IRIS AperTO

UNIVERSITÀ DEGLI STUDI DI TORINO

AperTO - Archivio Istituzionale Open Access dell'Università di Torino

Similarity Caching: Theory and Algorithms

This is a pre print version of the following article:

Original Citation:

Availability:

This version is available http://hdl.handle.net/2318/1844881 since 2022-03-02T08:58:25Z

Published version:

DOI:10.1109/TNET.2021.3126368

Terms of use:

Open Access

Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)

12 October 2024

Similarity Caching: Theory and Algorithms

Giovanni Neglia*, Michele Garetto†, Emilio Leonardi‡

*Inria, Université Côte d'Azur, France, giovanni.neglia@inria.fr
†Università di Torino, Italy, michele.garetto@unito.it
‡Politecnico di Torino, Italy, emilio.leonardi@polito.it

***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.**

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 potentially infinite and uncountable, as it happens when objects/requests are described by vectors of real-valued features [10]. Note that in this case exact caching

TABLE I
NOTATION

k	cache size
$\mathcal{X}$	set of objects that can be requested
$\mathcal{S}_t$	state of the cache at time t
$\|.\|$	a norm in $\mathbb{R}^p$
C_r	cost to retrieve an object from the remote server
$C_a(x,y)$	cost to approximate x with y
$C_a(x,\mathcal{S})$	minimum cost to approximate x with the elements in $\mathcal{S}$
$C_m(\mathcal{T},\mathcal{S})$	cost to move from cache state $\mathcal{T}$ to cache state $\mathcal{S}$
$C(x,\mathcal{S})$	cost to serve request x from cache state $\mathcal{S}$
$C_e(x,y)$	excursion cost to serve request x using (server) y
$C_A(\mathcal{S}_1,\mathbf{r}_T)$	time-average cost incurred by caching policy A to serve the request sequence $\mathbf{r}_T$ starting from state $\mathcal{S}_1$
$\mathcal{C}(\mathcal{S})$	expected cost to serve a request from cache state $\mathcal{S}$
λ_x	request rate for object x (finite case) or request rate density in x (continuous case)
$\mathcal{B}(x,V)$	ball in $\mathbb{R}^p$ centered in x and with volume V
$\mathcal{B}_r(x)$	ball in $\mathbb{R}^p$ centered in x and with radius r

policies like LRU would achieve zero hit ratio, when the spatial distribution of requests is described by a probability density function.

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.

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 compact subset of $\mathbb{R}^p$ and $C_a(x,y) = h(\|x-y\|)$, where $h : \mathbb{R}^+ \to \mathbb{R}^+$ is a non-decreasing non-negative function and $\|.\|$ is a norm in $\mathbb{R}^p$ (e.g. the Euclidean one). This case is more suitable for describing objects characterized by continuous features. For example, in the field of distance metric learning [12], supervised machine learning techniques are used to learn how to map similar objects to vectors in $\mathbb{R}^p$ that are close according to q-norm distances, cosine similarity, or Mahalanobis distances. We describe a specific application in Amazon recommendation systems in Sect. VI. 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 ($C_a(r_t,\mathcal{S}_t) \le C_r$). The caching policy will then always reply to a request r_t with an object o such that $C_a(r_t,o) \le C_r$. Indeed, one could otherwise devise a new caching policy that retrieves content r_t and then discards it, paying a smaller cost.

We also define the movement cost, i.e., the cost of updating the cache state from $\mathcal{T}$ to $\mathcal{S}$, as follows:

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

The definition captures the fact that it is possible to replace only one object in the cache paying the retrieval cost C_r. 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

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

where

$$C(o,\mathcal{S}) \triangleq \min(C_a(o,\mathcal{S}), C_r) \tag{3}$$

and we call it the service cost. 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 approximate answer or has retrieved (but not stored) r_t, paying $C(r_t,\mathcal{S}_t) \triangleq \min(C_r, C_a(r_t,\mathcal{S}_t))$. Note that the average cost depends on the caching policy 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 (MTS), first introduced by Borodin *et al.* [13], which is usually studied through competitive analysis (more in Sect. IV).

As mentioned in the introduction, similarity caching lacks a solid theoretical understanding. From an algorithmic viewpoint, 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 the 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 object whose next request is further in the future.

Adversarial: the request sequence is selected by an adversary who wants to maximize 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 in comparison to the 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 distinguish between two cases: i) a first preliminary scenario, in which a static set of objects has to be prefetched in the cache before the arrival of the first request (we refer to this scenario as static) and then the state of the cache can not be further modified; ii) a more general case in which the state of the cache can be modified upon every miss by inserting in it the requested content, in place of a previously stored content (we refer to this scenario as dynamic).

A. Static setting

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 the dominating set problem. Consider an undirected graph $\mathcal{G} = (V, E)$ with set of nodes V and set of edges E. Given $U \subset V$ we let $N[U] = \{U\} \cup \{u \in V : \exists v \in U, (u, v) \in E\}$ denote the (closed) neighborhood of U. The dominating set problem concerns testing, for a given graph $\mathcal{G}$ and value k, if there exists a set $U \subset V$ with size at most k such that $N[U] = V$. The dominating set problem is NP-complete [15, Sect. A1.1]. The corresponding optimization problem, which is NP-hard, can be formulated as:

$$\underset{U \subset V, |U| \leq k}{\text{maximize}} \quad |N[U]|. \tag{4}$$

We show that this problem can be reduced to a static offline similarity caching problem.

We consider the static offline similarity caching problem with $\mathcal{X} = V$, $C_r = 1$, and $C_a(x, y) = C_a(y, x) = 0$ if $(x, y) \in E$ and $C_a(x, y) = C_a(y, x) = +\infty$ otherwise. The request sequence $\mathbf{r}_T$ contains one and only one request for each object.

Let $\mathcal{S}$ denote the set of objects in the cache and $U \subset V$ the corresponding set of nodes in the graph $\mathcal{G}$. Consider a request r for object $v \in V$. The request generates a miss and incurs a cost equal to $C_r = 1$ if and only if v does not belong to $N[U]$. It follows that the total cost incurred by the similarity cache is equal to the number of nodes in $V \setminus N[U]$. Therefore finding the cache configuration $\mathcal{S}$ that minimizes the total cost is equivalent to identifying the k nodes U with the largest closed neighborhood $|N[U]|$ as in (4). □

Remark 1. We observe that the static cache allocation problem can also be formulated as a particular maximum coverage problem. The maximum coverage problem cannot be approximated by a constant larger than $1 - 1/e$, which is achieved by the greedy algorithm. The relation between the two problems suggests that a greedy algorithm would be a reasonable approach to the static offline similarity caching problem, but there is no guaranteed approximation ratio (because of the transformation from a maximization problem into a minimization one).

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 think that the problem could become simpler. The following theorem shows that this is not the case in general.

Theorem III.2. *Let $\mathcal{X} = \mathbb{R}^2$, and $C_a(x, y) = h(\|x - y\|)$, 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 objects to store in the cache to maximize the number of hits is then equivalent to the problem of finding k identical translated geometric shapes covering 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 [16]. □

Remark 2. In $\mathbb{R}$, DISC-COVER and BOX-COVER are solvable in linear time.

Remark 3. For geometric versions of the maximum coverage problem, better approximation ratios can be achieved. For example [17] presents some efficient polynomial-time approximation schemes when the objects are point on a plane.

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.

B. Dynamic setting

For the dynamic setting (i.e., when the state of the cache can be modified at every request), we propose a dynamic programming algorithm adapted from that proposed in [18] 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 ϵ denotes the empty sequence:

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

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

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. The number of possible states is $\binom{m}{k}$. We can arrive to each state $\mathcal{S}$ potentially from $(m-k)k+1$ other states. $d(T, S)$ and $C_a(x, \mathcal{S})$ can be evaluated with $O(k)$ operations. Algorithm's time complexity is $O\left((m-k)k^2\binom{m}{k}T\right)$. Space complexity is at least $\binom{m}{k}$. As this algorithm can only be applied to small cache/catalog sizes, we will derive more useful bounds for the optimal cost in Sect. V-C for the stochastic scenario.

Remark 4. This algorithm is especially suited for the case when the number of requests T is much larger than the number m of unique contents appearing in the sequence. For the classic k-server problem there exists also an alternative algorithm, based on minimum cost network flow, with complexity kT^2 [19], that is preferable in the opposite scenario. It is an open problem if a similar algorithm can be designed for similarity caching.

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 by simply asking at any time the content that is not currently stored in the cache.

For this reason, the seminal work of Sleator and Tarjan [20] introduced *competitive analysis* to characterize the relative performance of caching policies in comparison to the best possible offline policy with hindsight, i.e., under the assumption that the sequence of requests is known before caching decisions 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

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

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 [18] is perhaps the "most influential online problem [...] that manifests the richness of competitive analysis" [21]. 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. In particular, in the case when distances are all equal to one (the uniform metric space), the cost of k-server is equal to the total number of misses achieved by a caching policy that is forced to store the requested content.

Interestingly, Manasse and McGeoch's seminal paper on the k-server problem [18] 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

$$C_e(x, y) \triangleq \min(C_a(x, y), C_r). \tag{5}$$

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 in the scenario we are interested in (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 to the case when the cache can contain all objects

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

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. *[18, 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).*

Note that we can always select $\alpha_u = 1$.

Theorem IV.2. *[22, 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).*

A $(4k+1)$-competitive algorithm is proposed in [23]. [24] studies 1-server with limited look-ahead (i.e. considers policies that know a given number of future requests).

V. Stochastic request process

We now consider the case when requests arrival times follow a Poisson process with (normalized) intensity 1 and the requested object is drawn from $\mathcal{X}$ according to the same distribution independently from the past, i.e., requests 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[2] $\int_{\mathcal{A}} \lambda_x \, \mathrm{d}x$.

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:

$$\mathcal{C}(\mathcal{S}) \triangleq \begin{cases} \sum_x \lambda_x C(x,\mathcal{S}), & \text{finite case} \\ \int_{\mathcal{X}} \lambda_x C(x,\mathcal{S}) \, \mathrm{d}x, & \text{continuous case.} \end{cases} \tag{6}$$

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 $\min_{\mathcal{S}} \mathcal{C}(\mathcal{S})$,

$$\liminf_{T\to\infty} C_A(\mathcal{S}_1, \mathbf{r}_T) \geq \min_{\mathcal{S}} \mathcal{C}(\mathcal{S}), \text{ w.p.1.} \tag{7}$$

A proof is given in Appendix A. We then say that an online caching algorithm is optimal if its time-average cost achieves the lower bound in (7) 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 a vanishing fraction of time. Unfortunately, finding an optimal set of objects $\mathcal{S}^*$ to store is an NP-hard problem. In fact, minimizing (6) 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. [25]). 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 (6). 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;
- 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.*

The proof is in Appendix B.

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 [26], and we keep the same name here. OSA

[2] More in general, we could consider a generic compact metric space $\mathcal{X}$ endowed with the Haar measure, i.e., the unique (up to multiplicative constant) finite Borel measure, which is translation invariant over $\mathcal{X}$.

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 $\min(1, \exp((\mathcal{C}(\mathcal{S}_t) - \mathcal{C}(\mathcal{S}'))/T(t)))$.

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 becomes more and more "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

Theorem 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.*

The proof is in Appendix C.

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 $\mathcal{C}(\mathcal{S}^*) = 6/128$.

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 [27] to coordinate caching decisions across different base stations to maximize a generic utility function. Despite the different application scenario, there are deep similarities between how different copies of the same content interact in the dense cellular network scenario of [28] and how different contents interact in a similarity cache.

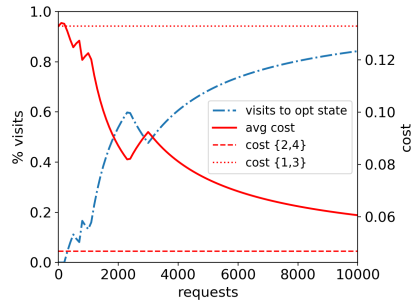

Fig. 1. OSA converges to the minimum cost state. Catalog: $\{1, 2, 3, 4\}$; $C_a(1, 2) = C_a(2, 1) = C_a(2, 3) = C_a(3, 2) = 1/16$; for all other pairs (x, y), $C_a(x, y) = \infty$; $C_r = 1$; $\lambda_2 = \lambda_4 = 1/8$, $\lambda_1 = \lambda_3 = 3/8$, $T(t) = 1/\sqrt{t}$.

In qLRU-ΔC the cache is managed as a ordered list (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.

Remark 5. An approximate hit may jointly lead to insert the new content x as well as to bring the approximating content z to the head of the queue.

When $|\mathcal{X}|$ is finite, the following result holds under the characteristic time (or Che's) approximation (CTA) [29] and the exponentialization approximation (EA), that has been recently proposed and validated in [28].

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.*

The proof is in Appendix D.

The 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 value of q, RND-LRU and qLRU-ΔC exhibit 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].

Remark 6. It is possible to consider admission probabilities of the form $q_{x,t} = a(x, \mathcal{S}_t)q$, i.e, probabilities that depend

on the requested content and the actual state of the cache. Theorem V.5 still holds, when q converges to 0. This flexibility can be exploited to obtain better performance, by avoiding inserting contents that look less promising.

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 $\mathcal{C}(\mathcal{S}'_t)$. 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 its counter exceeds the counter of y by more than δ within the duel duration τ. 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 β, it is matched to the closest object in the cache; with the complementary probability $1 - \beta$, 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. Moreover, we avoid running concurrent duellists y' and y'' which are too close to each other, because the counter of a duellist should not be perturbed by the possible insertion of a close-by duellist. To avoid such 'interferring duels', we do not admit a new duellist y' if its counter could be fed by a request that is already feeding the counter of another duellist y''.

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.[3] 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.[4]

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

$$\int_{\mathcal{A}} C(x, y)\, \mathrm{d}x \geq \int_{\mathcal{B}(y, |\mathcal{A}|)} C(x, y)\, \mathrm{d}x. \tag{8}$$

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 $C(x, y) = \min(C_a(x, y), C_r) = \min(h(\|x - y\|), C_r)$ 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 (8) does not depend on y but only on the volume $|\mathcal{A}|$, because $C(x, y)$ depends only on the distance $\|x - y\|$. 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 $\mathcal{S}$,*

$$\mathcal{C}(\mathcal{S}) \geq \lambda k F\left(\frac{|\mathcal{X}|}{k}\right). \tag{9}$$

Proof. Given $\mathcal{S} = \{y_1, y_2, \ldots, y_k\}$, we denote by $\mathcal{A}_h$ the set of objects in $\mathcal{X}$ having y_h as closest object in the cache, i.e., $\mathcal{A}_h = \{x \in \mathcal{X} \mid y_h \in \arg\min d(x, \mathcal{S})\}$. The family $\{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k\}$ is a Voronoi tessellation of $\mathcal{X}$. We have:

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

where the first inequality follows from Lemma V.6, and the second one from Jensen's inequality, since $F(\cdot)$ is a convex function (see Appendix E). □

In some cases, it is possible to show that specific cache configurations achieve the lower bound in (9) and then are optimal:

Corollary 1. *Let $\bar{d}$ be the distance for which the approximation cost is equal to the retrieval cost, i.e., $\bar{d} = \inf\{d : h(d) = C_r\}$. Let $\mathcal{B}_{\bar{d}}(x)$ be a ball of radius $\bar{d}$ centered in x. Any cache state $\mathcal{S} = \{y_1, \ldots y_k\}$, such that the balls $\mathcal{B}_{\bar{d}}(y_h)$*

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

[4] The geometric shape of a ball depends on the considered norm. For example, in $\mathbb{R}^2$, if $\|.\|$ is the usual norm-2, balls are circles; if it is the norm-1, balls are squares.

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, consider the case when the domain $\mathcal{X}$ lies in the plane ($\mathcal{X} \subset \mathbb{R}^2$), $C_r = \infty$, and $C_a(x, y) = \|x - y\|_1^\gamma$. We partition the domain into M disjoint regions of unitary area, on which the request rate can be approximately assumed to be constant and equal to λ_i, $1 \le i \le M$. Let $k_i \ge 0$ be the number of cache slots devoted to region i, and $\boldsymbol{k}$ the vector storing k_i values. Then each cache slot is used to approximate requests falling in a diamond of area $1/k_i$ and radius $r_i = \sqrt{1/(2k_i)}$, within region i (we ignore border effects, supposing $r_i \ll 1$). The approximation cost c_i for each cell belonging to region i is the same, as approximation costs are invariant to translation. Without lack of generality, we can then consider a cell centred in $y = 0$. The approximation cost can be easily computed as:

$$\begin{aligned} c_i(r_i) &= \int_{\mathcal{B}_{r_i}(0)} \|x - 0\|^\gamma \, \mathrm{d}x \\ &= 4 \int_0^{r_i} \int_0^{r_i - x_1} (x_1 + x_2)^\gamma \, \mathrm{d}x_2 \, \mathrm{d}x_1 = 4 \frac{r_i^{\gamma+2}}{\gamma + 2}, \end{aligned}$$

Expressing $c(r_i)$ as function of k_i, we obtain:

$$c_i(k_i) = \zeta k_i^{-\frac{\gamma+2}{2}},$$

where $\zeta \triangleq 2^{(2-\gamma)/2}/(\gamma + 2)$. Hence the total approximation cost in the whole domain is $\mathcal{C}(\boldsymbol{k}) = \sum_{i=1}^M k_i \lambda_i c_i(k_i)$.

We select the values $\boldsymbol{k}$ that minimize the expected cost:

$$\begin{aligned} \underset{k_1, \dots, k_M}{\text{minimize}} \quad & \zeta \sum_{i=1}^M \lambda_i k_i^{-\gamma/2} \\ \text{subject to} \quad & \sum_{i=1}^M k_i = k \end{aligned} \tag{10}$$

Employing the standard Lagrange method, we obtain that $\lambda_i k_i^{-(\gamma+2)/2}$ equals some unique constant for any region i, which means that k_i has to be proportional to $\lambda_i^{2/(\gamma+2)}$. In the limit of large M, we substitute the sums in (16) with continuous integrals, obtaining:

$$\min \mathcal{C}(\boldsymbol{k}) \approx \zeta k^{-\gamma/2} \left(\int_{\mathcal{X}} \lambda(x)^{\frac{2}{\gamma+2}} \, \mathrm{d}x \right)^{\frac{\gamma+2}{2}}. \tag{11}$$

The case when C_r is finite is slightly more complex and it is studied in Appendix F.

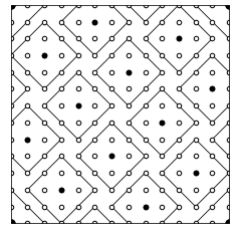

Fig. 2. Example of perfect tessellation of a square grid with wrap-around conditions, in the case $l = 2$, $L = 13$. Black dots correspond to a minimum cost cache configuration under homogenous popularities.

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 (11) to compute the minimum cost.

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 $\beta = 3/4$, $\delta = f \cdot \min_{x \neq y} C_a(x, y)$, $\tau = fL/\lambda$, which requires to choose a single parameter f.

Figures 3 and 4 show the instantaneous cost (6) 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 (11), and also exactly computed thanks to Corollary 2 in the

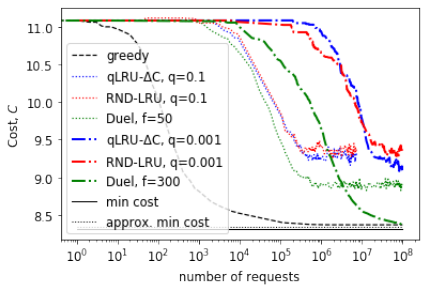

Fig. 3. Performance of different policies in the case of *homogeneous* traffic.

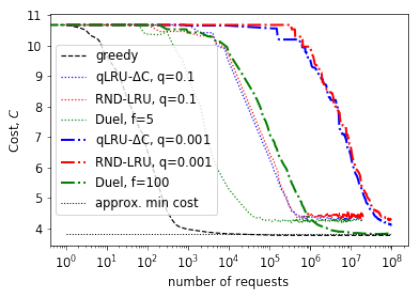

Fig. 4. Performance of different policies in the case of *Gaussian* traffic, with $\sigma = L/8$.

homogeneous case) is also reported as reference. In both cases, 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

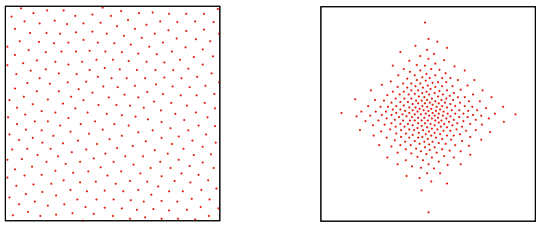

Fig. 5. Final configuration produced by the DUEL policy under *homogeneous* traffic (left plot, with $f = 300$), and *Gaussian* traffic (right plot, with $f = 100$).

roughly 418 million requests for about 13 million objects (more details about the trace are in [30]). 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 the qLRU-ΔC and DUEL policies, we performed a rough optimization of their (unique) parameter (i.e., either q or f), limiting ourselves to optimize the first significant digit.[5]

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 [31]. Since these policies produce a disproportionate number of misses, their total aggregate cost (2) is at least one order of magnitude larger than that qLRU-ΔC and DUEL. For a fair comparison, we only plot the aggregate approximation cost $\sum_t C_a(r_t, \mathcal{S}_t)$. 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. To quantify the extent of popularity variation throughout the trace, we separately computed the popularity distributions within the first and the second half of the trace, and used Kendall's tau measure,[6] as implemented in the scipy library [33], to compare the objects' rankings in the first and second half of the trace, obtaining the value 0.079, which is an indication of strong popularity variation along the trace.

At last, we tested our policies on an Amazon trace. In [34] the authors provide an image dataset for millions of items sold by Amazon, and use a neural network to embed such images into a d-dimensional "style" space. *"The notion of style is learned by training the model on pairs of objects which Amazon considers to be related"* [34] as multiple users purchased or viewed both objects. Style dissimilarity is captured by the Euclidean distance and close objects in the style space can be recommended to the customers. When a user is interested in

[5]This means that, for example, qLRU-ΔC with $q = 0.2$ achieves, at the end of trace, an accumulated cost smaller than that achieved using either $q = 0.1$ or $q = 0.3$.

[6]We adopt the "tau-b" version of this measure as defined in [32], which accounts for ties. Values close to 1 indicate strong agreement, values close to -1 indicate strong disagreement.

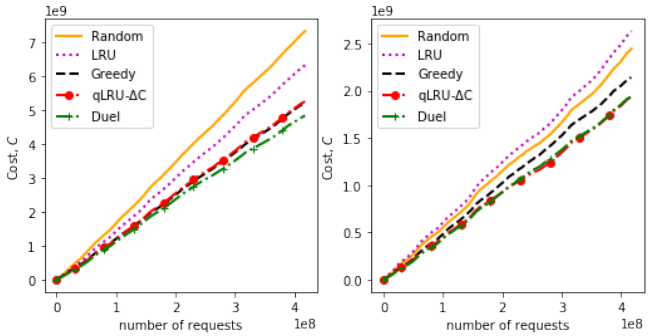

Fig. 6. Performance of different policies under Akamai trace: *uniform* mapping (left plot) and *spiral* mapping (right plot).

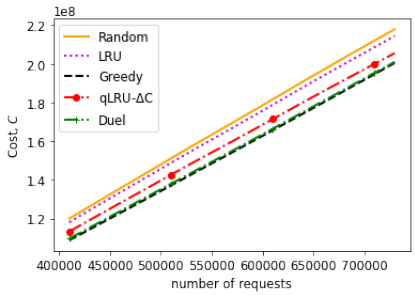

Fig. 7. Performance of different policies under Amazon trace.

a product, Amazon can check over the whole database which other products have a similar style. A similarity cache can be used to speed-up such search, similarly to what proposed for contextual advertising systems in [3]. We consider $d = 100$ and build a request trace from the timestamped reviews left by users for the 10000 most popular items belonging to the category "baby." The resulting trace, containing about 740K requests, is fed into a cache of size 100, and Euclidean distance is used as approximation cost (because it captures items' dissimilarity). Figure 6 shows the accumulated cost achieved by different policies as function of the number of arrived requests. We observe that DUEL performs almost the same as GREEDY, which suggests that the considered trace does not contain significant popularity variations over time. Actually, Kendall's tau measure of objects' ranking agreement between the two halves of the trace is 0.452, significantly larger than the value obtained for the Akamai trace.

VII. MODEL EXTENSIONS

Consider a request r_t that generates the retrieval of object o from the server with consequent cost C_r. Until now, we have assumed that 1) the retrieved object can only be the requested one ($o = r_t$), 2) the cache is not obliged to store the retrieved object o after serving it to the user, and 3) in any case the cache serves to the user a content o' such that $C_a(r_t, o') \leq C_r$ (this is actually a consequence of 1) and 2) as we discussed in Sect. II).

But it may also be meaningful to distinguish a cost paid by the user (C_r^u) and a cost paid by the network (C_r^n). The first one may capture the additional delay the user experiences to retrieve the content, the second one the additional network traffic or server load. We may want to serve a content o' to the user only if $C_a(r_t, o') \leq C_r^u$. Moreover, we may allow the cache to retrieve an approximation of r_t. Finally, many caching policies require the retrieved object to be stored locally.

We have then 8 different possible cases, depending if

- the cache is required to retrieve r_t, or not,
- the cache is required to store o, or not,
- the cache is allowed to serve a content o' to the user only if $C_a(r_t, o') \leq C_r^u$, or not.

Interestingly, all these cases can be captured by our model. In particular, $C_a(x, y)$ can be defined to be infinite any time the approximation cost exceeds C_r^u. Moreover, we can capture the choice about storing or not the retrieved object o through the constant

$$\chi \triangleq \begin{cases} +\infty & \text{if the cache is required to store } o, \\ C_r^u + C_r^n & \text{otherwise.} \end{cases}$$

The system can always be modeled as an MTS where the total cost can be expressed as sum of movement costs ($C_m(\mathcal{S}_t, \mathcal{S}_{t+1})$) to change from one state to another and service costs ($C(r_t, \mathcal{S}_{t+1})$). The movement cost has the same expression as in (1) with C_r replaced by $C_r^u + C_r^n$. The service cost needs now to be defined as

$$C(r, \mathcal{S}) \triangleq \min(C_a(r, \mathcal{S}), \chi). \tag{12}$$

Results for the offline and stochastic scenarios, respectively in Sect. III and Sect. V, hold also for these model variants. The adversarial scenario requires more discussion. Our system is not a k-server with excursions if the cache can retrieve an object different from the requested one (this would correspond to a server both moving to a new state and performing an excursion). In the other cases, excursion costs can be defined as

$$C_e(x, y) \triangleq \min(C_a(x, y), \chi).$$

Theorems IV.1 and IV.2 hold if we replace C_r by $C_r^u + C_r^n$ and add the additional requirement that $\alpha_u \leq 2$ and $\alpha \leq 2$, respectively.

VIII. 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 unknown 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 for 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 [35], [36], [37]. 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 compute 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 surely 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] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in Metric Spaces. *ACM Computing Surveys*, 33(3):273–321, September 2001.

[2] F. Falchi, C. Lucchese, S. Orlando, R. Perego, and F. 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] S. Pandey, A. Broder, F. Chierichetti, V. Josifovski, R. Kumar, and S. 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] P. Sermpezis, T. Giannakas, T. Spyropoulos, and L. 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] A. F. Auch, H. Klenk, and M. 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] J. Weston, S. Chopra, and A. Bordes. Memory networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[7] A. Graves, G. Wayne, and I. Danihelka. Neural Turing Machines. *arXiv, preprint* `arXiv:1410.5401 [CS.NE]`, 2014.

[8] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. Meta-Learning with Memory-Augmented Neural Networks. In *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] D. Crankshaw, X. Wang, J. E Gonzalez, and M. J. Franklin. Scalable training and serving of personalized models. In *NIPS 2015 Workshop on Machine Learning Systems (LearningSys)*, 2015.

[10] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. 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] F. Chierichetti, R. Kumar, and S. 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] A. Bellet, A. Habrard, and M. Sebban. *Metric learning*, volume 9. Morgan & Claypool Publishers, 2015.

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

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

[15] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

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

[17] K. Jin, J. Li, H. Wang, B. Zhang, and N. Zhang. Near-linear time approximation schemes for geometric maximum coverage. *Theoretical Computer Science*, 725:64 – 78, 2018.

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

[19] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. *SIAM J. Discret. Math.*, 4(2):172–181, March 1991.

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

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

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

[23] Y. Hollander and A. Itai. A (4k+1)-competitive algorithm for the k-server with fixed cost excursion problem. Technical Report CS0880, Computer science department, Technion, 1996.

[24] F. Chung and R. Graham. Dynamic location problems with limited look-ahead. *Theoretical Computer Science*, 261(2):213 – 226, 2001. Fourth Annual International Computing and Combinatorics Conference.

[25] 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.

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

[27] G. Neglia, E. Leonardi, G. Iecker, and T. Spyropoulos. A Swiss Army Knife for Dynamic Caching in Small Cell Networks. *arXiv preprint* `arXiv:1912.10149 [cs.NI]`, 2019.

[28] E. Leonardi and G. 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.

[29] H. Che, Y. 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.

[30] G. Neglia, D. Carra, M. Feng, V. Janardhan, P. Michiardi, and D. Tsigkari. Access-time-aware cache algorithms. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 2(4), December 2017.

[31] M. Garetto, E. Leonardi, and V. 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.

[32] M. G. Kendall. The treatment of ties in ranking problems. *Biometrika*, 33(3):239–251, 1945.

[33] Scipy.org: scipy.stats.kendalltau. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kendalltau.html. Accessed: September 2020.

[34] J. McAuley, C. Targett, Q. Shi, and A. van den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, page 43–52, New York, NY, USA, 2015. Association for Computing Machinery.

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

[36] C. Fricker, P. Robert, and J. 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.

[37] B. Jiang, P. Nain, and D. 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.

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

[39] N. E. Choungmo Fofack, P. Nain, G. Neglia, and D. Towsley. Analysis of TTL-based Cache Networks. In *ValueTools - 6th International Conference on Performance Evaluation Methodologies and Tools - 2012*, Cargèse, France, October 2012.

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

Supplementary Material for paper:

Similarity Caching: Theory and Algorithms

Appendix A
Proof of (7)

To ensure that the LHS in (7) is bounded, we assume that $\hat{C} = \sup_{x,\mathcal{S}\neq\emptyset} C(x,\mathcal{S}) < \infty$, i.e. the cost paid to serve any request from any state is finite.

Under such condition, we prove that:

$$\liminf_{T\to\infty} C_A(\mathcal{S}_1, \mathbf{r}_T) \geq \min_{\mathcal{S}} \mathcal{C}(\mathcal{S}), \text{ w.p.1.}$$

First, observe that

$$C_A(\mathcal{S}_1, \mathbf{r}_T) = \frac{1}{T}\sum_{t=1}^{T} C_m(\mathcal{S}_t, \mathcal{S}_{t+1}) + C(r_t, \mathcal{S}_{t+1}) \geq \frac{1}{T}\sum_{t=1}^{T} C(r_t, \mathcal{S}_t),$$

as $C(r_t, \mathcal{S}_{t+1}) = C_a(r_t, \mathcal{S}_t)$, if $\mathcal{S}_{t+1} = \mathcal{S}_t$, or $C_m(\mathcal{S}_t, \mathcal{S}_{t+1}) = C_r$ otherwise. Then it is sufficient to prove that

$$\liminf_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} C(r_t, \mathcal{S}_t) \geq \min_{\mathcal{S}} \mathcal{C}(\mathcal{S}), \text{ w.p.1.}$$

We define $Y_t = C(r_t, \mathcal{S}_t) - \mathcal{C}(\mathcal{S}_t)$ and observe that by construction $\mathbb{E}[C(r_t, \mathcal{S}_t)] = \mathcal{C}(\mathcal{S}_t)$, therefore $\mathbb{E}[Y_t] = 0$. Now let $M_t = \sum_{i=0}^{t} Y_i$ with $M_0 = 0$; M_t is a martingale, since i) $\mathbb{E}[M_t \mid \mathcal{F}_{t-1}] = M_{t-1} + \mathbb{E}[Y_t] = M_{t-1}$, where $\{\mathcal{F}_t\}_{t\in\mathbb{N}}$ is the associated natural filtration $\mathcal{F}_n = \sigma(\{Y_i\}_{i\leq n})$; ii) $\mathbb{E}[|M_t|] < t\hat{C} < \infty$, given that by construction $|M_{t+1} - M_t| = |Y_t| \leq \hat{C}$.

Let $S_t = \frac{1}{t}M_t$ (with $S_0 = 0$), by applying Azuma inequality to M_t, we have that, for any $\epsilon > 0$:

$$\mathbb{P}(|S_t| \geq \varepsilon) = \mathbb{P}(|M_t| \geq \varepsilon t) = \mathbb{P}(|M_t - M_0| \geq \varepsilon t) \leq 2\exp\left(-\frac{\varepsilon^2 t}{2\hat{C}^2}\right).$$

Therefore by a standard application of the Borel-Cantelli lemma we can claim that:

$$\lim_{t\to\infty} S_t = 0 \quad \text{w.p. 1.}$$

In other words. we have that:

$$\liminf_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} C(r_t, \mathcal{S}_t) - \min_{\mathcal{S}} \mathcal{C}(\mathcal{S}) \geq \liminf_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} [C(r_t, \mathcal{S}_t) - \mathcal{C}(\mathcal{S}_t)] = \liminf_{T\to\infty} S_T = \lim_{T\to\infty} S_T = 0, \quad \text{w.p. 1,}$$

from which the assertion follows immediately.

Appendix B
Proof of Theorem V.3

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. Let $\mathcal{Y}_n = \mathcal{Y}(t_n)$ be the sequence of caching configurations observed under policy Greedy (i.e., $\{t_n\}_n$ is the sequence of random instants at which the contents in the cache are modified), and $C(\mathcal{Y}_n)$. the corresponding average costs. We set conventionally $\mathcal{Y}_{n+1} = \mathcal{Y}_n$ whenever $t_{n+1} = \infty$.

Note that $\mathcal{Y}_n = \mathcal{Y}(t_n)$ is a stochastic sequence of set valued random variables (i.e $\mathcal{Y}_n \in \mathcal{X}^k$, where $\mathcal{X}^k$ is endowed with with the metric $d(\mathcal{Y}, \mathcal{Y}') = \min_\pi \sum_i d(y_i, y_{\pi(i)})$, where π denotes a generic permutation of order k), with associated law $\mathbb{P}_n(\mathrm{d}\mathcal{Y})$. Note that, by construction, the sequence of associated costs satisfies $\mathbb{E}[C(\mathcal{Y}_n) \mid \mathcal{Y}_{n-1}] \leq C(\mathcal{Y}_{n-1})$ i.e. sequence $C(\mathcal{Y}_n)$ it

is a non-negative uniformly-bounded super-martingale. Therefore $\lim_{n\to\infty} C(\mathcal{Y}_n) = C_\infty$ exists w.p.1 by Doob Convergence Theorem. Moreover since $C(\mathcal{Y}_n)$ is uniformly bounded $\mathbb{E}[|C(\mathcal{Y}_n) - C_\infty|] \to 0$ as well. Note that not necessarily $\{\mathcal{Y}_n\}_n$ converges point-wise: however, observe that, since the domain of configurations $\mathcal{X}^k$ is compact, by Prokhorov's theorem, there exists a sub-sequence $(\mathcal{Y}_{n_i})$ that converges weakly (i.e. in distribution) to a random variable $\mathcal{Y}_\infty$. In addition $\mathbb{E}[C_\infty] = \lim \mathbb{E}[C(\mathcal{Y}_n)] = \lim \mathbb{E}[C(\mathcal{Y}_{n_i})] = \mathbb{E}[C(\mathcal{Y}_\infty)]$, due to the continuity and boundness of $C()$ with respect to its argument.

Let $n_\epsilon < \infty$ be the minimum (discrete) time, such that $\mathbb{E}[C(\mathcal{Y}_{n_\epsilon})] < \mathbb{E}[C(\mathcal{Y}_\infty)] + \epsilon$ for an arbitrary $\epsilon > 0$. We denote with $\mathcal{P}_{n_\epsilon}(\mathrm{d}\mathcal{Y})$ the law of $\mathcal{Y}_{n_\epsilon}$.

Now we define as $\mathcal{Z}_{a,p}$ the open set of configurations $\mathcal{Y}$ such that:

$$B_a(\mathcal{Y}) := \{\mathcal{Y}' = \mathcal{Y} \cup y^c \setminus y^e \ : C(\mathcal{Y}) > C(\mathcal{Y}') + a, \mathcal{Y}' \in \mathcal{S}\} \neq \emptyset$$

and

$$p_a(\mathcal{Y}) = \frac{\int_{B_a(\mathcal{Y})} \lambda(x)\,\mathrm{d}x}{\int_{\mathcal{X}} \lambda(x)\,\mathrm{d}x} > p.$$

Note that $Z_{a,p}$, provided that $C_a()$ and $\lambda()$ are continuous, is by definition an open set. Now, note that any configuration $\mathcal{Y}$ which is not local optimal, necessarily lies in some $\mathcal{Z}_{a,p}$. In particular any configuration $\mathcal{Y}$ which is not local optimal must belong to $\mathcal{Z}_{1/\sqrt{m},1/\sqrt{m}}$ for sufficiently large m. Therefore the set of non local optimal configurations lies in $\cup_{m\geq 1}\mathcal{Z}_{1/\sqrt{m},1/\sqrt{m}}$.

Now by construction, we have:

$$\mathbb{E}[\mathbb{E}[C(\mathcal{Y}_\infty) \mid \mathcal{Y}_{n_\epsilon}]] \leq \mathbb{E}[\mathbb{E}[C(\mathcal{Y}_{n_\epsilon+1}) \mid \mathcal{Y}_{n_\epsilon}]] \leq \mathbb{E}[C(\mathcal{Y}_{n_\epsilon})] - ap\mathbb{P}(\mathcal{Y}_{n_\epsilon} \in \mathcal{Z}_{a,p})$$

but this is in contradiction with the fact that by construction $\mathbb{E}_{\mathbb{P}_M}[C(\mathcal{Y}_{n_\epsilon})] \leq \mathbb{E}[C(\mathcal{Y}_\infty) \mid \mathcal{Y}_{n_\epsilon}] + \epsilon$, unless $ap\mathbb{P}(\mathcal{Y}_{n_\epsilon} \in \mathcal{Z}_{a,p}) \leq \epsilon$. The assertion follows from the arbitrariness of ϵ. Indeed for any fixed $m \geq 1$ choosing $a = 1/\sqrt{m}$, $p = 1/\sqrt{m}$ and $\epsilon = 1/(mk)$ with $k \geq 1$ sufficiently large we obtain that:

$$\limsup_{n\to\infty} \mathbb{P}(\mathcal{Y}_n \in \mathcal{Z}_{\frac{1}{\sqrt{m}},\frac{1}{\sqrt{m}}}) = \limsup_{n\to\infty} \mathbb{P}(\mathcal{Y}_n \in \cup_{m'\leq m}\mathcal{Z}_{\frac{1}{\sqrt{m'}},\frac{1}{\sqrt{m'}}}) \geq \limsup_{i\to\infty} \mathbb{P}(\mathcal{Y}_{n_i} \in \cup_{m'\leq m}\mathcal{Z}_{\frac{1}{\sqrt{m'}},\frac{1}{\sqrt{m'}}}) = 0$$

Finally we have:

$$0 = \limsup_{i\to\infty} \mathbb{P}(\mathcal{Y}_{n_i} \in \cup_{m'\leq m}\mathcal{Z}_{\frac{1}{\sqrt{m'}},\frac{1}{\sqrt{m'}}}) \geq \liminf_{i\to\infty} \mathbb{P}(\mathcal{Y}_{n_i} \in \cup_{m'\leq m}\mathcal{Z}_{\frac{1}{\sqrt{m'}},\frac{1}{\sqrt{m'}}}) \geq \mathbb{P}(\mathcal{Y}_\infty \in \cup_{m'\leq m}\mathcal{Z}_{\frac{1}{\sqrt{m'}},\frac{1}{\sqrt{m'}}}).$$

where the last inequality descends from the Portmanteau's Theorem (recall that set $\cup_{m'\leq m}\mathcal{Z}_{\frac{1}{\sqrt{m'}},\frac{1}{\sqrt{m'}}}$ is open). Therefore, by continuity of probability with respect to increasing events, we have $\mathbb{P}(\mathcal{Y}_\infty \in \cup_{m'}\mathcal{Z}_{\frac{1}{\sqrt{m'}},\frac{1}{\sqrt{m'}}}) = 0$, i.e. w.p.1, $\mathcal{Y}_\infty$ is local optimal. The assertion holds since, as already observed, $\mathbb{E}[C_\infty] = \mathbb{E}[C(\mathcal{Y}_\infty)]$.

Observe that previous result can be strengthened under the additional assumption that system configuration $\mathcal{Y}_n$ converges w.p.1 to a variable $\mathcal{Y}_\infty$ (as for the case in which $|\mathcal{X}|$ is finite or denumerable). In such a case we can prove w.p.1 convergence to an optimal local configuration. □

Appendix C
Proof of Theorem V.4

Proof. 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 [26]. 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 [38].

In simulated annealing, it is usual to consider the transition probability from state $\mathcal{S}_t = \mathcal{S}$ to state $\mathcal{S}'$ as combination of two different random choices. First, $\mathcal{S}'$ is selected as a potential candidate with probability $R(\mathcal{S},\mathcal{S}')$. In our case, non null $R(\mathcal{S},\mathcal{S}')$ are associated only to transitions between states $\mathcal{S}$ and $\mathcal{S}' = \mathcal{S} \setminus \{y\} \cup \{x\}$. Furthermore, $\mathcal{S}'$ is selected as candidate only if x has been requested and y has been selected for eviction, i.e., $R(\mathcal{S},\mathcal{S}') = \lambda_x(p(\mathcal{S}))_y$. Second, this transition is accepted with a probability that depends on the corresponding costs of the two states, i.e., $\min(1, \exp((\mathcal{C}(\mathcal{S}) - \mathcal{C}(\mathcal{S}'))/T(t)))$.

