Cache Policies for Linear Utility Maximization

Giovanni Neglia*, Damiano Carra† and Pietro Michiardi‡ * Université Côte d'Azur, Inria, giovanni.neglia@inria.fr
†University of Verona, damiano.carra@univr.it §Eurecom, pietro.michiardi@eurecom.fr

***Abstract*—Cache policies to minimize the content retrieval cost have been studied through competitive analysis when the miss costs are additive and the sequence of content requests is arbitrary. More recently, a cache utility maximization problem has been introduced, where contents have stationary popularities and utilities are strictly concave in the hit rates. This paper bridges the two formulations, considering linear costs and content popularities. We show that minimizing the retrieval cost corresponds to solving an online knapsack problem, and we propose new dynamic policies inspired by simulated annealing, including DYNQLRU, a variant of QLRU. We prove that DYNQLRUasymptotic converges to the optimum under the characteristic time approximation. In a real scenario, popularities vary over time and their estimation is very difficult. DYNQLRU does not require popularity estimation, and our realistic, trace-driven evaluation shows that it significantly outperforms state-of-the-art policies, with up to 45% cost reduction.**

I. INTRODUCTION

Cache policies have often been designed with the purpose to maximize the hit rate, but different metrics can be meaningful in different contexts: data rate to be served from the upstream caches/servers, users' delivery time, ISP/AS operational costs [1], [2], damage to flash memories in hierarchical caches [3], service time from the HDD [4], etc. Performance optimization in all these cases can be abstracted to the same problem: given some cost c_i that is paid upon a miss to retrieve content i, minimize the sum of the retrieval costs. We provide a few examples below

- $c_i = 1$: minimize the cache miss rate,
- $c_i = s_i$, the size of content: minimize the traffic from upstream servers/caches,
- $c_i = \tau_i$, the retrieval time from the server where content i is stored: minimize user's retrieval time.

Our target is to design cache policies that minimize the time-average retrieval cost, when content requests exhibit some statistical regularity. When the request process is unpredictable, this problem has been studied under the name of *File Caching* (FC) problem [5]. In this case, no algorithm can provide absolute worst-case guarantees. Instead, there exist algorithms, like GreedyDual-Size (GDS), with a known (and optimal) competitive ratio, i.e. they achieve a cost at most a given factor larger than the cost of the optimal offline algorithm that knows the sequence of future requests. We want to go beyond FC, because in many practical cases, some contents can be requested more often than others during relatively long periods of time, so that a caching algorithm can exploit such regularity and perform much better. The Independent Reference Model (IRM) corresponds to the extreme case where content popularities are constant over time and contents requests are drawn independently according to a given probability distribution.

A related problem has been formulated in [6], considering the advantages from hits rather than the disadvantages from misses. In particular the authors have defined the following *Cache Utility Maximization* (CUM) problem under the IRM and constant content size:

$$\underset{h_1,\dots h_N \in [0,1]}{\text{maximize}} \sum_{i=1}^{N} U_i(h_i), \quad \text{subject to} \sum_{i=1}^{N} h_i = B, \qquad (1)$$

where B is the cache's size, h_i is the stationary hit probability of content i and $U_i(h_i)$ is the utility associated to the hit probability. The paper shows how to derive optimal TTL-cache policies [7] when the functions U_i are increasing and *strictly concave*. The constraint in (1) can be interpreted as an *average buffer occupancy constraint*.

Our first contribution is to bridge the FC and CUM formulations, by showing that the FC problem under the IRM (our focus) corresponds to a CUM problem where the utility functions U_i are linear and the constraint takes into account content sizes. This linear case is then particularly important to study, because most of the usual cache performance metrics are additive over different misses (as shown above). Strictly concave functions are instead of interest if fairness across contents is an issue, because the optimization of linear utilities can lead to performance dishomogeneity.

The second contribution is the proposal of new dynamic policies to solve the linear utility maximization problem. We leverage the fact that a CUM problem with linear utilities corresponds to a Knapsack Problem (KP). Recognizing this parallelism does not lead to a trivial algorithm, because the optimal cache policy needs then to solve an *online* KP *under partial information* (e.g. the catalogue is not known). We design then two new dynamic algorithms, OSA and DYNQLRU, based on simulated annealing ideas, and we prove that they asymptotically store the optimal set of contents under some hypotheses. As an example of the difficulties indicated above, convergence to the optimum does not follow immediately from known results for simulated annealing. Indeed simulated annealing methods work offline and can freely explore the solution space, while in our online setting the possibility to change the current tentative solution is limited by the request process. Our analysis also leads to a novel characterization of the convergence sets of simulated annealing methods in terms of a specific potential function.

As a third contribution, we consider a realistic setting, where popularities keep varying over time. Their estimation is a very difficult task. In particular, we show through some numerical examples that estimation may require a significant amount of memory and estimation errors can jeopardize performance. For these reasons, policies that do not require to estimate popularities, like our DYNQLRU, can be more of practical interest.

TABLE I
CACHING POLICIES CONSIDERED IN THIS PAPER. FOR EACH OF THEM WE INDICATE IN WHICH SECTION IT IS DESCRIBED AND IF IT REQUIRES THE KNOWLEDGE OF CONTENT POPULARITIES.

Policy	Section	Needs Popularities	
LRU	II	No	Least Recently Used.
GDS	II	No	Greedy Dual Size. See Alg. 1
VGREEDY	III	Yes	Value Greedy. It evicts content $i = \arg\max\{p_i c_i\}$.
DGREEDY	III	Yes	Density Greedy. It evicts content $i = \arg\max\{p_i c_i/s_i\}$.
OSA	IV	Yes	Online Simulated Annealing. It mimics a simulated annealing optimization algorithm.
DYNQLRU	VI	No	Dynamic qLRU. It evicts contents as LRU. Upon a miss it admits a content with a time-varying content-dependent probability q.

In order to use DYNQLRU also in this realistic non-IRM setting, we propose a change detector that resets DYNQLRU and restarts its exploration phase when the request process appears to have significantly changed. A simple formula allows us to configure the change detector.

We use request traces from Akamai content delivery network to tune IRM parameters and validate our theoretical results. Moreover, we test the performance of DYNQLRU coupled with the change detector under the actual traces and four different realistic retrieval costs: miss ratio, upstream traffic, retrieval time and HDD load. DYNQLRU outperforms other policies like LRU or GDS always but in the case of the upstream traffic when all the policies perform equally well. Cost reduction can be as high as 45%.

The paper is organized as follows. In Sec. II we introduce the FC and CUM problems and other related works. We then formalize the retrieval minimization problem in Sec. III and prove that optimal static policies exist and they solve some specific KPs. We discuss how some heuristics for KP lead naturally to cache policies. Then, in Sec. IV we introduce the policy OSA. After having shown the difficulties to estimate popularities in Sec. V, we illustrate the policy DYNQLRU in Sec. VI and the change detector in Sec. VII. Simulation results both under IRM and real content request traces are in Sec. VIII. The policies we compare in this paper are listed in Table I and their algorithmic complexity is discussed in Sec. IX.

II. BACKGROUND AND RELATED WORKS

Let $\mathcal{N}$ denote the (potentially infinite) catalogue of contents and $\mathbf{r}_L \in \mathcal{N}^L$ a sequence of L requests for the contents. The *File Caching* (FC) problem [5] is formulated as follows: given a cache with integer size B, and files with positive integer sizes and non-negative retrieval costs, maintain in the cache files to minimize the total retrieval cost. We denote by s_i and c_i respectively the size and the cost of content $i \in \mathcal{N}$.

Let $X(n) \subseteq \mathcal{N}$ denote the state of the cache at time n, i.e. the set of the contents stored in the cache when the n-th request arrives. A possible state $\mathbf{x}$ needs to satisfy an *instantaneous buffer occupancy constraint*, i.e. $\sum_{i\in\mathbf{x}} s_i \leq B$. Then, *replacement-policies* are required to decide which contents should be evicted to make space for a new content. The retrieval cost experienced by a cache policy π under an arrival sequence $\mathbf{r}_L$ when the cache has size B is

$$C(\pi, B, \mathbf{r}_L) = \sum_{n=1}^{L} c_{r_L(n)} \mathbb{1}(r_L(n) \notin X(n)) \,. \tag{2}$$

It is always possible to find a specific sequence of content requests such that any cache policy performs arbitrarily bad. It is then standard to perform a competitive analysis [8], [9], [10]. Let π_{id} denote the ideal optimal policy that knows in advance the sequence of requests. A policy π is said to be $f(B', B)$-*competitive* if on any sequence the total retrieval cost incurred by π with a cache of size B is at most $f(B', B)$ times the cost obtained by π_{id} with a cache of size $B' \leq B$, i.e.

$$\max_{\mathbf{r}_L} \frac{C(\pi, B, \mathbf{r}_L)}{C(\pi_{id}, B', \mathbf{r}_L)} \leq f(B', B), \ \ \forall L.$$

It is possible to prove [11] that the best possible competitive ratio for any deterministic online algorithm (i.e. an algorithm that does not know the future requests) is $B/(B - B' + 1)$. In [12], [11] the algorithm GDS was proven to be $B/(B-B'+1)$-competitive and then optimal. This algorithm will be used later for comparison and is shown in Alg. 1. When the two caches have the same size, i.e. $B' = B$, the best competitive ratio is simply B, that may be huge, and then of limited interest. Nevertheless, the performance of these algorithms degrades in practice much slower than linearly with the cache size B.

Algorithm 1 GDS algorithm

Input: Sequence of content requests $\mathbf{r}$

```
W ← 0
while n ≤ |r| do
    i ← r(n)
    if i ∈ X(n) then
        H(i) ← W + c_i/s_i
    else
        while (s_i + Σ_{j∈X(n)} s_j > B) do    ▷ not enough space for content i
            W ← min_{l∈X(n)} H(l)
            arbitrarily select j|H(j) = W
            X(n) ← X(n) − {j}
        end while
        X(n) ← X(n) ∪ {i}
        H(i) ← W + c_i/s_i
    end if
    n ← n+1
end while
```

Differently from replacement-policies, *TTL-policies* associate a timer to each content and the content is evicted only when the timer expires. As a consequence, TTL-caches ideally operate with an infinite cache size and impose only an *average constraint on the buffer occupancy*, that should be equal to a given value. We denote also this value as B.[1] The timer

[1] A practical implementation will require a buffer only slightly larger than B, see [6].

of a given content may or may not be renewed upon a hit. TTL-policies were first proposed as a modeling tool to study existing replacement-policies starting from the seminal work on LRU(the policy that evicts, if needed, the least recently used content) from Fagin [13] and Che et al. [14]. In this paper we use the expression *characteristic time approximation* to denote the possibility to approximate a replacement policy with an opportunely tuned TTL-policy. This approach has been shown to be very accurate [15], [16]. More recently, the practical use of TTL-policies has been advocated because of their flexibility [7], [6]. In particular, as we mentioned in the introduction, [6] derives TTL-policies that can solve the CUM problem (1) when the utility functions U_i are strictly concave. The framework considers a finite catalogue $\mathcal{N}$ and requests arriving according to the (continuous-time) IRM: the request process is a Poisson process and a request is for content i with probability p_i (called the content popularity) independently from previous requests.

Many papers consider cache policies minimizing specific retrieval costs (e.g. [1], [2], [3], [4] mentioned in the introduction). None of them tries to address the general problem we target in this paper, but we rely on two results from our previous work [4] that do not actually depend on the specific cost considered there. There we study which set of contents $\mathcal{M}^*$ should be duplicated in the RAM in order to reduce the expected HDD workload generated from the next request, that we call the *one-step lookahead expected cost*. We prove that $\mathcal{M}^*$ is the solution of the following problem:

$$\underset{\mathcal{M}\subseteq\mathcal{N}}{\text{maximize}} \sum_{i\in\mathcal{M}} p_i c_i, \quad \text{subject to} \sum_{i\in\mathcal{M}} s_i \leq B, \tag{3}$$

i.e. *minimizing* the expected retrieval cost is equivalent to *maximizing* the objective function in (3), i.e. the utility from storing the contents $\mathcal{M}$ in the cache. We formally define the utility $\mathcal{U}$ of a set of contents $\mathcal{M}$ as

$$\mathcal{U}(\mathcal{M}) \triangleq \sum_{i\in\mathcal{M}} p_i c_i. \tag{4}$$

Problem (3) is a KP where the knapsack has capacity B and objects have value $p_i c_i$ and weight s_i. We extend this result by showing that minimizing the one-step lookahead expected retrieval cost (and then problem (3)) is actually equivalent to minimizing the time-average retrieval cost. We show a similar result when TTL-policies with average occupancy constraints are considered as in the original CUM problem. Our DYNQLRU, to be described in Sec. VI, can be considered a dynamic version of the policy q_i-LRU, proposed in [4], according to which a new content i is introduced in the cache upon a miss with a probability that depends on the ratio c_i/s_i. The idea to probabilistically differentiate content management according to the ratio c_i/s_i had already been considered in [17], where, upon a hit, content i is moved to the front of the queue with some probability $\tilde{q}_i$. Under Zipf's law for popularities, the authors prove that the asymptotic hit ratio is optimized when the probabilities $\tilde{q}_i$ are chosen to be inversely proportional to document sizes.

The interactions of caches at different ASs has been investigated through game theory in [2], where a stochastic potential "à la Young" [18] (as we do in Sec. IV) is introduced to study Nash equilibria stability. While our caching algorithms are randomized by choice (to explore the solution space), in [2] randomization is rather a collateral effect of noisy popularity estimates. Moreover, [2] does not consider the non-homogeneous dynamics rising when the noise "converges" to zero as time goes on, whereas we do.

Finally, we observe that, once the analogy between KP and caching is clearly identified, it may appear natural to explore approaches like simulated annealing to design caching policies, but, to the best of our knowledge, this was never done before. The annealed Gibbs sampler was instead used in [19] to jointly solve the AP channel selection problem and the users association problem. Moreover, we are aware that there exists a rich literature on online KP where a sequence of objects arrive over time (see e.g. [20] and references therein), but i) it relies on some assumptions that do not suit a caching application (e.g. contents cannot be removed from the knapsack once stored), and ii) the focus is on a competitive analysis as for the FC problem.

III. Retrieval cost minimization under IRM

We want to minimize the retrieval cost under the assumptions that i) the total cost is the sum of the retrieval costs due to each miss (as in FC) and ii) contents have different popularities and in particular requests follow the IRM (as in CUM). The catalogue $\mathcal{N}$ is then finite with size $N = |\mathcal{N}|$. We are interested in replacement-policies and TTL-policies that are optimal for long content request sequences. Given an infinite request sequence $\mathbf{r} = (r(1), r(2), \dots)$, we denote by $\lfloor \mathbf{r} \rfloor_n$ its subsequence containing the first n elements. It seems natural to define the cost of a policy π to be the time-average retrieval cost

$$\lim_{n\to\infty} \frac{C(\pi, B, \lfloor \mathbf{r} \rfloor_n)}{n} = \lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} c_{r(k)} \mathbb{1}(r(k) \notin X(k)), \tag{5}$$

but one may (rightly) wonder if the cost in (5) is well defined, i.e. if this limit always exists. It is indeed possible to build policies for which the average would keep oscillating. The main results of this section are that i) TTL or replacement policies minimizing the one-step lookahead expected cost also minimize the time average cost defined above and ii) they implicitly solve two related Knapsack Problems (KPs).

We first consider classic replacement-policies that satisfy the instantaneous occupancy constraint. We say that a replacement-policy π^*_{rep} is *expected-cost optimal*, if it guarantees that after a finite number of requests a set of contents $\mathcal{M}^*$, solution of problem (3), is stored in the cache almost surely (a.s.). For example, a policy that "waits" for the contents in a given $\mathcal{M}^*$ to be requested, and then stores them forever is expected-cost optimal, because any content is asked by a finite time a.s. and the set $\mathcal{M}^*$ is finite. We prove now that any of such policies π^* is optimal in the average-cost sense.[2]

[2] To stress that the request sequence is a sequence of random variables, we denote it by using capital letters.

Proposition III.1. *For any replacement-policy π_{rep}, any expected-cost optimal policy π^*_{rep}, and an IRM sequence of content requests $\mathbf{R}$ it holds*

$$\liminf_{n\to\infty} \frac{C(\pi_{rep}, B, \lfloor\mathbf{R}\rfloor_n)}{n} \geq \lim_{n\to\infty} \frac{C(\pi^*_{rep}, B, \lfloor\mathbf{R}\rfloor_n)}{n} \quad a.s. \tag{6}$$

The proof is in the supplementary material, Appendix A.

We consider now TTL-policies with an infinite buffer size and a constraint on the average buffer occupancy, i.e., $\sum_{i\in\mathcal{N}} h_i s_i = B$. A TTL-policy ($\pi_{TTL}$) is identified by the timers it associates to each content. The following results are valid both if timers are renewed or not upon a hit. We want to find the hit probabilities h^*_i that maximize the one-step lookahead expected retrieval cost for a given request. They are the solution of the following problem:

$$\underset{h_1,\dots h_N\in[0,1]}{\text{maximize}} \sum_{i\in\mathcal{N}} p_i h_i c_i, \quad \text{subject to} \sum_{i\in\mathcal{N}} h_i s_i = B. \tag{7}$$

We denote by π^*_{TTL} a TTL-policy whose timers have been selected so that the corresponding hit probability for any content i is h^*_i and we call it an *expected-cost optimal policy*.

The following proposition (whose proof is in the supplementary material, Appendix B) is the analogue of Prop. III.1 for the case of TTL policies.

Proposition III.2. *For any TTL-policy π_{TTL}, any expected-cost optimal policy π^*_{TTL}, and an IRM sequence of content requests $\mathbf{R}$ it holds*

$$\lim_{n\to\infty} \frac{C(\pi_{TTL}, B, \lfloor\mathbf{R}\rfloor_n)}{n} \geq \lim_{n\to\infty} \frac{C(\pi^*_{TTL}, B, \lfloor\mathbf{R}\rfloor_n)}{n} \quad a.s. \tag{8}$$

We have then shown that, both under instantaneous and average buffer occupancy constraints, a policy that minimizes the one-step lookahead expected retrieval cost, i.e. the expected cost from the next request, also minimizes the time-average retrieval cost. In particular, an optimal replacement-policy stores, after some finite time, the set of contents that solves the knapsack problem (3). An optimal TTL-policy stores each content i in the cache a fraction h^*_i of time, where h^*_i are solutions of problem (7). Problem (7) is an instance of the CUM problem (1), where utilities are proportional to the hit probabilities $U_i = p_i c_i h_i$.[3] The two problems (3) and (7) are strongly related because (7) is the fractional knapsack problem corresponding to a relaxation of (3).

In the rest of this paper, we focus on replacement-cache policies. Nevertheless, the characteristic time approximation and the fractional KP (7) will still make their appearance as approximate solutions. Our purpose is to design expected-cost optimal policies or good heuristics. We already mentioned a possible implementation, if an optimal solution $\mathcal{M}^*$ of problem (3) is known: store forever the contents in $\mathcal{M}^*$ as soon as they are retrieved. This policy is not practical because knowing $\mathcal{M}^*$ would require to solve the NP-hard problem (3). An additional difficulty is that in general the set of contents and their popularities p_i are not known, but we assume for the moment that this is the case and we postpone this issue until Sec. V.

For example we call VGREEDY a policy that keeps contents ordered according to their expected *value* $p_i c_i$ and removes the contents with smallest values when space is needed. We observe that, when retrieval costs are equal to 1, VGREEDY corresponds to LFU, the policy that evicts the least frequently used content. Instead, the policy DGREEDY is a policy that evicts the contents with the smallest *density* $p_i c_i / s_i$, i.e. the expected value per byte occupied in the cache. None of these policies is guaranteed to converge to a global optimum as we show in the following example.

Example 1 (DGREEDY and VGREEDY may not converge to the optimum). *Let $s_1 = 51$, $s_2 = 100$, $s_3 = s_4 = 50$, $p_1 = 0.26$, $p_2 = 0.27$, $p_3 = p_4 = 0.235$ and costs $c_i = 1$ for $i = 1, 2, 3, 4$ and $B = 100$. As soon as content 1 with value 0.26 is required,* DGREEDY *would store it and would never evict it. Similarly,* VGREEDY *would get stuck with content 2 with value 0.27. The optimal policy should instead store contents 3 and 4 with a utility $\mathcal{U}(\{3,4\}) = 0.47$.*

In the next section, we investigate if approaches based on simulated annealing can converge to the optimal solution.

IV. A SIMULATED ANNEALING APPROACH

In this section we show a new approach based on simulated annealing to design an optimal cache policy that implicitly solves the KP problem (3). Simulated annealing [21] is based on the idea of exploring in a random way the neighbourhood of a potential solution accepting occasional changes that may worsen the solution with a probability that decreases over time. The application of simulated annealing to caching is, to the best of our knowledge, new. As it will be evident from the discussion below, convergence to the optimal solution does not follow directly from standard results for simulated annealing because in this online setting we do not have the possibility to design the neighbourhood structure. The analysis is then more involved.

