_C can be expressed locally as a $\mathcal{C}^1$ function of β and

$$\frac{\partial \tau_C}{\partial \beta} = -\frac{\partial f/\partial \beta}{\partial f/\partial \tau_C} > 0.$$

This is true in some open set (whose existence is assured by the theorem) containing the points (τ_C, β) that verify $f(\tau_C, \beta) = C$. So, τ_C is an increasing function with respect to β and the limit $\lim_{\beta\to\infty} \tau_C(\beta)$ exists.

We prove by contradiction that the limit is equal to $+\infty$. Suppose that $\lim_{\beta\to\infty} \tau_C(\beta) < \infty$, then, by (21), we get $\lim_{\beta\to\infty} f(\tau_C(\beta), \beta) = 0$. This would contradict the fact that $f(\tau_C, \beta) = C$ and therefore we conclude that $\lim_{\beta\to\infty} \tau_C = +\infty$.

Appendix B
When contents fill exactly the cache

In this section, we study the case where $\sum_{i=1}^{c} s_i = C$. Note that the results up to Lemma A.3 (included) are general, i.e, they do not make any assumption on $\sum_{i=1}^{c} s_i$, while the rest of the section focuses on the case where $\sum_{i=1}^{c} s_i = C$.

We start introducing some additional notation. Remember that contents are labeled according to the reverse order of the values $\lambda_i \frac{T(s_i)}{s_i}$. Given a point y, we denote by $r(y)$ the largest index such that $\lambda_i \frac{T(s_i)}{s_i}$ is larger than y (or 0 if all the values are smaller), and by $l(y)$ the smallest index i such that $\lambda_i \frac{T(s_i)}{s_i}$ is smaller than y (or $N+1$ if all the values are larger), i.e. we have

$$\begin{aligned} r(y) &= \max\left(\left\{0\right\} \cup \left\{k = 1, \dots N \middle| \lambda_k \frac{T(s_k)}{s_k} > y\right\}\right), \\ l(y) &= \min\left(\left\{N+1\right\} \cup \left\{k = 1, \dots N \middle| \lambda_k \frac{T(s_k)}{s_k} < y\right\}\right). \end{aligned}$$

We recall here the definition of a cluster value [34, Exercise 5.10.11], that allows us to express more synthetically some of the following results.[14]

Definition B.1. Given a function $f : A \to \mathbb{R}$, where $A \subset \mathbb{R}$, and $x_0 \in [-\infty, +\infty]$ an accumulation point of A, we say that $y^* \in \mathbb{R}$ is a cluster value of $f(x)$ at x_0 if it exists a sequence $x_n \in A - \{x_0\}$ such that $\lim_{n\to\infty} x_n = x_0$ and $\lim_{n\to\infty} f(x_n) = y^*$. We also say that $f(x)$ has a cluster value y^* at x_0.

In what follows we only consider cluster values at $+\infty$. For the sake of conciseness, we will omit to specify "at $+\infty$."

We start establishing some connections between the asymptotic behaviour of $\frac{\beta}{\tau_c(\beta)}$ and $h_i(\beta)$ in terms of their cluster values.

Lemma B.1. *If y^* is a cluster value of $\frac{\beta}{\tau_c(\beta)}$, then it exists a diverging sequence β_n such that, for all $i \leq r(y^*)$, $h_i(\beta_n)$ converges to 1 and, for all $j \geq l(y^*)$, $h_j(\beta_n)$ converges to 0.*

Proof. From the definition of a cluster value, it exists a diverging sequence β_n such that $\lim_{n\to\infty} \beta_n/\tau_c(\beta_n) = y^*$. For each $i \leq r(y^*)$, it holds

$$\lim_{n\to\infty}\left(\frac{\beta_n}{\tau_c(\beta_n)} - \lambda_i \frac{T(s_i)}{s_i}\right) = y^* - \lambda_i \frac{T(s_i)}{s_i} < 0.$$

Since $\lim_{\beta\to\infty} \tau_c(\beta) = \infty$, it holds

$$\lim_{n\to\infty} \tau_c(\beta_n)\left(\frac{\beta_n}{\tau_c(\beta_n)} - \lambda_i \frac{T(s_i)}{s_i}\right) = -\infty.$$

From Equation (11), it follows that

$$\lim_{n\to\infty} h_i(\beta_n) = 1.$$

The reasoning for $j \geq l(y^*)$ is analogous. □

A consequence of Lemma A.1 is that if y^* is a cluster value of $\beta/\tau_c(\beta)$, then 1 is a cluster value of $h_j(\beta)$ for all $j \leq r(y^*)$ and 0 is a cluster value of $h_j(\beta)$ for all $j \geq l(y^*)$.

We can derive results about the convergence of the hit probabilities if we know bounds for the cluster values of $\beta/\tau_c(\beta)$.

Lemma B.2. *If the set of cluster values of $\beta/\tau_c(\beta)$ is a subset of the interval $[a, b]$, then, when β diverges, $h_i(\beta)$ converges to 1, for $i < r(b)$, and to 0, for $i > l(a)$.*

Proof. For all $\epsilon > 0$, it exists a β_ϵ such that, for all $\beta > \beta_\epsilon$,

$$\frac{\beta}{\tau_c(\beta)} < b + \epsilon$$

and

$$\frac{\beta}{\tau_c(\beta)} - \lambda_i \frac{T(s_i)}{s_i} < b - \lambda_i \frac{T(s_i)}{s_i} + \epsilon.$$

For $i < r(b)$, it is $\lambda_i T(s_i)/s_i > b$ and we can choose ϵ sufficiently small so that the left term is bounded away from 0 by a negative constant for large β

$$\frac{\beta}{\tau_c(\beta)} - \lambda_i \frac{T(s_i)}{s_i} < -\delta < 0.$$

From Equation (11), it follows that, for large β,

$$1 \geq h_i(\beta) \geq \frac{1 - e^{-\lambda_i \tau_c(\beta)}}{e^{-\tau_c(\beta)\frac{s_i}{T(s_i)}\delta} + 1 - e^{-\lambda_i \tau_c(\beta)}}$$

and then $h_i(\beta)$ converges to 1 when β diverges.

The other result can be proven following a similar reasoning. □

The constraint on the expected cache's occupancy under the Che's model leads to the following result:

Lemma B.3. *If y^* is a cluster value of $\frac{\beta}{\tau_c(\beta)}$, then*

$$\sum_{i=1}^{r(y^*)} s_i \leq C \leq \sum_{i=1}^{l(y^*)-1} s_i.$$

Proof. Consider the following inequalities that are true for any value of β:

$$\sum_{i=1}^{r(y^*)} h_i s_i \leq \sum_{i=1}^{N} h_i s_i \leq \sum_{i=1}^{l(y^*)-1} s_i + \sum_{i=l(y^*)}^{N} h_i s_i.$$

Because of Equation (5), the middle term is equal to C for all β, then:

$$\sum_{i=1}^{r(y^*)} h_i s_i \leq C \leq \sum_{i=1}^{l(y^*)-1} s_i + \sum_{i=l(y^*)}^{N} h_i s_i.$$

Finally, Lemma A.1 leads to conclude that the terms h_i in the left (resp. right) sum can be made simultaneously arbitrarily close to 1 (resp. 0). □

From now on we consider that $\sum_{i=1}^{\underline{c}} s_i = C$. Bounds for the cluster values of $\beta/\tau_c(\beta)$ easily follow from Lemma A.3.

Lemma B.4. *All the cluster values of $\frac{\beta}{\tau_c(\beta)}$ are in the interval*

$$\left[\lambda_{\underline{c}+1} \frac{T(s_{\underline{c}+1})}{s_{\underline{c}+1}}, \lambda_{\underline{c}} \frac{T(s_{\underline{c}})}{s_{\underline{c}}}\right].$$

Proof. We prove it by contradiction. Let y^* be a cluster value of $\frac{\beta}{\tau_c(\beta)}$ and assume that $y^* < \lambda_{\underline{c}+1} T(s_{\underline{c}+1})/s_{\underline{c}+1}$. Then, it would be $r(y^*) \geq \underline{c} + 1$, leading to

$$C < \sum_{i=1}^{\underline{c}+1} s_i \leq \sum_{i=1}^{r(y^*)} s_i \leq C,$$

where the first inequality follows from the definition of $\underline{c}$ and the second inequality from Lemma A.3.

If we assume that $y^* > \lambda_{\underline{c}} T(s_{\underline{c}})/s_{\underline{c}}$ we arrive also to a contradiction. □

[14] It is also referred to as a cluster point or a limit point (in analogy to the corresponding concept for a sequence).

Proposition B.5. *If $\sum_{i=1}^{\underline{c}} s_i = C$, then*

$$\lim_{\beta\to\infty} h_i(\beta) = \begin{cases} 1, & \text{for } i \le \underline{c}, \\ 0, & \text{for } i > \underline{c}+1. \end{cases}$$

Proof. We first observe that, from Lemma A.2 and Lemma A.4, it immediately follows that $h_i(\beta)$ converges to 1 for $i < \underline{c}$ and to 0 for $i > \underline{c}+1$. We need to consider only $i = \underline{c}$ and $i = \underline{c}+1$.

We prove that $h_{\underline{c}+1}(\beta)$ converges to 0. Let us assume that it is not the case, then $h_{\underline{c}+1}(\beta)$ has a cluster value $h^* > 0$. Because of Lemmas A.2 and A.4 this implies that $\beta/\tau_c(\beta)$ has a cluster value in $\lambda_{\underline{c}+1}T(s_{\underline{c}+1})/s_{\underline{c}+1}$. But from Lemma A.1 it follows that it exists a diverging sequence β_n such that $\lim_{n\to\infty} h_i(\beta_n) = 1$, for all $i \le \underline{c}$. Then, for each $\epsilon > 0$, it exists an n_ϵ, such that for $n \ge n_\epsilon$,

$$C = \sum_{i=1}^{N} h_i(\beta_n)s_i \ge \sum_{i=1}^{\underline{c}+1} h_i(\beta_n)s_i \ge C + h^* s_{\underline{c}+1} - \epsilon,$$

leading to a contradiction.

We have shown that $h_{\underline{c}+1}(\beta)$ converges to 0. Because $\sum_{i=1}^{N} h_i s_i = C$, it follows that

$$h_{\underline{c}}(\beta) = \frac{C - \sum_{i\neq\underline{c}} h_i(\beta)s_i}{s_{\underline{c}}}$$

converges to 1. □

Appendix C
The Langrange method for the utility maximization problem

In this appendix, we study q_i-LRU in the cache utility maximization framework introduced in [11]. We derive the corresponding utility functions that appear in the maximization problem (6).

We look for increasing, continuously differentiable, and strictly concave functions $U_i(.)$. Moreover, we look for the following functional dependency

$$U_i(h_i) = \lambda_i s_i U_0(h_i, q_i),$$

where U_0 is increasing and concave in h_i. In what follows we will consider s_i, λ_i and q_i to be constant parameters, so that U_i and $U_0(h_i, q_i)$ are only functions of h_i.

The Lagrange function associated to problem (6) is

$$\mathcal{L}(\mathbf{h}, \alpha) = \sum_{i=1}^{N} \Big(U_i(h_i) - \alpha h_i s_i \Big) + \alpha C,$$

where $\mathbf{h}$ is the vector of the hit probabilities and α is the Lagrange multiplier associated to the constraint.

Under q_i-LRU (for finite $\beta > 0$) the hit probabilities h_i are in $(0, 1)$, because every content has some chance to be stored and no content is guaranteed to be stored. Then, if the hit probabilities of q_i-LRU are the solutions of problem (6) for a given choice of the functions $U_i(.)$, they belong to the interior part of the definition set of the concave problem (6). The hit probabilities can then be obtained by equating to 0 the Lagrangian derivatives:

$$\frac{\partial \mathcal{L}}{\partial h_i} = \frac{\mathrm{d}U_i}{\mathrm{d}h_i} - \alpha s_i = 0.$$

Therefore, from the above equation we get[15]

$$h_i = U_i'^{-1}(\alpha s_i).$$

Taking into account the specific functional dependency in Equation (A), it holds:

$$h_i = U_0'^{-1}\left(\frac{\alpha}{\lambda_i}, q_i\right).$$

We equate the expression above to that in Equation (9) and we obtain

$$\frac{1 - e^{-\lambda_i \tau_C}}{\frac{1}{q_i} e^{-\lambda_i \tau_C} + 1 - e^{-\lambda_i \tau_C}} = U_0'^{-1}\left(\frac{\alpha}{\lambda_i}, q_i\right).$$

