GRADES: Gradient Descent for Similarity Caching

Anirudh Sabnis, *Student Member, IEEE*, Tareq Si Salem, *Student Member, IEEE*, Giovanni Neglia, *Member, IEEE*, Michele Garetto, *Member, IEEE*, Emilio Leonardi, *Senior Member, IEEE*, and Ramesh K. Sitaraman, *Fellow, IEEE, ACM*

***Abstract*— A similarity cache can reply to a query for an object with similar objects stored locally. In some applications of similarity caches, queries and objects are naturally represented as points in a continuous space. This is for example the case of 360° videos where user's head orientation—expressed in spherical coordinates—determines what part of the video needs to be retrieved, or of recommendation systems where a metric learning technique is used to embed the objects in a finite dimensional space with an opportune distance to capture content dissimilarity. Existing similarity caching policies are simple modifications of classic policies like LRU, LFU, and *q*LRU and ignore the continuous nature of the space where objects are embedded. In this paper, we propose GRADES, a new similarity caching policy that uses gradient descent to navigate the continuous space and find appropriate objects to store in the cache. We provide theoretical convergence guarantees and show GRADES increases the similarity of the objects served by the cache in both applications mentioned above.**

***Index Terms*— Content distribution networks, approximate computing.**

I. INTRODUCTION

SIMILARITY searching [1] is a key building block for a large variety of applications including multimedia retrieval [2]–[4], recommender systems [5]–[7], genome study [8], [9], machine learning training [10]–[12], and serving [13]–[22]. Given a query for an object, the goal is to retrieve one or more similar objects from a repository. In the traditional setting, a cache is used to speed up object retrieval: once similarity search has identified the set of similar objects in the global catalog, the system checks if some of these objects are stored in the cache memory. In this setting, the cache performs a local *exact* lookup for the objects. Similarity search over the catalog can be itself a time-consuming operation, equivalent to linearly scanning the whole catalog [23]. Moreover, if users generating the queries are located far from the repository, they may experience long delays.

In order to solve these problems, the seminal papers [3], [5] proposed, almost at the same time, a different use of the cache: clients' requests are directly forwarded to the cache; then the cache performs a similarity search over the set of locally stored objects and possibly serves the requests without the need to forward the query to the (remote) repository. The cache thus reduces the overall serving time at the cost of providing objects *less similar* than those the repository would provide. This operation was named *similarity caching*, in contrast to the traditional *exact caching*. As recognized in [24], the idea of similarity caching has been rediscovered a number of times under different names: recognition caches [15], [16], approximate deduplication [18], semantic caches [19], prediction caches [14], approximate caches [20], and soft caches [7], [25].

In many applications similarity is quantified using supervised machine learning techniques that collectively go under the name of distance metric learning [26]. These techniques learn how to map similar objects to vectors in $\mathbb{R}^d$ (called *embeddings*) that are close according to p-norm distances, cosine similarity, or Mahalanobis distances. Requests and objects live then in a continuous space. For instance, in augmented reality applications we often require to identify similar objects: the image is coded into a query, i.e., an embedding in $\mathbb{R}^d$, and the application logic finds similar images to be returned to the user [15]–[17], [19]. This is also the case for other potential applications of similarity caching like 360° videos, where requests for parts of the video are implicitly dictated by the user's head orientation expressed in spherical coordinates.

Existing dynamic policies for similarity caches adapt and extend well-known exact caching policies, like LRU and LFU, to deal with the notion of a *dissimilarity cost*, i.e., how distant is a cached object from the request. As a consequence, they treat requests and objects as discrete entities and ignore the continuous representation space.

This paper proposes GRADES, the first similarity caching policy designed to exploit object embeddings in $\mathbb{R}^d$ with a distance that captures dissimilarity costs. While previous policies update the cache state by replacing a cached object with (in general) a distant one—corresponding to "jumps" in the representation space—GRADES incrementally updates the (embedding of) each object using a gradient descent step to progressively reduce the dissimilarity cost. Qualitatively, as shown in Fig. 1, the objects in the cache smoothly move in the representation space to find their optimal position,

Manuscript received 25 June 2021; revised 24 March 2022; accepted 26 May 2022; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor J.-W. Lee. Date of publication 13 July 2022; date of current version 16 February 2023. This work was supported in part by the National Science Foundation under Grant CNS-1763617 and Grant CNS-1901137 and in part by Inria under the Exploratory Action MAMMALS. *(Corresponding author: Anirudh Sabnis.)*

Anirudh Sabnis and Ramesh K. Sitaraman are with the College of Information and Computer Sciences, University of Massachusetts Amherst, Amherst, MA 01003 USA (e-mail: asabnis@cs.umass.edu; ramesh@cs.umass.edu).

Tareq Si Salem and Giovanni Neglia are with Inria, Université Côte d'Azur, 06902 Sophia Antipolis, France (e-mail: tareq.si-salem@inria.fr; giovanni.neglia@inria.fr).

Michele Garetto is with the Dipartimento di Informatica, Università degli Studi di Torino, 10149 Turin, Italy (e-mail: garetto@di.unito.it).

Emilio Leonardi is with the Politecnico di Torino, 10129 Turin, Italy (e-mail: leonardi@polito.it).

This article has supplementary downloadable material available at https://doi.org/10.1109/TNET.2022.3187044, provided by the authors.

Digital Object Identifier 10.1109/TNET.2022.3187044

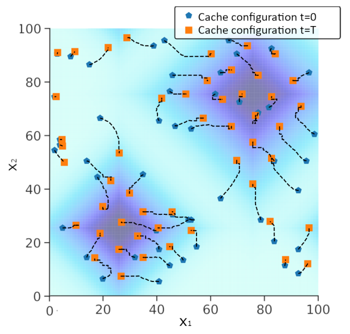


Fig. 1. Cached objects' movements in the representation space ($\mathbb{R}^2$) during $[0, T]$ when the cache is managed by GRADES. The catalog is made by the points in a 100×100 grid, dark shaded areas correspond to more popular objects. Dissimilarity cost $C_d(x, y) = 1/10 \, \|x - y\|_1$, retrieval cost $C_r = 1$, cache size $k = 50$. See the description in Sec. V.

i.e., where they can serve a large number of requests with small dissimilarity cost.

We prove that in a stationary setting, with an opportune choice of the gradient step sizes, GRADES converges to a cache configuration that corresponds to a critical point of the service cost (likely a local minimum). Our experiments based on realistic traces (made available online [27]) shows that GRADES outperforms existing similarity caching policies both for 360° videos and recommendation systems applications.

The paper is organized as follows. After an overview of the related work in Sec. II, we present the formal problem definition in Sec. III. We introduce GRADES and its theoretical guarantees in Sec. IV. Finally, experimental results are presented in Sec. V.

II. Related Work

Most existing policies for similarity caches generalize well-known exact caching policies, like LRU and LFU, to the new context, where besides the exact hits and misses, *approximate hits* are also possible. For example SIM-LRU [5], [13] maintains objects in an ordered queue and serves an object from the cache if its distance to the requested object is lesser than a given threshold (an approximate hit occurs). The object is then moved to the front of the queue. When no object in the cache is close enough to the request, there is a *miss*. The object is then retrieved from the server and inserted at the front of the queue, possibly evicting objects from the back. RND-LRU [5] is a variant of SIM-LRU where the threshold is replaced by a random variable that is a function of the dissimilarity cost i.e., the cost associated to the distance between a cached object and the request. As with SIM-LRU and RND-LRU that are adaptations of LRU, qLRU-ΔC [24] modifies qLRU [28] by introducing a refresh probability that depends on the similarity. Finally, DUEL [24] is inspired by LFU, and decides which object to evict by tracking the dissimilarity cost i.e., the cost associated to the distance between a cached object and the request accumulated over a given time-window. In our experiments we compare GRADES with SIM-LRU, qLRU-ΔC, and DUEL. Recently, there has been preliminary work that shows that coordinating decisions in a network of similarity caches is particularly challenging [29], [30]. GRADES can be adapted to work with a network of similarity caches by tweaking the algorithm to use the cost to retrieve content from a parent/neighboring cache in place of the cost to retrieve content from a remote server. This change, however, does not ensure optimal coordination amongst the similarity caches in the network.

Our algorithm was inspired by the work from Jorge Cortés *et al.* on coordination algorithms for mobile agents [31]–[33]). In their setting mobile agents (e.g., drones) place themselves in the space to be able to detect the largest number of events in the environment. Similarly, the objects in the cache need to position themselves to optimally serve the requests appearing over the space. Despite similarities at a high-level, their work focuses on a two-dimensional space and needs to take into account agents' movement and communication constraints that do not hold in our context.

From another point of view, GRADES gradient update can be considered as a generalization of stochastic K-means algorithms, where the function we want to minimize is not necessarily the squared Euclidean distance (as it is the case for K-means). Our proofs rely on techniques for non-convex optimization originally proposed in [34] also to study K-means.

Online caching policies based on gradient methods have been proposed in the stochastic request setting (see, e.g., [35], [36]), and, more recently, in the adversarial setting [37], [38]. In these papers, the gradient step updates a vector of length equal to the catalog size, whose component i (in $[0, 1]$) represents which fraction of object i should be stored in the cache or alternatively the probability to store i in the cache. Differently from this line of work, GRADES uses the gradient step to modify the objects in the cache and maintains a vector of size equal to the cache—then much smaller than the catalog size.

A costly operation in any similarity search system is to find the closest object to the request. A simple solution is to index the collection, e.g., with a tree based data structure, to find the exact closest object. Unfortunately, when the number of dimensions d of the representation space exceeds 10, such an approach has a computational cost comparable to a full scan of the collection [23]. For this reason a number of approximate search techniques have been developed, which trade accuracy for speed and provide one or more points close to the request, but not necessarily the closest. Prominent examples are the solutions based on locality sensitive hashing [39], product quantization [40], [41], pivots [42], or graphs [43]. In the experiments in this paper, we have performed an exact similarity search, but any of these approximated search techniques could be used in GRADES.

The paper [44] models different caching policies as Markov chains to study the tradeoff between the time required to estimate the popularity distribution of the requested objects and the accuracy of their estimation. GRADES allows to balance these contrasting goals by tuning two parameters: the learning rate and the grafting parameters (see e.g., Fig. 7 and Fig. 8). Our experiments show that GRADES achieves a better tradeoff than state-of-the-art similarity caching policies.

The original envisaged applications of similarity caches were content-based image retrieval (CBIR) [3] and contextual advertising [5]. In a CBIR system, given an image (used as a query), users can query the CBIR system to obtain images that are most similar to the query by comparing their visual contents. Here, a cache can respond with the most similar images that are available locally. Similarly, in the case of contextual advertising, the cache can provide ads similar to those matching the user profile [5]. Likewise, recommender systems can leverage similarity caches [7], [25]: a recommender system can save operating costs and decrease its response time through recommendation of relevant contents from a cache to user-generated queries, i.e., in case of a cache miss, an application proxy (e.g., YouTube) running close to the helper node (e.g., at a multi-access edge computing server) can recommend the most related files that are locally cached. More recently, similarity caches have been employed extensively for machine learning based inference systems to store queries and the respective inference results to serve future requests, for example, prediction serving systems [14], image recognition systems [15], [16], [18], object classification on the cloud [19], caching hidden layer outputs of a neural network to accelerate computation [20], network traffic classification tasks [45]. The cache can indeed respond with the results of a previous query that is very similar to the current one, thus reducing the computational burden and latency of running complex machine learning inference models.

We remark that a prior version of the work has been published in [46]. Here, we extend it by providing a complete proof for our theoretical claims in the Appendix and new experimental results in Sec. V (depicted in Fig. 6, Fig. 9, and Fig. 11).

III. Problem Definition

We consider a similarity search system where a server answers users' queries with the most similar object from a locally stored catalog. In some applications, it is required to serve k similar cached objects instead of a single object. GRADES can be augmented to provide k similar answers using the same techniques introduced in [3], [5], [47]. For example, the cache may store key-value pairs, where the key is a past query and the value is the set of k closest objects to the query. Upon a new query, the cache looks for the most similar key stored locally and returns the corresponding set of objects.

Requests satisfied by the server incur a *retrieval cost* C_r, which quantifies the delay the user experiences to retrieve the object from the remote server, and/or the additional load for the server, and/or the additional load for the network. Alternatively, the request may be satisfied by a *similarity cache* which stores a subset of the catalog. The cache provides, in general, a less similar object than what the server could provide, but incurs a negligible retrieval cost, as, for example, it is located closer to the user, or uses a faster memory storage or can perform faster lookup operations on the smaller set of stored contents.

We assume that each request or object in the catalog can be represented as a point in the d-dimensional Euclidean space $\mathbb{R}^d$. In what follows we will refer to such representations as *embeddings* and, for the sake of simplicity, we will identify each object/request with its embedding (e.g., we will say that object x belongs to $\mathbb{R}^d$). We assume all objects have the same size and the cache can store up to k objects.

Our model of the system is similar to the one considered in previous papers on similarity caching in the continuous setting like [5], [24]. Let $\mathcal{Z}$ and χ denote the catalog and the set of possible requests, respectively. Both sets may be finite or infinite, but we require them to be compact (to be able to retrieve a closest object to a given request). The "quality" of a similarity search for x depends on how similar the response object z is to the request. We assume the *dissimilarity cost* is quantified by the function $C_d(x,z) = h(\|x-z\|)$, where $h : \mathbb{R} \to \mathbb{R}^+$ is a non-decreasing non-negative function and $\|\cdot\|$ is a norm in $\mathbb{R}^d$ (e.g., the Euclidean one). For example Faiss (Facebook AI Similarity Search) library [48] for multimedia retrieval supports all p-norms for $p \in [1, \infty]$.

The state of the cache at time t is given by the set of objects $\mathcal{S}_t$ currently stored in it, $\mathcal{S}_t = \{y_t^1, y_t^2, \ldots, y_t^k\}$, with $y_t^i \in \chi \subset \mathbb{R}^d$. Requests arrive first at the cache. Given a request for object x_t at time t, let i_t denote the index of the most similar object to the request, i.e., $i_t \in \arg\min_{i=1,\ldots,k} C_d\left(x_t, y_t^{(i)}\right)$ (if there are many equally similar ones we arbitrarily select one). If the cache satisfies the request x_t, it will use content $y_t^{(i_t)}$, and the user will incur the dissimilarity cost $C_d(x_t, \mathcal{S}_t) \triangleq C_d\left(x_t, y_t^{(i_t)}\right) = \min_{y\in\mathcal{S}_t} C_d(x_t, y)$, but the retrieval cost is negligible. Alternatively, the cache can forward the request to the server, where it will be satisfied by the most similar object in the catalog. The request will generate the retrieval cost C_r, and the user will experience the dissimilarity cost $C_d(x_t, \mathcal{Z}) = \min_{z\in\mathcal{Z}} C_d(x_t, z) \le C_d(x_t, \mathcal{S}_t)$.

Ideally, the cache should compare the costs of serving request locally ($C_d(x_t, \mathcal{S}_t)$) and from the server ($C_d(x_t, \mathcal{Z}) + C_r$) and select the most convenient action. But, in order to evaluate $C_d(x_t, \mathcal{Z})$, the cache would need to store locally metadata for the whole set $\mathcal{Z}$ and find the closest object in it. The memory and computation requirements could defeat the whole utility to have a cache. For this reason we consider the cache does not know $C_d(x_t, \mathcal{Z})$, but it is easy to adapt our algorithm when it is not the case and its theoretical guarantees still hold.

In the impossibility to compare $C_d(x_t, \mathcal{S}_t)$ with the request-dependent value $C_d(x_t, \mathcal{Z}) + C_r$, the cache compare $C_d(x_t, \mathcal{S}_t)$ with a constant threshold value C_θ. If $C_d(x_t, \mathcal{S}_t) \le C_\theta$, the cache serves the request locally, otherwise it forwards it to the server. We assume C_θ is set once and for all offline. Figure 2 illustrates how requests are served.

As $C_d(x_t, \mathcal{S}_t) = C_d\left(x_t, y_t^{(i_t)}\right)$, the final cost to serve request x (denoted by $C(x_t, \mathcal{S}_t)$) depends only on $y_t^{(i_t)}$:

$$C(x_t, \mathcal{S}_t) = C\left(x_t, y_t^{(i_t)}\right) = \begin{cases} C_d\left(x_t, y_t^{(i_t)}\right), & \text{if } C_d\left(x_t, y_t^{(i_t)}\right) \le C_\theta, \\ C_r + C_d\left(x_t, \mathcal{Z}\right), & \text{otherwise.} \end{cases} \tag{1}$$

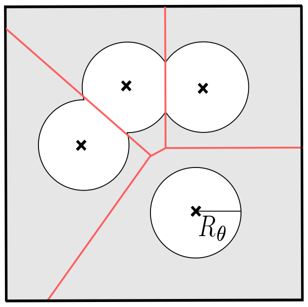


Fig. 2. Coverage of the request space ($\subset \mathbb{R}^2$) by 4 objects in the cache with norm-2 as dissimilarity cost. Crosses represent the objects. Each object is the closest point to requests in the corresponding Voronoi cell delimited by the red lines. Consider a request x falling in the Voronoi cell of an object y_i. If x is closer to y_i than the critical radius R_θ (such that $h(R_\theta) = C_\theta$), x receives y_i as reply, otherwise (it falls in the gray shaded area) it generates a miss.

After a request, the cache can update its state. As updates can themselves generate retrieval costs, we restrain to reactive policies that can only update their state by inserting the object retrieved from the server to satisfy a request.

In our setting, we assume that the two costs, C_r and C_d, can be expressed in the same unit. For instance, the two costs can quantify different aspects contributing to the overall user's quality of experience (QoE). The dissimilarity cost function $C_d(x, y) = h(||x - y||)$ can describe the QoE loss for the end user upon receiving a dissimilar object, while C_r can capture the QoE loss due to the experienced delay. As in our model, the ITU-T E-model combines additively different metrics, such as signal-to-noise-ratio, packet loss ratio, and delay, to obtain a single scalar QoE rating for voice communications [49]. More in general, it is quite common in multi-objective resource allocation problems to express the cost as a weighted sum of the different objective functions [50]–[52].

In our theoretical analysis in Sec. III, we consider the case when requests arrive according to a Poisson process and are i.i.d. distributed. In the finite case ($|\chi| < \infty$), we recover the classic independent reference model [53], where object x is requested with rate λ_x. In the continuous case, we need to consider a spatial density of requests and objects in a set $\mathcal{A} \subset \chi$ are requested with rate $\int_{\mathcal{A}} \lambda_x \, \mathrm{d}x$.

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:

$$\mathcal{C}(\mathcal{S}) \triangleq \begin{cases} \sum_x \lambda_x C(x, \mathcal{S}), & \text{finite case} \\ \int_\chi \lambda_x C(x, \mathcal{S}) \, \mathrm{d}x, & \text{continuous case.} \end{cases} \tag{2}$$

Finding an optimal set of objects $\mathcal{S}^*$ to store in the cache that minimizes the cost $\mathcal{C}(\mathcal{S})$ is NP-hard as it is a generalization of the problem considered in [24] (where $C_d(x, \mathcal{Z}) = 0$ for each $x \in \chi$). Nevertheless, for the continuous case, we propose a dynamic gradient descent based algorithm, that, under the stationary request process, can achieve a stationary point of $\mathcal{C}(\mathcal{S})$. The gradient descent based algorithm is a natural choice for the continuous setting; the algorithm takes a descent step to decrease $\mathcal{C}(\mathcal{S})$ at each time step. Further, in Sec. IV-C, we describe an adaptation of the proposed algorithm for the finite case.

IV. A Gradient-Based Algorithm

The key idea of our algorithm is to let the objects stored in the cache gradually "move" in the space $\mathbb{R}^d$ to reach a configuration where they can be used as approximate answers for a large number of requests (see Fig. 1). Upon a request at time t for x_t, the most similar object in the cache, $y_t^{(i_t)}$, is moved in the direction opposite to the gradient of the service cost ($\nabla_y C(x_t, y_t^{(i_t)})$) proportionally to a time-varying step-size (or learning rate) η_t:

$$y_{t+1}^{(i_t)} = y_t^{(i_t)} - \eta_t \nabla_y C\left(x_t, y_t^{(i_t)}\right). \tag{3}$$

It is possible to prove that $C(x, y)$ is differentiable everywhere and then the gradients in (3) exist with probability 1 when the request process is continuous.[1] When the request process is discrete, the probability that the gradient $\nabla_y C(x_t, y_t)$ does not exist may be non-zero, but we can then perturb the request by a small random vector $\epsilon \in \mathbb{R}^d$ and consider $\nabla_y C(x_t + \epsilon, y_t)$.

Note that the algorithm is oblivious to the request rate (λ_{x_t}) of the object x_t. However, every time a request is made for object x_t, the most similar object in the cache ($y_t^{(i_t)}$) is moved in the direction of the requested object. Therefore, the algorithm dynamics are sensitive to λ_{x_t}.

An attentive reader may frown upon the simple algorithm (3). First, it potentially updates the cache upon every request, even when $C_d(x_t, y_t^{(i)}) \leq C_\theta$ and the cache would not need to retrieve any object. Second, even if $y_t^{(i)}$ is the embedding of an object in the catalog, $y_{t+1}^{(i)}$ may not correspond to any object in the catalog.

In the following sections we address all issues mentioned above. After having refined the update rule (3) (Sec. IV-A), we prove that this idealized algorithm indeed converges to a critical point of $\mathcal{C}(\mathcal{S})$ (Sec. IV-B). Then, in Sec. IV-C we present a practical algorithm which 1) satisfies all our requirements, 2) keeps the state of the cache "close" to the state of the idealized algorithm, and 3) is more reactive and thus more suitable to non-stationary request processes.

A. Introducing a Projection

As requests are only for objects in the bounded set χ, there exists a norm-2 ball with radius R—$\mathcal{B}_2(R) = \{y \in \mathbb{R}^d, \|y\|_2 \leq R\}$—such that $\chi \subset \mathcal{B}_2(R)$ and $C(x, y) = C_r$ for each $y \notin \mathcal{B}_2(R)$ and $x \in \chi$. There is no advantage to store in the cache objects that do not belong to $\mathcal{B}_2(R)$ as they do not contribute to approximate any request. We then modify (3) in order to make closer to $\mathcal{B}_2(R)$ any cached object $y_t^{(i)}$ that the

[1] As far as all the $y_t^{(i)}$ are different, $\nabla_y C(x, y_t^{(i_t)})$ exists for all the points in $x \in \mathcal{X}$ but at most for a measure zero set: the set of points for which $\nabla_y C_d(x, y_t^{(i)})$ does not exist, the points for which $\|x - y_t^{(i)}\| = \|x - y_t^{(j)}\|$ for $i \neq j$, and finally the set of points in $\partial\{x \in \mathbb{R}^d : C_d(x, y_t^{(i)}) > C_\theta\}$, where ∂A denotes the boundary of the set A.

gradient update may have brought out of $\mathcal{B}_2(R)$. We write

$$y_{t+1}^{(i)} = y_t^{(i)} - \eta_t g_t^{(i)}, \tag{4}$$

$$g_t^{(i)} = \begin{cases} \nabla_y C\left(x_t, y_t^{(i)}\right), & \text{if } (i = i_t) \wedge \left(y_t^{(i)} \in \mathcal{B}_2(R)\right) \\ f(\|y_t^{(i)}\|_2 - R)\frac{y_t^{(i)}}{\|y_t^{(i)}\|_2}, & \text{if } y_t^{(i)} \notin \mathcal{B}_2(R), \\ \mathbf{0}, & \text{otherwise,} \end{cases}$$

where $f(\cdot)$ is an increasing non negative function (so that $-g_t^{(i)}$ points to the origin of the space). For technical reasons that will be required in the proof of Theorem 4.2, we want $f(0) = f'(0) = f''(0) = f'''(0) = 0$, and $f'(\infty) \in \Theta(1)$. For these reasons, we select $f(u) = \frac{\mathrm{d}}{\mathrm{d}u}\frac{u^4}{1+u^2} = 2\ u^3\frac{2+u^2}{(1+u^2)^2}$ for the proof.

B. Convergence

In this section we provide convergence results for the basic algorithm described in (4). We assume the cache update rule generates embeddings that always correspond to objects in the catalog. Moreover, we will ignore the cost of updates made *after* the request is served. These two simplifications will be removed in the next section.

It will be useful to denote the cache state as a vector $\mathbf{y}_t = (y_{t,1}, \dots, y_{t,k}) \in \mathbb{R}^{k \times d}$, obtained by concatenating the embeddings of the different objects in the cache. Similarly, we define the different costs as function of $\mathbf{y}_t$ and then write $C_d(\mathbf{y}_t)$, $C(\mathbf{y}_t)$, and $\mathcal{C}(\mathbf{y}_t)$. We are now going to prove that algorithm (4) converges almost surely to a stationary point of $\mathcal{C}(\mathbf{y})$ and the trajectory of $\mathbf{y}_t$ is bounded almost surely.

Lemma 4.1: Let the learning rate η_t be selected so that $\sum_{t=1}^{+\infty} \eta_t = +\infty$ and $\sum_{t=1}^{+\infty} \eta_t^2 < +\infty$. The sequence $(\mathbf{y}_t)$ is bounded almost surely.

The proof of the lemma is in Appendix A. The lemma is used in the proof of the following convergence result.

Theorem 4.2: Let the learning rate η_t be selected so that $\sum_{t=1}^{+\infty} \eta_t = +\infty$ and $\sum_{t=1}^{+\infty} \eta_t^2 < +\infty$. If $\mathcal{C}(\cdot)$ is continuously differentiable up to the second order then

$$\liminf_{t\to\infty} \|\nabla_{\mathbf{y}} \mathcal{C}(\mathbf{y}_t)\|_2 = 0 \ a.s.$$

If $\mathcal{C}(\cdot)$ is continuously differentiable up to the third order then

$$\lim_{t\to\infty} \nabla_{\mathbf{y}_t} \mathcal{C}(\mathbf{y}_t) = 0 \ a.s.$$

The proof of Theorem 4.2 is in Appendix B. Our proof relies on techniques for non-convex optimization originally proposed in [34]. We think it is possible to derive similar results, under different hypotheses, using the approach based on ordinary differential equations proposed in [54].

Theorem 4.2 states that the sequence $(\mathbf{y}_t)$ converges to a critical point of $\mathcal{C}(\cdot)$, i.e., a point where the gradient is zero. This may be a saddle point, a local maximum or a local minimum of $\mathcal{C}(\cdot)$. The latter is more likely as it is the only one locally stable. The saddle points and local maxima of $\mathcal{C}(\cdot)$ are not stable, as on reaching either of these two types of points, requests that appear in the neighborhood may perturb $\mathbf{y}_t$ and the gradient descent algorithm moves $\mathbf{y}_t$ away from these points. Given the stochastic nature of the request process this is highly likely to happen.

C. Implementation

In this section we present our complete caching policy GRADES, whose pseudo-code is in Algorithm 1. Theorem 4.2 shows that the basic gradient update (4) attains a critical point of the expected cost $\mathcal{C}(\cdot)$. Nevertheless, we have assumed that this update rule always generates embeddings in $\mathbb{R}^d$ that correspond to objects in the catalog. However, if the catalog has a finite number of objects, this is unlikely to happen, as the update (4) can potentially generate any real vector. Moreover, the update (4) may modify an object in the cache upon each request and then generate a high load on the server and the network to retrieve the new modified objects.

In Sec. IV-C.1 we describe how our algorithm addresses these issues. We then move on in Sec. IV-C.2 to describe some additional features that provide a higher adaptivity of the algorithm to deal with highly non-stationary request processes, allowing for some random insertions with probability p.