A. The algorithm

We start describing our policy that we call Online Simulated Annealing (OSA). Upon a miss for content i at time n, we select a set $\mathbf{v}$ of contents potentially to be evicted to free space for content i as follows. The set $\mathbf{v}$ is initially empty. We draw at random a content j among those stored in the cache and we put it in $\mathbf{v}$. If removing the contents in $\mathbf{v}$ frees enough space to store content i, we are done, otherwise we keep selecting at random other contents from the cache (without resampling) until this condition is not satisfied. Now, we actually evict the contents in $\mathbf{v}$ to store i with probability $p(i, \mathbf{v})$

$$p(i, \mathbf{v}) = \min\left(1, \frac{\sum_{j\in\mathbf{v}} p_j}{p_i}\right) \times \min\left(1, e^{\frac{\mathcal{U}(\{i\}) - \mathcal{U}(\mathbf{v})}{T(n)}}\right) \tag{9}$$

where $T(n) > 0$ is a parameter decreasing to 0 over time and $\mathcal{U}()$ is defined in Eq. (4).

[3] Additionally, different sizes are taken into account in (7), but the CUM framework developed in [6] can be immediately extended to consider such case considering the same equality constraint as in (7).

Let $\mathcal{X}$ be the set of all the possible sets of contents that can be stored at the cache, i.e. if $\mathbf{x} \in \mathcal{X}$, then $\sum_{i \in \mathbf{x}} s_i \leq B$. If the state of the cache at time n is $\mathbf{x}$ ($X(n) = \mathbf{x}$) and the object required is i ($r(n) = i$), it is possible that the state stays unchanged for example if the content i was already in the cache, or that it changes to some other state $\mathbf{z} = \mathbf{y} \cup \{i\}$ where $\mathbf{x} = \mathbf{y} \cup \mathbf{v}$, and $\mathbf{v}$ and then $\mathbf{y}$ are determined by the eviction algorithm described above. We define the neighbourhood of state $\mathbf{x}$ as all the possible states that are reachable from $\mathbf{x}$ as a consequence of the following request, and we denote it by $\mathcal{I}(\mathbf{x})$. It is evident that the policy OSA implicitly defines a non-homogeneous Markov Chain (MC) over the set $\mathcal{X}$, whose probability transition matrices we denote by $\{P(n)\}_{n\in\mathbb{N}}$. When we talk about the MC $P(n)$ we refer instead to the homogeneous MC that at any step use the transition probability matrix $P(n)$. We observe that the second term on the right hand side of (9) is always equal to 1 if $\mathcal{U}(\{i\}) \geq \mathcal{U}(\mathbf{v})$, and then if $\mathcal{U}(\mathbf{z}) = \mathcal{U}(\mathbf{y}) + \mathcal{U}(\{i\}) \geq \mathcal{U}(\mathbf{y}) + \mathcal{U}(\mathbf{v}) = \mathcal{U}(\mathbf{x})$, i.e. if the utility of the state $\mathbf{z}$ is higher than the utility of the current state $\mathbf{x}$. If this is not the case, the cache can still move to the new state with a probability exponentially decreasing in the utility loss ($0 > \mathcal{U}(\{i\}) - \mathcal{U}(\mathbf{v}) = \mathcal{U}(\mathbf{z}) - \mathcal{U}(\mathbf{x})$). Because the parameter $T(n)$ is decreasing over time, the probability to move to $\mathbf{z}$ converges to 0 over time: the algorithm will explore more the solution space at the beginning and will become more and more “greedy” as time goes on.

The policy has been designed to operate as a simulated annealing algorithm. While the neighbourhood and the transition probabilities can be arbitrarily chosen in the offline simulated annealing, here we cannot completely control them, because they depend on the request sequence. We will come back later to the consequences of such difference.

B. Convergence

As we discussed in Sec. III, we look for policies that asymptotically store a set of contents $\mathcal{M}^*$ that is solution of problem (3). Note that the objective function of problem (3) is $\mathcal{U}(\mathcal{M})$ (by definition (4)), hence we would like OSA to asymptotically store a set of contents that is a global maximizer of $\mathcal{U}()$. The average utility (or the average retrieval cost) achieved by OSA does not change if the cache state keeps changing over time, but only a vanishing fraction of time is spent in states that are not global optimizers of $\mathcal{U}()$. These observations motivate us to study which states have an asymptotical non-zero probability to be visited by the MC $\{P(n)\}_{n\in\mathbb{N}}$. We call such states *stochastically stable*.

The following theorem IV.1 provides a sufficient condition for the existence of a stationary distribution for the non-homogeneous MC $\{P(n)\}_{n\in\mathbb{N}}$, and then shows that stochastically stable sets are well defined. Moreover, the theorem relates the stationary distribution of this non-homogeneous MC to the stationary distributions of the sequence of homogeneous MCs each with (constant) probability matrix $P(n)$. Observe that for a given n, the matrix $P(n)$ identifies a homogeneous finite state MC, that is irreducible and aperiodic. Indeed, given two states $\mathbf{x}$ and $\mathbf{y}$, $\mathbf{y}$ is reachable from $\mathbf{x}$ in at most $|\mathbf{y}|$ transitions corresponding to a sequence of requests for each of the contents in $\mathbf{y}$. The chain is aperiodic because self-transitions are possible. It follows that there exists a stationary probability $\mu(n)$.

Let $P(n,k)$ denote the product $P(n)P(n+1)\dots P(n+k)$, $\Delta\mathcal{U}_{\max}$ the maximum absolute difference of utilities between two neighbouring states and b the maximum number of contents that may be stored in the cache (b depends on B and the content sizes).

Proposition IV.1. *If $T(n) = \Delta\mathcal{U}_{\max} b/\log(n)$, the non-homogeneous Markov Chain with transitions matrices $\{P(n)\}_{n\in\mathbb{N}}$ is strongly ergodic, i.e. it exists a probability vector μ such that $\lim_{k\to\infty} P_{\mathbf{x},\mathbf{y}}(n,k) = \mu_{\mathbf{y}}$ for all $\mathbf{x},\mathbf{y} \in \mathcal{X}$. Moreover, μ is the limit of the stationary distributions $\mu(n)$ of the Markov Chains $P(n)$, i.e. $\lim_{n\to\infty} \mu(n) = \mu$.*

The stochastically stable sets are the states $\mathbf{y}$ for which $\mu_{\mathbf{y}} > 0$. The proof is in the supplementary material Appendix C and it follows from standard results for simulated annealing (see e.g. [22]).

We are now ready to prove that OSA is expected-cost optimal.

Proposition IV.2. *If all the contents have the same size and $T(n) = \Delta\mathcal{U}_{\max} b/\log(n)$, the stochastically stable sets of* OSA *store all and only the contents that are included in the solution of the knapsack problem* (3).

Proof. Because of Proposition IV.1, we know that asymptotically OSA will store in the cache a set of contents corresponding to a stochastically stable state of the non-homogeneous MC $\{P(n)\}$. In order to prove the thesis we can simply show that all the stochastically stable states are global maximizers of the optimization problem considered, and then only the states with maximum utility (or equivalently minimum cost) have a positive probability to be selected by the algorithm asymptotically.

We observe that, when all the contents have the same size (say it equal to 1), the policy evicts a single element, say it j, to make space for a new content i. Let $\mathbf{x} = \mathbf{y} \cup \{j\}$ and $\mathbf{z} = \mathbf{y} \cup \{i\}$ be two neighbouring states. If $X(n) = \mathbf{x}$, the system moves to state $\mathbf{z}$ if 1) i is requested, 2) j is selected for eviction and 3) the replacement is actually accepted. The corresponding probability is then:

$$P_{\mathbf{x},\mathbf{z}}(n) = p_i \times \frac{1}{B} \times \min\left(1, \frac{p_j}{p_i}\right) \times \min\left(1, e^{\frac{\mathcal{U}(\{i\}) - \mathcal{U}(\mathbf{v})}{T(n)}}\right).$$

It can be easily checked that each homogeneous MC $P(n)$ is reversible, because the probability distribution

$$\mu_{\mathbf{x}}(n) = \frac{e^{-\frac{\mathcal{U}(\mathbf{x})}{T(n)}}}{\sum_{\mathbf{y}\in\mathcal{X}} e^{-\frac{\mathcal{U}(\mathbf{x})}{T(n)}}} \tag{10}$$

satisfies the detailed balance equation $\mu_{\mathbf{x}}(n)P_{\mathbf{x},\mathbf{z}}(n) = \mu_{\mathbf{z}}(n)P_{\mathbf{x},\mathbf{z}}(n)$. The probability distribution in (10) is then the stationary distribution of the MC $P(n)$. Because of Proposition IV.1 $\mu_{\mathbf{x}} = \lim_{n\to\infty} \mu_{\mathbf{x}}(n)$ and it is easy to verify that $\lim_{n\to\infty} \mu_{\mathbf{x}}(n) = 0$ if $\mathbf{x}$ is not a global maximizer. The thesis follows. $\square$

This optimality result has not much practical interest, because when all the contents have the same size, problem (3) can be solved in linear time. Unfortunately, it is not possible to extend this proof to the case when contents do not have the same size. The difficulty rises from the fact that the neighbourhood set is not symmetric, i.e. $\mathbf{z} \in \mathcal{I}(\mathbf{x})$ does not imply $\mathbf{x} \in \mathcal{I}(\mathbf{z})$. For example, if introducing object i requires to evict two objects from the cache, then it will not be possible to go back from $\mathbf{z}$ to $\mathbf{x}$ with a single transition. As a consequence the MC cannot be made reversible.

A few convergence results are known for simulated annealing in the non-reversible case. In [23] convergence to the optimum is proven under a *weak reversibility* condition. Weak reversibility requires that for any pair of states $\mathbf{x}$ and $\mathbf{y}$, if there is a path from $\mathbf{x}$ to $\mathbf{y}$ (i.e. a sequence of states $\mathbf{x} = \mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_p = \mathbf{y}$ such that for each $n = 1, \dots p-1$, $\mathbf{x}_{n+1} \in \mathcal{I}(\mathbf{x}_n)$) along which the utility does not go below a level L, then there is a path from $\mathbf{y}$ to $\mathbf{x}$ for which this is also true. Unfortunately this is not the case in our problem, as the following example shows.

Example 2 (Weak reversibility does not hold)**.** *Let* $s_1 = s_2 = 1$ *and* $s_3 = 2$*,* $p_1 c_1 = p_2 c_2 = 4$*,* $p_3 c_3 = 7$*,* $B = 2$*. Consider the two states* $x = \{1, 2\}$ *and* $y = \{3\}$*. The only way to move from* $\mathbf{x}$ *to* $\mathbf{y}$ *is directly (*$\mathbf{y} \in \mathcal{I}(\mathbf{x})$*), once a request for* 3 *occurs. Along this path the utility decreases from* $\mathcal{U}(\mathbf{x}) = 8$ *to* $\mathcal{U}(\mathbf{y}) = 7$*. There are two possible ways to move from* y *to* x*, corresponding to two requests for contents* 1 *and* 2*. In both cases, the system passes through an intermediate state* $\mathbf{z}$ *with utility* $\mathcal{U}(\mathbf{z}) = 4 < 7$*.*

A generalization of the weak reversibility condition is in [24], but the condition for convergence to the global maximum is implicit, because it requires to run an algorithm on the matrix embedding all the possible transitions, produce a specific set of states and check that this is a subset of the set of optimal solutions. The approach is computationally infeasible in our problem. Moreover, the same author doubts that the condition he found can be satisfied "without some form of reversibility."

In what follows we provide an alternative characterization of the states to which our algorithm converges. To the best of our knowledge, this result was never observed in the simulated annealing community. We prove that the stochastically stable sets are the global minimizers of a potential function $V(\mathbf{x})$, that is defined below. Our analysis follows the regular perturbation approach made popular by Young to study the stochastically stable equilibria in games with trembling hands [18]. Using this new characterization, we will be able to show that OSA does not converge in general to the optimum. The reader who is not interested in the characterization may skip what follows and start reading again from Sec. IV-C.

Let ϵ denote $e^{-1/T(n)}$ and $P(\epsilon)$ be the extension of $P(n)$ obtained by replacing $e^{-1/T(n)}$ by ϵ. Observe that $P(\epsilon)$ is continuous in 0, i.e.. $\lim_{\epsilon \to 0} P(\epsilon) = \lim_{n \to \infty} P(n) = P(0)$. Moreover, for each pair $\mathbf{x}$, $\mathbf{y}$, such that $P_{\mathbf{x},\mathbf{y}}(\epsilon) > 0$, there exists a non negative real number $w_{\mathbf{x},\mathbf{y}}$ such that $0 < \lim_{\epsilon \to 0} P_{\mathbf{x},\mathbf{y}}(\epsilon)/\epsilon^{w_{\mathbf{x},\mathbf{y}}} < \infty$. Under these properties $P(\epsilon)$ is called a *regular perturbation* of $P(0)$ [25].

In our setting $w_{\mathbf{x},\mathbf{y}}$ is equal to

$$w_{\mathbf{x},\mathbf{y}} = \begin{cases} 0 & \text{if } \mathcal{U}(\mathbf{y}) \geq \mathcal{U}(\mathbf{x}) \\ \mathcal{U}(\mathbf{x}) - \mathcal{U}(\mathbf{y}) & \text{otherwise.} \end{cases}$$

It is called the *resistance* of the system to move from $\mathbf{x}$ to $\mathbf{y}$. There is no resistance if the state $\mathbf{y}$ has larger utility. Otherwise, the resistance is equal to the immediate loss of utility. Let $\mathcal{G}$ be the graph corresponding to the possible transitions of $P(\epsilon)$ for $\epsilon > 0$, whose links have weight equal to the corresponding resistance of the transition. The graph $\mathcal{G}$ for Example 1 is in Fig. 1.

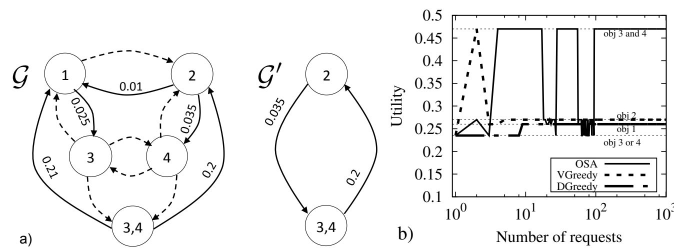

Fig. 1. Example 1: a) Resistance graphs to calculate the potentials (dashed lines indicates transitions with null resistance), b) Utility over time for different policies.

We say that $\mathbf{x}$ is a local maximizer of the function $\mathcal{U}()$ (with respect to the neighborhood relation defined above), if $\mathcal{U}(\mathbf{x}) \geq \mathcal{U}(\mathbf{z})$ for all $\mathbf{z} \in \mathcal{I}(\mathbf{x})$.

In the limit for $\epsilon \to 0$, only the transitions with null resistance are possible, and these are the transitions possible in the matrix $P(0)$. The recurrent communicating classes of $P(0)$ are the local maximizers of the function $\mathcal{U}()$. More precisely, the recurrent communicating classes contain only the local maximizers. Let $\mathcal{B}(\mathbf{x})$ be the recurrent communicating class containing the local maximizer $\mathbf{x}$. If all the states $\mathbf{y} \in \mathcal{I}(\mathbf{x})$ have smaller utility than $\mathbf{z}$, then $\mathcal{B}(\mathbf{x}) = \{\mathbf{x}\}$, i.e. the class reduces to the single point $\mathbf{x}$. Instead, if there is a state $\mathbf{z} \in \mathcal{I}(\mathbf{x})$ such that $\mathcal{U}(\mathbf{z}) = \mathcal{U}(\mathbf{x})$, then $\mathcal{B}(\mathbf{x}) = \mathcal{B}(\mathbf{z})$, i.e. both states belong to the same class.

We are going to prove that the $\lim_{\epsilon \to 0} \mu(\epsilon)$ exists and it is obviously equal to μ. Only the states in the recurrent communicating classes can be stochastically stable, but not all of them are so. We introduce a new directed graph $\mathcal{G}'$, whose nodes are the recurrent communicating classes of $P(0)$, denoted by $\mathcal{B}_1, \mathcal{B}_2, \dots \mathcal{B}_l$. The graph is full meshed and the link from $\mathcal{B}_a$ to $\mathcal{B}_b$ has weight equal to the resistance of the minimum-resistance path between any state $\mathbf{x} \in \mathcal{B}_a$ and $\mathbf{y} \in \mathcal{B}_b$ in the graph $\mathcal{G}$.[4] We denote such weight as $w_{\mathcal{B}_a,\mathcal{B}_b}$. Fig. 1, also shows the graph $\mathcal{G}'$ for Example 1, using two particular states to identify the corresponding communicating classes. Given a class $\mathcal{B}_a$ we define its potential $V(\mathcal{B}_a)$ to be the resistance of the minimum-resistance spanning tree in $\mathcal{G}'$, where from any node there is a path to $\mathcal{B}_a$. The potential

[4] The resistance of a path is defined as the sum of the resistances of each link in the path. It is immediate to check that the resistance of the minimum-resistance path does not depend on the specific states $\mathbf{x}$ and $\mathbf{y}$ chosen in the two classes.

$V(\mathcal{B}_a)$ can be considered as a global measure of the difficulty to reach a state in $\mathcal{B}_a$ from the other classes. With some abuse of notation we can define the stochastic potential of a local maximizer $\mathbf{x}$ of $\mathcal{U}()$ to be the potential of the class it belongs to, i.e. $V(\mathbf{x}) = V(\mathcal{B}(\mathbf{x}))$. The interpretation is the same: states with lower potential are easier to reach. The following result formalizes this intuition and is an immediate consequence of [25, Chapter 3, Theorem 3.1].

Proposition IV.3. *A cache state* $\mathbf{x}$ *is stochastically stable* ($\mu_{\mathbf{x}} > 0$) *if and only if* $\mathbf{x}$ *is a global minimizer for* $V()$.

A consequence of the discussion above is that all the nodes of $\mathcal{G}'$ correspond to local maximizers of $\mathcal{U}()$, and then only the local maximizers of $\mathcal{U}()$ may be stochastically stable (as it was intuitively expected). More importantly, the proposition indicates which of these local maximizers the policy OSA will converge to.

In Example 1, potentials are $V(\{3,4\}) = 0.035$ and $V(\{2\}) = 0.2$. The state $\{3,4\}$ is the unique global minimizer for the function $V()$, and by Prop. IV.3 is the only stochastically stable cache state for OSA. In this case OSA converges to state $\{3,4\}$ that is the optimal solution of problem (3). Figures 1 shows caches dynamics over time in terms of the utility of the current states for VGREEDY, DGREEDY and OSA and confirms that they respectively converge to the states $\{2\}$, $\{1\}$ and $\{3,4\}$ (we simulated 10^8 request, but there is no change after the first hundred requests).

Unfortunately the following example shows that OSA does not always converge to the optimum.

Example 3 (Convergence to the optimum may fail)**.** *Let* $s_1 = s_2 = 1$ *and* $s_3 = 2$, $p_1c_1 = p_2c_2 = 4$, $p_3c_3 = 7$, $B = 2$. *The system has four possible states:* $\mathbf{x} = \{1,2\}$, $\mathbf{y} = \{3\}$, $\mathbf{z}_1 = \{1\}$, $\mathbf{z}_2 = \{2\}$. *Among those states, only* $\mathbf{x}$ *and* $\mathbf{y}$ *are points of local maximum of* $\mathcal{U}()$ *and* $\mathbf{x}$ *is the point of global maximum. Resistances have the following values:* $w_{\mathbf{x},\mathbf{y}} = 1$, $w_{\mathbf{y},\mathbf{x}} = 3$. *It follows that there is a unique minimum-resistance spanning tree in* $\mathcal{G}'$ *and it is routed in* $\mathbf{y}$. OSA *converges to* $\mathbf{y}$ *and not to the point of global maximum.*

It is definitely interesting to study under which conditions (if any) the minimum-resistance spanning trees are rooted at global maximizers of $\mathcal{U}()$ and then optimality of OSA follows. For example, we expect it to be the case under the conditions identified in [23], [24] and we hope that our characterization may allow us to further extend such conditions. Moreover, even when the convergence to the optimum cannot be guaranteed, if the difference between the utility of the global minimizers of $V()$ and the maximum utility can be bounded, then it is possible to guarantee approximation factors for OSA. We leave this investigation for future research and we move now to more practical considerations for our original problem.

C. Quasi Weak Reversibility

Although our system is not weakly reversible in general, in typical scenarios we expect its dynamics to be close to those of a weakly reversible system and then in particular we expect OSA to converge to the global optimum of the problem or to a close point.

Our support to the previous claim originates from the success of the characteristic time approximation discussed in Sec. II. If we consider a TTL-policy mimicking OSA (as it has been done successfully for LRU, FIFO, RANDOM, QLRU..., see e.g. [16]), then the corresponding system is weakly reversible. This follows immediately from the fact that for any path from $\mathbf{x}$ to $\mathbf{y}$, e.g. $\mathbf{x} = \mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_p = \mathbf{y}$ with $\mathbf{x}_{n+1} \in \mathcal{I}(\mathbf{x}_n)$ for $n = 1, \dots p-1$, the reverted sequence of states is now a possible path from $\mathbf{y}$ to $\mathbf{x}$.

V. INTERLUDE: ESTIMATION OF CONTENT POPULARITY

All the policies described in Sections III and IV require to know content popularities p_i. A possibility is to let the policies unchanged, but replace popularities with their estimates. Unfortunately, making timely estimates of varying content popularity is a difficult task. Classic approaches essentially use compact data structures to perform autoregressive moving averages of the current number or requests for each content [26]. Results are far from being satisfactory and popularity estimation is still an open research topic itself (see for example the recent papers [27], [28]). This is one of the reasons for which simple policies like LRU are a de facto standard, even when content sizes are uniform and the key performance metric is the hit ratio.

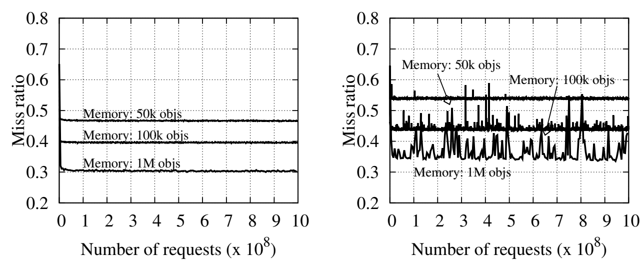

Fig. 2. Miss ratio over time for the DGREEDY (left) and the OSA (right) policies with estimated popularity: impact of the number of objects for which we maintain popularity estimates.

Here, we show that popularity estimation can be tricky even under the simple IRM. In such case, the asymptotically optimal estimator for the content request rate is simply the total number of requests divided by the observation period. If the memory available for estimation is of the order of the catalogue size ($\Theta(N)$), then it is possible to track the popularity of each content and, after some time, the estimates are precise enough for the policies to run as in the exact-knowledge case. If memory is more limited, then performance rapidly degrades. For example Fig. 2 shows the performance of DGREEDY and OSA under IRM (details in Sec. VIII) when the popularities of the W most recently requested contents are tracked. The values of W considered correspond roughly to 2, 4 and 40 times the average number of objects stored in the cache (the catalogue has 110 millions objects). A similar observation for the case when Bloom counting filters are used is in [29]: the counting error floor (due to false positives) does not allow to evaluate correctly the popularity but for the most popular m contents, where m is the number of counters used.

Given the difficulty to estimate content popularities, we would like to design a policy that does not rely on popularity estimation, but can still asymptotically store the optimal set of contents. The next section shows that this goal is feasible.

VI. How to avoid popularity estimation: DynqLRU

The new policy we propose here is a variant of qLRU including the dynamics of OSA. This policy, that we call DynqLRU is almost as simple to implement as qLRU, but inherits the convergence properties of OSA, without the need to explicitly estimate online popularities. DynqLRU works as follows. Contents are stored in a queue ordered from the most recently requested to the least recently requested object. It is more convenient in this case to consider the cache state to be this sequence. With some abuse of notation, we will still write $i \in X(n)$ to indicate that content i is stored in the cache at the time of the n-th request. If the n-th request generates a miss, the content, say i, is retrieved and inserted at the head of the queue with probability

$$q(n,i) = \frac{1}{n^{\alpha d_{\min} \frac{s_i}{c_i}}}, \tag{11}$$

where $\alpha > 0$ is an adimensional parameter and $d_{\min} = \min_{i \in \mathcal{N}} c_i/s_i$ is the minimum density across all the catalogue.[5] If space is needed to store the new content, objects are removed from the tail. Upon a hit, the content is served and moved to the front of the queue.

We observe that the policy q_i-LRU proposed in [4] stores a content in the cache upon a miss with probability $q_i = \exp\left(-\beta \frac{s_i}{c_i}\right)$ (in that paper c_i is the content retrieval time from the HDD). DynqLRU can be considered as a version of q_i-LRU where the parameter β changes over time according to $\beta(n) = \ln(n) \alpha d_{\min}$.

As for OSA, $X(n)$ can be modeled as a non-homogeneous MC with transition probability matrices $\{P(n)\}_{n \in \mathbb{N}}$. The following proposition (whose proof is in the supplementary material, Appendix D) corresponds to Prop. IV.1 for OSA, even if the proof does not follow exactly the same steps.

Proposition VI.1. *If $\alpha \leq 1/b$, the non-homogeneous Markov Chain with transitions matrices $\{P(n)\}_{n\in\mathbb{N}}$ is (strongly) ergodic, i.e. it exists a probability vector μ such that $\lim_{k\to\infty} P_{\mathbf{x},\mathbf{y}}(n,k) = \mu_{\mathbf{y}}$ for all $\mathbf{x},\mathbf{y} \in \mathcal{X}$. Moreover, μ is the limit of the stationary distributions of the Markov Chains $P(n)$, i.e. $\lim_{n\to\infty} \mu(n) = \mu$.*

Now, as in Sec. IV, we should characterize the stochastically stable states of the MC. The following result shows that under the characteristic time approximation, DynqLRU with $\alpha \leq 1/b$ converges to the solution of the fractional knapsack problem (7).

Proposition VI.2. *Under the characteristic time approximation, when $\alpha \leq 1/b$, the stochastically stable sets of* DynqLRU *store all and only the contents that are included in the solution of the fractional knapsack problem* (7).

[5] In a practical implementation, it can simply be replaced by the minimum density value seen until now.

The proof is in the supplementary material, Appendix E. This result corresponds to the optimality result for OSA in Prop. IV.2.

VII. Learning in a non-stationary setting

In the discussion above we considered a stationary content request process. Here we discuss how the policies can be adapted in a setting where content popularities vary over time. Policies like LRU or GDS are intrinsically robust to such changes. For the policies that require to know popularities, like DGreedy, VGreedy and OSA, the most natural approach is to keep dynamic estimates of popularities, for example using moving-average or autoregressive filters. This approach requires to tune the filters by estimating the timescale over which popularities may be considered constant. Moreover, the simulated annealing approaches OSA and DynqLRU explore the solution space less and less over time. The risk is to maintain stale cache states. A standard approach is to stop decreasing the parameters $T(n)$ or $q(n,i)$ when they reach a given (small) positive value, in order that some exploration is still possible. But in this case we lose the advantage of the fast initial exploration phase. Moreover, the final value has to be carefully selected for the policy to be able to follow popularity changes.

In this section, we propose a different solution that leads to a more adaptive and simpler configuration. The idea is to couple the system with a change detector to decide when to "reset" the policies, bringing them back to the initial high temperature/high q phase where they explore more. Our solution is based on the standard CUSUM sequential analysis technique to detect online changes of a system parameter [30], [31]. CUSUM computes cumulative sums of the deviation of some process samples from their expected value and it declares that a change has happened when this sum exceeds a given value. In our case, we use CUSUM to detect increases in the expected miss cost, that may suggest that popularities have changed and a new optimal set of contents to be stored need to be found.

Let $R(n)$ be the content requested by the n-th request and $C(n)$ be the corresponding cost. Hence, $C(n) = 0$ if $R(n)$ is stored in the cache and $C(n) = c_{R(n)}$ otherwise. Until no change occurs the costs $C(n)$ are assumed to be i.i.d. random variables with expected value μ_C and variance σ_C^2. We implement a one-sided CUSUM filter to detect an increase of the average cost of relative amplitude f. Algorithm 2 describes the pseudo-code. The expected value μ_C and the variance σ_C^2 are not known and are estimated through a sample average (the maximum likelihood estimator). Costs of value larger than $\hat{\mu}_C(1+f/2)$ (then $\mu_C f/2$ larger than the expected value) contribute to increase the cumulative sum S. When S is larger than the threshold h, it is assumed that a change has happened and both the dynamic policy and the CUSUM filter are reset.

The CUSUM filter requires to select two parameters f and h. As we said f corresponds to the minimum level of change in the expected cost that we want to detect. Below we consider $f = 0.1$. The threshold h allows us to trade off false positive

Algorithm 2 CUSUM change detector

Input: Sequence of costs $(C(1), C(2), \dots)$, relative change to detect (f), threshold (h)
  $n \leftarrow 1$
  while true **do**
    $k \leftarrow 1$ ▷ requests since last reset
    $\hat{\mu}_C \leftarrow 0$ ▷ estimate current expected cost
    $\hat{\sigma}^2_C \leftarrow 0$ ▷ estimate current cost variance
    $S \leftarrow 0$
    while $S \leq h$ **do**
      $S \leftarrow \left\{S + \hat{\mu}_C f/\hat{\sigma}^2_C \left(C(n) - \hat{\mu}_C(1+f/2)\right)\right\}^+$
      $\hat{\mu}_C \leftarrow (\hat{\mu}_C(k-1) + C(n))/k$
      $\hat{\sigma}^2_C \leftarrow \left(\hat{\sigma}^2_C(k-1) + (C(n)-\hat{\mu}_C)^2\right)/k$
      $k \leftarrow k+1$
      $n \leftarrow n+1$
    end while
    reset cache policy
  end while

versus false negative rates. In the supplementary material, Appendix F, we show that h can be chosen from the inequality

$$e^h - h - 1 \geq 10^{\theta/\alpha},$$

if we consider the exploration phase to be ended when probabilities decrease by a factor 10^θ.

VIII. Simulation results

In this section we evaluate the performance of the different policies using an anonymized, aggregated set of requests for objects collected over 30 days from Akamai. The actual identity of the requested objects was obfuscated, but the size of the object was known. The trace contains $2\cdot 10^9$ requests for 110 millions contents, whose size varies from few bytes to tens of MB. Figure 3 (left-hand side) shows the number of requests for each object, sorted by rank (in terms of popularity). The right-hand side shows the empirical Cumulative Distribution Function (CDF) for the size of the requested objects (without aggregating requests for the same object).

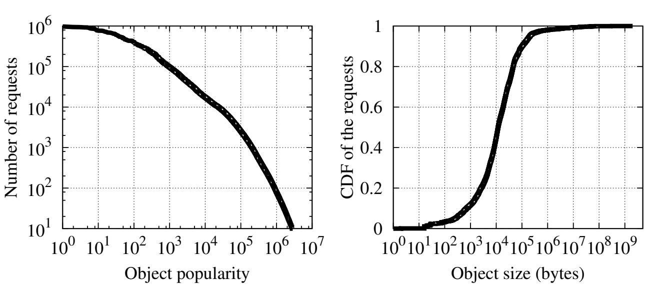

Fig. 3. Number of requests per object, ordered by rank (left), and cumulative fraction of the requests for objects up to a given size (right).

Along with each object, the traces report an additional parameter called *retrieval time*, which is the time needed to fetch the object either from the original server, the cache hierarchy, the disk or the memory, along with the necessary computation (e.g., unzipping or encoding the content). Considering the objects retrieved from the original server and the cache hierarchy, their retrieval times are an effective measure of the pressure on back-end servers each object impose, as computed by the content delivery network management system. Thus, in some of our experiments, we use as cost this retrieval time. Due to internal Akamai confidentiality policies, the retrieval time has been re-normalized to an integer between 1 and 10'000. It is important to note that the retrieval time is not necessarily correlated to object sizes: Fig. 4 shows the relation between the object size and the normalized retrieval time (each point represents an object). We have also computed the correlation coefficient between the size and the cost, obtaining a value equal to 0.013, which indicates no correlation.

