No-Regret Caching via Online Mirror Descent

Tareq Si Salem
Université Côte d'Azur, Inria
tareq.si-salem@inria.fr

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

Stratis Ioannidis
Northeastern University
ioannidis@ece.neu.edu

***Abstract*—We study an online caching problem in which requests can be served by a local cache to avoid retrieval costs from a remote server. The cache can update its state after a batch of requests and store an arbitrarily small fraction of each content. We study no-regret algorithms based on Online Mirror Descent (OMD) strategies. We show that the choice of OMD strategy depends on the request diversity present in a batch and that OMD caching policies may outperform traditional eviction-based policies.**

I. Introduction

Caches are deployed at many different levels in computer systems: from CPU hardware caches to operating system memory caches, from application caches at clients to CDN caches deployed as physical servers in the network or as cloud services like Amazon's ElastiCache [1]. They aim to provide a faster service to the user and/or to reduce the computation/communication load on other system elements, like hard disks, content servers, etc.

Most prior work has assumed that caches serve requests generated according to a stochastic process, ranging from the simple, memory-less independent reference model [2] to more complex models trying to capture temporal locality effects and time-varying popularities (e.g., the shot-noise model [3]). An alternative modeling approach is to consider the sequence of requests is generated by an adversary, and compare online caching policies to the optimal offline policy that views the sequence of requests in advance [4].

Recently, Paschos et al. [5] proposed studying caching as an online convex optimization (OCO) problem [6]. OCO generalizes previous online problems like the experts problem [7], and has become widely influential in the learning community [6], [8]. This framework considers an adversarial setting, where the metric of interest is the *regret*, i.e., the difference between the costs incurred over a time horizon T by the algorithm and by the optimal offline *static* solution. Online algorithms whose regret grows sublinearly with T are called *no-regret* algorithms, as their time-average regret becomes negligible for large T. Paschos et al. proposed a no-regret caching policy based on the classic online gradient descent method (OGD), under the assumption that (a) the cache can store arbitrarily small fractions of each content (the so-called fractional setting), and (b) the cache state is updated after each request. Bhattacharjee et al. [9] extended this work proving tighter lower bounds for the regret and proposing new caching policies for the networked setting that do not require content coding.

In this paper, we extend and generalize the analysis of Paschos et al. in two different directions:

1) We assume the cache can update its state after processing a batch of $R \geq 1$ requests. This is of interest both in high-demand settings, as well as in cases when updates may only occur infrequently, because they are costly w.r.t. either computation or communication.
2) We consider a broad family of caching policies based on online mirror descent (OMD); OGD can be seen as a special instance of this family.

Our contributions are summarized as follows. First, we show that caching policies based on OMD enjoy $\mathcal{O}\left(\sqrt{T}\right)$ regret. Most importantly, we show that bounds for the regret crucially depend on the diversity of the request process. In particular, the regret depends on the *diversity ratio* R/h, where R is the size of the request batch, and h is the maximum multiplicity of a request in a given batch. We observe that OMD with neg-entropy mirror map ($\mathrm{OMD_{NE}}$) surmounts OGD in the *high diversity* regime, and it is overturned in the *low diversity* regime.

Second, all OMD algorithms include a gradient update followed by a projection to guarantee that the new solution is in the feasible set (e.g., it does not violate the cache capacity constraints). The projection is often the most computationally expensive step of the algorithm. We show that efficient polynomial algorithms exist both for OGD (slightly improving the algorithm in [5]) and for $\mathrm{OMD_{NE}}$.

The remainder of this paper is organized as follows. We introduce our model assumptions in Sect. II. We present OMD algorithms and quantify their regret and their computational complexity in Sect. III. Finally, numerical results are presented in Sect. IV.

The complete proofs are available in [10].

II. System description

Remote Service and Local Cache. We consider a system in which requests for files are served either remotely or by an intermediate cache of finite capacity. A cache miss incurs a file-dependent remote retrieval cost. This cost could be, e.g., an actual monetary cost for using the network infrastructure, or a quality of service cost incurred due to fetching latency. Costs may vary, as each file may be stored at a different remote location.

Our goal is to study online caching algorithms that attain sublinear regret. Formally, we consider a stream of requests for files of equal size from a catalog $\mathcal{N} = \{1, 2, \ldots, N\}$. These requests can be served by a remote server at cost $w_i \in \mathbb{R}_+$ per request for file $i \in \mathcal{N}$. We denote by $\mathbf{w} = [w_i]_{i\in\mathcal{N}} \in \mathbb{R}_+^N$ the vector of costs and assume that $\mathbf{w}$ is known.

A local cache of finite capacity is placed in between the source of requests and the remote server(s). The local cache's role is to reduce the costs incurred by satisfying requests locally. We denote by $k \in \{1, \ldots, N\}$ the capacity of the cache. The cache is allowed to store arbitrary fractions of files, a common assumption in the literature [5], [11], [12] and a good approximation when the cache can store chunks of a file and chunk sizes are much smaller than file size. We assume that time is slotted, and denote by $x_{t,i} \in [0, 1]$ the fraction of file $i \in \mathcal{N}$ stored in the cache at time slot $t \in \{1, 2, \ldots, T\}$. The cache state is then given by vector $\mathbf{x}_t = [x_{t,i}]_{i\in\mathcal{N}} \in \mathcal{X}$, where $\mathcal{X}$ is the capped simplex determined by the capacity constraint, i.e.:

$$\mathcal{X} = \left\{ \mathbf{x} \in [0, 1]^N : \textstyle\sum_{i=1}^N x_i = k \right\}. \tag{1}$$