Algorithm 1 GRADES
```
1: Let $k$ be the cache size and $x$ the object requested
2: if $(|\mathcal{S}_{V,t}| < k) \wedge (x \notin \mathcal{S}_{V,t})$ then          ▷ still space in cache
3:     Insert $x$ in VC
4:     Retrieve and insert $\rho(x)$ in PC
5:       $\mu(x) = \rho(x)$
6: $y_V = \arg\min_{y \in \mathcal{S}_{V,t}} C_d(x, y)$
7: Update $S_{V,t}$ according to (4)
8: if $C_d(x, y_V) \le C_\theta$ then                                          ▷ virtual hit
9:     if $\|x - y_V\| < \|\mu(y_V) - y_V\|$ then        ▷ $x$ approximates $y_V$ better than $\mu(y_V)$
10:        Evict $\mu(y_V)$
11:        Retrieve and Insert $\rho(x)$ in PC
12:          $\mu(y_V) = \rho(x)$
13:    GRAFT_HIT_UPDATE($x, S_{V,t}$)
14: $y_P = \arg\min_{y \in \mathcal{S}_{P,t}} C_d(x, y)$
15: $\xi \sim$ Uniform(0, 1)
16: if $\xi < p$ then
17:     (update, $\omega$) = GRAFT_MISS_UPDATE($S_{V,t}, x, \rho(x)$)
18:     if update then
19:         Evict $\omega$ and $\mu(\omega)$
20:         Retrieve and Insert $\rho(x)$ in VC and PC
21:           $\mu(\rho(x)) = \rho(x)$
22:         $y_P = \rho(x)$
23: if $(C_d(x, y_P) \le C_\theta) \vee (\rho(x)$ inserted in PC$)$ then
24:     Serve $y_P$
25: else
26:     Retrieve and Serve $\rho(x)$
```

1) Dealing With Finite Catalogue and Reducing Server Load

1) Dealing With Finite Catalogue and Reducing Server Load: We propose to maintain a virtual cache (VC) and a physical cache (PC). The VC only stores some metadata, but no actual object; its use is common to other policies like 2-LRU [28] or AdaptSize [55]. The VC is sometimes called shadow cache.

In our case the VC stores k vectors in $\mathbb{R}^d$ that are updated upon each request according to the basic algorithm in (4). These vectors are the embeddings of the objects we would like to store in the cache, but, as discussed above, such objects may not exist, or they may not have been retrieved yet from the server. The PC contains objects from the catalog together with their embeddings.

At a high level, the main idea behind GRADES is to maintain the PC as close as possible to the VC. We use then the current state of the VC to drive updates at the PC, i.e., object eviction and insertion. In particular each vector y_V in the VC is matched to an actual object $\mu(y_V)$ in the PC and GRADES will opportunistically update $\mu(y_V)$ to make it as close as possible to y_V.

We now describe in details Algorithm 1 using the following additional notation:

- $\mathcal{S}_{V,t}$ and $\mathcal{S}_{P,t}$ denote the state of the VC and the PC, respectively.
- $\rho(x)$ denotes the closest object in the catalogue to x.

The gray lines correspond to changes to increase algorithm adaptivity and will be discussed in Sec. IV-C.2.

Upon a request for x, if there is still space in the cache, we retrieve the most similar object in the catalogue $\rho(x)$. GRADES inserts x and $\rho(x)$ in the VC and in the PC, respectively, and matches them ($\mu(x) = \rho(x)$). These operations are described in lines 2–5. The cache will finally serve $\rho(x)$.

If the cache is already full, the closest object in VC y_V will be updated according to (4) (lines 6–7). Upon a virtual hit, if x is closer to y_V than the currently matching object $\mu(y_V)$ in the PC, GRADES takes advantage of this request to replace $\mu(y_V)$ with $\rho(x)$ (lines 9–12). In a stationary setting, the state of VC converges to a critical point of the cost (Theorem 4.2) and the PC should become closer and closer to it. Finally, the most similar object in PC is served if it is close enough to x, or if in any case $\rho(x)$ has been retrieved (line 11).

2) Increasing Adaptivity: According to what we described above, only the closest object in VC is updated upon a request (unless some projection back to $\mathcal{B}(R)$ is needed). A potential problem is that if an object x far from any other object has been accidentally inserted in VC (and the corresponding object $\rho(x)$ in PC), it may never be updated and may uselessly occupy cache space. Moreover, if at some point the request process changes abruptly, some objects in the cache that were initially useful may find themselves too far from the new requests. Again, the gradient algorithm, by itself, would not update such objects. To overcome this problem, we can graft to GRADES a more dynamic caching policy that occasionally (with probability p) updates the VC, hopefully evicting the least useful object in the VC.

The "grafting" is described by the green-shaded lines in Algorithm 1 and has been designed to support general cache eviction algorithms like LRU, LFU, and their variants. The grafted caching policy internally maintains its own data structure, e.g., an ordered queue for LRU. Upon an approximate hit, the hit update rule of the grafted policy is called (line 13). For example, LRU would move the requested object (if present in the cache) to the front of the queue. Also, with probability p, GRADES invokes the miss update rule of the grafted policy, that may lead to select an element ω to be evicted. GRADES then updates accordingly the VC and the PC (lines 19–22).

3) Algorithm Complexity: A straightforward implementation of GRADES has a time complexity of O(kd), where k is the cache capacity and d is the embedding dimension. This is because one has to iterate through the k objects in the cache to find the most similar object to the requested object. However, one could use approximate nearest neighbor index, such as those based on hierarchical navigable small worlds (HNSW) graphs [43]. Numerically, HNSW is able to answer a 10NN query over a dataset with 1 million objects in a 128-dimensional space in less than 0.5 ms with a recall greater than 97% [56]. As for the memory footprint, a typical configuration of the HNSW index requires $O(d)$ bytes per objects, where d is the number of dimensions. For instance, in case of $d = 128$ dimensional vectors, the memory required to index 10 million objects is approximately 5 GB.

TABLE I
TRACES DESCRIPTION

Trace	Number of requests	Catalog size	Dimension (d)
Synthetic	2,000,000	97,969	2
360° videos	10,000,000	25,393	3
Amazon	908,179	63,891	100
CiteULike	2,411,819	153,277	100
Movielens	620,222	136,677	200

V. EXPERIMENTS

In this section, we empirically validate our algorithm through simulations. First we demonstrate the benefit of the algorithm by using synthetic traces. Next, to demonstrate real world applicability of our algorithm, we use GRADES in the domain of caching for 360° videos and recommendation systems. We assume that the catalog coincides with the set of possible requests ($\mathcal{Z} = \mathcal{X}$) and then set $C_\theta = C_r$. The retrieval cost C_r is always equal to 1. Table I summarizes the main characteristics of the traces. Further details about the experimental setup and the properties of request traces will be described in the corresponding subsections. To the best of our knowledge, there are no public traces for similarity caching; we made our traces available online [27].

We compare GRADES with the following algorithms.

a) GREEDY is an offline static algorithm that progressively fills the cache inserting the object that provides the largest cost saving given the set of objects already inserted. The algorithm provides a $\frac{1}{2}$ approximation in terms of cost savings [57].

b) LRU+ updates the cache as the classic LRU evicting the least recently used content when needed, but it can provide approximate objects.

c) SIM-LRU [5] maintains the content in an ordered queue as LRU. It moves objects to the front upon an approximate hit, and evicts objects from the back when needed.

d) qLRU-ΔC [24] is a variant of qLRU [28] that, upon an approximate hit, moves the object to the front with a probability which is proportional to the service cost reduction the object has guaranteed on the current request.

e) DUEL [24], upon a request for object x not in cache, x is matched with an object y in the cache in a tournament aimed at deciding if x is a better candidate to be stored in the cache as compared to y. The decision is made by comparing the cost savings x and y provide over a fixed interval of time (f). If the

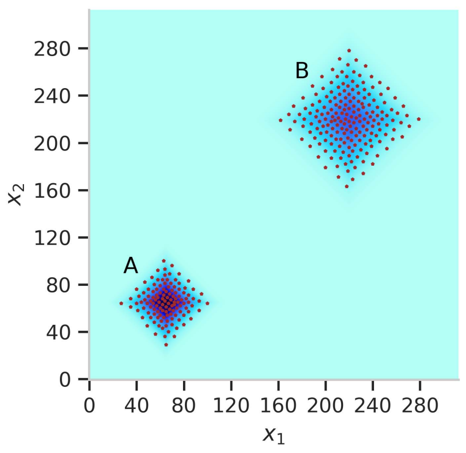


Fig. 3. The heatmap depicts the popularity distribution of objects in the grid. The darker regions A and B contain the most popular content. The circles represent the final configuration produced by the GRADES policy ($\eta = 0.64$) under the trace *Synthetic*.

new object x provides a larger cost saving, then x replaces y in the cache.

A. Synthetic Traces

We consider a setting similar to [24]. The catalog is made by the points of a $L \times L$ bi-dimensional grid with $L = 313$. The cache hs size $k = 313$.[2]

For any two objects x and y on the grid we define the approximation cost to be proportional to the norm-1 distance between the two points x and y, in particular $C_d(x, y) = \frac{1}{10}||x - y||_1$. In our experiments, we observe that GRADES converges to an expected cost that is slightly better than the approximate optimal cost as computed in [24], suggesting that GRADES converges very close to optimal.

The traffic is generated under the *Independent Reference Model* [53]. There are two popular regions A and B centered around coordinates $(65, 65)$ and $(220, 220)$, respectively; they are produced by a mix of two Gaussian distributions. In particular, an object at (norm-1) distances d_1 from the center of A and d_2 from the center of B is requested with probability

$$(d_1, d_2) \propto 0.4 \times \frac{e^{\frac{-d_1^2}{2\times 15^2}}}{\sqrt{2\pi} \times 15} + 0.6 \times \frac{e^{\frac{-d_2^2}{2\times 25^2}}}{\sqrt{2\pi} \times 25}.$$

The popularity distribution of the objects in the grid is depicted in the heat-map in Fig. 3. Figure 1 corresponds to a rescaled version of the same process.

Figure 4 shows the performance of GRADES without any graft and with different grafts (LRU+, SIM-LRU, qLRU-ΔC) for a quite large value of the grafting parameter ($p = 10^{-2}$). GRADES/X denotes GRADES grafted with policy X. We observe that GRADES achieves the smallest cost, and by

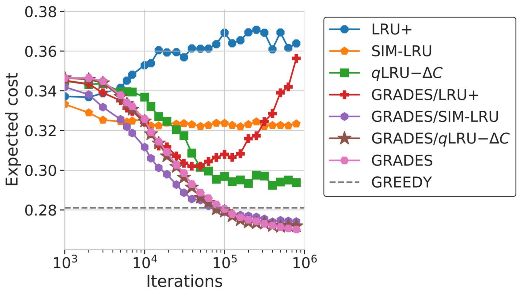


Fig. 4. Expected cost $\mathcal{C}(\cdot)$ incurred by different policies under the trace *Synthetic*. *LRU+*, SIM-LRU, qLRU-ΔC ($q = 10^{-2}$), GREEDY, and GRADES ($\eta = 0.64$, plain and grafted with $p = 10^{-1}$). Cache size $k = 313$.

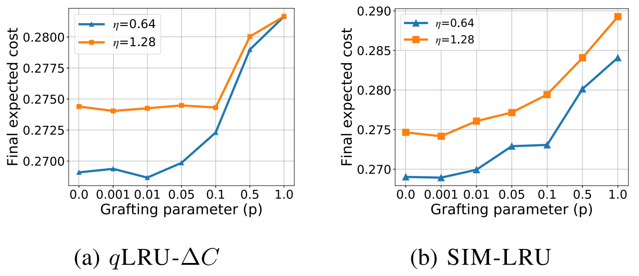


Fig. 5. Effect of the grafting parameter p on the final expected cost $\mathcal{C}(\cdot)$. GRADES/qLRU-ΔC ($q = 10^{-2}$) and GRADES/SIM-LRU for different learning rates under the trace *Synthetic*. Cache size $k = 313$.

grafting GRADES with SIM-LRU we can make the initial transient faster. Figure 3 also shows the final cache configuration reached by GRADES: as expected, the density of the objects in the cache is higher where the request density is higher.

The effect of the grafting parameter p is shown in Fig. 5 and depends on the specific grafted policy. We see that GRADES/qLRU-ΔC[3] is relatively insensitive to the grafting up to $p = 0.05$, but for larger values of p the cost increases, and approaches the cost of qLRU-ΔC alone (about 0.295 as it can bee seen in Fig. 4). This happens because, for large p, more and more cache updates are due to qLRU-ΔC, which, even upon an approximate hit, may introduce the requested object with probability proportional to q. For GRADES/SIM-LRU, the cost again increases as p increases, but it is always much smaller than the cost of SIM-LRU alone (about 0.32). The explanation is that SIM-LRU never introduces new objects on approximate hits. Hence, as far as the current cache allocation is providing approximate answers, the function GRAFT_MISS_UPDATE in Algorithm 1 does not modify the current cache allocation.

In Fig. 6, we study the effect of the initialization on the the performance of GRADES. We test three different initialization schemes: uniform box, uniform grid, and request-based depicted in Fig. 6 (b), (c), (d), respectively. We sample the initial set of objects uniformly at random without replacement among those at the boundary ($x \in \{0, 312\}$ or $y \in \{0, 312\}$) and from the whole catalog (313×313 grid), respectively

[2]As noted in [24], when object requests fall uniformly over the points of a $L \times L$ grid (with wrap-around conditions), with $k = L = 1 + 2l(l + 1)$, for some positive integer l, an optimal cache configuration can be computed. The value $k = L = 313$ results from the particular choice of $l = 12$. For a non-homogeneous request process an approximate optimal cost can be computed as well (see Appendix F in [24]).

[3]Note that GRADES grafted with $p = p'$ to a qLRU-ΔC with $q = q'$ is equivalent to GRADES grafted with $p = 1$ to a qLRU-ΔC with parameter $q = p' \times q'$.

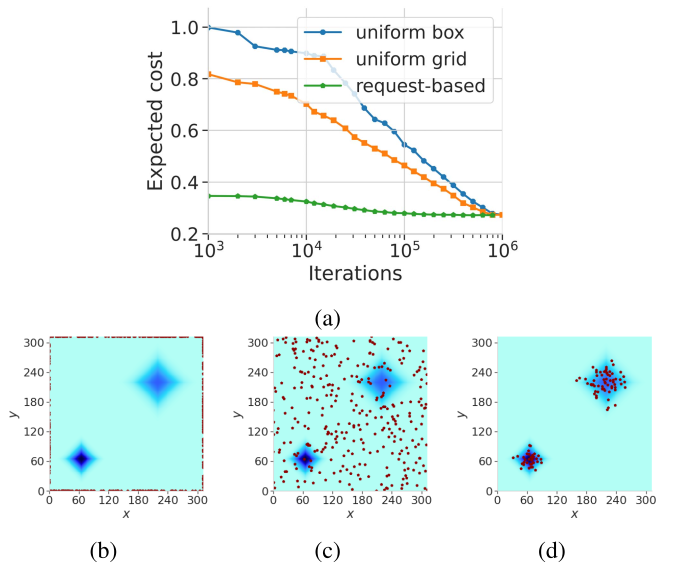


Fig. 6. Subfigure (a) shows the expected cost $\mathcal{C}(\cdot)$ incurred by GRADES/qLRU-ΔC ($\eta = 0.64$ grafted with $p = 1$ and $q = 10^{-3}$) for different initialization (uniform box, uniform grid, and request-based depicted in (b), (c), and (d), respectively) under the trace *Synthetic*. Cache size is $k = 313$. The heatmap depicts the popularity distribution of objects in the grid.

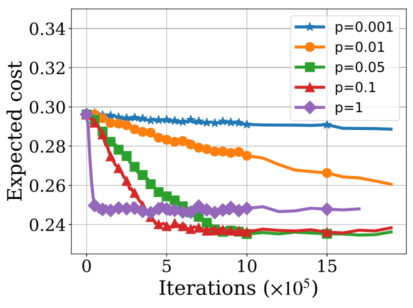


Fig. 7. Effect of grafting parameter p on the expected cost $\mathcal{C}(\cdot)$ in a dynamic setting for GRADES/qLRU-ΔC ($\eta = 0.64$). The cache is initialized to a stationary configuration as in Fig. 3. Now, only requests corresponding to region B from the trace *Synthetic* are made. Cache size $k = 313$.

under the *uniform box* initialization and the *uniform grid* one. Instead, we take the first k distinct requested objects to obtain the *request-based* initialization. Note how these three schemes lead to store initially in the cache progressively more popular objects. Figure 6 (a) shows the three initializations provide different initial costs, with the configuration with the least (resp. most) popular objects leading to the highest (resp. lowest) initial cost. The figure shows also that, independently of the initial cache configuration, GRADES is able to move to configurations with smaller cost (similar to the one in Fig. 3).

Until now, we have considered a stationary request scenario, where there is no evident advantage from grafting a more reactive policy to GRADES. In Fig. 7 we consider a highly non-stationary setting. At time 0, the cache is initialized as in Fig. 3, i.e., the cache configuration reached by GRADES after a large number of requests (a million) made from a mix of the two gaussian distributions. Then, the request process changes abruptly and no more requests for objects in region A are generated. The cache should reach a new configuration

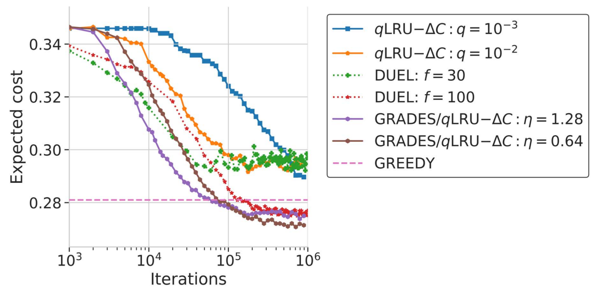


Fig. 8. Expected cost $\mathcal{C}(\cdot)$ incurred by the different policies under the trace *Synthetic*: qLRU-ΔC ($q = 10^{-3}, 10^{-2}$), DUEL , GREEDY, and GRADES ($\eta = 1.28, 0.64$ grafted with $p = 1$ and $q = 10^{-3}$). Cache size $k = 313$.

where all cached objects are located in region B, achieving a lower cost, as now the same number of objects should cover a smaller area. Fig. 7 shows that a higher value of p enables faster migration of objects from region A to region B in the cache.

Figure 8 shows the synergy between GRADES and the grafted policy. qLRU-ΔC, DUEL, and GRADES, all have parameters (q, η, and f) that can be tuned to find an optimal trade off between convergence speed and final cost. They can converge fast to configurations within a large neighborhood of a critical point (for high q, high η, and low f, respectively), or slowly to configurations within a smaller neighborhood.

Experiments in [24], in a setting similar to ours, show that qLRU-ΔC achieves a worse cost-vs-speed tradeoff than DUEL. Figure 8 confirms that this is the case, but when qLRU-ΔC is grafted on GRADES, the resulting policy improves on top of DUEL. In fact GRADES/qLRU-ΔC achieves a better trade-off as it is able to converge to a cost comparable to DUEL in a shorter time, or equivalently to a smaller cost in roughly the same time. Note also that DUEL's expected cost at steady state is noisier than GRADES/qLRU-ΔC's cost, showing the advantage of smoothly updating the state using gradients.

We now study the impact of the approximability threshold (C_θ) on the expected cost using the *Synthetic* trace. We vary the value of C_θ from 0.5 to 1.4 in increments of 0.1. Figure 9 shows that, as C_θ increases, the expected cost first decreases and then increases. In fact, for $C_\theta \ll C_r = 1$, a larger C_θ increases the number of approximate hits and then avoids the need to pay the cost C_r to retrieve the objects from the server. On the contrary, for $C_\theta \gg C_r = 1$, a larger C_θ is not beneficial, because misses are finally less costly than approximate hits. Figure 9 suggests that the optimal configuration is $C_\theta \approx C_r = 1$ (in our experiments the server can always provide an exact hit). We observe that, while this choice minimizes the cost to serve a request *given the current cache configuration*, the choice of C_θ also influences how the cache state evolves. This is because newer objects are only introduced into the cache on misses. Therefore, the optimal value for C_θ could be, in principle, different.

B. 360° Videos

We test our algorithms on 360° video traces. A 360° video is an immersive, spherical video [58], [59]. The video is first

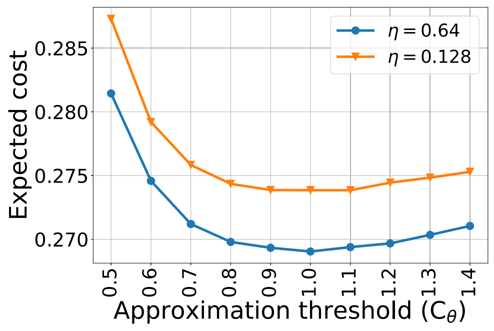


Fig. 9. Expected cost $\mathcal{C}(\cdot)$ incurred by the GRADES/SIM-LRU under the trace *Synthetic* for different values of the approximation threshold C_θ. Cache size $k = 313$, $C_r = 1$, $\eta = 0.64$, 1.28 and $p = 0.001$.

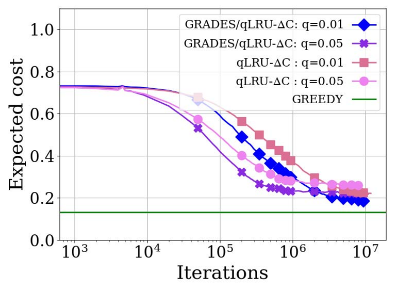


Fig. 10. Expected cost $\mathcal{C}(\cdot)$ incurred by the different caching policies under the trace 360° *videos*. GRADES/qLRU-ΔC ($q = 0.01, 0.05$) with $\eta = 1.0$ and qLRU-ΔC ($q = 0.01, 0.05$). Cache size $k = 4000$.

projected on to a 2D plane to be encoded by classic 2D video encoders. The video is divided into time segments, and each segment is further spatially divided into tiles. The VR headset is optimized to fetch the required tiles based on the head position of the user. A system responsible for the delivery of 360° videos can store the popular tiles in nearby caches [60]. Moreover, tiles at the periphery of the user's field of view could be approximated by neighbouring tiles that are stored at the cache and can then be served with low latency. Similarity caching may then be useful in this context, specially in the future when the number of tiles will increase and close tiles will become more similar.

We generated a sequence of tiles' requests for 360° videos using the approach proposed in [61]. We took real traces from 8 videos watched by 48 users each, and then built a *navigation graph* for each video, i.e., a Markov Chain that represents the spatial and temporal viewing correlations for the video. The videos we considered have on average 207 segments, each with 25 tiles. From each navigation graph we can generate an arbitrary number of possible views of the video. We generated then a trace with 10,000 users as follows. At time $t = 0$, each user selects one of the videos at random and starts watching the video from a random segment of the video. The user then walks through the navigation graph to view the complete video. Once the user reaches the last segment in the video, it selects a new video uniformly at random (with replacement) and starts watching the selected video from the first segment. The process is repeated till 10 million requests are generated. We assume each tile can approximate at most 4 tiles (the adjacent ones), with a fixed approximation cost $C_a = 0.1$.

Figure 10 compares the performance of GRADES/qLRU-ΔC and qLRU-ΔC. Note that in this setting, the representation space exhibits a very rough granularity, as the tiles of a segment cannot be used to approximate those of another segment and each segment is decomposed in a 5×5 grid of tiles. Nevertheless, GRADES/qLRU-ΔC shows significant improvement with respect to existing similarity caching policies and approaches the cost of GREEDY.

C. Machine Learning Traces

We study the performance of similarity caching under the following traces in high-dimensional spaces.

1) Amazon Trace: The paper [62] proposes a technique to embed the images of Amazon products in a 100-dimensional space, where the Euclidean distance between two items captures the similarity of the sets of users who purchased or viewed both items. We have restricted ourselves to the products in the category "Baby" and we have assumed that a request for a given item was issued at time t, if a user left a review for the considered item at the same time.

2) CiteULike Trace: The CiteULike dataset [63] contains a bipartite network of 22,715 users and 153,277 tags, where each edge represents a timestamped tag creation. The embeddings in a 100-dimensional space are obtained using the collaborative metric learning model proposed in [64]. As for the Amazon trace, the Euclidean distance within this space encodes the similarity between users and items, where the items here are the tags. We generated the trace considering that an object (tag) is requested when one user adds the corresponding tag.

3) Movielens Trace: We have trained the RecVAE collaborative filtering model from [65] on the Movielens dataset [66] to embed users' rating histories in a $d = 200$ dimensional space. Users with similar rating histories are mapped to vectors close according to the Euclidean distance. We have generated the trace by embedding every batch of 38 ratings from the same user (38 is the median number of ratings across all users) and assigning it the timestamp of the latest rating in the batch.

In all previous traces similarity is captured by the Euclidean distance. We then assume the dissimilarity cost to be proportional to the squared Euclidean distance, i.e., $C_d(x, y) = b\|x - y\|_2^2$. As the absolute value of such distance has not a clear meaning, we select the constant b so that on average an object can approximate a given fraction α of the catalogue. We say that x can approximate y if $C_d(x, y) \leq C_r = 1$, and we call α the *approximability* value. We set the cache size to $k = 100$.

The time-average cost of different caching policies is shown in Fig. 12 for the Amazon trace and 10% approximability. Although an object is able to approximate only 10% of the catalog on average, similarity caching policies significantly reduce the cost in comparison to an exact caching policy like LRU, with qLRU-ΔC and GRADES/qLRU-ΔC achieving the lowest service cost.

The empirical distribution of pairwise distances in Amazon trace is shown in Fig. 11. The figure shows that objects

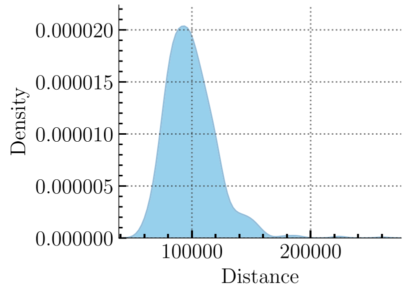


Fig. 11. The empirical distribution of pairwise distances under the trace *Amazon*. The distribution is computed on a random subset of the catalog containing 1000 products.

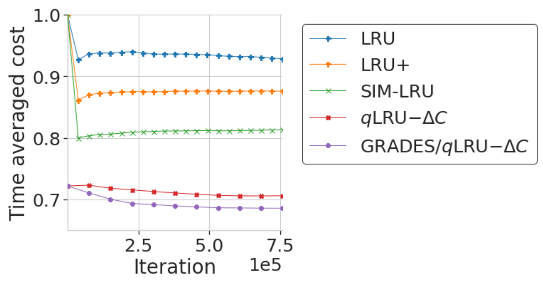


Fig. 12. The time averaged cost incurred by different policies under the trace *Amazon*: LRU, LRU+, SIM-LRU, qLRU-ΔC ($q = 10^{-3}$) and GRADES/qLRU-ΔC ($\eta = 6.9 \times 10^2$, grafted with p $=$ 1). The level of approximability is 10%. Cache size $k = 100$.