In [38] weak-reversibility is a structural property of the MC, which is determined by the configuration of possible state transitions $\mathcal{S} \to \mathcal{S}'$ (i.e., those that have $R(\mathcal{S},\mathcal{S}') > 0$) and by the behavior of state function $\mathcal{C}(\mathcal{S})$, which deterministically determines the acceptance probabilities. We say that state $\mathcal{U}$ is reachable from state $\mathcal{S} \neq \mathcal{U}$ at height E, if there exists a sequence of states $\mathcal{S} = \mathcal{S}_1, \mathcal{S}_2, \cdots, \mathcal{S}_n = \mathcal{U}$, such that all transitions $\mathcal{S}_i \to \mathcal{S}_{i+1}$ have positive probability $R(\mathcal{S}_i,\mathcal{S}_{i+1})$ and $\mathcal{C}(\mathcal{S}_i) \leq E$ for each i. The MC is weakly-reversible if, for every value E, state $\mathcal{U}$ is reachable at height E from state $\mathcal{S}$ if and only if state $\mathcal{S}$ is reachable at height E from state $\mathcal{U}$. The policy OSA defines a weakly-reversible MC. Observe, indeed, that $R(\mathcal{S}',\mathcal{S}) > 0$ if and only if $R(\mathcal{S},\mathcal{S}') > 0$. Moreover note that the MC is irreducible. In particular, any state $\mathcal{U}$ is reachable from any state $\mathcal{S}$ at height at most $\mathcal{C}(\mathcal{S}) + k\Delta\mathcal{C}_{\max}$. In fact, at most k transitions are required to move from $\mathcal{S}$ to $\mathcal{U}$, corresponding to a request for each object that is in $\mathcal{U}$, but not in $\mathcal{S}$. In particular, any global minimum is reachable from any local minimum $\mathcal{S}$ at height at most $\mathcal{C}(\mathcal{S}) + k\Delta\mathcal{C}_{\max}$. The thesis then follows from a direct application of [38, Thm. 1]. □

Appendix D
Proof of Theorem V.5

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 stays in the cache until eviction if 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 ordered 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 [39]. 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 can violate the hard buffer constraint.[7] The value of T_c is obtained by imposing the expected occupancy to be equal to the buffer size, i.e.,

$$\sum_{x \in \mathcal{X}} \pi_x = k, \tag{13}$$

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

$$\Delta\mathcal{C}_x(\mathcal{S}) \triangleq \mathcal{C}(\mathcal{S} \setminus \{x\}) - \mathcal{C}(\mathcal{S}). \tag{14}$$

If the state of the cache does not change, the expected sojourn time of content x in the cache can be computed as:

$$\mathbb{E}[T_S] = \frac{e^{\frac{\Delta\mathcal{C}_x(\mathcal{S})}{C_r} T_c} - 1}{\frac{\Delta\mathcal{C}_x(\mathcal{S})}{C_r}} \triangleq \frac{1}{\nu_x(\mathcal{S})}. \tag{15}$$

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 (15), 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 $\mathcal{S}$). [28] shows that EA is very precise in practice for complex systems of interacting caches.

Results for regular perturbations of Markov chains [40] 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 [27].

Reference [27] 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 scenario, there is an analogy between the two problems. Here, two similar/close contents interact because they can satisfy the same requests. In [27], 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. More formally, the gain from a given allocation of copies of content f, $G_f(\mathbf{x}_f))$, corresponds here to the cost reduction $\mathcal{C}(\emptyset) - \mathcal{C}(\mathcal{S})$. Similarly, the gain from the copy at base station b, $\Delta G_f^{(b)}(\mathbf{x}_f)$, corresponds to the gain from content x, $\Delta\mathcal{C}_x(\mathcal{S})$. Moreover, transition rates of the MC when adding/removing a content x, as function of the current set of contents $\mathcal{S}$, have the same "structure" of the transition rates in [27]. It follows that many results in [27] have a corresponding result for our problem, that can often be obtained simply replacing the matching quantities. For example, from the rate expressions and the constraint (13) we can conclude that there exists a sequence (q_n) converging to 0, such that $T_c(q_n) = C_r/\gamma \log(1/q_n)$ [27, Sect. IV.C].

In particular, we can adapt the proof of [27, 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. More formally, given two states that differ only for one element, i.e. $\mathcal{S} = \mathcal{U} \cup \{x\}$ and $\mathcal{S}' = \mathcal{U} \cup \{x'\}$ with $x, x' \notin \mathcal{U}$, we can prove that if $\mathcal{S}$ is stochastically stable then $\mathcal{C}(\mathcal{S}) \leq \mathcal{C}(\mathcal{S}')$.

We briefly present the steps of the proofs that need to be adapted. As for any cache state $\mathcal{S} = \{y_1, y_2, \ldots, y_l\}$ holds $\mathcal{C}(\emptyset) - \mathcal{C}(\mathcal{S}) = \sum_{i=1}^{l} \Delta\mathcal{C}_{y_i}\left(\cup_{j=1}^{i}\{y_j\}\right)$ (corresponding to [27, Eq. (5)]), the inequality $\mathcal{C}(\mathcal{S}) \leq \mathcal{C}(\mathcal{S}')$ holds if and only if $\Delta\mathcal{C}_x(\mathcal{S}) \geq \Delta\mathcal{C}_{x'}(\mathcal{S}')$. We can prove the last inequality showing that $\Delta\mathcal{C}_x(\mathcal{S}) \geq \gamma$ and $\Delta\mathcal{C}_{x'}(\mathcal{S}') \leq \gamma$. The proof exploits an additional state function $\phi(\mathcal{S}) \triangleq \mathcal{C}(\emptyset) - \mathcal{C}(\mathcal{S}) - \gamma|\mathcal{S}|$, that characterizes the dominant transitions [27, Sect. IV.E]. □

[7] 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.

Appendix E
Convexity of $F(v)$

Let $F(v) = \int_{\mathcal{B}(y,v)} C(x,y)\,\mathrm{d}x$, where $\mathcal{B}(y,v)$ denotes the ball of volume v with center y. Remember that $C(x,y)$ depends on x and y through the norm-induced distance $d(x,y) = \|x-y\|$. By the change of variable $z = x - y$, we obtain

$$F(v) = \int_{\mathcal{B}(0,v)} C(z,0)\,\mathrm{d}z.$$

We use the short form $\mathcal{B}(v)$ for $\mathcal{B}(0,v)$ and $c(z)$ for $C(z,0)$. Note that $c(z)$ is an increasing function of $\|z\|$.

We prove convexity of $F(v)$ over $\mathbb{R}^+$ using the definition. Consider $v_1, v_2 \in \mathbb{R}^+$ with $v_1 \le v_2$, $\alpha \in [0,1]$ and $v_\alpha = \alpha v_1 + (1-\alpha)v_2$. We also denote by $c_\alpha = \max_{z\in\mathcal{B}(v_\alpha)} c(z)$ (that is achieved at the points z at the boundary of the ball $\mathcal{B}(v_\alpha)$). We want to prove that $\alpha F(v_1) + (1-\alpha)F(v_2) \ge F(v_\alpha)$.

$$\begin{aligned}
\alpha F(v_1) + (1-\alpha)F(v_2) - F(v_\alpha) &= \alpha \int_{\mathcal{B}(v_1)} c(z)\,\mathrm{d}z + (1-\alpha)\int_{\mathcal{B}(v_2)} c(z)\,\mathrm{d}z - \int_{\mathcal{B}(v_\alpha)} c(z)\,\mathrm{d}z \\
&= \alpha \int_{\mathcal{B}(v_1)} c(z)\,\mathrm{d}z + (1-\alpha)\int_{\mathcal{B}(v_2)} c(z)\,\mathrm{d}z - \alpha\int_{\mathcal{B}(v_\alpha)} c(z)\,\mathrm{d}z - (1-\alpha)\int_{\mathcal{B}(v_\alpha)} c(z)\,\mathrm{d}z \\
&= -\alpha \int_{\mathcal{B}(v_\alpha)\setminus\mathcal{B}(v_1)} c(z)\,\mathrm{d}z + (1-\alpha)\int_{\mathcal{B}(v_2)\setminus\mathcal{B}(v_\alpha)} c(z)\,\mathrm{d}z \\
&\ge -\alpha \int_{\mathcal{B}(v_\alpha)\setminus\mathcal{B}(v_1)} c_\alpha\,\mathrm{d}z + (1-\alpha)\int_{\mathcal{B}(v_2)\setminus\mathcal{B}(v_\alpha)} c_\alpha\,\mathrm{d}z \\
&= (-\alpha(v_\alpha - v_1) + (1-\alpha)(v_2 - v_\alpha))c_\alpha = 0,
\end{aligned}$$

where the inequality follows from the monotonicity of $c(z)$ (note indeed that $c(z) \ge c_\alpha$ for $z \in \mathcal{B}(v_2)\setminus\mathcal{B}(v_\alpha)$, while $c(z) \le c_\alpha$ for $z \in \mathcal{B}(v_\alpha)\setminus\mathcal{B}(v_1)$).

Appendix F
Performance bound in the continuous scenario ($C_r < \infty$)

When C_r is finite, the cost c_i within a cell takes two different expression depending on whether r_i is smaller than or greater than $\bar{d} = C_r^{1/\gamma}$:

$$c_i(r_i) = \begin{cases} 4\frac{r_i^{\gamma+2}}{\gamma+2}, & \text{if } r_i \le \bar{d}, \\ 4\frac{\bar{d}^{\gamma+2}}{\gamma+2} + C_r\left(\frac{1}{k_i} - 2\bar{d}^2\right), & \text{if } r_i > \bar{d}, \end{cases}$$

or, as a function of k_i:

$$c_i(k_i) = \begin{cases} \zeta\bar{k}^{-\frac{\gamma+2}{2}} + C_r\left(\frac{1}{k_i} - \frac{1}{\bar{k}}\right), & \text{if } k_i < \bar{k}, \\ \zeta k_i^{-\frac{\gamma+2}{2}}, & \text{if } k_i \ge \bar{k}, \end{cases}$$

where $\bar{k}$ is the value for which diamonds in the region have radius $\bar{d}$, i.e., $\bar{k} \triangleq 1/(2\bar{d}^2) = 1/(2C_r^{2/\gamma})$. We observe that $c_i(k_i)$ is a decreasing function of k_i (as intuitively expected).

The optimization problem becomes:

$$\begin{aligned}
\underset{k_1,\dots,k_M}{\text{minimize}} \quad & \sum_{i=1}^{M} \lambda_i \left(\zeta\bar{k}^{-\frac{\gamma}{2}} + C_r\left(1 - \frac{k_i}{\bar{k}}\right)\right)\mathbb{1}_{k_i<\bar{k}}, \\
& + \lambda_i \zeta k_i^{-\frac{\gamma}{2}}\mathbb{1}_{k_i\ge\bar{k}} \\
\text{subject to} \quad & \sum_{i=1}^{M} k_i = k.
\end{aligned} \tag{16}$$

Let us suppose that $\lambda_1 \ge \lambda_2 \ge \cdots \ge \lambda_M$. Consider an allocation of cache slots $\boldsymbol{k}$ such that there exist i and j with $i < j$ such that $k_i < \bar{k}$ and $k_j > 0$. We can improve such allocation by moving cache slots from region j to region i at least as far as the number of slots for region i does not reach $\bar{k}$. Formally, one can check that the allocation $\boldsymbol{k}'$ such that $k'_i = k_i + \min(\bar{k} - k_i, k_j)$, $k'_j = k_j - \min(\bar{k} - k_i, k_j)$, and $k'_l = k_l$ for $l \notin \{i,j\}$, has a smaller cost than $\boldsymbol{k}$. From this observation it follows that the optimal allocation has the following structure: the most popular i^* regions receive more than $\bar{k}$ slots, region i^*+1 may receive $k_{i^*+1} > 0$ slots (but $k_{i^*+1} < \bar{k}$) , and all other regions do not receive any slot.

In the limit for large M we can ignore the effect of the region i^*+1 and only consider the regions that are popular enough to receive more than $\bar{k}$ slots and those that do not receive any cache slot. If the measure of each level set of $\lambda(x)$ is null, a

threshold λ^* can be identified, such that the first (resp. second) set of regions corresponds to the part of the Euclidean space where the request rate exceeds (resp. does not exceed) λ^*. We obtain the approximate cost:

$$\min \mathcal{C}(\boldsymbol{k}) \approx \zeta k^{-\gamma/2} \left(\int_{\mathcal{X}:\lambda(x)>\lambda^*} \lambda(x)^{\frac{2}{\gamma+2}} \, \mathrm{d}x \right)^{\frac{\gamma+2}{2}} + C_r \left(\int_{\mathcal{X}:\lambda(x)<\lambda^*} \lambda(x) \, \mathrm{d}x \right). \tag{17}$$

Under the assumptions above and that the density $\lambda(x)$ is a continuous function, λ^* can be determined considering that in the popular space region

$$k(x) = k \frac{\lambda(x)^{\frac{2}{\gamma+2}}}{\int_{\mathcal{X}:\lambda(x)>\lambda^*} \lambda(x)^{\frac{2}{\gamma+2}} \, \mathrm{d}x} \geq \bar{k} = \frac{1}{2C_r^{\frac{2}{\gamma}}}.$$

As $\lambda(x)$ is continuous, we can select points x whose request rate is arbitrarily close to (but larger than) λ^*, and then we obtain the inequality

$$k(\lambda^* + \epsilon)^{\frac{2}{\gamma+2}} \geq \frac{1}{2C_r^{2/\gamma}} \left(\int_{\mathcal{X}:\lambda(x)>\lambda^*} \lambda(x)^{\frac{2}{\gamma+2}} \, \mathrm{d}x \right).$$

Moreover, given the arbitrariness of ϵ, and recalling (17), we identify λ^* with the smallest value (the inf) that satisfies this inequality, which necessarily is the only solution of

$$k\lambda^{*\frac{2}{\gamma+2}} = \frac{1}{2C_r^{2/\gamma}} \left(\int_{\mathcal{X}:\lambda(x)>\lambda^*} \lambda(x)^{\frac{2}{\gamma+2}} \, \mathrm{d}x \right).$$