Optimistic online caching for batched requests

Francescomaria Faticanti, Giovanni Neglia

Optimistic Online Caching for Batched Requests

Francescomaria Faticanti[a], Giovanni Neglia[b]

[a]*École Normale Supérieure de Lyon, 46 Allée d'Italie, Lyon, 69007, France*
[b]*Inria, 2004 route des Lucioles, BP 93, Sophia Antipolis, 06902, France*

Abstract

In this paper, we investigate 'optimistic' online caching policies, distinguished by their use of future request predictions derived, for example, from machine learning models. Traditional online optimistic policies, grounded in the Follow-The-Regularized-Leader (FTRL) algorithm, incur a higher computational cost compared to classic policies like Least Frequently Used (LFU) and Least Recently Used (LRU). This is due to each cache state update necessitating the resolution of a constrained optimization problem. To address this problem, we introduce and analyze the 'batched' version of two distinct FTRL-based optimistic policies. In this approach, the cache updates occur less frequently, thereby amortizing the update cost over time or over multiple requests. Rather than updating the cache with each new request, the system accumulates a batch of requests before modifying the cache content. First, we present a batched version of the *Optimistic Bipartite Caching* (OBC) algorithm, that works for single requests, then we introduce a new optimistic batched caching policy, the *Per-Coordinate Optimistic Caching* (PCOC) algorithm, derived from the per-coordinate-based FTRL. We demonstrate that these online algorithms maintain 'vanishing regret' in the batched case, meaning their average performance approaches over time that of an optimal static file allocation, regardless of the sequence of file requests. We then compare the performance of these two strategies with each other and against optimistic versions of LFU and LRU. Our experimental results indicate that this batched optimistic approach outperforms traditional caching policies on both stationary and real-world file request traces.

Keywords: Caching, Online Optimization, Predictions, Batched Requests

1. Introduction

Caching systems represent one of the most deeply studied research areas that span from the design of CPU hardware [2, 3] to the development of caching services in cloud computing, e.g., elastic caching systems for cloud and edge [4, 5]. The main objective of such systems is to reduce specific costs for the users, the network operator or the content provider. Caching policies have been studied under various assumptions on the arrival process of file requests. Recently online learning theory has been proposed to deal with caching settings where requests do not exhibit a regular pattern, and can be thought to be selected by an adversary [6, 7, 8]. Such an approach for the requests modeling stands in contrast to traditional stochastic models which can fail, e.g., in cases of small users' populations [9].

Online caching has been studied in the online convex optimization (OCO) framework [10] starting from the work [6]. In this setting, the main objective is to design algorithms that minimize the *regret*, i.e., the difference between the costs incurred over a fixed time horizon by the proposed solution and by the optimal static content allocation. Specifically, if the regret of an algorithm increases at a sublinear rate relative to the length of the time horizon, the algorithm achieves the same cost of the optimal static allocation in the long run. Later contributions analyzed other online learning algorithms [11] and provided new lower bounds on the regret [7].

Motivation. Nowadays, thanks to the huge availability of data and resources in cloud systems, reliable predictions for future requests can be generated by machine learning (ML) models [12, 13]. Online caching policies can exploit such predictions to incur lower future costs by anticipating future requests. Online algorithms with access to predictions are called *optimistic* [14, 15]. References [14, 16] provide example of optimistic online algorithms based on the Follow-The-Regularized-Leader (FTRL) and Online Mirror Descent (OMD) frameworks [10]. Mhaisen et al. [15] presented one of the first applications of optimistic online algorithms to a caching problem. They proved that predictions, even if not perfectly accurate, can improve the performance of online algorithms. They designed an optimistic FTRL algorithm that requires the cache to be updated each time a new file request is received. However, these updates are computationally very expensive, as they require to solve a constrained optimization problem, and can then limit the applicability of online caching policies. Hence, the main question we want to investigate in this work is the following: "*how will such algorithms perform if the caching system collects a batch of requests before updating the cache in order to amortize the computational cost?*". We show that, in order to amortize the update cost over time and over multiple requests, a *batched* approach can be adopted without loss in the performance, i.e., the presented batched algorithms still enjoy sublinear regret. In

⋆Research supported by Inria under the exploratory action MAMMALS. A preliminary version of this paper has been presented at IEEE ICC 2023 [1].

this batched scenario the caching system serves each request as it arrives, but updates the cache less frequently on the basis of the batch of requests collected since the last update [11]. We stress that the batched approach does not cause any additional delay for the user.

The novelty of this work resides in the study of optimistic online caching policies able to work on batches of requests. Our main contributions are the following:
1) We present a batched version of the optimistic caching policy in [15] and prove that it still enjoys sublinear regret.
2) We introduce a new optimistic batched caching policy based on the per-coordinate-based algorithm in [14].
3) We analytically characterize under which conditions each of these two caching policies outperforms the other.
4) We determine when a batched operation provides better performance in terms of regret under different models for the predictions' error.
5) We design optimistic versions of classical caching policies like LFU and LRU.
6) We experimentally show, both on stationary traces and real ones, that our optimistic batched online caching policies outperform classical caching policies like LRU and LFU achieving both smaller service cost and per-request computational cost.

The reminder of this paper is organized as follows. The next section discusses the main related works. