Similarity Caching: Theory and Algorithms

Michele Garetto, Emilio Leonardi, Giovanni Neglia

Similarity Caching: Theory and Algorithms

Michele Garetto
Università di Torino, Italy
michele.garetto@unito.it

Emilio Leonardi
Politecnico di Torino, Italy
emilio.leonardi@polito.it

Giovanni Neglia
Inria, Université Côte d'Azur, France
giovanni.neglia@inria.fr

***Abstract*—This paper focuses on similarity caching systems, in which a user request for an object o that is not in the cache can be (partially) satisfied by a similar stored object o', at the cost of a loss of user utility. Similarity caching systems can be effectively employed in several application areas, like multimedia retrieval, recommender systems, genome study, and machine learning training/serving. However, despite their relevance, the behavior of such systems is far from being well understood. In this paper, we provide a first comprehensive analysis of similarity caching in the offline, adversarial, and stochastic settings. We show that similarity caching raises significant new challenges, for which we propose the first dynamic policies with some optimality guarantees. We evaluate the performance of our schemes under both synthetic and real request traces.**

***Index Terms*—Caching systems, similarity searching**

I. Introduction

Caching at the network edge plays a key role in reducing user-perceived latency, in-network traffic, and server load. In the most common setting, when a user requests a given object o, the cache provides o if locally available (hit), and retrieves it from a remote server (miss) otherwise. In other cases, a user request can be (partially) satisfied by a similar object o'. For example, a request for a high-quality video can still be met by a lower resolution version. In other scenarios, a user query is itself a query for objects similar to a given object o. This situation goes under the name of *similarity searching*, proximity searching, or also metric searching [1]. Similarity searching plays an important role in many application areas, like multimedia retrieval [2], recommender systems [3], [4], genome study [5], machine learning training [6], [7], [8], and serving [9], [10]. In all these cases, a cache can deliver to the user one or more objects similar to o among those locally stored, or decide to forward the request to a remote server. The answer provided by the cache is in general an *approximate* one in comparison to the best possible answer the server could provide. Following the seminal papers [2], [3], we refer to this setting as *similarity caching*, and to the classic one as *exact caching*.

To the best of our knowledge, the first paper introducing the problem of caching for similarity searching was [2]. The authors considered how caches can improve the scalability of content-based image retrieval systems. Almost at the same time, [3] studied caches in content-match systems for contextual advertisement. Both papers propose some simple modifications to the least recently used policy (LRU) to account for the possibility of providing approximate answers. More recently, in [6] and [7], similarity caching has been used to retrieve similar feature vectors from a memory unit with the goal of improving the performance of sequence learning tasks. Previous approaches led to the concept of memory-augmented neural networks [8]. Clipper [10]—a distributed system to serve machine learning predictions—includes similarity caches to provide low-latency, approximate answers. A preliminary evaluation of the effect of different caching strategies for this purpose can be found in [9]. Recently, [4] and a series of papers by the same authors have studied recommendation systems in a cellular setting, where contents can be stored close to the users. They focus on how to statically allocate the contents in each cache assuming to know the contents' popularities and the utility for a user interested in content o to receive a similar content o'.

Exact caching has been studied for decades in many areas of computer science, and there is now a deep understanding of the problem. Optimal caching algorithms are known in specific settings, and general approaches to study caching policies have been developed both under adversarial and stochastic request processes. On the contrary, despite the many potential applications of similarity caching, there is still almost no theoretical study of the problem, specially for dynamic policies. The only one we are aware of is the competitive analysis of a particular variant of similarity caching in [11] (details in Sect. IV). Basic questions are still unanswered: are similarity and exact caching fundamentally different problems? Do low-complexity optimal similarity caching policies exist? Are there margins of improvement with respect to heuristics proposed in the literature, like in [2], [3]? This paper provides the first answers to the above questions. Our contributions are the following:

1) we show that similarity caching gives rise to NP-hard problems even when exact caching lends itself to simple polynomial algorithms;
2) we provide an optimal pseudo-polynomial algorithm when the sequence of future requests is known;
3) we recognize that, in the adversarial setting, similarity caching is a k-server problem with excursions;
4) we propose optimal dynamic policies, both when objects' popularities are known, and when they are unknown;
5) we show by simulation that our dynamic policies provide better performance than existing schemes, both under the independent reference model (IRM) and under real request traces.

A major technical challenge of our analysis is that we allow the object catalog to be uncountable—as it happens when objects/requests are described by vectors of real-valued features [10]—and the request process to be characterized by a probability density function. Note that in this case exact caching policies like LRU would achieve zero hit ratio.

The rest of the paper is organized as follows. Section II introduces our main assumptions on request processes and caching policies. Sections III and IV present results on similarity caching respectively in the offline and in the adversarial setting. Our new dynamic policies are described in Sect. V, together with their optimality guarantees in the stochastic setting. We numerically explore the performance of our policies in Sect. VI. An extended version of this paper is available online [12].

II. Main assumptions

Let $\mathcal{X}$ be the (finite or infinite) set of objects that can be requested by the users. We assume that all objects have equal size and the cache can store up to k objects. The state of the cache at time t is given by the set of objects $\mathcal{S}_t$ currently stored in it, $\mathcal{S}_t = \{y_1, y_2, \dots y_k\}$, with $y_i \in \mathcal{X}$.

We assume that, given any two objects x and y in $\mathcal{X}$, there is a non-negative (potentially infinite) cost $C_a(x,y)$ to approximate x with y. We consider $C_a(x,x) = 0$. Given a set $\mathcal{A}$ of elements in $\mathcal{X}$, let $C_a(x,\mathcal{A})$ denote the minimum approximation cost provided by elements in $\mathcal{A}$, i.e., $C_a(x,\mathcal{A}) = \inf_{y\in\mathcal{A}} C_a(x,y)$.

In what follows, we consider two main instances for $\mathcal{X}$ and $C_a()$. In the first instance, $\mathcal{X}$ is a finite set of objects and thus the approximation cost can be characterized by an $|\mathcal{X}| \times |\mathcal{X}|$ matrix of non-negative values. This case could well describe the (dis)similarity of contents (e.g. videos) in a finite catalog. In the second instance, $\mathcal{X}$ is a subset of $\mathbb{R}^p$ and $C_a(x,y) = h(d(x,y))$, where $h : \mathbb{R}^+ \to \mathbb{R}^+$ is a non-decreasing non-negative function and $d(x,y)$ is a metric in $\mathbb{R}^p$ (e.g. the Euclidean one). This case is more suitable for describing objects characterized by continuous features. We will refer to the above two instances as *finite* and *continuous*, respectively.

Our goal is to design effective and efficient cache management policies that minimize the aggregate cost to serve a sequence of requests for objects in $\mathcal{X}$. We assume that the function $C_a : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+ \cup \{+\infty\}$ is available for caching decisions and that the cache is able to compute the set of best approximators for x, i.e., $\arg\min_{y\in\mathcal{S}_t} C_a(x,y)$. This can be efficiently done using locality sensitive hashing (LSH) [3]. Moreover, we will restrict ourselves to online policies in which object insertion into the cache is triggered by requests (i.e., the cache cannot pre-fetch arbitrary objects). Upon a request for object r_t at time t, if the content is locally stored ($r_t \in \mathcal{S}_t$), then the cache directly provides r_t incurring a null cost ($C_a(r_t,\mathcal{S}_t) = 0$) and we have an *exact hit*. Otherwise, the cache can either i) provide the best approximating object locally stored, i.e., $y \in \arg\min_{y\in\mathcal{S}_t} C_a(x,y)$, incurring the approximation cost $C_a(r_t,\mathcal{S}_t)$ (*approximate hit*) or ii) retrieve the content from the server incurring a fixed cost $C_r > 0$ (*miss*). Upon a miss, the cache retrieves the object r_t, serves it to the user, and then may replace one of the currently stored objects with r_t. We stress the caching policy is not required to store r_t. Without loss of generality, we can restrict to caching policies providing an approximate hit only if the approximation cost is smaller than the retrieval cost ($C_a(r_t,\mathcal{S}_t) \le C_r$). Indeed, one could otherwise devise a new caching policy that retrieves content r_t and then discards it, paying a smaller cost. As a consequence, when the cache state does not change, the cost to serve r_t is equal to $C(r_t,\mathcal{S}_t) \triangleq \min(C_r, C_a(r_t,\mathcal{S}_t))$.

We also define the movement cost from cache state $\mathcal{T}$ to cache state $\mathcal{S}$ as

$$C_m(\mathcal{T},\mathcal{S}) \triangleq \begin{cases} 0, & \text{if } \mathcal{S} = \mathcal{T}, \\ C_r, & \text{if } |\mathcal{S} \setminus \mathcal{T}| = 1, \\ +\infty, & \text{otherwise.} \end{cases} \quad (1)$$

Given a finite sequence of requests $\mathbf{r}_T = r_1, r_2, \dots, r_T$ and an initial state $\mathcal{S}_1$, the average cost paid by a given caching policy A is

$$\mathcal{C}_A(\mathcal{S}_1, \mathbf{r}_T) = \frac{1}{T}\sum_{t=1}^{T} \left[C_m(\mathcal{S}_t,\mathcal{S}_{t+1}) + C(r_t,\mathcal{S}_{t+1})\right]. \quad (2)$$

In fact, if $\mathcal{S}_{t+1} \neq \mathcal{S}_t$, the cache has retrieved r_t paying the retrieval cost $C_r = C_m(\mathcal{S}_t,\mathcal{S}_{t+1})$, but no approximation cost ($C(r_t,\mathcal{S}_{t+1}) = C_a(r_t,\mathcal{S}_{t+1}) = 0$). If $\mathcal{S}_{t+1} = \mathcal{S}_t$, the cache has provided an approximated answer or has retrieved (but not stored) r_t, paying $C(r_t,\mathcal{S}_t) = \min(C_r, C_a(r_t,\mathcal{S}_t))$. Note that the average cost depends on A, because the policy determines the evolution of the cache state $\mathcal{S}_t$. Policies differ in the choice of which requests are approximate hits or misses (even if $C_a(r_t,\mathcal{S}_t) \le C_r$, the cache can decide to retrieve and store r_t) and in the choice of which object is evicted upon insertion of a new one. We observe that, if $C_a(x,y) = \infty$ for all $x \neq y$, we recover the exact caching setting. If, in addition, $C_r = 1$, Eq. (2) provides the miss ratio. The cost structure (2), together with a movement cost like (1) that satisfies the triangle inequality, defines a metrical task system, first introduced by Borodin *et al.* [13] and usually studied through competitive analysis (more in Sect. IV).

As mentioned in the introduction, similarity caching lacks a solid theoretical understanding. From an algorithmic view-point, it is not clear if similarity caching is a problem intrinsically more difficult than exact caching. From a performance evaluation view-point, we do not know if similarity caching can be studied resorting to the same approaches adopted for exact caching. In this paper we provide a first answers to these questions, which depend crucially on the nature of the requests' sequence. Three scenarios are commonly considered in the literature:

Offline: the request sequence is known in advance. This assumption is made when one wants to determine the best possible performance of any policy. In the case of exact caching, it is well known that the minimum cost (miss ratio) is achieved by Bélády's policy [14], that evicts at each time the cached object, whose next request is further in the future.

Adversarial: the request sequence is selected by an adversary who aims at maximizing the cost incurred by a given caching policy. This approach leads to competitive analysis, which determines how much worse an online policy (without knowledge of future requests) performs with respect to an optimal offline policy.

Stochastic: requests arrive according to a stationary exogenous stochastic process. One example is the classic IRM, where requests for different objects are generated by independent time-homogeneous Poisson processes. The goal here is to minimize the expected cost or equivalently the average cost in (2) over an infinite time horizon.

We separately consider the above three scenarios in the next sections.

III. Offline optimization

In this section we consider the offline setting in which a finite sequence of requests $\mathbf{r}_T$ is known in advance. We first address the problem of finding a static set of objects to be prefetched in the cache, so as to minimize the cost in (2), i.e., we want to find:

$$\mathcal{S}^* \in \arg\min_{\mathcal{S}_1} \sum_{t=1}^{T} C(r_t, \mathcal{S}_1).$$

Note that the corresponding version of this (static, offline) problem for exact caching has a simple polynomial solution with $T \log T$ time complexity and T space complexity: one simply needs to store in the cache the k most requested objects in the trace. For similarity caching the problem is much more difficult, in fact:

Theorem III.1. *The static offline similarity caching problem is NP-hard.*

Proof. The result follows from a reduction of maximum coverage problem (NP-hard) to a static offline similarity caching problem. Let $\mathcal{G} = (V, E)$ be an undirected graph with set of nodes V and set of edges E. We consider the static offline similarity caching problem with $\mathcal{X} = V$, $C_r = 1$, and $C_a(x, y) = 0$ if $(x, y) \in E$ and $C_a(x, y) = +\infty$ otherwise. The request sequence $\mathbf{r}_T$ contains one and only one request for each content. Minimizing the total cost of this instance of the similarity caching problem is equivalent to finding the k nodes that cover the largest number of nodes in V. $\square$

In the continuous case, where objects are points in $\mathbb{R}^p$, and $C_a(x, y)$ is a function of a distance $d()$, one may expect that the problem becomes simpler. The following theorem shows that this is not the case in general.

Theorem III.2. *Let $\mathcal{X} = \mathbb{R}^2$, and $C_a(x, y) = h(d(x, y))$, where $h(z) = 0$ for $z \leq 1$ and $h(z) = C_r = 1$ otherwise. Finding the optimal static set of objects to store in the cache is NP-hard both for norm-2 and norm-1 distance.*

Proof. We prove NP-hardness in the restricted case when every object is requested only once. We observe that any object y stored in the cache can satisfy requests for all the points in a disc (resp. square) centered in y in the case of norm-2 (resp. norm-1) distance. The problem of determining the optimal static set of cached objects that maximizes the number of hits is then equivalent to the problem of finding the placement of k identical geometric shapes that covers the largest number of points in the request sequence. These shapes are, respectively, discs and squares in the case of norm-2 and norm-1 distance. NP-hardness follows immediately from the NP-hardness of the two covering problems on the plane known as DISC-COVER and BOX-COVER [15]. $\square$

We have already observed that exact caching is a particular case of similarity caching. Theorems III.1 and III.2 show that similarity caching is an intrinsically more difficult problem.

For the dynamic setting, we propose a dynamic programming algorithm adapted from that proposed in [16] for exact caching.

Let $\mathbf{r}$ denote a finite sequence of requests for m distinct objects, and $\mathbf{r}x$ the sequence obtained appending to $\mathbf{r}$ a new request for content x. We denote by $\mathcal{S}_1$ the initial cache state, and by $C_{\text{OPT}}(\mathbf{r}, \mathcal{S})$, the minimum aggregate cost achievable under the request sequence $\mathbf{r}$, when the *final* cache state is $\mathcal{S}$. It is possible to write the following recurrence equations, where $\boldsymbol{\epsilon}$ denotes the empty sequence:

$$C_{\text{OPT}}(\boldsymbol{\epsilon}, \mathcal{S}) = \begin{cases} 0, & \text{if } \mathcal{S} = \mathcal{S}_1, \\ +\infty, & \text{otherwise.} \end{cases}$$

$$C_{\text{OPT}}(\mathbf{r}x, \mathcal{S}) = \begin{cases} \min\limits_{\mathcal{T}} \left(C_{\text{OPT}}(\mathbf{r}, \mathcal{T}) + C_m(\mathcal{T}, \mathcal{S}) \right), & \text{if } x \in \mathcal{S}, \\ C_{\text{OPT}}(\mathbf{r}, \mathcal{S}) + C(x, \mathcal{S}) & \text{otherwise.} \end{cases}$$

These equations lead to a dynamic programming procedure that iteratively computes the optimal cost for $\mathbf{r}_T$ and determine the corresponding sequence of caching decisions. Algorithm's time complexity is $\mathcal{O}\left((m-k)k^2\binom{m}{k}T\right)$. Space complexity is at least $\binom{m}{k}$. As this algorithm can only be applied to scenarios with small cache/catalog sizes, we will derive more useful bounds for the optimal cost in Sect. V-C.

IV. Competitive analysis under adversarial requests

The usual worst case analysis is not particularly illuminating for caching problems: if an adversary can arbitrarily select the request sequence, then the performance of any caching policy can be arbitrarily bad. For example, with a catalog of $k + 1$ objects, the adversary can make any deterministic algorithm achieve a null hit rate.

For this reason, the seminal work of Sleator and Tarjan [17] introduced *competitive analysis* to characterize the relative performance of caching policies in comparison to the best possible offline policy with hindsight, i.e., when the sequence of requests selected by the adversary is known when caching decision are taken.[1] In particular, an online caching algorithm A is said to be ρ-competitive, if its performance is within a factor ρ (plus a constant) from the optimum. More formally, there exists a such that

$$\mathcal{C}_A(\mathcal{S}_1, \mathbf{r}_T) \leq \rho\, \mathcal{C}_B(\mathcal{S}_1, \mathbf{r}_T) + \frac{a}{T}, \text{ for all } B \text{ and } \mathbf{r}_T.$$

[1]By now, competitive analysis has become a standard approach to study the performance of many other algorithms.

A competitive analysis of similarity caching in the particular case when $C_a(x,y) = 0$ if $d(x,y) \leq r$, where $d()$ is a distance in $\mathcal{X}$, is in [11] (the only theoretical study of similarity caching we are aware of). In this section we present results for other particular cases, relying on existing work for the k-server problem with excursions.

The k-server problem [16] is perhaps the "most influential online problem [...] that manifests the richness of competitive analysis" [18]. In the k-server problem, at each time instant a new request arrives over a metric space and the user has to decide which server to move to serve it, paying a cost equal to the distance between the previous position of the server and the request. It is well known that the k-server problem generalizes the exact caching problem.

Interestingly, Manasse and McGeoch's seminal paper on the k-server problem [16] also introduces the following variant: a server can perform an excursion to serve the new request and then come back to the original point paying a cost determined by a different function. Similarity caching problem can be considered as a k-server problem with excursions where server movements have uniform cost C_r and the excursion of a server in y to serve a request for x has cost $C_e(x,y) \triangleq \min(C_a(x,y), C_r)$.

Unfortunately, while we have found a noble relative of our problem in the algorithmic field, not much is known about the k-server problem with excursions for the exact scenario we are targeting (uniform metric space for movements and generic metric space for excursions). We rephrase a few existing results in terms of the similarity caching problem. The first one applies when the cache can contain all objects but one. The second one applies to the uniform scenario where each object can equally well approximate any other object. We hope that the important applications of similarity caching will motivate further research on the k-server problem with excursions.

Theorem IV.1. *[16, Sect. 6, Thm 10] Let α_u be an upper bound for the set $\{C_e(x,y)/C_r \mid x, y \in \mathcal{X}\}$. If $|\mathcal{X}| = k+1$, then the competitive ratio of any algorithm is bounded below by $(2k+1)(1+\alpha_u)/(1+2\alpha_u)$. Moreover, there exists a $(2k+1)$-competitive deterministic algorithm (BAL).*

Theorem IV.2. *[19, Thms 4.1-2] If $|\mathcal{X}| > k$ and there exists $0 < \alpha$ such that $C_e(x,y) = \alpha C_r$ for all $x, y \in \mathcal{X}$ with $x \neq y$, then the competitive ratio of any algorithm is at least $2k+1$. Moreover, there exists a $(2k+1)$-competitive deterministic algorithm (RFWF).*

V. Stochastic request process

We now consider the case when requests arrive according to a Poisson process with (normalized) intensity 1 and are i.i.d. distributed. In the finite case ($|\mathcal{X}| < \infty$), we have a request rate λ_x for each content x and we essentially obtain the classic IRM. In the continuous case, we need to consider a spatial density of requests defined by a Borel-measurable function $\lambda_x : \mathcal{X} \to \mathbb{R}_+$, i.e., for every Borel set $\mathcal{A} \subseteq \mathcal{X}$, the rate with which contents in $\mathcal{A}$ are requested is given by $\int_{\mathcal{A}} \lambda_x \, \mathrm{d}x$.

Under the above assumptions, for a given cache state $\mathcal{S} = \{y_1 \dots y_k\}$, we can compute the corresponding expected cost to serve a request:

$$\mathcal{C}(\mathcal{S}) \triangleq \begin{cases} \sum_x \lambda_x C(x, \mathcal{S}), & \text{finite case} \\ \int_{\mathcal{X}} \lambda_x C(x, \mathcal{S}) \, \mathrm{d}x, & \text{continuous case.} \end{cases} \tag{3}$$

We observe that, as the sequence of future requests does not depend on the past, the average cost incurred over time by any online caching algorithm A is bounded with probability 1 (w.p. 1) by the minimum expected cost $\min_{\mathcal{S}} \mathcal{C}(\mathcal{S})$,[2] i.e.,

$$\begin{aligned} \liminf_{T\to\infty} \mathcal{C}_A(\mathcal{S}_1, \mathbf{r}_T) &\geq \liminf_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} C(r_t, \mathcal{S}_{t+1}) \\ &\geq \min_{\mathcal{S}} \mathcal{C}(\mathcal{S}), \text{ w.p.1.} \end{aligned} \tag{4}$$

We then say that an online caching algorithm is optimal if its time-average cost achieves the lower bound in (4) w.p. 1. For example, an algorithm that reaches a state $\mathcal{S}^* \in \arg\min_{\mathcal{S}} \mathcal{C}(\mathcal{S})$ and, then, does not change its state is optimal. More in general, an optimal algorithm visits states with non minimum expected cost only for a vanishing fraction of time. Unfortunately, finding an optimal set of objects $\mathcal{S}^*$ to store is an NP-hard problem. In fact, minimizing (3) is a weighted version of the problem considered in Sect. III. Despite the intrinsic difficulty of the problem, we present some online caching policies that achieve a global or local minimum of the cost. We call a policy λ-aware (resp. λ-unaware), if it relies (resp. does not rely) on the knowledge of λ_x.

In practice, λ-aware policies are meaningful only when objects' popularities do not vary wildly over time, remaining approximately constant over time-scales in which λ_x can be estimated through runtime measurements, similarly to what has been done in the case of exact caching by various implementations of the Least Frequently Used (LFU) policy (see e.g. [21]). In contrast, λ-unaware policies do not suffer from this limitation.

Sections V-A and V-B below are respectively devoted to λ-aware and λ-unaware policies. Section V-C presents some lower bounds for the cost of the optimal cache configuration in the continuous scenario.

A. Online λ-aware policies

The first policy we present, Greedy, is based on the simple idea to systematically move to states with a smaller expected cost (3). It works as follows. Upon a request for content x at time t, Greedy computes the maximum decrement in the expected cost that can be obtained by replacing one of the objects currently in the cache with x, i.e., $\Delta\mathcal{C} \triangleq \min_{y \in \mathcal{S}} \mathcal{C}(\mathcal{S}_t \cup \{x\} \setminus \{y\}) - \mathcal{C}(\mathcal{S}_t)$.

- if $\Delta\mathcal{C} < 0$ (x contributes to decrease the cost), then the cache retrieves x, serves it to the user, and replaces $y_e \in \arg\min_{y \in \mathcal{S}} \mathcal{C}(\mathcal{S}_t \cup \{x\} \setminus \{y\})$ with x;

[2]This is a quite intuitive result, but a formal proof is not trivial. The corresponding result for exact caching is in [20].

- if $\Delta\mathcal{C} \geq 0$, the cache state is not updated. If $C_a(x, \mathcal{S}_t) > C_r$, x is retrieved to serve the request; otherwise the request is satisfied by one of the best approximating object in $\mathcal{S}_t$.

Intuitively, we expect GREEDY to converge to a local minimum of the cost. In the continuous case, special attention is required to correctly define and prove this result.

Definition V.1. *A content y_c is said significant if, for any $\delta > 0$, it holds: $\int_{\mathcal{B}(y_c,\delta)} \lambda_x \, \mathrm{d}x > 0$, where $\mathcal{B}(y_c, \delta)$ is the ball of volume δ centered at y_c.*

Definition V.2. *A cache configuration $\mathcal{S}$ is locally optimal if $\mathcal{C}(\mathcal{S}) \leq \mathcal{C}(\mathcal{S}')$, for all $\mathcal{S}'$ obtained from $\mathcal{S}$ by replacing only one of the contents in the cache with a significant content y_c.*

Theorem V.3. *If $C_a()$ and $\lambda()$ are smooth and $\mathcal{X}$ is a compact set, the expected cost of* GREEDY *converges to the expected cost of a configuration that is locally optimal w.p. 1. If $\mathcal{X}$ is a finite set, the cache state converges to a locally optimal configuration in finite time w.p. 1.*

Proof. We start with the finite case. Let (t_n) be the sequence of time instants at which contents are requested, and $(\mathcal{S}_n) = (\mathcal{S}_{t_n})$ be the corresponding sequence of cache configurations. The sequence $(\mathcal{C}(\mathcal{S}_n))$ is non increasing and then convergent. Therefore, there exists a finite random variable $\mathcal{C}_\infty$ such that $\lim_{n\to\infty} \mathcal{C}(\mathcal{S}_n) = \mathcal{C}_\infty$ w.p. 1. Moreover, as the set of possible cache configurations (and then the set of possible costs) is finite, the limit is necessarily reached within a finite number of requests. Also the sequence $(\mathcal{S}_n)$ converges after a finite number of requests w.p. 1 to a random configuration $\mathcal{S}_\infty$, such that $\mathcal{C}(\mathcal{S}_\infty) = \mathcal{C}_\infty$. Observe, indeed, that no configuration can be visited more than once by construction. We prove that $\mathcal{S}_\infty$ is locally optimal w.p. 1. Consider the event E composed of all path trajectories of $(\mathcal{S}_n)$ converging to a specific configuration $\mathcal{S}'$ that is not locally optimal. By definition, there exists a significant object $y_c \notin \mathcal{S}'$, whose insertion in the cache strictly reduces the cost of $\mathcal{S}'$. By construction, in E, y_c can be requested only before the convergence of $(\mathcal{S}_n)$ to $\mathcal{S}'$. Then, necessarily $\mathbb{P}(E) = 0$, because $\lambda_{y_c} > 0$ and, therefore, w.p. 1 sample-paths contain an unbounded sequence of time-instants at which requests for y_c arrive.

The proof for the continuous case is significantly more technical and complex. It is still possible to prove that the sequence $(\mathcal{C}(\mathcal{S}_n))$ converges to a random variable through the convergence theorem for super-martingales, but the sequence $(\mathcal{S}_n)$ may not converge. We then use Prokhorov's theorem to prove that there exists a subsequence $(\mathcal{S}_{n_k})$ that converges in distribution to $\mathcal{S}_\infty$. Working with convergence in distribution makes the rest of the proof more involuted (see [12] for the technical details). $\square$

The GREEDY policy converges to a locally optimal configuration. In the finite catalog case, under knowledge of content popularities, it is possible to asymptotically achieve the global optimal configuration using a policy that mimics a simulated annealing optimization algorithm. This policy is adapted from the OSA policy (Online Simulated Annealing) proposed in [20], and we keep the same name here. OSA maintains a dynamic parameter $T(t)$ (the temperature). Upon a request for content x at iteration t, OSA modifies the cache state as follows:

- If $x \in \mathcal{S}_t$, the state of the cache is unchanged.
- If $x \notin \mathcal{S}_t$, a content $y \in \mathcal{S}$ is randomly selected according to some vector of positive probabilities $p(\mathcal{S}_t)$, and the state of the cache is changed to $\mathcal{S}' = \mathcal{S}_t \setminus \{y\} \cup \{x\}$ with probability $\min\left(1, \exp((\mathcal{C}(\mathcal{S}_t) - \mathcal{C}(\mathcal{S}'))/T(t))\right)$.

In the first case, OSA obviously serves x (a hit). In the second case, if the state changes to $\mathcal{S}'$, the cache serves x. Otherwise, it serves x or $x' \in \arg\min_{z\in\mathcal{S}} C_a(x, z)$, respectively, if $C_a(x, \mathcal{S}) > C_r$ or $C_a(x, \mathcal{S}) \leq C_r$. OSA always stores a new content if this reduces the cost (as GREEDY does), but it does not get stuck in a local minimum because it can also accept apparently harmful changes with a probability that is decreasing in the cost increase. By letting the temperature $T(t)$ decrease over time, the probability to move to worse states converges to 0 over time: the algorithm explores a larger part of the solution space at the beginning and behaves more and more similarly to "greedy" as time goes by. The eviction probability vector $p(\mathcal{S})$ can be arbitrarily chosen, as far as each content in $\mathcal{S}$ has a positive probability to be selected. In practice, we want to select with larger probability contents in $\mathcal{S}$, whose contribution to the cost reduction is smaller.

OSA provides the following theoretical guarantees. Let $\Delta\mathcal{C}_{\max}$ be the maximum absolute difference of costs between two neighboring states, then

Proposition V.4. *When $|\mathcal{X}| < \infty$, if $T(t) = \Delta\mathcal{C}_{\max} k/(1 + \log t)$, asymptotically only the states with minimum cost have a non-null probability to be visited.*

If the content to be evicted were selected uniformly at random from the cache, then the proof would be the same as the one of Proposition IV.2 in [20]. A key point in that proof is that, when the temperature is constant ($T(t) = T$), the homogeneous Markov chains induced by OSA are reversible, so that one can easily write their stationary probability distributions. Here, it is not the case, but we can use the more general result for *weakly-reversible* time-variant Markov chains in [22, Thm. 1] (see [12] for the proof).

As it is usual for simulated annealing results, convergence is guaranteed under very slow decrease of the temperature parameter (inversely proportional to the logarithm of the number of iterations). In practice, much faster cooling rates are adopted and convergence is still empirically observed.

Figure 1 shows a toy case with a catalog of 4 contents and cache size equal to 2, for which GREEDY with probability at least 9/20 converges to a suboptimal state $\mathcal{S} = \{1, 3\}$ with corresponding cost $\mathcal{C}(\mathcal{S}) = 17/128$. On the contrary, OSA escapes from this local minimum and asymptotically converges to the optimal state $\mathcal{S}^* = \{2, 4\}$ with $\mathcal{C}(\mathcal{S}^*) = 6/128$.

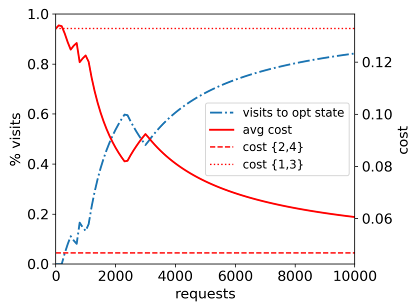

Fig. 1. OSA converges to the minimum cost state. Catalog: $\{1, 2, 3, 4\}$; $C_a(1, 2) = C_a(2, 1) = C_a(2, 3) = C_a(3, 2) = 1/16$; for all other pairs (x, y), $C_a(x, y) = \infty$; $C_r = 1$; $\lambda_2 = \lambda_4 = 1/8$, $\lambda_1 = \lambda_3 = 3/8$, $T(t) = 1/\sqrt{t}$.

B. Online λ-unaware policies

In this section we present two new policies, qLRU-ΔC and DUEL, that, without knowledge of λ_x, bias admission and eviction decisions so to statistically favour configurations with low cost $\mathcal{C}$.

Policy qLRU-ΔC is inspired by qLRU-Δ proposed in [23], which coordinates caching decisions across different base stations so as to maximize a generic utility function. In qLRU-ΔC the cache is managed as an ordered queue as follows. Let x be the content requested at time t.

- If $C_a(x, \mathcal{S}_t) > C_r$, there is a miss. The cache retrieves the content x to serve it to the user. The content is inserted at the front of the queue, with probability q.
- If $C_a(x, \mathcal{S}_t) \leq C_r$, there is an approximate hit. The cache serves a content $z \in \arg\min_{y \in \mathcal{S}_t} C_a(x, y)$, that is *refreshed*, i.e., it is moved to the front of the queue, with probability $\frac{C(x,\mathcal{S}_t \setminus \{z\}) - C_a(x,z)}{C_r}$. With probability $qC_a(x, z)/C_r$ the content x is still retrieved from the remote server and inserted at the head of the queue.

If needed, contents are evicted from the tail of the queue. We observe that $C(x, \mathcal{S}_t \setminus \{z\}) - C_a(x, z)$ corresponds to the cost saving for the request x due to the presence of z in the cache.

When $|\mathcal{X}|$ is finite, the following result holds under the characteristic time (or Che's) approximation (CTA) [24] and the exponentialization approximation (EA), that has been recently proposed in [25].

Theorem V.5. *Under CTA and EA, when $|\mathcal{X}| < \infty$ and q converges to 0, qLRU-ΔC stores a set of contents that corresponds to a local minimum of the cost.*

Proof. We start by extending the CTA to similarity caching. Given an object x, let $T_c^{(x)}(\mathcal{S})$ denote the time content x spends in cache until eviction provided that 1) the cache is in state $\mathcal{S}$ just after its insertion and 2) x is never refreshed (i.e., moved to the front) during its sojourn in the cache . In general $T_c^{(x)}(\mathcal{S})$ is a random variable, whose distribution depends both on x and on the cache state $\mathcal{S}$. The basic assumption of CTA is that $T_c^{(x)}(\mathcal{S}) =_d T_c^{(x')}(\mathcal{S}')$ for each x, x', $\mathcal{S}$, and $\mathcal{S}'$, i.e., we can ignore dependencies on the content and on the state. Moreover, for caching policies where contents are maintained in a priority queue sorted by the time of the most recent refresh, and where evictions occur from the tail (as in LRU, qLRU, and qLRU-ΔC), CTA approximates T_c with a constant.

The strong advantage of CTA is that the interaction among different contents in the cache is now greatly simplified as in a TTL-cache [26]. In a TTL-cache, upon insertion, a timer with value T_c is activated. It is restarted upon each new request for the same content. Once the timer expires, the content is removed from the cache. Under CTA, the instantaneous cache occupancy ($|\mathcal{S}_t|$) can violate the hard buffer constraint.[3] The value of T_c is obtained by imposing the expected occupancy to be equal to the buffer size, i.e.,

$$\sum_{x \in \mathcal{X}} \pi_x = k, \tag{5}$$

where π_x is the stationary probability that x is stored in the cache. For exact caching, it is relatively easy to express π_x as function of T_c and, then, to numerically compute the value of T_c. For similarity caching, additional complexity arises because the timer refresh rate for each content x depends on the other contents in the cache (as x can be used to provide approximate answers), i.e., dynamics of different contents are still coupled. Nevertheless, the TTL-cache model allows us to study this complex system as well.

The expected marginal cost reduction due to x in state $\mathcal{S}$ is $\Delta\mathcal{C}_x(\mathcal{S}) \triangleq \mathcal{C}(\mathcal{S} \setminus \{x\}) - \mathcal{C}(\mathcal{S})$. If the state of the cache does not change, the expected sojourn time of content x in the cache can be computed as:

$$\mathbb{E}[T_S] = \frac{e^{\frac{\Delta\mathcal{C}_x(\mathcal{S})}{C_r} T_c} - 1}{\frac{\Delta\mathcal{C}_x(\mathcal{S})}{C_r}} \triangleq \frac{1}{\nu_x(\mathcal{S})}. \tag{6}$$

EA assumes that $\mathcal{S}_t$ evolves as a Markov Chain (MC) with transition rate from $\mathcal{S}$ to $\mathcal{S} \setminus \{x\}$ equal to $\nu_x(\mathcal{S})$ from (6), and from $\mathcal{S}$ to $\mathcal{S} \cup \{x\}$ equal to $q\lambda_x C_a(x, \mathcal{S})/C_r$ (if x is not already in $\mathcal{S}$). [25] shows that EA is very precise in practice for complex systems of interacting caches.

Results for regular perturbations of Markov chains [27] allow us to study the asymptotic behavior of the MC $(\mathcal{S}_t)$ when q vanishes, and in particular to determine which states are *stochastically stable*, i.e., have a non-null probability to occur as q converges to 0. The proof is analogous to the one in [23].

Reference [23] proposes a caching policy that can coordinate content allocation across different caches in a dense cellular network to maximize a performance metric. Despite the different application scenarios, there is an analogy between the two problems. Here, two similar/close contents interact because they can satisfy the same requests. In [23], two copies of the same content at two close base stations interact because they can satisfy requests from users in the transmission range of both base stations. In particular, we can adapt the proof

[3] Under CTA the number of contents stored in the cache is a Poisson random variable with expected value equal to k. Since its coefficient of variation tends to 0 as k grows large, CTA is expected to be asymptotically accurate.

of [23, Prop. IV.1] to our problem, and show that the stochastically stable states are locally optimizers in the sense that it is not possible to replace a content in such states while reducing the cost [12]. $\square$

Paper [3] proposes two policies for similarity caching: RND-LRU and SIM-LRU. In RND-LRU a request produces a miss with a probability that depends on the distance from the best approximating object z. If it does not produce a miss, it refreshes the timer of z. Interestingly, RND-LRU can emulate in part qLRU-ΔC, by using $qC_a(x, \mathcal{S}_t)/C_r$ as its miss probability. The only difference is the refresh probability: in RND-LRU the best approximating content z is refreshed with probability $1 - qC_a(x, \mathcal{S}_t)/C_r$ (instead of $(C_r - C_a(x, \mathcal{S}_t))/C_r$ as in qLRU-ΔC). Our simulations in Sect. VI confirm that, for the same q, RND-LRU and qLRU-ΔC exhibit a very similar performance. Given our result in Theorem V.5, it is not surprising that RND-LRU performs better than SIM-LRU [3, Fig. 9].

As we will show in Sect. VI, qLRU-ΔC approaches the minimum cost only for very small values of q. This is undesirable when contents' popularities change rapidly. To obtain a more responsive cache behavior, we propose a novel online λ-unaware policy, that we call DUEL.

Similarly to GREEDY, upon a request at time t for a content y' which is not in the cache, DUEL estimates the potential advantage of replacing a cached content y with y', i.e., to move from the current state $\mathcal{S}_t$ to state $\mathcal{S}' = \mathcal{S}_t \setminus \{y\} \cup \{y'\}$. As popularities are unknown, it is not possible to evaluate instantaneously the two costs $\mathcal{C}(\mathcal{S}_t)$ and $\mathcal{C}(\mathcal{S}')$. Then, the two contents engage in a 'duel', i.e., they are compared during a certain amount of time (during this time we need to store only a reference to y'). When a duel between a real content y and its virtual challenger y' starts, we initialize to zero a counter for each of them. If y (resp. y') is the best approximating object for a following request $r_{\tilde{t}}$ occurring at time $\tilde{t} > t$, then the corresponding counter is incremented by $C(r_{\tilde{t}}, \mathcal{S}_{\tilde{t}} \setminus \{y\}) - C_a(r_{\tilde{t}}, y)$ (resp. $C(r_{\tilde{t}}, \mathcal{S}_{\tilde{t}} \setminus \{y\}) - C_a(r_{\tilde{t}}, y')$). The counter associated to a dueling content accumulates then the aggregate cost savings due to that content. A duel finishes in one of two possible ways: 1) counters get separated by more than a fixed quantity δ (a tunable parameter), or 2) a maximum delay τ (another parameter) has elapsed since the start of the duel. Duellist y' replaces y if and only if, at the end of the duel, its counter exceeds the counter of y by more than δ. Otherwise y' is evicted, and y becomes available again for a new duel.

A requested content is matched, whenever possible, with a content in the cache that is not engaged in an ongoing duel. At a given time, then, there can be up to k ongoing duels. A challenger y' is matched to a stored object y in two possible ways: with probability α, it is matched to the closest object in the cache; with the complementary probability $1 - \alpha$, it is matched to a content selected uniformly at random. Duels between nearby contents allow for fine adjustments of the current cache configuration, while duels between far contents enable fast macroscopic changes in the density of stored objects. In essence, DUEL provides a distributed, stochastic version of GREEDY with delayed decisions (due to lack of knowledge of λ_x).

C. Performance bound in the continuous scenario

Besides being appropriate to describe objects/queries in some applications, the continuous scenario is particularly interesting, because it marks a striking difference with exact caching.[4] For this scenario, we can derive some exact bounds and approximations of the minimum cost, exploiting simple geometric considerations.

We start considering a homogeneous request process where $\lambda_x = \lambda$ over a bounded set $\mathcal{X}$. In what follows, all integrals are Lebesgue ones and all sets are Lebesgue measurable. Given a set $\mathcal{A} \subset \mathbb{R}^p$, let $|\mathcal{A}|$ denote its volume (its measure), and $\mathcal{B}(x, |\mathcal{A}|)$ the ball with the same volume centered in x.[5]

Lemma V.6. *For any $y \in \mathcal{X}$ and a set $\mathcal{A} \subset \mathbb{R}^p$ it holds:*

$$\int_{\mathcal{A}} C(x, y)\, \mathrm{d}x \ge \int_{\mathcal{B}(y, |\mathcal{A}|)} C(x, y)\, \mathrm{d}x. \tag{7}$$

The lemma provides the intuitive result that, among all sets $\mathcal{A}$ with a given volume, the approximation cost for requests falling in $\mathcal{A}$ is minimized when $\mathcal{A}$ is a ball centered in y, since $C(x, y) = \min(C_a(x, y), C_r) = \min(h(d(x, y)), C_r)$ is a non-decreasing function of the distance between x and y. We omit the simple proof.

Observe that the integral on the right hand size of (7) does not depend on y, but only on the volume $|\mathcal{A}|$. We then write $F(|\mathcal{A}|) \triangleq \int_{\mathcal{B}(y, |\mathcal{A}|)} C(x, y)\, \mathrm{d}x$. We are now able to express the following bound for the expected cost:

Theorem V.7. *In the continuous scenario with constant request rate λ over $\mathcal{X}$, for any cache state $\mathcal{S}$,*

$$\mathcal{C}(\mathcal{S}) \ge \lambda k F\left(\frac{|\mathcal{X}|}{k}\right). \tag{8}$$

Proof. Given $\mathcal{S} = \{y_1, y_2, \dots, y_k\}$, we denote by $\mathcal{A}_h$ the set of objects in $\mathcal{X}$ having y_h as closest object in the cache, i.e., $\mathcal{A}_h = \{x \in \mathcal{X} \mid y_h \in \arg\min d(x, \mathcal{S})\}$. We have:

$$\begin{aligned}
\mathcal{C}(\mathcal{S}) &= \lambda \int_{\mathcal{X}} C(x, \mathcal{S})\, \mathrm{d}x = \lambda \sum_{h=1}^{k} \int_{\mathcal{A}_h} C(x, y_h)\, \mathrm{d}x \\
&\ge \lambda \sum_{h=1}^{k} \int_{\mathcal{B}(y_h, |\mathcal{A}_h|)} C(x, y_h) \mathrm{d}x = \lambda \sum_{h=1}^{k} F\left(|\mathcal{A}_h|\right) \\
&\ge \lambda k F\left(\frac{\sum_{h=1}^{k} |\mathcal{A}_h|}{k}\right) = \lambda k F\left(\frac{|\mathcal{X}|}{k}\right),
\end{aligned}$$

where the first inequality follows from Lemma V.6, and the second one from Jensen's inequality, since $F(\cdot)$ is convex (as it can be easily checked). $\square$

[4]Recall that in the continuous case the rate of exact hits is null.

[5]The geometric shape of a ball depends on the considered distance function $d(x, y)$. For example, in $\mathbb{R}^2$, if $d(x, y)$ is the usual norm-2, balls are circles; if it is the norm-1, balls are squares.

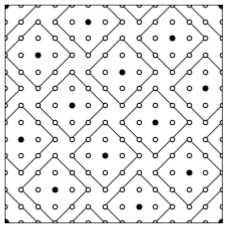

Fig. 2. Example of perfect tessellation of a square grid with wrap-around conditions, in the case $l = 2$, $L = 13$. Black dots correspond to a minimum cost cache configuration under homogeneous popularities.

In some cases, it is possible to show that specific cache configurations achieve the lower bound in (8) and then are optimal:

Corollary 1. *Let d^* be the distance for which the approximation cost is equal to the retrieval cost, i.e., $d^* = \inf\{d : h(d) = C_r\}$. Let $\mathcal{B}_{d^*}(x)$ be a ball of radius d^* centered in x. Any cache state $\mathcal{S} = \{y_1, \dots, y_k\}$, such that the balls $\mathcal{B}_{d^*}(y_h)$ are contained in $\mathcal{X}$ and have intersections with null volume, is optimal.*

Corollary 2. *Any cache state $\mathcal{S} = \{y_1, \dots, y_k\}$, such that, for some d, the balls $\mathcal{B}_d(y_h)$ for $h = 1, \dots, k$ are a tessellation of $\mathcal{X}$ (i.e., $\cup_h \mathcal{B}_d(y_h) = \mathcal{X}$ and $|\mathcal{B}_d(y_i) \cap \mathcal{B}_d(y_j)| = 0$ for each i and j), is optimal.*

If the request rate is not space-homogeneous, one can apply the results above over small regions $\mathcal{X}_i$ of $\mathcal{X}$ where λ_x can be approximated by a constant value $\lambda_{\mathcal{X}_i}$, assuming a given number k_i of cache slots is devoted to each area (with the constraint that $\sum_i k_i = k$). In the regime of large cache size k, it is possible to determine how k_i should scale with the local request rate $\lambda_{\mathcal{X}_i}$, obtaining an approximation of the minimum achievable cost through Theorem V.7. For example, if $\mathcal{X}$ is a square, $d(x, y)$ is the norm-1, $C_r = \infty$, and $C_a(x, y) = d(x, y)^\alpha$, we obtain [12]:

$$\min \mathcal{C}(\mathcal{S}) \approx \frac{2\, k^{-\alpha/2}}{2^{\alpha/2}(\alpha + 2)} \left(\int_{\mathcal{X}} \lambda(x)^{\frac{2}{2+\alpha}} \, \mathrm{d}x \right)^{\frac{2+\alpha}{2}}. \quad (9)$$

VI. Experiments

To evaluate the performance of different caching policies, we have run extensive Monte-Carlo simulations in the following reference scenario: a bidimensional $L \times L$ square grid of points, with unitary step and wrap-around conditions, and $C_a(x, y) = \|x - y\|_1$, i.e., the approximation cost equals the minimum number of hops between x and y. This is a finite scenario (with catalog size equal to L^2) that approximates the continuous scenario in which $\mathcal{X}$ is a square. We let $k = L = 1 + 2l(l + 1)$, for some positive integer l. When $L = 1 + 2l(l + 1)$, there exists a regular tessellation of the grid with L balls (squares in this case), each with L points. Figure 2 provides an example of such regular tessellation in the case $l = 2$, $L = 13$. When $k = L$, we can apply (the discrete versions of) Corollary 2 and approximation (9) to compute the minimum cost.

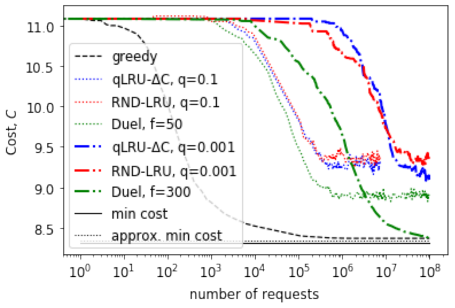

Fig. 3. Performance of different policies in the case of *homogeneous* traffic.

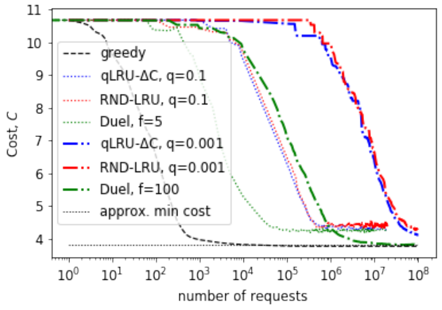

Fig. 4. Performance of different policies in the case of *Gaussian* traffic, with $\sigma = L/8$.

We first consider traffic synthetically generated according to the IRM, in two cases: *homogeneous*, in which all objects are requested with the same rate; *Gaussian*, in which the request rate of object i is proportional to $\exp(-d_i^2/(2\sigma^2))$, where d_i is the hop distance from the grid center. Under homogeneous traffic, Corollary 2 guarantees that a cache configuration storing the centers of the balls of any tessellation like the one in Fig. 2 is optimal. The case of homogeneous popularities then tests the ability of similarity caching policies to converge to one of the L optimal configurations (corresponding to translated tessellations). The case of Gaussian popularities, instead, tests their ability to reach a heterogeneous configuration richer of stored objects close to the center of the grid.

We consider the case $l = 12$, $L = 313$, with catalog size slightly less than 10^5 objects. We set $C_r = 1000 > 2L = \max_{x,y} C_a(x, y)$, i.e., a setting very far from exact caching, where any request can in principle be approximated by any object. For a fair comparison, all algorithms start from the same initial state, corresponding to a set of (distinct) objects drawn uniformly at random from the catalog. For the Duel policy, we experimentally found that, in the general case of grids with unitary step, a good and robust way to set its various parameters is $\alpha = 3/4$, $\delta = f \cdot \min_{x \neq y} C_a(x, y)$, $\tau = fL/\lambda$, which requires to choose a single parameter f.

Figures 3 and 4 show the instantaneous cost (3) achieved by different policies as function of the number of arrived requests, respectively for homogeneous and Gaussian traffic (with $\sigma = L/8$). The optimal cost (approximated by (9), and also exactly computed thanks to Corollary 2 in the homogeneous case) is also reported as reference. In both cases,

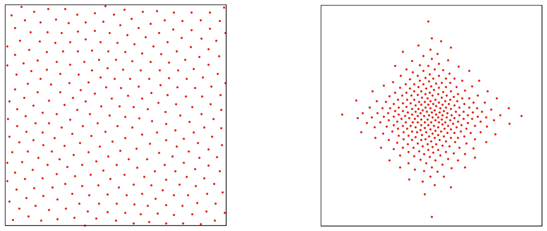

Fig. 5. Final configuration produced by DUEL under *homogeneous* traffic (left plot, with $f = 300$), and *Gaussian* traffic (right plot, with $f = 100$).

as expected, GREEDY outperforms all λ-unaware policies and reaches an almost optimal cache configuration after a number of arrivals of the order of the catalog size. For qLRU-ΔC, RND-LRU, and DUEL, we show two curves for different settings of their parameter (either q or f), leading to a faster convergence to more costly states (thin dotted curves), or a slower one to less costly states (thick dash-dotted curves). As we mentioned in Sect. V-B, qLRU-ΔC and RND-LRU are close (provided that we match their miss probability), and indeed they exhibit very similar performance, with a slight advantage of qLRU-ΔC for small values of q (remember that the local optimality in Theorem V.5 holds for vanishing q). DUEL achieves the best accuracy-responsiveness trade-off, i.e., for a given quality (cost) of the final configuration, it achieves it faster than the other λ-aware policies. Figure 5 shows the cache configuration achieved by DUEL after 10^8 arrivals, for both types of traffic.

We have also evaluated the performance of the different policies using a real content request trace collected over 5 days from Akamai content delivery network. The trace contains roughly 418 million requests for about 13 million objects [28]. By discarding the 116 least popular objects (all requested only once) from the original trace, we obtain a slightly reduced catalog that can be mapped to a square grid with $L = 3643$. We tested two extremely different ways to carry out the mapping. In the *uniform* mapping, trace objects are mapped to the grid points according to a random permutation: popularities of close objects on the grid are, then, uncorrelated. In the *spiral* mapping, trace objects are ordered from the most popular to the least popular, and then mapped to the grid points along an expanding spiral starting from the center: popularities of close-by objects are now strongly correlated, similarly to what happens under synthetic *Gaussian* popularities.

Figure 6 shows the accumulated cost achieved by different policies as function of the number of arrived requests, for both mappings. For qLRU-ΔC and DUEL we performed a (coarse) optimization of their parameter (resp. q and f), so as to achieve the smallest final accumulated cost. To better appreciate the possible gains achievable by similarity caching, we have also added the curves produced by a cache whose state evolves according to two exact caching policies: LRU and RANDOM [29]. These policies produce a disproportionate number of misses. As a consequence, in our experiments, their total aggregate cost (2) is at least one order of magnitude larger than that of qLRU-ΔC or DUEL. For a fair comparison, we only plot the aggregate approximation cost $\sum_t C_a(r_t, \mathcal{S}_t)$.

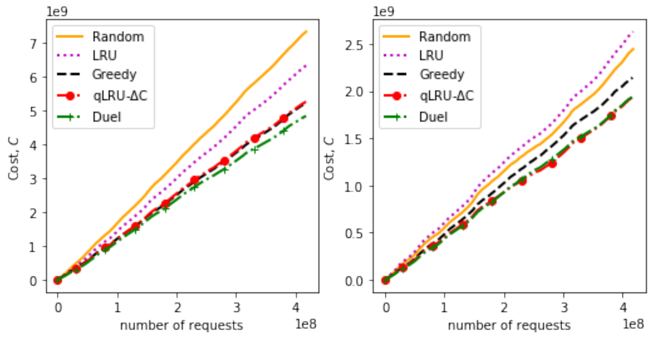

Fig. 6. Performance of different policies under real traffic: *uniform* mapping (left plot) and *spiral* mapping (right plot).

Although retrieval costs incurred are ignored, LRU and RANDOM still perform between 30% and 50% worse than DUEL.

The figure also shows the performance of GREEDY, using as λ_x the empirical popularity distribution measured on the entire trace. Interestingly, under non stationary, realistic traffic conditions, GREEDY no longer outperforms λ-unaware policies. In particular, DUEL takes the lead under both mappings, due to its ability to dynamically adapt to shifts in contents' popularity.

VII. CONCLUSION AND FUTURE WORK

The analysis provided in this paper constitutes a first step toward the understanding of similarity caching, however it is far from being exhaustive. In the offline dynamic setting, it is unclear if and under which conditions there exists an efficient polynomial clairvoyant policy corresponding to Bélády's one [14] for exact caching. The adversarial setting calls for more general results about the k-server problem with excursions. For exact caching, the characteristic time approximation is rigorously justified under an opportune scaling of cache and catalogue sizes [30], [31], [32]. It would be interesting to understand if and to what extent analogue results hold for similarity caching. Moreover, is it possible to use the CTA to estimate the expected cost of a similarity caching policy, similarly to what can be done for the miss ratio of LRU, qLRU, RANDOM, and other policies in the classic setting? There is still room for the design of efficient λ-unaware policies. Interestingly, in our experiments the smallest cost is achieved by DUEL, a novel policy that completely differs from exact caching policies, suggesting that similarity caching may require to depart from traditional approaches. Another interesting direction would be to consider networks of similarity caches. At last, the above issues should be specified in the context of the different application domains mentioned in the introduction, ranging from multimedia retrieval to recommender systems, from sequence learning tasks to low-latency serving of machine learning predictions. The design of computationally efficient algorithms can, indeed, strongly depend on the specific application context. In conclusion, our initial theoretical study and performance evaluation of similarity caching opens many interesting directions and brings new challenges into the caching arena.

References

[1] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in Metric Spaces. *ACM Computing Surveys*, 33(3):273–321, September 2001.

[2] Fabrizio Falchi, Claudio Lucchese, Salvatore Orlando, Raffaele Perego, and Fausto Rabitti. A Metric Cache for Similarity Search. In *Proceedings of the 2008 ACM Workshop on Large-Scale Distributed Systems for Information Retrieval*, LSDS-IR '08, pages 43–50, New York, NY, USA, 2008. ACM.

[3] Sandeep Pandey, Andrei Broder, Flavio Chierichetti, Vanja Josifovski, Ravi Kumar, and Sergei Vassilvitskii. Nearest-neighbor Caching for Content-match Applications. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 441–450, New York, NY, USA, 2009. ACM.

[4] Pavlos Sermpezis, Theodoros Giannakas, Thrasyvoulos Spyropoulos, and Luigi Vigneri. Soft Cache Hits: Improving Performance Through Recommendation and Delivery of Related Content. *IEEE Journal on Selected Areas in Communications*, 36(6):1300–1313, June 2018.

[5] Alexander F. Auch, Hans-Peter Klenk, and Markus Göker. Standard operating procedure for calculating genome-to-genome distances based on high-scoring segment pairs. *Standards in genomic sciences*, 2(1):142, 2010.

[6] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[7] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing Machines. *arXiv, preprint* `arXiv:1410.5401 [CS.NE]`, 2014.

[8] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-Learning with Memory-Augmented Neural Networks. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1842–1850, New York, New York, USA, 20–22 Jun 2016. PMLR.

[9] Daniel Crankshaw, Xin Wang, Joseph E Gonzalez, and Michael J Franklin. Scalable training and serving of personalized models. In *NIPS 2015 Workshop on Machine Learning Systems (LearningSys)*, 2015.

[10] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. Clipper: A Low-Latency Online Prediction Serving System. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 613–627, Boston, MA, 2017. USENIX Association.

[11] Flavio Chierichetti, Ravi Kumar, and Sergei Vassilvitskii. Similarity Caching. In *Proceedings of the Twenty-eighth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '09, pages 127–136, New York, NY, USA, 2009. ACM.

[12] Michele Garetto, Emilio Leonardi, and Giovanni Neglia. Similarity Caching: Theory and Algorithms. *arXiv preprint* `arXiv:1912.03888 [cs.NI]`, 2019.

[13] Allan Borodin, Nathan Linial, and Michael E. Saks. An Optimal On-line Algorithm for Metrical Task System. *J. ACM*, 39(4):745–763, October 1992.

[14] László A. Bélády. A Study of Replacement Algorithms for a Virtual-storage Computer. *IBM Systems Journal*, 5(2):78–101, June 1966.

[15] Robert J. Fowler, Michael S. Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Information Processing Letters*, 12(3):133 – 137, 1981.

[16] Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. Competitive Algorithms for Server Problems. *J. Algorithms*, 11(2):208–230, May 1990.

[17] Daniel D. Sleator and Robert E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Commun. ACM*, 28(2):202–208, February 1985.

[18] Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, May 2009.

[19] Yair Bartal, Moses Charikar, and Piotr Indyk. On page migration and other relaxed task systems. *Theoretical Computer Science*, 268(1):43 – 66, 2001. On-line Algorithms '98.

[20] Giovanni Neglia, Damiano Carra, and Pietro Michiardi. Cache Policies for Linear Utility Maximization. *IEEE/ACM Transactions on Networking*, 26(1):302–313, 2018.

[21] G. Einziger and R. Friedman. TinyLFU: A Highly Efficient Cache Admission Policy. In *22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 146–153, Feb 2014.

[22] Bruce Hajek. Cooling Schedules for Optimal Annealing. *Mathematics of Operations Research*, 13(2):311–329, 1988.

[23] Giovanni Neglia, Emilio Leonardi, Guilherme Iecker, and Thrasyvoulos Spyropoulos. A Swiss Army Knife for Dynamic Caching in Small Cell Networks. *arXiv preprint* `arXiv:1912.10149 [cs.NI]`, 2019.

[24] Hao Che, Ye Tung, and Z. Wang. Hierarchical Web caching systems: modeling, design and experimental results. *IEEE Journal on Selected Areas in Communications*, 20(7):1305–1314, Sep 2002.

[25] Emilio Leonardi and Giovanni Neglia. Implicit coordination of caches in small cell networks under unknown popularity profiles. *IEEE Journal on Selected Areas in Communications*, 36(6):1276–1285, June 2018.

[26] Nicaise Eric Choungmo Fofack, Philippe Nain, Giovanni Neglia, and Don Towsley. Analysis of TTL-based Cache Networks. In *ValueTools - 6th International Conference on Performance Evaluation Methodologies and Tools - 2012*, Cargèse, France, October 2012.

[27] H Peyton Young. The Evolution of Conventions. *Econometrica*, 61(1):57–84, January 1993.

[28] Giovanni Neglia, Damiano Carra, Mingdong Feng, Vaishnav Janardhan, Pietro Michiardi, and Dimitra Tsigkari. Access-time-aware cache algorithms. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 2(4), December 2017.

[29] Michele Garetto, Emilio Leonardi, and Valentina Martina. A Unified Approach to the Performance Analysis of Caching Systems. *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 1(3):12:1–12:28, May 2016.

[30] Ronald Fagin. Asymptotic miss ratios over independent references. *Journal of Computer and System Sciences*, 14(2):222 – 250, 1977.

[31] Christine Fricker, Philippe Robert, and James Roberts. A Versatile and Accurate Approximation for LRU Cache Performance. In *Proceedings of the 24th International Teletraffic Congress*, ITC '12, pages 8:1–8:8. International Teletraffic Congress, 2012.

[32] Bo Jiang, Philippe Nain, and Don Towsley. On the Convergence of the TTL Approximation for an LRU Cache Under Independent Stationary Request Processes. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 3(4):20:1–20:31, September 2018.