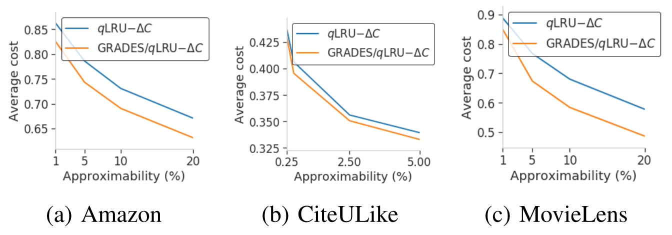


Fig. 13. The average cost incurred by qLRU-ΔC ($q = 10^{-3}$) and GRADES/qLRU-ΔC (grafted with $p = 1$) under the machine learning traces. The learning rates picked for different approximability levels are: (a) ($\eta = 5.4 \times 10^2, 6.3 \times 10^2, 6.9 \times 10^2, 7.7 \times 10^2$), (b) ($\eta = 2.4 \times 10^{-2}, 2.6 \times 10^{-2}, 3.0 \times 10^{-2}, 3.2 \times 10^{-2}$) and (c) ($\eta = 1.6 \times 10^{-1}, 2.7 \times 10^{-1}, 3.2 \times 10^{-1}, 3.8 \times 10^{-1}$). Cache size $k = 100$.

are quite scattered in this high-dimensional space with mean distance of around 85,000 between any two objects. The objects in the virtual cache are then in general far from any object in the catalog and we could expect gradient methods to perform poorly. Nevertheless, Fig. 12 shows that GRADES/qLRU-ΔC outperforms existing similarity caching policies. Similar results hold for the other two traces.

Finally, Fig. 13 reports the costs obtained for the three traces under different values of approximability. As expected, the service cost reduces as the approximability becomes larger. In the CiteULike trace, the service cost flattens rapidly (it is almost constant after 10% approximability): a close look at the dataset shows that popular objects are clustered in a small region of space. Once the approximability value guarantees that these objects can approximate each other, the marginal improvement from further increasing approximability becomes negligible. The other two traces show instead a similar behaviour, with the service cost that is still decreasing after 20% approximability. We also observe that the relative improvement of GRADES/qLRU-ΔC in comparison to qLRU-ΔC becomes larger as the approximability increases.

VI. Conclusion

In this paper we have proposed GRADES, a new caching policy for similarity caching systems that takes advantage from the fact that objects and requests can many times be embedded in a continuous metric space. GRADES outperforms traditional caching policies in stationary scenarios, converging to provably optimal configurations under mild assumptions. Moreover we have shown that GRADES can be grafted to any traditional caching policy, obtaining flexible schemes that achieve arbitrary trade-offs between convergence speed and average costs at steady state. The performance of GRADES and its extensions has been evaluated in several synthetic and realistic scenarios.

References

[1] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquin, “Searching in metric spaces,” *ACM Comput. Surv.*, vol. 33, no. 3, pp. 273–321, Sep. 2001.

[2] S. Berchtold, C. Böhm, B. Braunmüller, D. A. Keim, and H.-P. Kriegel, “Fast parallel similarity search in multimedia databases,” *ACM SIGMOD Rec.*, vol. 26, no. 2, pp. 1–12, Jun. 1997.

[3] F. Falchi, C. Lucchese, S. Orlando, R. Perego, and F. Rabitti, “A metric cache for similarity search,” in *Proc. ACM Workshop Large-Scale Distrib. Syst. Inf. Retr. (LSDS-IR)*, New York, NY, USA, 2008, pp. 43–50.

[4] F. Falchi, C. Lucchese, S. Orlando, R. Perego, and F. Rabitti, “Similarity caching in large-scale image retrieval,” *Inf. Process. Manage.*, vol. 48, no. 5, pp. 803–818, Sep. 2012.

[5] S. Pandey, A. Broder, F. Chierichetti, V. Josifovski, R. Kumar, and S. Vassilvitskii, “Nearest-neighbor caching for content-match applications,” in *Proc. 18th Int. Conf. World Wide Web (WWW)*, New York, NY, USA, 2009, pp. 441–450.

[6] D. Asanov, “Algorithms and methods in recommender systems,” Berlin Inst. Technol., Berlin, Germany, 2011.

[7] P. Sermpezis, T. Giannakas, T. Spyropoulos, and L. Vigneri, “Soft cache hits: Improving performance through recommendation and delivery of related content,” *IEEE J. Sel. Areas Commun.*, vol. 36, no. 6, pp. 1300–1313, Jun. 2018.

[8] B. Yan, D. R. Lovley, and J. Krushkal, “Genome-wide similarity search for transcription factors and their binding sites in a metal-reducing prokaryote *Geobacter sulfurreducens*,” *Biosystems*, vol. 90, no. 2, pp. 421–441, Sep. 2007.

[9] A. F. Auch, H. Klenk, and M. Göker, “Standard operating procedure for calculating genome-to-genome distances based on high-scoring segment pairs,” *Standards Genomic Sci.*, vol. 2, no. 1, p. 142, 2010.

[10] J. Weston, S. Chopra, and A. Bordes, “Memory networks,” in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–15.

[11] A. Graves, G. Wayne, and I. Danihelka, “Neural Turing machines,” 2014, *arXiv:1410.5401*.

[12] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, “Meta-learning with memory-augmented neural networks,” in *Proc. Int. Conf. Mach. Learn.*, New York, NY, USA, Jun. 2016, pp. 1842–1850.

[13] D. Crankshaw, X. Wang, J. E. Gonzalez, and M. J. Franklin, “Scalable training and serving of personalized models,” in *Proc. NIPS Workshop Mach. Learn. Syst. (LearningSys)*, 2015, pp. 1–6.

[14] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, “Clipper: A low-latency online prediction serving system,” in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Boston, MA, USA, 2017, pp. 613–627.

[15] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, “Cachier: Edge-caching for recognition applications,” in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 276–286.

[16] U. Drolia, K. Guo, and P. Narasimhan, "Precog: Prefetching for image recognition applications at the edge," in *Proc. 2nd ACM/IEEE Symp. Edge Comput.*, Oct. 2017, pp. 1–13.

[17] P. Guo, B. Hu, R. Li, and W. Hu, "FoggyCache: Cross-device approximate computation reuse," in *Proc. MobiCom*, 2018, pp. 19–34.

[18] P. Guo and W. Hu, "Potluck: Cross-application approximate deduplication for computation-intensive mobile applications," in *Proc. 23rd Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2018, pp. 271–284.

[19] S. Venugopal, M. Gazzetti, Y. Gkoufas, and K. Katrinis, "Shadow puppets: Cloud-level accurate AI inference at the speed and economy of edge," in *Proc. USENIX (HotEdge)*, 2018, pp. 1–6.

[20] A. Kumar, A. Balasubramanian, S. Venkataraman, and A. Akella, "Accelerating deep learning inference via freezing," in *Proc. 11th USENIX Workshop Hot Topics Cloud Comput. (HotCloud)*, 2019, pp. 1–9.

[21] A. Balasubramanian, A. Kumar, Y. Liu, H. Cao, S. Venkataraman, and A. Akella, "Accelerating deep learning inference via learned caches," 2021, *arXiv:2101.07344*.

[22] T. Si Salem, G. Castellano, G. Neglia, F. Pianese, and A. Araldo, "Towards inference delivery networks: Distributing machine learning with optimality guarantees," in *Proc. 19th Medit. Commun. Comput. Netw. Conf. (MedComNet)*, Jun. 2021.

[23] R. Weber, H. J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *Proc. VLDB*, vol. 98, 1998, pp. 194–205.

[24] G. Neglia, M. Garetto, and E. Leonardi, "Similarity caching: Theory and algorithms," *IEEE/ACM Trans. Netw.*, vol. 30, no. 2, pp. 475–486, Apr. 2021.

[25] T. Spyropoulos and P. Sermpezis, "Soft cache hits and the impact of alternative content recommendations on mobile edge caching," in *Proc. 11th ACM Workshop Challenged Netw.*, New York, NY, USA, Oct. 2016, pp. 51–56.

[26] A. Bellet, A. Habrard, and M. Sebban, *Metric Learning* (Synthesis Lectures on Artificial Intelligence and Machine Learning), vol. 9. San Rafael, CA, USA: Morgan & Claypool, 2015.

[27] *Similarity Caching Trace Repository*. Accessed: Aug. 2021. [Online]. Available: https://sim-cache.gitlabpages.inria.fr/similarity-caching-traces/

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

[29] M. Garetto, E. Leonardi, and G. Neglia, "Content placement in networks of similarity caches," *Comput. Netw.*, vol. 201, Dec. 2021, Art. no. 108570.

[30] J. Zhou, O. Simeone, X. Zhang, and W. Wang, "Adaptive offline and online similarity-based caching," *IEEE Netw. Lett.*, vol. 2, no. 4, pp. 175–179, Dec. 2020.

[31] J. Cortes, S. Martinez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *IEEE Trans. Robot. Autom.*, vol. 20, no. 2, pp. 243–255, Apr. 2004.

[32] J. Cortés and F. Bullo, "Coordination and geometric optimization via distributed dynamical systems," *SIAM J. Control Optim.*, vol. 44, no. 5, pp. 1543–1574, 2005.

[33] J. Cortés, S. Martínez, and F. Bullo, "Spatially-distributed coverage optimization and control with limited-range interactions," *ESAIM, Control, Optim. Calculus Variat.*, vol. 11, no. 4, pp. 691–719, Oct. 2005.

[34] L. Bottou, "On-line learning and stochastic approximations," in *On-Line Learning in Neural Networks*. New York, NY, USA: Cambridge Univ. Press, 1998.