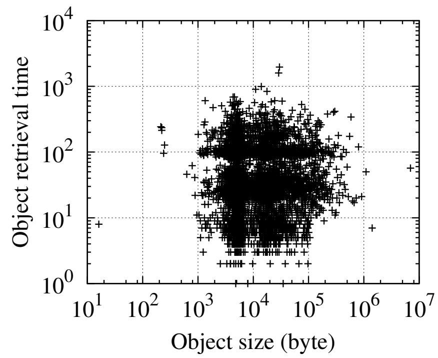

Fig. 4. Relation between object size and normalized retrieval time. Each point represents an object.

We use the trace directly (reading the request arrival times from the trace itself), and also to tune the parameters of IRM from the empirical joint popularity-size distribution.

In the previous sections we have proved that OSA and DynqLRU asymptotically store the optimal set of contents under the characteristic time approximation and provided that the parameters $T(n)$ and $q(n,i)$ decrease slow enough. In many applications the sufficient conditions for convergence can lead to a too long convergence time for realistic cache size and are then of low practical interest. Moreover, in the case of DynqLRU, when the request process is non-stationary, the policy may be reset before the admission probability has significantly decreased. In this case, if α is too small, DynqLRU will tend to behave as LRU, whose performance are someway the reference setpoint. In practice, we can work with much larger α values than $1/b$. The larger α is, the faster the admission probability decreases, but then the more likely the policy is to get stuck storing a suboptimal set of contents, until the change detector does not restart it. The sweet spot between improving on top of LRU, while not hurting performance by converging too fast needs some exploration. Similar considerations hold for OSA. In what follows we consider $T(n) = 0.001\hat{\mathcal{U}}_{max}/\log n$, where $\hat{\mathcal{U}}_{\max}$ is the maximum content utility seen until the current time. DynqLRU is configured with $\alpha = 10$, and $d_{\min}$ is set to the minimum density value seen.[6]

We start evaluating the performance of the different policies under the trace-tuned IRM, considering as target the mini-

[6]In our experiments changing α by a factor 10 was not leading to remarkable differences.

mization of the miss ratio, i.e. $c_i = 1$. For each policy, we evaluated its performance on 100 IRM request traces generated with different seeds. Each IRM trace has 10^8 requests, the miss ratio is calculated over the last 10^6 requests because we are interested in their convergence properties. We consider the ideal estimators that track the cumulative number of requests for each content ever seen.

We present results for cache sizes $B = 1$KB and $B = 1$GB (respectively in the top and bottom row of Fig. 5). When $B = 1$KB, only requests for the about 30 thousand contents with size between 1 and 10 bytes are considered. This particular scenario allow us to study a small cache for which the settings considered for OSA and DYNQLRU are closer to those that would guarantee convergence to the optimum. The left-hand side of Fig. 5 shows the empirical CDF of the miss ratio for the policies that require to estimate popularity.[7] DGREEDY achieves a small miss ratio. Indeed when objects have relatively small size in comparison to the knapsack size, the policy that greedily stores the objects with largest density is known to lead to very good approximations. OSA succeeds to find a slightly better set of contents, even if the parametrization does not allow it to consistently converge to them. The right-hand side of Fig. 5 shows the results for the policies that do not require the knowledge of popularities, DYNQLRU, GDS, and LRU, as well as the DGREEDY as a reference. DYNQLRU has a behaviour similar to OSA (not appreciable at this scale), while the policies GDS and LRU perform significantly worse.

When the cache has size 1GB and all the content requests are considered, DGREEDY achieves the lowest miss ratio as shown in the bottom row of Fig. 5. The OSA policy does not perform equally well: the temperature does not decrease slow enough to reach the optimal allocation and the policy gets stuck in some local minimizer of the miss ratio. We tried temperatures up to 100 times larger, but there was no significant improvement. On the contrary, for the largest temperature values the transient becomes so long, that performance can actually worsen: OSA is still randomly exploring the solution space at the end of the simulation. Despite of this OSA still outperforms VGREEDY policy that easily gets stuck in local minima for the miss ratio.

DYNQLRU shows performance similar to OSA, but with less variability and less sensitivity to parameter setting. The gap with DGREEDY has the same explanation. On the other hand, DYNQLRU outperforms both GDS and LRU, whose miss ratios are respectively between 40% and 60% and between 75% and 100% larger than those of DYNQLRU.

From now on, we compare the policies using directly the actual trace. We illustrated in Sec. V the difficulty to estimate popularities online. Here we provide an additional experiment, comparing the performance of DGREEDY, the "winner" under IRM, with those of DYNQLRU coupled with a CUSUM (configured as described in Sec. VII with $f = 0.1$ and $\theta = 2$). For DGREEDY the average request rate of *each* content ever seen is maintained. Note that a comparison of popularities would require ideally to update all the estimated request rates at the arrival of each request, that may not be feasible.

[7] Remember that in this case VGREEDY corresponds to LFU.

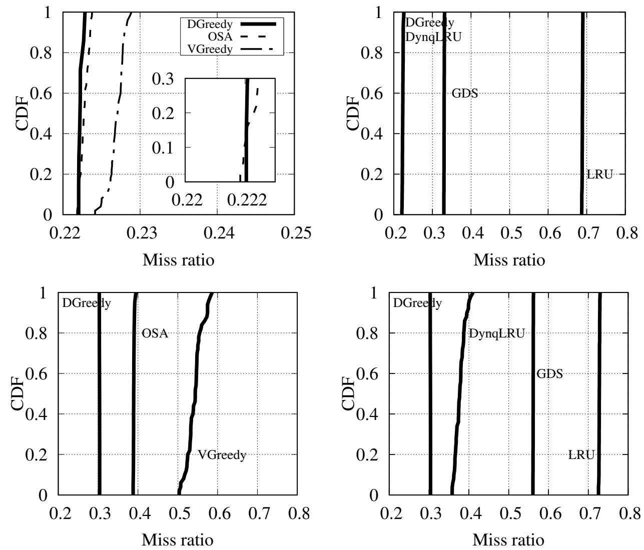

Fig. 5. Miss ratio over time for B=1KB (top) and B=1GB (bottom), policies with known object popularity (left) and unknown object popularity (right). In both cases we use DGREEDY as a reference, which requires the popularity to be known.

Figure 6 shows the miss ratio over time for two different DGREEDY settings. In the first one, the request rate for a content is updated only at the arrival of a request for that content. In the second one, all the estimates are *also* updated every 10^7 requests, i.e. every 6 hours. The corresponding plots are respectively labeled without/with updates. The experiment shows that even when memory for estimation is not a concern, computation constraints may affect the popularity estimation quality, to the point that the result in Fig. 5 may be reversed and DYNQLRU may perform better than DGREEDY.

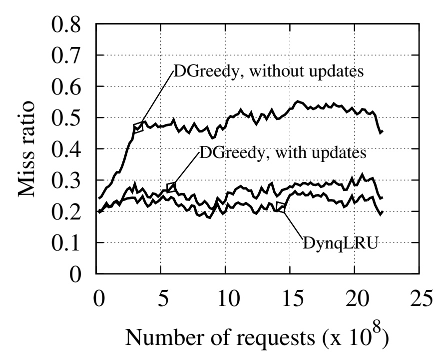

Fig. 6. Impact of the popularity on DGREEDY policy: no updates in the estimate, with updated, and comparison with DYNQLRU.

In the following we show the results for the DYNQLRU, GDS, and LRU policies and four different retrieval costs: the miss ratio, the upstream traffic, the retrieval time from the server, and the HDD load. The upstream traffic is the amount of data to be retrieved by parent caches or the authoritative content servers, it corresponds to setting $c_i = s_i$. For the retrieval time, the cost c_i is the average retrieval time for content i as measured in the Akamai network we consider.

Finally for the HDD load, the cost of i is the work imposed to the HDD to retrieve content i. We have estimated it as a function of the content size and HDD characteristics using the following empirical formula proposed in [4]:

$$T(s_i) = (\sigma + \rho)\left\lceil \frac{s_i}{b} \right\rceil + \left(\frac{1}{\mu} + \sigma_r\right) s_i + \phi, \qquad (12)$$

where σ denotes the average seek time, ρ the average rotation time, μ the transfer speed, σ_r the seek time for read, ϕ the controller overhead and b the block size. All the metrics have been normalized to 1, by dividing them from the cost that would be incurred if the cache were not present. Results in Fig. 7 show significant improvement from DYNQLRU, but for the upstream traffic, for which all the policies have almost the same performance. Average cost reductions in comparison to the second best policy range from 15% for the HDD load up to 30% for the retrieval time and 45% for the miss ratio.

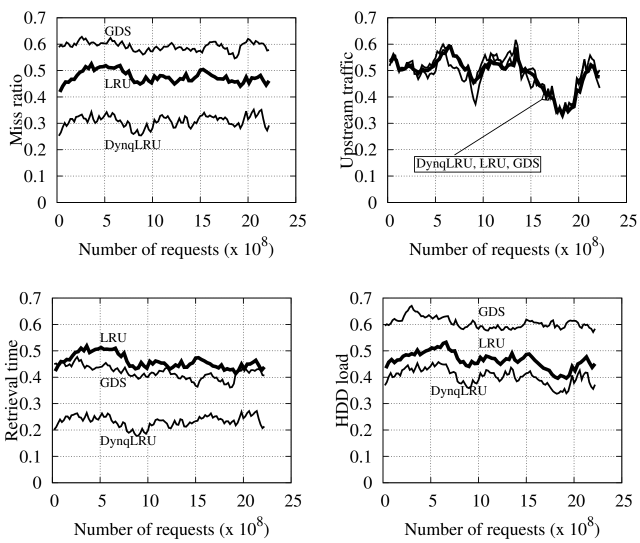

Fig. 7. Miss ratio (top-left), upstream traffic (top-right), retrieval time from origin (bottom-left) and HDD load (bottom-right).

We observe that both DYNQLRU and GDS rely on the knowledge of the size and the retrieval cost of each content, while LRU does not use such information. Note that the size s_i is required when content i is at the cache, and then this information is immediately available. The retrieval cost c_i can correspond to different physical quantities depending on what the policy is trying to optimize, as discussed above using 4 different examples. For two of them c_i is immediately known at the cache. Indeed for the miss cost, we have $c_i = 1, \forall i$, for the upstream traffic we have $c_i = s_i$. For the HDD load, the cost can be simply computed using (12). Regarding the retrieval times, if they are relatively constant, they are parameters that could be set directly by the CDN operator, e.g. as a function of the location of the corresponding content provider. On the contrary, if they vary because of network or servers' congestion, the cache can keep running estimates of c_i, by measuring the time needed to retrieve the content upon each miss.

IX. COMPLEXITY

DYNQLRU differs from LRU only for its admission policy, that requires to compute the probability $q(n,i)$ and to generate a random number. Its complexity is then $\mathcal{O}(1)$ as for LRU. OSA requires additionally to be able to randomly access elements to evict, but also this procedure requires only $\mathcal{O}(1)$ time. VGREEDY, DGREEDY and GDS require to extract the smallest element among a given set of numbers: an implementation using a heap would lead to $\mathcal{O}(\log b)$ complexity where b is the number of contents in the cache. For VGREEDY, when costs are expressed by integer values, one can adapt the LFU implementation in [32] that takes $\mathcal{O}(1)$ time for each operation.