The expressions on the LHS and the RHS depend on λ_i respectively through the products $\lambda_i \tau_C$ and λ_i/α. It follows that we should consider α proportional to $1/\tau_C$, in particular we choose:

$$\alpha = \frac{1}{\tau_C}.$$

By substituting the above equation into the formula of h_i (as given in (9)), we obtain

$$h_i = \frac{q_i(1 - e^{-\frac{\lambda_i}{\alpha}})}{e^{-\frac{\lambda_i}{\alpha}} + q_i(1 - e^{-\frac{\lambda_i}{\alpha}})}. \tag{22}$$

Next, we solve (22) with respect to α and we get

$$\alpha = \frac{\lambda_i}{\ln\left(1 + \frac{h_i}{q_i(1-h_i)}\right)}.$$

Finally, by replacing this expression for α in the equation$U_i'(h_i) = \alpha s_i$

$$U_i'(h_i) = \frac{\lambda_i s_i}{\ln\left(1 + \frac{h_i}{q_i(1-h_i)}\right)}. \tag{23}$$

By integrating (23) we obtain, for $h_i \in (0, 1]$,

$$\begin{aligned} U_i(h_i) &= -\lambda_i s_i \int_{h_i}^{1} \frac{\mathrm{d}x}{\ln\left(1 + \frac{x}{q_i(1-x)}\right)} \\ &= -\lambda_i s_i \int_{0}^{1-h_i} \frac{\mathrm{d}x}{\ln\left(1 + \frac{1-x}{q_i x}\right)}. \end{aligned}$$

The function is well defined for $h_i \in (0, 1]$, since

$$\begin{aligned} \int_{h_i}^{1} \frac{\mathrm{d}x}{\ln\left(1 + \frac{x}{q_i(1-x)}\right)} &\le \int_{h_i}^{1} \frac{\mathrm{d}x}{\ln\left(1 + \frac{x}{q_i}\right)} \\ &\le q_i \int_{1+\frac{h_i}{q_i}}^{1+\frac{1}{q_i}} \frac{\mathrm{d}y}{\ln y} < \infty. \end{aligned}$$

For $h_i \to 0^+$, the integral diverges.

[15]The existence of the inverse functions of $U_i'(\cdot)$ follows from the assumption that $U_i(\cdot)$ are strictly concave.

REFERENCES

[1] M. Garetto, E. Leonardi, and V. Martina, "A unified approach to the performance analysis of caching systems," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 1, no. 3, pp. 12:1–12:28, May 2016.

[2] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai Network: A Platform for High-performance Internet Applications," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 3, pp. 2–19, Aug. 2010.

[3] G. Neglia, D. Carra, M. D. Feng, V. Janardhan, P. Michiardi, and D. Tsigkari, "Access-time aware cache algorithms," in *Proc. of ITC-28*, September 2016.

[4] R. Barve, E. Shriver, P. B. Gibbons, B. K. Hillyer, Y. Matias, and J. S. Vitter, "Modeling and Optimizing I/O Throughput of Multiple Disks on a Bus," in *Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '99. New York, NY, USA: ACM, 1999, pp. 83–92.

[5] S. W. Ng, "Advances in Disk Technology: Performance Issues," *IEEE Computer*, vol. 31, pp. 75–81, 1998.

[6] E. G. Coffman and P. J. Denning, *Operating systems theory*, ser. Prentice-Hall series in automatic computation. Englewood Cliffs: Prentice-Hall, 1973. [Online]. Available: http://opac.inria.fr/record=b1079172

[7] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack problems.* Berlin Heidelberg: Springer, 2004.

[8] R. Fagin, "Asymptotic miss ratios over independent references," *Journal of Computer and System Sciences*, vol. 14, no. 2, pp. 222 – 250, 1977.

[9] H. Che, Y. Tung, and Z. Wang, "Hierarchical Web caching systems: modeling, design and experimental results," *Selected Areas in Communications, IEEE Journal on*, vol. 20, no. 7, pp. 1305–1314, Sep 2002.

[10] N. C. Fofack, P. Nain, G. Neglia, and D. Towsley, "Performance evaluation of hierarchical TTL-based cache networks," *Computer Networks*, vol. 65, pp. 212 – 231, 2014.