Requests. We assume that a batch of multiple requests may arrive within a single time slot. Moreover, a file may be requested multiple times within a single time slot (e.g., by different users, whose aggregated requests form the stream reaching the cache). We denote by $r_{t,i} \in \mathbb{N}$ the number of requests—also called multiplicity—for file $i \in \mathcal{N}$ at time t, and by $\mathbf{r}_t = [r_{t,i}]_{i\in\mathcal{N}} \in \mathbb{N}^N$ the vector of such requests, representing the entire batch. We assume that the number of requests (i.e., the batch size) at each timeslot is given by $R \in \mathbb{N}$. We also assume that the maximum multiplicity of a file in a batch is bounded by $h \in \mathbb{N}$. As a result, $\mathbf{r}_t$ belongs to set:

$$\mathcal{R}_{R,h} = \left\{ \mathbf{r} \in \{0, \ldots, h\}^N : \textstyle\sum_{i=1}^N r_i = R \right\}. \tag{2}$$

Intuitively, the ratio $\frac{R}{h}$ defines the diversity of request batches in a timeslot. For example, when $\frac{R}{h} = 1$, requests are concentrated on a single content. When $\frac{R}{h} = N$, requests are spread evenly across the catalog $\mathcal{N}$. For that reason, we refer to $\frac{R}{h}$ as the *diversity ratio*. We note that our request model generalizes the setting by Paschos et al. [5] or Bhattacharjee et al. [9], which can be seen as the case $R = h = 1$ under our request model. We make no additional assumptions on the request arrival process; put differently, we operate in the adversarial online setting, where a potential adversary may select an arbitrary request sequence in $\mathcal{R}_{R,h}$ to increase system costs.

Service Cost Objective. When a request batch $\mathbf{r}_t$ arrives, the cache incurs the following cost:

$$f_{\mathbf{r}_t}(\mathbf{x}_t) = \textstyle\sum_{i=1}^N w_i r_{t,i}(1 - x_{t,i}). \tag{3}$$

In other words, for each file $i \in \mathcal{N}$, the system pays a cost proportional to the file fraction $(1 - x_{t,i})$ missing from the local cache, weighted by the file cost w_i and by the number of times $r_{t,i}$ file i is requested in the current batch $\mathbf{r}_t$.

The cost objective (3) captures several possible real-life settings. First, it can be interpreted as a QoS cost paid by each user for the additional delay to retrieve part of the file from the server. Second, assuming that the R requests arrive and are served individually (e.g., because they are spread-out within a timeslot), Eq. (3) can represent the load on the servers or on the network to provide the missing part of the requested objects.

Online Caching Algorithms and Regret. Cache contents are determined online: that is, the cache has selected a state $\mathbf{x}_t \in \mathcal{X}$ at the beginning of a time slot. The request batch $\mathbf{r}_t$ arrives, and the linear cost $f_{\mathbf{r}_t}(\mathbf{x}_t) : \mathcal{X} \to \mathbb{R}_+$ is incurred; the state is subsequently updated to $\mathbf{x}_{t+1}$. Formally, the cache state is determined by an online policy $\mathcal{A}$, i.e., a sequence of mappings $\{\mathcal{A}_t\}_{t=1}^{T-1}$, where for every $t \geq 1$, $\mathcal{A}_t : (\mathcal{R}_{R,h} \times \mathcal{X})^t \to \mathcal{X}$ maps the sequence of the request batches and previous decisions $\{(\mathbf{r}_s, \mathbf{x}_s)\}_{s=1}^t$ to the next state $\mathbf{x}_{t+1} \in \mathcal{X}$. We assume that the policy is initialized with a feasible state $\mathbf{x}_1 \in \mathcal{X}$.

We measure the performance of an online algorithm $\mathcal{A}$ in terms of regret, i.e., the difference between the total cost experienced by a policy $\mathcal{A}$ over a time horizon T and that of the best static state $\mathbf{x}_*$ in hindsight, i.e., $\mathbf{x}_* = \arg\min_{\mathbf{x}\in\mathcal{X}} \sum_{t=1}^T f_{\mathbf{r}_t}(\mathbf{x})$. Formally,

$$\text{Regret}_T(\mathcal{A}) = \sup_{\{\mathbf{r}_1,\ldots,\mathbf{r}_T\}\in\mathcal{R}_{R,h}^T} \left\{ \sum_{t=1}^T f_{\mathbf{r}_t}(\mathbf{x}_t) - \sum_{t=1}^T f_{\mathbf{r}_t}(\mathbf{x}_*) \right\}, \tag{4}$$

Note that, by taking the supremum in Eq. (4), we indeed measure regret in the adversarial setting, i.e., against an adversary that potentially picks requests in $\mathcal{R}_{R,h}$ trying to jeopardize cache performance. In Sect. IV we show our caching policies performs well also under stochastic request processes and real request traces.

III. Online Mirror Descent Algorithms

Upon seeing $\mathbf{r}_t$, the policy $\mathcal{A}$ could select as $\mathbf{x}_{t+1}$ the state that would have minimized (on hindsight) the aggregate cost up to time t (i.e., $\sum_{\tau=1}^t f_{\mathbf{r}_\tau}(\mathbf{x})$). This can fail catastrophically, because of the adversarial nature of the requests. A more conservative approach, that indeed leads to sublinear regret, is to take gradient steps, moving in the direction of a better decision according to the latest cost. OMD denotes a family of gradient techniques.