X. CONCLUSIONS

In this paper we have bridged the two cache utility maximization frameworks proposed until now and proved that when costs are linear over the misses and requests follow the IRM, an optimal policy solves online a knapsack problem. We have proposed two new policies based on simulated annealing that are optimal. Experiments on real traces show that DYNQLRU outperforms both LRU and the competitive-ratio-optimal GDS.

REFERENCES

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

[2] V. Pacifici and G. Dán, "Coordinated selfish distributed caching for peering content-centric networks," *IEEE/ACM Trans. on Networking*, 2016.

[3] S. Shukla and A. A. Abouzeid, "On designing optimal memory damage aware caching policies for content-centric networks," in *Proc. of WiOpt 2016*, 2016, pp. 163–170.

[4] 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.

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

[6] M. Dehghan, L. Massoulié, D. Towsley, D. Menasche, and Y. Tay, "A Utility Optimization Approach to Network Cache Design," in *Proc. of IEEE INFOCOM 2016*, 2016.

[7] 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.

[8] 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.

[9] N. Buchbinder and S. Naor, "Online primal-dual algorithms for covering and packing problems," in *Proc. of 13th Annual European Symposium on Algorithms (ESA 2005)*, 2005.

[10] S. Albers, "Competitive online algorithms," BRIC, Lecture Series LS-96-2, 1996.

[11] N. E. Young, "On-line file caching," *Algorithmica*, vol. 33, no. 3, pp. 371–383, 2002.

[12] P. Cao and S. Irani, "Cost-aware www proxy caching algorithms," in *Proc. of the USENIX USITS*, 1997.

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

[14] 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.

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

[16] 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.

[17] 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.

[18] H. P. Young, "The Evolution of Conventions," *Econometrica*, vol. 61, no. 1, pp. 57–84, January 1993.

[19] B. Kauffmann, F. Baccelli, A. Chaintreau, V. Mhatre, K. Papagiannaki, and C. Diot, "Measurement-based self organization of interfering 802.11 wireless access networks," in *Proc. of IEEE INFOCOM 2007*, May 2007, pp. 1451 –1459.

[20] H.-J. Böckenhauer, D. Komm, R. Královič, and P. Rossmanith, "The online knapsack problem: Advice and randomization," *Theor. Comput. Sci.*, vol. 527, pp. 61–72, Mar. 2014.

[21] P. J. M. Laarhoven and E. H. L. Aarts, Eds., *Simulated Annealing: Theory and Applications*. Norwell, MA, USA: Kluwer Academic Publishers, 1987.

[22] S. Anily and A. Federgruen, "Simulated Annealing method with general acceptance probabilities," *Journal of Applied Probability*, vol. 24, no. 3, pp. 657–667, 1987.

[23] B. Hajek, "Cooling schedules for optimal annealing," *Mathematics of Operations Research*, vol. 13, May 1988.

[24] J. N. Tsitsiklis, "Markov chains with rare transitions and simulated annealing," *Math. Oper. Res.*, vol. 14, no. 1, pp. 70–90, 1989.

[25] H. Young, *Individual Strategy and Social Structure: An Evolutionary Theory of Institutions*. Princeton University Press, 2001.

[26] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet Math.*, vol. 1, no. 4, pp. 485–509, 2003.

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

[28] 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.

[29] G. Bianchi, K. Duffy, D. J. Leith, and V. Shneer, "Modeling conservative updates in multi-hash approximate count sketches," in *Proc. of ITC-24*, 2012.

[30] E. S. Page, "Continuous Inspection Schemes," *Biometrika*, vol. 41, no. 1-2, pp. 100–115, 1954.

[31] P. Granjon, "The CUSUM algorithm a small review," 2012. [Online]. Available: http://chamilo2.grenet.fr/inp/courses/ENSE3A35EMIAAZ0/document/change_detection.pdf

[32] K. Shah, A. Mitra, and D. Matani, "An $O(1)$ algorithm for implementing the LFU cache eviction scheme," 2010. [Online]. Available: http://dhruvbird.com/lfu.pdf

[33] D. Williams, *Probability with Martingales*. Cambridge University Press, 1991.

[34] S. Anily and A. Federgruen, "Ergodicity in parametric non stationary Markov chains: An application to simulated annealing methods," *Operations Research*, vol. 35, no. 6, pp. 867–874, 1987.

[35] P. Brémaud, *Markov chains : Gibbs fields, Monte Carlo simulation and queues*. New York, Berlin, Heidelberg: Springer, 1999.

[36] M. Basseville and I. V. Nikiforov, *Detection of Abrupt Changes: Theory and Application*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993.

Giovanni Neglia received the master's degree in electronic engineering and the PhD degree in telecommunications from the University of Palermo, Italy, in 2001 and 2005, respectively. He has been a researcher at Inria, Sophia Antipolis, France, since September 2008. In 2005, he was a research scholar with the University of Massachusetts, Amherst, visiting the Computer Networks Research Group. Before joining Inria, he was a post-doctorate with the University of Palermo and an external scientific advisor with the Maestro Team at Inria. His research is focused on modeling and performance evaluation of networks.

Damiano Carra received his Laurea in Telecommunication Engineering from Politecnico di Milano, and his Ph.D. in Computer Science from University of Trento. He is currently an Assistant Professor in the Computer Science Department at University of Verona. His research interests include modeling and performance evaluation of large scale distributed systems.

Pietro Michiardi received his M.S. in Computer Science from EURECOM and his M.S. in Electrical Engineering from Politecnico di Torino. Pietro received his Ph.D. in Computer Science from Telecom ParisTech (former ENST, Paris), and his HDR (Habilitation) from UNSA. Today, Pietro is a Professor of Computer Science at EURECOM, where he leads the Distributed System Group, which blends theory and system research focusing on large-scale distributed systems (including data processing and data storage), and scalable machine learning algorithms. Currently, Pietro is the Head of the Data Science Department.

Supplementary material for the article

Cache Policies for Linear Utility Maximization

APPENDIX A
PROOF OF PROPOSITION III.1

We first prove that the LHS and the RHS are well defined. The limit inferior in the LHS always exists because $C(\pi, B, \lfloor \mathbf{R} \rfloor_n)/n \geq 0$. For the limit in the RHS, observe that with probability 1 there is a request m such that $X(n) = \mathcal{M}^*$ for $n \geq m$. The status of the cache in the first $m-1$ timeslots does not affect the limit, we can simply consider that the cache always stored the contents in $\mathcal{M}^*$. By the strong law of large numbers it follows then that

$$\lim_{n\to\infty} \frac{C(\pi^*, B, \lfloor \mathbf{R} \rfloor_n)}{n} = \sum_{i \notin \mathcal{M}^*} p_i c_i.$$

We observe that

$$\sum_{i \notin \mathcal{M}^*} p_i c_i = \sum_{i \in \mathcal{N}} p_i c_i - \sum_{i \in \mathcal{M}^*} p_i c_i = \mathcal{U}(\mathcal{N}) - \mathcal{U}(\mathcal{M})$$

and similarly

$$\begin{aligned}\frac{C(\pi, B, \lfloor \mathbf{r} \rfloor_n)}{n} &= \frac{1}{n}\sum_{k=1}^{n} c_{r(n)} \mathbb{1}(r(n) \notin X(n)) \\ &= \frac{1}{n}\sum_{k=1}^{n} c_{r(n)} - \frac{1}{n}\sum_{k=1}^{n} c_{r(n)} \mathbb{1}(r(k) \in X(k)).\end{aligned}$$

The first term converges by the strong law of large numbers to the expected cost per request, i.e. to $\mathcal{U}(\mathcal{N})$. If follows then that (6) is equivalent to

$$\limsup_{n\to\infty} \frac{1}{n}\sum_{k=1}^{n} c_{r(k)} \mathbb{1}(r(k) \in X(k)) \leq \mathcal{U}(\mathcal{M}^*) \quad a.s. \tag{13}$$

If the states $X(n)$ were independent from the request sequence, the result would follow immediately by the strong law of large numbers for independent r.v.s and the fact that $\mathcal{M}^*$ is a solution of problem (3), but this is not the case. We are going to define some auxiliary supermartingales.

Let $Y_n \triangleq c_{r(n)} \mathbb{1}(r(k) \in X(k)) - \mathcal{U}(X(k))$, then $\mathbf{E}[Y_n] = 0$. Moreover, the variance of Y_n is finite for each n, in particular $\mathrm{Var}(Y_n) \leq c_{\max}^2$, where $c_{\max} \triangleq \max_{i\in\mathcal{N}}\{c_i\}$. The stochastic process defined by $M_0 = 0$ and $M_{n+1} = M_n + Y_{n+1}$ is a martingale relative to the filtration $\{\mathcal{F}_n, n = 1, 2, \dots\}$ induced by the request process. In fact $\mathbf{E}[|M_n|] < \infty$ and $\mathbf{E}[M_n|\mathcal{F}_{n-1}] = M_{n-1}$ for each n. Because of the Pythagoras's theorem for martingales [33, Sec. 12.1], it holds $\mathbf{E}[M_n^2] \leq n c_{\max}^2$.

We consider now the stochastic process $S_n = M_n/n$. From what we proved for the process M_n, it follows that $\mathbf{E}[S_n] = 0$ and it variance converges to 0 because $\mathrm{Var}(S_n) \leq c_{\max}^2/n$. The process S_n can also been written recursively as $S_0 = 0$ $S_{n+1} = S_n n/(n+1) + Y_{n+1}/(n+1)$. It holds

$$\mathbf{E}[S_{n+1}|\mathcal{F}_n] = \mathbf{E}[S_n|\mathcal{F}_n]\frac{n}{n+1} < \mathbf{E}[S_n|\mathcal{F}_n],$$

and then S_n is a supermartingale. Moreover, S_n is $\mathcal{L}^1$ bounded because $\sup_n \mathbf{E}[|S_n|] \leq c_{\max}$. Doob's convergence theorem [33, Th. 11.5] guarantees that, almost surely $\lim_{n\to\infty} S_n$ exists and is finite. We denote by S_∞ the limit r.v.. By Fatou-Lebesgue theorem it follows that $\mathbf{E}[S_\infty] = \mathbf{E}[\lim_{n\to\infty} S_n] = \lim_{n\to\infty} \mathbf{E}[S_n] = 0$ and $\mathrm{Var}[S_\infty] \leq \liminf_{n\to\infty} \mathrm{Var}[S_n] = 0$. Then S_∞ is a.s. the constant 0. In conclusion we have proved that

$$\lim_{n\to\infty} \frac{1}{n}\sum_{k=1}^{n}\left(c_{r(n)} \mathbb{1}(r(k) \in X(k)) - \mathcal{U}(X(k))\right) = 0 \quad a.s. \tag{14}$$

We are now ready to prove Eq. (13) by contradiction. If Eq. (13) were not true, there would exist a diverging sequence n_m such that

$$\lim_{m\to\infty} \frac{1}{n_m}\sum_{k=1}^{n_m} c_{r(k)} \mathbb{1}(r(k) \in X(k)) > \mathcal{U}(\mathcal{M}^*) \quad a.s. \tag{15}$$

It holds:

$$\begin{aligned}\lim_{m\to\infty} \frac{1}{n_m}\sum_{k=1}^{n_m} c_{r(k)} \mathbb{1}(r(k) \in X(k)) &= \lim_{m\to\infty} \frac{1}{n_m}\sum_{k=1}^{n_m} \mathcal{U}(X(k)) \\ &\leq \lim_{m\to\infty} \frac{1}{n_m}\sum_{k=1}^{n_m} \mathcal{U}(\mathcal{M}^*) = \mathcal{U}(\mathcal{M}^*) \end{aligned} \tag{16}$$

where the first equality follows from Eq. (14) and the inequality from $\mathcal{M}^*$ being the solution of Problem (3). Equation (16) contradicts (15) and then the thesis follows.