[35] S. Ioannidis, L. Massoulie, and A. Chaintreau, "Distributed caching over heterogeneous mobile networks," in *Proc. ACM Int. Conf. Meas. Modeling Comput. Syst. (SIGMETRICS)*, 2010, pp. 311–322.

[36] S. Ioannidis and E. Yeh, "Adaptive caching networks with optimality guarantees," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 44, pp. 113–124, Jun. 2016.

[37] G. S. Paschos, A. Destounis, L. Vigneri, and G. Iosifidis, "Learning to cache with no regrets," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2019, pp. 235–243.

[38] T. Si Salem, G. Neglia, and S. Ioannidis, "No-regret caching via online mirror descent," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2021, pp. 1–6.

[39] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," in *Proc. 47th Annu. IEEE Symp. Found. Comput. Sci. (FOCS)*, 2006, pp. 459–468.

[40] J. Johnson, M. Douze, and H. Jegou, "Billion-scale similarity search with GPUs," *IEEE Trans. Big Data*, vol. 7, no. 3, pp. 535–547, Jul. 2021.

[41] A. Babenko and V. Lempitsky, "The inverted multi-index," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 6, pp. 1247–1260, Jun. 2014.

[42] B. Naidan, L. Boytsov, and E. Nyberg, "Permutation search methods are efficient, yet faster search is possible," 2015, *arXiv:1506.03163*.

[43] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 4, pp. 824–836, Apr. 2020.

[44] J. Li, S. Shakkottai, J. C. S. Lui, and V. Subramanian, "Accurate learning or fast mixing? Dynamic adaptability of caching algorithms," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 6, pp. 1314–1330, Jun. 2018.

[45] A. Finamore, J. Roberts, M. Gallo, and D. Rossi, "Accelerating deep learning classification with error-controlled approximate-key caching," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, May 2022, pp. 2118–2127.

[46] A. Sabnis, T. S. Salem, G. Neglia, M. Garetto, E. Leonardi, and R. K. Sitaraman, "GRADES: Gradient descent for similarity caching," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, May 2021, pp. 1–10.

[47] T. Si Salem, G. Neglia, and D. Carra, "AÇAI: Ascent similarity caching with approximate indexes," in *Proc. 33rd Int. Teletraffic Congr. (ITC)*, Avignon, France, Aug. 2021, pp. 1–9.

[48] *Faiss: A Library for Efficient Similarity Search*. Accessed: Mar. 2017. [Online]. Available: https://engineering.fb.com/data-infrastructure/faiss-a-library-for-efficient-similarity-search/

[49] J. A. Bergstra and C. A. Middelburg, "Itu-t recommendation g.107 : The e-model, a computational model for use in transmission planning," Citeseer, Tech. Rep., 2003.

[50] H. Liu, Y. Li, Z. Duan, and C. Chen, "A review on multi-objective optimization framework in wind energy forecasting techniques and applications," *Energy Convers. Manage.*, vol. 224, Nov. 2020, Art. no. 113324.

[51] K. Poularakis, G. Iosifidis, A. Argyriou, and L. Tassiulas, "Video delivery over heterogeneous cellular networks: Optimizing cost and performance," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2014, pp. 1078–1086.

[52] D. Carra, G. Neglia, and P. Michiardi, "Elastic provisioning of cloud caches: A cost-aware TTL approach," *IEEE/ACM Trans. Netw.*, vol. 28, no. 3, pp. 1283–1296, Jun. 2020.

[53] E. G. Coffman and P. J. Denning, *Operating Systems Theory*, vol. 973. Englewood Cliffs, NJ, USA: Prentice-Hall, 1973.

[54] V. S. Borkar, *Stochastic Approximation: A Dynamical Systems Viewpoint*. Cambridge, U.K.: Cambridge Univ. Press, 2008.

[55] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter, "Adaptsize: Orchestrating the hot object memory cache in a content delivery network," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Boston, MA, USA, 2017, pp. 483–498.

[56] M. Aumüller, E. Bernhardsson, and A. Faithfull, "ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms," in *Similarity Search and Applications*, C. Beecks, F. Borutta, P. Kröger, and T. Seidl, Eds. Cham, Switzerland: Springer, 2017, pp. 34–49.

[57] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey, *An Analysis of Approximations for Maximizing Submodular Set Functions—II*. Berlin, Germany: Springer, 1978, pp. 73–87.

[58] X. Corbillon, G. Simon, A. Devlic, and J. Chakareski, "Viewport-adaptive navigable 360-degree video delivery," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2017, pp. 1–7.

[59] F. Qian, L. Ji, B. Han, and V. Gopalakrishnan, "Optimizing 360 video delivery over cellular networks," in *Proc. 5th Workshop Things Cellular: Oper., Appl. Challenges*, Oct. 2016, pp. 1–6.

[60] M. Zink, R. Sitaraman, and K. Nahrstedt, "Scalable 360 video stream delivery: Challenges, solutions, and opportunities," *Proc. IEEE*, vol. 107, no. 4, pp. 639–650, Apr. 2019.

[61] J. Park and K. Nahrstedt, "Navigation graph for tiled media streaming," in *Proc. 27th ACM Int. Conf. Multimedia*, Oct. 2019, pp. 447–455.

[62] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel, "Image-based recommendations on styles and substitutes," in *Proc. 38th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, Aug. 2015, pp. 43–52.

[63] H. Wang, N. Wang, and D.-Y. Yeung, "Collaborative deep learning for recommender systems," in *Proc. 21st ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, New York, NY, USA, Aug. 2015, pp. 1235–1244.

[64] C.-K. Hsieh, L. Yang, Y. Cui, T.-Y. Lin, S. Belongie, and D. Estrin, "Collaborative metric learning," in *Proc. 26th Int. Conf. World Wide Web*, Geneva, Switzerland, Apr. 2017, pp. 193–201.

[65] I. Shenbin, A. Alekseev, E. Tutubalina, V. Malykh, and S. I. Nikolenko, "RecVAE: A new variational autoencoder for top-N recommendations with implicit feedback," in *Proc. 13th Int. Conf. Web Search Data Mining*, New York, NY, USA, Jan. 2020, pp. 528–536.
[66] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, pp. 1–19, Dec. 2015.

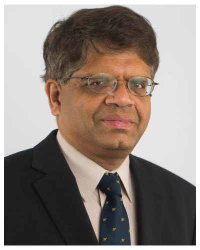
Anirudh Sabnis (Student Member, IEEE) received the B.Tech. degree in information technology from the National Institute of Technology Karnataka, Surathkal, and the master's degree in computer science from the University of Massachusetts Amherst, where he is currently pursuing the Ph.D. degree in computer science. His research interests include distributed systems, caching, and networks.

Tareq Si Salem (Student Member, IEEE) received the M.Sc. degree in computer science from Côte d'Azur University in 2019, where he is currently pursuing the Ph.D. degree in computer science. He is a Ph.D. Visitor at the Delft University of Technology. His research interests include networking, optimization, and machine learning.

Giovanni Neglia (Member, IEEE) received the Ph.D. degree and the degree in electronic engineering from the University of Palermo, Italy, in 2005 and 2001, respectively, and the Habilitation degree from Université Côte d'Azur, France, in 2017. Before joining Inria as a Permanent Researcher, he was a Research Scholar at University Massachusetts Amherst in 2005 and a Postdoctoral Researcher at Inria (2006–2007). He has been a Researcher at Inria, France, since 2008, and has been holding the Chair of Pervasive Sustainable Learning Systems at the 3IA Côte d'Azur (one of the French Interdisciplinary Institutes on Artificial Intelligence), since 2021. His research activity focuses on modeling and performance evaluation of networked systems and proposals of new mechanisms to improve their performance. Currently, his main interests are networks of caches and distributed optimization for machine learning.

Michele Garetto (Member, IEEE) received the Dr. (Ing.) degree in telecommunication engineering and the Ph.D. degree in electronic and telecommunication engineering from the Politecnico di Torino, Italy, in 2000 and 2004, respectively. In 2002, he was a Visiting Scholar with the Networks Group, University of Massachusetts Amherst, and in 2004 he held a postdoctoral position at the ECE Department, Rice University, Houston. He is currently an Associate Professor at the Computer Science Department, University of Torino, Italy.

Emilio Leonardi (Senior Member, IEEE) is currently a Professor with the Department of Electronics and Telecommunications, Politecnico di Torino. He visited: CS Department, UCLA, in 1995, Lucent Bell-Laboratories, Holmdel, in 1999, Stanford EE Department, in 2001, Sprint Laboratories, Burlingame, in 2003, NEC Laboratories, Heidelberg, in 2012, and INRIA, Sophia Antipolis, in 2016. His research interests include performance evaluation of computer networks and distributed systems, dynamics over networks, and human centric computation.



Ramesh K. Sitaraman (Fellow, IEEE) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Madras, and the Ph.D. degree in computer science from Princeton University. He is currently a Distinguished Professor in computer science at the University of Massachusetts Amherst. His research focuses on all aspects of internet-scale distributed systems, including algorithms, architectures, performance, energy efficiency, security, and economics. He is best known for pioneering content delivery networks (CDNs) that currently deliver much of the world's web content, streaming videos, and online applications. He helped create the world's first major CDNs and edge computing services while at Akamai. He retains a part-time role as an Akamai's Chief Consulting Scientist. He is a Fellow of the ACM.