The main premise behind OMD [6, Sect. 5.3] is that variables and gradients live into two distinct spaces: the *primal space*, for variables, and the *dual space*, for gradients. The two are linked via a function known as a *mirror map*. Updates using the gradient occur on the dual space; the mirror map is used to invert this update to a change on the primal variables. The standard online gradient descent OGD is a particular OMD algorithm where the mirror map is the squared Euclidean distance. Other mirror maps may lead to better performance depending on the specific problem [13, Section 4.3].

OMD for Caching. Applied to our caching problem, both the primal and dual spaces are $\mathbb{R}^N$. To disambiguate between the two, we denote primal points by $\mathbf{x}, \mathbf{y} \in \mathbb{R}^N$ and dual points by $\hat{\mathbf{x}}, \hat{\mathbf{y}} \in \mathbb{R}^N$, respectively. Formally, OMD is parameterized by (a) a fixed learning rate $\eta \in \mathbb{R}_+$, and (b) a differentiable map $\Phi : \mathcal{D} \to \mathbb{R}$, strictly convex over $\mathcal{D}$ and ρ-strongly convex w.r.t. a norm $\|\cdot\|$ over $\mathcal{X} \cap \mathcal{D}$, where $\mathcal{X}$ is included in the closure of $\mathcal{D}$. The set $\mathcal{X}$ is the constraint set over which we optimize. The function Φ is called the *mirror map*, that links the primal to the dual space. Two additional technical assumptions on Φ and $\mathcal{D}$ must hold. First, the gradient of Φ must diverge at the boundary of $\mathcal{D}$. Second, the image of $\mathcal{D}$ under the gradient of Φ should take all possible values, that is $\nabla\Phi(\mathcal{D}) = \mathbb{R}^N$.

Given η and Φ, an OMD iteration proceeds as follows. After observing the request batch $\mathbf{r}_t$ and incurring the cost $f_{\mathbf{r}_t}(\mathbf{x}_t)$, the current state $\mathbf{x}_t$ is first mapped from the primal to the dual space via:

$$\hat{\mathbf{x}}_t = \nabla\Phi(\mathbf{x}_t). \tag{5}$$

Then, a regular gradient descent step is performed *in the dual space* to obtain an updated dual point:

$$\hat{\mathbf{y}}_{t+1} = \hat{\mathbf{x}}_t - \eta \nabla f_{\mathbf{r}_t}(\mathbf{x}_t). \tag{6}$$

This updated dual point is then mapped back to the primal space using the inverse of mapping $\nabla\Phi$, i.e.:

$$\mathbf{y}_{t+1} = (\nabla\Phi)^{-1}(\hat{\mathbf{y}}_{t+1}). \tag{7}$$

The resulting primal point $\mathbf{y}_{t+1}$ may lie outside the constraint set $\mathcal{X}$. To obtain the final feasible point $\mathbf{x}_{t+1} \in \mathcal{X}$, a projection is made using the Bregman divergence associated with the mirror map Φ; the final cache state becomes:

$$\mathbf{x}_{t+1} = \textstyle\prod_{\mathcal{X}\cap\mathcal{D}}^{\Phi}(\mathbf{y}_{t+1}), \tag{8}$$

where $\prod_{\mathcal{X}\cap\mathcal{D}}^{\Phi} : \mathbb{R}^N \to \mathcal{X}$ is the Bregman projection defined as $\Pi_{\mathcal{X}\cap\mathcal{D}}^{\Phi}(\mathbf{y}) = \underset{\mathbf{x}\in\mathcal{X}\cap\mathcal{D}}{\arg\min}\, D_\Phi(\mathbf{x}, \mathbf{y})$. $D_\Phi(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x}) - \Phi(\mathbf{y}) - \nabla\Phi(\mathbf{y})^T(\mathbf{x} - \mathbf{y})$ is the Bregman divergence associated with the mirror map Φ.

A. *Euclidean mirror map*

The selection of the mirror map $\Phi(\mathbf{x}) = \frac{1}{2}\|\mathbf{x}\|_2^2$ and $\mathcal{D} = \mathbb{R}^N$ yields the identity mapping $\nabla\Phi(\mathbf{x}) = \mathbf{x}$, for all $x \in \mathcal{D}$. Furthermore, the Bregman divergence associated with this map is just the Euclidean distance $D_\Phi(\mathbf{x}, \mathbf{y}) = \frac{1}{2}\|\mathbf{x} - \mathbf{y}\|_2^2$ [13, Section 4.3]. Upon receiving a request batch $\mathbf{r}_t$, the algorithm updates its current state according to steps (5)-(8), this gives the OGD update rule:

$$\mathbf{x}_{t+1} = \Pi_{\mathcal{X}}(\mathbf{x}_t - \eta\nabla f_{\mathbf{r}_t}(\mathbf{x}_t)), \forall t \in \{1, \ldots, T-1\}, \tag{9}$$

where $\Pi_{\mathcal{X}}(\cdot)$ is the Euclidean projection onto $\mathcal{X}$. This setup of OMD coincides with OGD, studied by Paschos et al. [5]. We present a new regret bound in Theorem III.1, that takes into account diversity, and improves over the bound in [5] (valid only for $R = h = 1$) by a factor at least $\sqrt{2}$.

Theorem III.1. *For* $\eta = \sqrt{\frac{k\left(1-\frac{k}{N}\right)}{\|\mathbf{w}\|_\infty^2 hRT}}$ *the regret of OGD, satisfies:*

$$\mathrm{Regret}_T(\mathrm{OGD}) \leq \|\mathbf{w}\|_\infty \sqrt{hRk\left(1 - \tfrac{k}{N}\right)T} \tag{10}$$

Proof. Bubeck [13, Theorem 4.2] provides a general bound on the regret of OMD, which in our caching setting becomes

$$\mathrm{Regret}_T(\mathrm{OMD}_\Phi) \leq \tfrac{D_\Phi(\mathbf{x}_*, \mathbf{x}_1)}{\eta} + \tfrac{\eta}{2\rho}\textstyle\sum_{t=1}^{T}\|\nabla f_{\mathbf{r}_t}(\mathbf{x}_t)\|_*^2 \tag{11}$$

where $\|.\|_*$ is the dual norm of $\|.\|$, and $\mathbf{x}_*$ is the best fixed decision in hindsight. The mirror map for OGD is $\Phi(\mathbf{x}) = \frac{1}{2}\|\mathbf{x}\|_2^2$ that is 1-strongly convex with regard to the Euclidean norm ($\rho = 1$) and the dual norm is also the Euclidean one.

Let $\mathbf{x}_1$ be the minimizer of $\Phi(\mathbf{x})$, then we have $\nabla\Phi^T(\mathbf{x}_1)(\mathbf{x} - \mathbf{x}_1) \geq 0, \forall \mathbf{x} \in \mathcal{X}$ [6, Theorem 2.2] and, from the definition of the Bregman divergence, $D_\Phi(\mathbf{x}_*, \mathbf{x}_1) \leq \Phi(\mathbf{x}_*) - \Phi(\mathbf{x}_1)$.

The minimum value of $\Phi(\mathbf{x})$ over $\mathcal{X}$ is achieved when $x_i = \frac{k}{N}, i \in \mathcal{N}$, and $\mathbf{x}_*$ is an integral solution (i.e., k components of $\mathbf{x}_*$ equal 1 and the others equal 0). Then $\Phi(\mathbf{x}_*) = \frac{1}{2}k$ and we get:

$$D_\Phi(\mathbf{x}_*, \mathbf{x}_1) \leq \frac{1}{2}k\left(1 - k/N\right) \tag{12}$$

We now bound the second term in (11). The maximum of $\|\mathbf{r}\|_2$ is achieved when $\frac{R}{h}$ components are set to h, then:

$$\max_{r\in\mathcal{R}} \|\nabla f_{\mathbf{r}}(\mathbf{x})\|_2 \leq \max_{\mathbf{r}\in\mathcal{R}} \|\mathbf{w}\|_\infty \|\mathbf{r}\|_2 = \|\mathbf{w}\|_\infty \sqrt{Rh}. \tag{13}$$

Plugging (13) and (12) in (11), and selecting the learning rate $\eta = \sqrt{k\left(1 - \frac{k}{N}\right)/(\|\mathbf{w}\|_\infty^2 hRT)}$, we obtain the upper bound in (10). □

The OGD Algorithm requires computing an Euclidean projection at each iteration, which, for general values of R and h, can be performed in $\mathcal{O}\left(N^2\right)$ steps using the projection algorithm by Wang and Lu [14]. The algorithm proposed by Paschos et al. [5], for the case $R = h = 1$, has $\mathcal{O}\left(N\log(N)\right)$ time-complexity, but we can achieve $\mathcal{O}\left(N\right)$ complexity by replacing the preliminary sorting operation with a $\mathcal{O}\left(\log(N)\right)$ binary search and insertion.

B. *Neg-Entropy Mirror Map*

We now consider a different mirror map, i.e., the neg-entropy mirror map, defined as follows:

$$\Phi(\mathbf{x}) = \sum_{i=1}^{N} x_i \log\left(x_i\right), \text{ and } \mathcal{D} = \mathbb{R}_{>0}^N. \tag{14}$$

It can be checked that this map satisfies all requirements in Sect. III., We refer to the OMD algorithm with the neg-entropy mirror map as $\mathrm{OMD}_{\mathrm{NE}}$.

We first characterize the regret of $\mathrm{OMD}_{\mathrm{NE}}$:

Theorem III.2. *For* $\eta = \sqrt{\frac{2\log(N/k)}{\|\mathbf{w}\|_\infty^2 h^2 T}}$*, the regret of* $\mathrm{OMD}_{\mathrm{NE}}$ *satisfies:*

$$\mathrm{Regret}_T(\mathrm{OMD}_{\mathrm{NE}}) \leq \|\mathbf{w}\|_\infty hk\sqrt{2\log(N/k)T}. \tag{15}$$

Proof. The proof is similar to the proof of Theorem III.1. The neg-entropy mirror map is $\rho = \frac{1}{k}$ strongly convex w.r.t the $\|.\|_1$ over $\mathcal{X}$ [8, Example 2.5] and the corresponding dual norm is $\|\cdot\|_\infty$. Moving from (11), we bound the diameter of $\mathcal{X}$ w.r.t. to the Bregman divergence as well as the $\|\cdot\|_\infty$ of

Algorithm 1 Neg-Entropy Bregman projection onto $\mathcal{X}$

```
Require: $N$; $k$; $\|\mathbf{y}\|_1$; $P$; Sorted $y_N \geq .. \geq y_{N-k+1} \geq y_i, \forall i \leq N-k$
    // $\mathbf{y}$ is the decision vector that lies outside of $\mathcal{X}$.
    // The scale $P$ is a global variable initialized to 1.
 1: $y_{N+1} \leftarrow +\infty$
 2: for $b \in \{N, \dots, N-k+1\}$ do
 3:     $m_b \leftarrow \frac{k+b-N}{\|\mathbf{y}\|_1 - \sum_{i=b+1}^{N} y_i P}$
 4:     if $y_b m_b P < 1 \leq y_{b+1} m_b P$ then
                                   // Appropriate $b$ is found.
 5:         for i $\geq$ b+1 do
 6:             $y_i \leftarrow \frac{1}{m_b P}$
 7:         end for
 8:         $P \leftarrow m_b P$
 9:         return $\mathbf{y}P$      // $\mathbf{y}P$ is the result of the projection.
10:     end if
11: end for
```

gradients $\nabla f_{\mathbf{r}_t}(\mathbf{x}_t)$. Finally, selecting the learning rate that gives the tightest upper bound $\eta = \sqrt{2\log(\frac{N}{k})/(\|\mathbf{w}\|_\infty^2 h^2 T)}$, we obtain the desired regret upper bound. □

The following theorem characterizes under which diversity regime OMD_{NE} outperforms OGD, and contrariwise the regime where OGD surmounts.

Theorem III.3. OMD_{NE} *has a tighter regret bound than* OGD *when* $\frac{R}{h} > 2\sqrt{Nk}$, *while* OGD *has a tighter regret bound than* OMD_{NE} *when* $\frac{R}{h} < 2k$.

The result is obtained by relaxing the regret bound in Eq. (15) using a sharp upper bound [15], which is subsequently compared to Eq. (10), and the second regime is shown by using the lower bound $\log(1+x) \geq \frac{x}{x+1}$ for $x > -1$.

Theorem III.3 immediately implies the advantage of OMD_{NE} over OGD in the high-diversity ratio regime, and it is overturned in the low-diversity regime.

Beyond its improved performance in terms of regret the neg-entropy mirror map comes with an additional computational advantage: its projection step admits a highly efficient implementation. As $\frac{\partial \Phi(\mathbf{x})}{\partial x_i} = 1 + \log x_i$, the inverse mapping is given by $\left((\nabla\Phi)^{-1}(\hat{\mathbf{y}}_{t+1})\right)_i = \exp(\hat{y}_{t,i} - 1)$. Hence, the map to the dual space and back in Eqs. (5)–(7) can be concisely written as:

$$y_{t+1,i} = x_{t,i}\, e^{-\eta \frac{\partial f_{\mathbf{r}_t}(\mathbf{x}_t)}{\partial x_i}}, \quad \text{for all } i \in \mathcal{N}. \tag{16}$$

In other words, OMD under the neg-entropy adapts the cache state via *a multiplicative rule*, as opposed to the additive rule of OGD.

Finally, the projection algorithm onto the capped simplex can be implemented in $\mathcal{O}(N + k\log(k))$ time for arbitrary R, h using the waterfilling-like Algorithm 1. The algorithm receives as input the largest k components of $\mathbf{y}$ sorted in a descending order. It then re-scales the b largest components and $N-b$ smallest ones by different constant factors; the value of b is determined via a linear search. The following theorem holds:

Theorem III.4. *Algorithm 1 returns the projection* $\Pi^{\Phi}_{\mathcal{X}\cap\mathcal{D}}(\mathbf{y})$ *under the neg-entropy* Φ, *onto the capped simplex* $\mathcal{X}$ *in* $\mathcal{O}(k)$ *steps per iteration. This results in an overall time complexity of* $\mathcal{O}(N + k\log(k))$ *per iteration of OMD, for general* R, h *values, and* $\mathcal{O}(k)$ *per iteration of OMD, when* $R = h = 1$.

Proof. We adapt the Euclidean projection algorithm in [14]. Finding the projection $\mathbf{x} = \Pi^{\Phi}_{\mathcal{X}\cap\mathcal{D}}(\mathbf{y})$ corresponds to solving a convex problem as $D_\Phi(\mathbf{x}, \mathbf{y})$ is convex in $\mathbf{x}$ and $\mathcal{X}\cap\mathcal{D}$ is a convex set. Without loss of generality, we assume the components of $\mathbf{x} = \Pi^{\Phi}_{\mathcal{X}\cap\mathcal{D}}(\mathbf{y})$ to be in non-decreasing order. Let $b \in \mathcal{N}$ be the index of the largest component of $\mathbf{x}$ smaller than 1. The KKT conditions lead to conclude that if the components of $\mathbf{y}$ are ordered in ascending order, so are the components of $\mathbf{x}$, and it holds $y_b e^\gamma < 1 \leq y_{b+1} e^\gamma$, where γ is the Lagrangian multiplier associated with the capacity constraint. The value of b can be found by checking this condition for the possible values of b. We observe that necessarily $b \in \{N-k+1, \dots, N\}$. In fact, we cannot have $b \leq N-k$. If $b \leq N-k$, we get $\sum_{i=N-k+1}^{N} x_i \geq k$ and the capacity constraint implies that $x_i = 0, \forall i \leq b$, but we must have $x_i > 0$ since $\mathbf{x} \in \mathcal{X}\cap\mathcal{D}$ and $\mathcal{D} = R^N_{>0}$. As $b > N-k$, then we only require the largest k components of $\mathbf{y}$ to perform the projection.

For a given b let:

$$m_b := e^\gamma = \frac{k+b-N}{\sum_{i=1}^{b} y_i} = \frac{k+b-N}{\|\mathbf{y}\|_1 - \sum_{i=b+1}^{N} y_i} \tag{17}$$

The projection corresponds to setting map the components $y_{b+1}, \dots, y_N$ to 1 and multiply the other $N-b$ components by m_b. In order to avoid updating all components at each step, we can simply set the components $x_i, i > b$ (those that should be set equal to 1) to $\frac{1}{m_b}$. Then, at any time t, we can recover the value of $x_{t,i}$ by $P = \prod_{i=1}^{t} m_{b,i}$, where $m_{b,t}$ is the returned m_b from the Bregman projection at time step t. Then at any time step, the actual value of the decision variable is $\mathbf{x}P$.

For general values of R and h, the projection step takes $\mathcal{O}(k)$ steps per iteration and a partial sort is required to maintain top-k components of $\mathbf{y}_t$ sorted; this can be done using partial sorting in $\mathcal{O}(N + k\log(k))$ [16].

When $R = h = 1$, Alg. 1 leads to only a single state coordinate update, and requires $\mathcal{O}(\log(k))$ steps to maintain top-k components of $\mathbf{x}_t$ sorted online.

□

Theorem III.4 implies that OMD_{NE} has a much smaller computational cost than OGD. In fact, for general value of R and h, OMD_{NE} requires $\mathcal{O}(N + k\log(k))$ operations per iteration, while OGD requires $\mathcal{O}(N^2)$ operations. When $R = h = 1$, the overall time complexity is $\mathcal{O}(k)$ per iteration for OMD_{NE} and $\mathcal{O}(N)$ for OGD.

IV. Numerical Experiments

A. Experimental setup

1) Datasets: Throughout all experiments, we assume equal costs per file, i.e., $w_i = w'_i = 1, \forall i \in \mathcal{N}$.

Synthetic traces We generate synthetic datasets, as summarized in Table I. Individual file requests are i.i.d. and sampled from a catalog of N files according to a *Zipf* distribution

TABLE I: Trace Summary

Trace	# Batches	T	N	R	h	α
Fixed Popularity	10^5	10^5	200	1	1	0.8
Batched Fixed Popularity (1)	10^4	10^4	200	5×10^3	40	0.1
Batched Fixed Popularity (2)	10^4	10^4	200	5×10^3	53	0.2
Batched Fixed Popularity (3)	10^4	10^4	200	5×10^3	346	0.7
Partial Popularity Change (1)	5×10^3	10^3	500	5×10^3	40	0.1
Partial Popularity Change (2)	5×10^3	10^3	500	5×10^3	169	0.4
Partial Popularity Change (3)	5×10^3	10^3	500	5×10^3	473	0.7
Akamai Trace	1.7×10^4	10^2	10^3	5×10^3	380	n/a

with exponent α. As we increase the exponent α, the requests become more concentrated. Table I shows the value of h observed in each trace. The requests are grouped into batches of size R, and η^* denotes the learning rate value specified in Theorem III.1 and in Theorem III.2 for OGD and $\mathrm{OMD}_{\mathrm{NE}}$, respectively.

We also generate non-stationary request traces (*Partial Popularity Change* traces), where the popularity of a subset of files is modified every $T = 10^3$ time slots. In particular the 5% most popular files become the 5% least popular ones and vice versa. We want to model a situation where the cache knows the timescale over which the request process changes and which files are affected (but not how their popularity changes). Correspondingly, the time horizon is also set to T and, at the end of each time horizon, the cache redistributes uniformly the cache space currently allocated by those files. The cache size is $k = 5$.

Akamai Trace We consider also a real file request from a the Akamai CDN provider [17]. The trace spans 1 week, and we extract from it about 8.5×10^7 requests for the $N = 10^3$ most popular files. We group requests in batches of size $R = 5 \times 10^3$, and we consider a time horizon $T = 100$ time slots corresponding roughly to 1 hour. The cache size is $k = 25$.

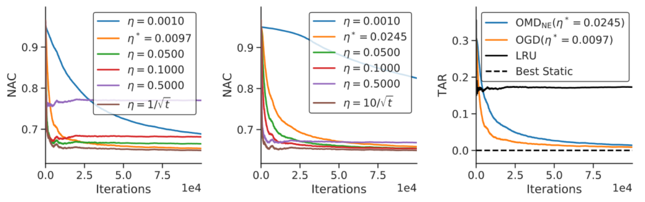
2) Online Algorithms: In addition to the gradient based algorithms, we implemented three caching eviction policies: LRU, LFU, and W-LFU. LRU and LFU evict the least recently used and least frequently used, respectively. W-LFU [18] is an LFU variant that only considers the most requested contents during a recent time window W, which we set equal to $T \times R$ in our experiments. The policies LRU, LFU, and W-LFU are allowed to update the cache state after each request. Finally, we define *Best Static* to be the optimal static allocation $\mathbf{x}^*$. We also define *Best Dynamic* to be the cache that stores the k most popular files at any time for the synthetic traces (for which the instantaneous popularity is well defined).

3) Performance Metrics: We measure performance w.r.t. three metrics, that we define here. The Normalized Average Cost $\mathrm{NAC}(\mathcal{A}) \in [0, 1]$ corresponds to the time-average cost over the first t time slots, normalized by the batch size R.

$$\mathrm{NAC}(\mathcal{A}) = \tfrac{1}{Rt} \textstyle\sum_{s=0}^{t} f_{\mathbf{r}_s}(\mathbf{x}_s) \tag{18}$$

The Normalized Moving Average Cost $\mathrm{NMAC}(\mathcal{A}) \in [0, 1]$ is computed similarly using a moving average instead over a time window $\tau > 0$ (we use $\tau = 500$ in our experiments):

$$\mathrm{NMAC}(\mathcal{A}) = \tfrac{1}{R \min(\tau, t)} \textstyle\sum_{s=t-\min(\tau,t)}^{t} f_{\mathbf{r}_s}(\mathbf{x}_s) \tag{19}$$



(a) NAC of OGD (b) NAC of $\mathrm{OMD}_{\mathrm{NE}}$ (c) Time-Average Regret

Fig. 1: NAC of OGD (a) and $\mathrm{OMD}_{\mathrm{NE}}$ (b). Subfigure (c) compares their TAC. The cache capacity is $k = 10$, and $\alpha = 0.8$.

Finally, we consider the Time Average Regret $\mathrm{TAR}(\mathcal{A}) \in [0, R]$, which is precisely the time average regret over the first t time slots.

$$\mathrm{TAR}(\mathcal{A}) = \tfrac{1}{t} \left(\textstyle\sum_{s=1}^{t} f_{\mathbf{r}_s}(\mathbf{x}_s) - \sum_{s=1}^{t} f_{\mathbf{r}_s}(\mathbf{x}_*) \right) \tag{20}$$

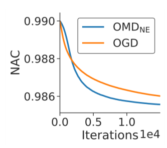
B. Results

1) Stationary Requests: Figures 1 (a) and 1 (b) show the performance w.r.t. NAC of OGD and $\mathrm{OMD}_{\mathrm{NE}}$, respectively, under different learning rates η on the *Fixed Popularity* trace. We observe that both algorithms converge slower under small learning rates, but reach a final lower cost, while larger learning rates lead to faster convergence, albeit to higher final cost. This may motivate the adoption of a diminishing learning rate, that combines the best of the two options, starting large to enable fast convergence, and enabling eventual fine-tuning. We show one curve corresponding to a diminishing learning rate both for OGD and $\mathrm{OMD}_{\mathrm{NE}}$, and indeed they achieve the smallest costs. The learning rate denoted by η^* is the learning rate that gives the tightest worst-case regrets for OGD and $\mathrm{OMD}_{\mathrm{NE}}$, as stated in Theorems III.1 and III.2). While this learning rate is selected to protect against any (adversarial) request sequence, it is not too pessimistic: Figures 1 (a) and 1(b) show it performs well when compared to other learning rates.

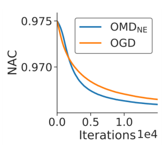
Figure 1 (c) shows the time-average regret TAR of OGD and $\mathrm{OMD}_{\mathrm{NE}}$ over the *Fixed Popularity* trace. As both algorithms have sub-linear regret, their time average regret goes to 0 for $T \to \infty$. Note how instead LRU exhibits a constant time average regret.

2) Effect of Diversity: Figure 2 shows the NAC performance of $\mathrm{OMD}_{\mathrm{NE}}$ and OGD on the traces *Batched Fixed Popularity* (1), (2), and (3) under different cache capacities k and exponent values α. We observe that $\mathrm{OMD}_{\mathrm{NE}}$ outperforms OGD in the more diverse regimes ($\alpha \in \{0.1, 0.2\}$). This is more apparent for smaller values of k. In contrast, OGD outperforms $\mathrm{OMD}_{\mathrm{NE}}$ when requests are less diverse ($\alpha = 0.7$); again, this is more apparent for larger k. These observations agree with Theorems III.3, which postulated that high diversity favors $\mathrm{OMD}_{\mathrm{NE}}$.

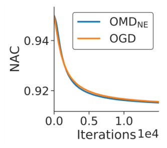
3) Robustness to Transient Requests: Figure 3 shows the normalized average cost of $\mathrm{OMD}_{\mathrm{NE}}$ and OGD over the *Partial Popularity Change* traces, evaluated under different diversity



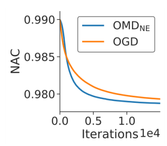
(a) $k = 2, \alpha = 0.1$



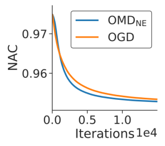
(b) $k = 5, \alpha = 0.1$



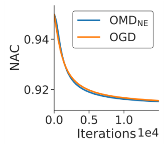
(c) $k = 10, \alpha = 0.1$



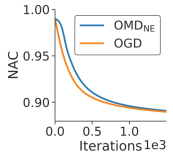
(d) $k = 2, \alpha = 0.2$



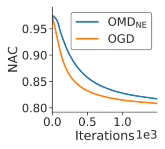
(e) $k = 5, \alpha = 0.2$



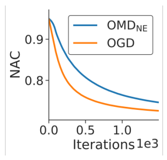
(f) $k = 10, \alpha = 0.2$



(g) $k = 2, \alpha = 0.7$

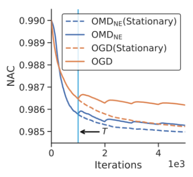


(h) $k = 5, \alpha = 0.7$

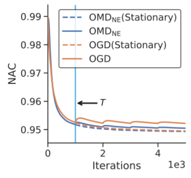


(i) $k = 10, \alpha = 0.7$

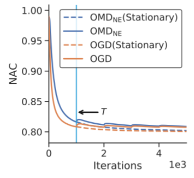
Fig. 2: NAC of $\mathrm{OMD_{NE}}$ and OGD evaluated under different cache sizes and diversity regimes.



(a) $\alpha = 0.1$



(b) $\alpha = 0.4$



(c) $\alpha = 0.7$

Fig. 3: NAC of OGD and $\mathrm{OMD_{NE}}$ evaluated under different diversity regimes when 10% of the files change popularity over time.

regimes. Dashed lines indicate the projected performance in the stationary setting (if request popularties stay fixed). The diversity regimes are selected to provide different performance: in (a) $\mathrm{OMD_{NE}}$ outperforms OGD, in (b) $\mathrm{OMD_{NE}}$ has similar performance to OGD, and in (c) $\mathrm{OMD_{NE}}$ performs worse than OGD.

Across the different diversity regimes, we find the $\mathrm{OMD_{NE}}$ is more robust to popularity changes. In (a) and (b) $\mathrm{OMD_{NE}}$ outperforms OGD in the non-stationary popularity setting: we observe a wider performance gap as compared to the stationary setting. In (c), the algorithms exhibit similar performance.

4) Akamai Trace: Figure 4 shows that the two gradient algorithms, $\mathrm{OMD_{NE}}$ and OGD, perform similarly over the Akamai Trace w.r.t. NMAC (the curves almost overlap). LRU, LFU, and W-LFU may update the cache state after each request, while the gradient algorithms can only update the cache after $R = 5000$ requests. Nevertheless, the gradient algorithms consistently outperform the classic ones.

V. Conclusions

We studied no-regret caching algorithms based on OMD with the neg-entropy mirror map. Our analysis indicates that batch diversity impacts regret performance; a key finding

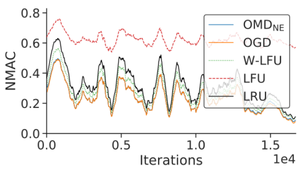


Fig. 4: NMAC of the different caching policies evaluated on the *Akamai Trace*.

is that OGD is favourable in low-diversity regimes, while $\mathrm{OMD_{NE}}$ outperforms OGD under high diversity. Our preliminary evaluation results also suggest that gradient algorithms outperform classic eviction-based policies. We plan to conduct a more thorough comparison as future research.

Acknowledgement. G. Neglia and S. Ioannidis acknowledge support from Inria under the exploratory action MAMMALS and the National Science Foundation (NeTS:1718355), respectively.

References

[1] AWS, "Amazon Web Service ElastiCache," 2018. [Online]. Available: https://aws.amazon.com/elasticache/
[2] E. G. Coffman and P. J. Denning, *Operating systems theory*. Prentice-Hall Englewood Cliffs, NJ, 1973, vol. 973.
[3] S. Traverso *et al.*, "Temporal Locality in Today's Content Caching: Why It Matters and How to Model It," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 5, pp. 5–12, Nov. 2013.
[4] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young, "Competitive paging algorithms," *Journal of Algorithms*, vol. 12, no. 4, pp. 685 – 699, 1991.
[5] G. S. Paschos, A. Destounis, L. Vigneri, and G. Iosifidis, "Learning to cache with no regrets," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 235–243.
[6] E. Hazan, "Introduction to online convex optimization," *Found. Trends Optim.*, vol. 2, no. 3–4, p. 157–325, Aug. 2016.
[7] N. Littlestone and M. Warmuth, "The weighted majority algorithm," *Information and Computation*, vol. 108, no. 2, pp. 212 – 261, 1994.
[8] S. Shalev-Shwartz, "Online learning and online convex optimization," *Found. Trends Mach. Learn.*, vol. 4, no. 2, p. 107–194, Feb. 2012.
[9] R. Bhattacharjee, S. Banerjee, and A. Sinha, "Fundamental limits on the regret of online network-caching," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 4, no. 2, Jun. 2020.
[10] T. Si Salem, G. Neglia, and S. Ioannidis, "No-regret caching via online mirror descent," *arXiv:2101.12588*, 2021.
[11] L. Maggi, L. Gkatzikis, G. Paschos, and J. Leguay, "Adapting caching to audience retention rate," *Computer Communications*, vol. 116, pp. 159–171, 2018.
[12] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.
[13] S. Bubeck, "Convex optimization: Algorithms and complexity," *Found. Trends Mach. Learn.*, vol. 8, no. 3–4, p. 231–357, Nov. 2015.
[14] W. Wang and C. Lu, "Projection onto the capped simplex," *arXiv:1503.01002*, 2015.
[15] C. Chesneau and Y. J. Bagul, "New sharp bounds for the logarithmic function," *Electronic Journal of Mathematical Analysis and Applications*, vol. 8, no. 1, pp. 140–145, 2020.
[16] R. Paredes and G. Navarro, "Optimal incremental sorting," in *2006 Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2006, pp. 171–182.
[17] G. Neglia, D. Carra, M. Feng, V. Janardhan, P. Michiardi, and D. Tsigkari, "Access-time-aware cache algorithms," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 2, no. 4, Nov. 2017.
[18] G. Karakostas and D. N. Serpanos, "Exploitation of different types of locality for web caches," in *Proceedings ISCC 2002 Seventh International Symposium on Computers and Communications*, 2002, pp. 207–212.