APPENDIX B
PROOF OF PROPOSITION III.2

Proof. The proof is simpler than that of Prop. III.1 because in this case contents management at the cache are decoupled. It holds

$$\begin{aligned}\lim_{n\to\infty} \frac{C(\pi_{TTL}, B, \lfloor \mathbf{R} \rfloor_n)}{n} &= \sum_{i\in\mathcal{N}} \lim_{n\to\infty} \frac{1}{n}\sum_{k=1}^{n} c_i \mathbb{1}(R(k) \neq i) \\ = \sum_{i\in\mathcal{N}} c_i(1-h_i) &\geq \sum_{i\in\mathcal{N}} c_i(1-h_i^*) = \lim_{n\to\infty} \frac{C(\pi_{TTL}^*, B, \lfloor \mathbf{R} \rfloor_n)}{n}\end{aligned}$$

where h_i is the occupancy/hit probability for content i. The second equality follows from standard renewal arguments and the inequality from h_i^* being a solution of (7). □

APPENDIX C
PROOF OF PROPOSITION IV.1

Proof. The acceptance probabilities $t_{\mathbf{x},\mathbf{y}}(n)$ can be lower bounded as follows

$$t_{\mathbf{x},\mathbf{y}}(n) \geq \underline{t}(n) = e^{-\frac{\Delta \mathcal{U}_{\max}}{T(n)}} \quad \forall \mathbf{x}, \mathbf{y} \in \mathcal{X},$$

and it holds

$$\sum_{k=1}^{\infty} \underline{t}(kb)^b = \sum_{k=1}^{\infty}\left(e^{-\frac{\log(bk)}{b}}\right)^b = \sum_{k=1}^{\infty} \frac{1}{bk} = \infty.$$

The result follows from [34, Theorem 2]. □

Appendix D
Proof of Prop. VI.1

Proof. We first prove that the MC is weakly ergodic.

Let y_i denote the i-th element of the sequence $\mathbf{y}$. Given two states $\mathbf{x}$ and $\mathbf{y}$, it is always possible to move from $\mathbf{x}$ to $\mathbf{y}$ in at most b steps. for example if the following sequence of content requests occurs: $y_{|\mathbf{y}|}$, $y_{|\mathbf{y}|-1}$, ..., y_1 and all these contents are stored in the cache (if not already present), followed by $b-|\mathbf{y}|$ further requests for content y_1. The probability that a given content in the cache is requested at time n and it is then stored in the cache is at least $p_{\min} 1/n^\alpha$, where $p_{\min} = \min_{i\in\mathcal{N}}\{p_i\}$ is the minimum popularity. Then the probability to move from state $\mathbf{x}$ to state $\mathbf{y}$ between step nb and step $(n+1)b$ is bounded as follows

$$P_{\mathbf{x},\mathbf{y}}(nb,(n+1)b) \geq \left(p_{\min}\frac{1}{((n+1)b)^\alpha}\right)^b. \qquad (17)$$

Remember that the Dobrushin's index of a (finite) transition matrix A with state space $\mathcal{X}$ is defined as follows

$$\delta(A) = 1 - \min_{\mathbf{x},\mathbf{y}\in\mathcal{X}} \sum_{\mathbf{k}\in\mathcal{X}} \min(A_{\mathbf{x},\mathbf{k}}, A_{\mathbf{y},\mathbf{k}})$$

Then from bound (17), it follows

$$\delta(P(nb,(n+1)b)) \leq 1 - |\mathcal{X}|\frac{p_{\min}^b}{b^{b\alpha}}\frac{1}{(n+1)^{\alpha b}}$$

and

$$\sum_{n=0}^{\infty}(1-\delta(P(nb,(n+1)b))) \geq |\mathcal{X}|\frac{p_{\min}^b}{b^{b\alpha}}\sum_{n=0}^{\infty}\frac{1}{(n+1)^{\alpha b}},$$

but this series is divergent whenever $\alpha b \leq 1$. It follows from the block criterion [35, Ch. 6, Th. 8.2] that the MC is weakly ergodic whenever $\alpha \leq 1/b$.

We now move to prove strong ergodicity. We consider that costs c_i can be expressed by integer values, and we let γ denote the least common multiple of the set of costs $\gamma = LCM\{c_i, i \in \mathcal{N}\}$. Consider that the variable n can assume any positive real number value and define the matrix function over $(0,1]$ as follows $\bar{P}(a) = P\left(1/a^{\frac{\gamma}{d_{\min}\alpha}}\right)$. $\bar{P}(a)$ is a *regular extension* of the matrix $P(n)$ [34, Def. 1]. Moreover it can be checked that it is polynomial in the variable a and then all its entries belong to a closed class of asymptotically monotone functions (CAM) [34, Def. 3]. These properties of the regular extension $\bar{P}(a)$, together with the weak ergodicity of the MC {P(n)} imply strong ergodicity of the MC [34, Th. 2]. Moreover, for n large enough there is a unique stationary distribution $\mu(n)$ of the homogeneous MC $P(n)$, and $\lim_{n\to\infty}\mu(n) = \mu$. □

Appendix E
Proof of Prop. VI.2

Proof. Without loss of generality we assume that contents are ordered so that $\lambda_i c_i/s_i > \lambda_j c_j/s_j$ for $i < j$. Moreover, let $\hat{b}$ be the largest index value such that $\sum_{i=1}^{\hat{b}-1} s_i \leq B$ and $\sum_{i=1}^{\hat{b}+1} s_i > C$. Let $\mathcal{A}^*$ be the set of stochastically stable states of DynqLRU. The probability h_i to find content i asymptotically in the cache is

$$h_i = \sum_{\mathbf{x}\in\mathcal{X}|i\in\mathbf{x}} \mu_{\mathbf{x}} = \sum_{\mathbf{x}\in\mathcal{A}^*|i\in\mathbf{x}} \mu_{\mathbf{x}}.$$

It follows that 1) if i has null hit probability, all the states $\mathbf{x}$ containing i have zero probability and then they are not stochastically stable, and 2) if i has positive hit probability, it needs to belong to at least one stochastically stable state. Then, the stochastically stable states contain all and only the contents that have a positive hit probability asymptotically.

When n diverges, $\beta(n) = \ln(n)\alpha d_{\min}$ diverges and it has been proved in [4, Prop III.1] that, under Che's approximation, the hit probabilities converge to the solution of the fractional knapsack problem (7)

$$\lim_{n\to\infty} h_i^* = \begin{cases} 1 & \text{if } i < \hat{b} \\ 0 & \text{if } i > \hat{b} \\ \frac{C-\sum_{i=1}^{\hat{b}-1} s_i}{s_{\hat{b}}} & \text{for } i = \hat{b} \end{cases}$$

Combining the two remarks the thesis follows. □

Appendix F
CUSUM configuration

It is usual to express the performance of CUSUM filters in terms of the Average Run Length (ARL), i.e. the expected number of requests before a reset. In particular, one distinguishes the ARL under the hypothesis that no change happened (ARL_0) or that a change happened (ARL_1). ARL_0 quantifies how often false positives occur, while ARL_1 corresponds to the delay before a change is detected. Ideally we would like ARL_0 to be large and ARL_1 to be small, but the two goals are conflicting. The threshold h allows us to trade off the two conflicting issues. In our case we want ARL_0 to be longer than a characteristic timescale of the exploration process of the dynamic policy to avoid false positive to reset the policy when it is still in the exploration phase. We can define such timescale as the number of requests required for the policy to reduce the probability values of a factor 10^θ for the contents with the smallest density c_i/s_i. A typical value for θ may be 2. Then the characteristic exploration timescale of DynqLRU is $10^{\theta/\alpha}$. We want to select h so that $\text{ARL}_0 \geq 10^{\theta/\alpha}$. The exact expression of ARL_0 requires to solve some complex integral equations [30]. Here we adopt the Wald's approximation [36, Chapter 5, eq. (5.2.44)]:

$$\text{ARL}_0(h) \approx \frac{1}{\mathbf{E}[\Delta S]}\left(h + \frac{e^{-\omega_0 h}}{\omega_0} - \frac{1}{\omega_0}\right),$$

where $\Delta S = \hat{\mu}_C f/\hat{\sigma}_C^2\,(C(n) - \hat{\mu}_C(1+f/2))$ and ω_0 is the unique non-zero solution of $\mathbf{E}[e^{-\omega_0\Delta S} = 1]$. Approximating $C(n)$ with a gaussian variable, we obtain

$$\mathbf{E}[e^{-\omega_0\Delta S}] = \exp\left(-\omega_0\mathbf{E}[\Delta S] + \omega_0\frac{\text{Var}(\Delta S)}{2}\right).$$

It holds $\mathbf{E}[\Delta S] = \frac{1}{2}\left(\frac{\mu_C f}{\sigma_C}\right)^2$ and $\mathrm{Var}(\Delta S) = \left(\frac{\mu_C f}{\sigma_C}\right)^2$. Then the unique non-zero solution of $\mathbf{E}[e^{-\omega_0 \Delta S} = 1]$ is $\omega_0 = -1$. Imposing $\mathrm{ARL}_0(h) \geq 10^{\theta/\alpha}$, we obtain

$$e^h - h - 1 \geq \frac{1}{2}\left(\frac{\mu_C f}{\sigma_C}\right)^2 10^{\theta/\alpha}.$$

In practical settings content retrieval costs exhibit high variability so that $\mu_C/\sigma_C << 1$, and we can consider the simpler inequality:

$$e^h - h - 1 \geq 10^{\theta/\alpha},$$

from which h can easily be determined.