[11] M. Dehghan, L. Massoulié, D. Towsley, D. Menasche, and Y. C. Tay, "A utility optimization approach to network cache design," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, April 2016, pp. 1–9.

[12] M. Leconte, G. Paschos, L. Gkatzikis, M. Draief, S. Vassilaras, and S. Chouvardas, "Placing dynamic content in caches with small population," in *Proc. of IEEE INFOCOM 2016*, 2016.

[13] S. Li, J. Xu, M. van der Schaar, and W. Li, "Popularity-driven content caching," in *Proc. of IEEE INFOCOM 2016*, 2016.

[14] G. Neglia, D. Carra, and P. Michiardi, "Cache policies for linear utility maximization," in *Proc. of the IEEE International Conference on Computer Communications (INFOCOM 2017)*, May 2017.

[15] V. Martina, M. Garetto, and E. Leonardi, "A unified approach to the performance analysis of caching systems," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, April 2014, pp. 2040–2048.

[16] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox, "Caching Proxies: Limitations and Potentials," in *Proceedings of the Fourth International WWW Conference*, Boston,MA, December 1995.

[17] B. M. Maggs and R. K. Sitaraman, "Algorithmic nuggets in content delivery," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 3, pp. 52–66, Jul. 2015. [Online]. Available: http://doi.acm.org/10.1145/2805789.2805800

[18] M. Z. Shafiq, A. R. Khakpour, and 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.

[19] C. Fricker, P. Robert, J. Roberts, and N. Sbihi, "Impact of traffic mix on caching performance in a content-centric network," in *Proc. of IEEE INFOCOM 2012*, 2012.

[20] S. Traverso, M. Ahmed, M. Garetto, P. Giaccone, E. Leonardi, and S. Niccolini, "Temporal Locality in Today's Content Caching: Why It Matters and How to Model It," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 5, pp. 5–12, Nov. 2013.

[21] A. Araldo, D. Rossi, and F. Martignon, "Cost-aware caching: Caching more (costly items) for less (isps operational expenditures)," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1316–1330, May 2016.

[22] V. Pacifici and G. Dán, "Coordinated selfish distributed caching for peering content-centric networks," *IEEE/ACM Transactions on Networking*, vol. PP, no. 99, pp. 1–12, 2016.

[23] S. Shukla and A. A. Abouzeid, "On designing optimal memory damage aware caching policies for content-centric networks," in *14th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks, WiOpt 2016*, 2016, pp. 163–170.

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

[25] G. Bianchi, A. Detti, A. Caponi, and N. Blefari Melazzi, "Check before storing: What is the performance price of content integrity verification in LRU caching?" *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 3, pp. 59–67, 2013.

[26] G. Rossini, D. Rossi, M. Garetto, and E. Leonardi, "Multi-Terabyte and multi-Gbps information centric routers," in *INFOCOM, 2014 Proceedings IEEE*, 2014, pp. 181–189.

[27] N. E. Young, *Encyclopedia of Algorithms*. Boston, MA: Springer US, 2008, ch. Online Paging and Caching, pp. 601–604.

[28] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young, "Competitive paging algorithms," *Journal of Algorithms*, vol. 12, pp. 685–699, 1991.

[29] N. Buchbinder and J. Naor, *Online Primal-Dual Algorithms for Covering and Packing Problems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 689–701.

[30] P. Cao and S. Irani, "Cost-aware www proxy caching algorithms," in *Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems*, ser. USITS'97. Berkeley, CA, USA: USENIX Association, 1997, pp. 18–18. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267279.1267297

[31] D. Starobinski and D. Tse, "Probabilistic methods for web caching," *Performance Evaluation*, vol. 46, no. 2-3, pp. 125–137, 2001.

[32] P. R. Jelenkovic and A. Radovanovic, "Optimizing LRU Caching for Variable Document Sizes," *Comb. Probab. Comput.*, vol. 13, no. 4-5, pp. 627–643, Jul. 2004.

[33] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter, "Adaptsize: Orchestrating the hot object memory cache in a content delivery network," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI'17*, 2017.

[34] B. S. Thomson, J. B. Bruckner, and A. M. Bruckner, *Elementary Real Analysis*. Prentice-Hall, 2001.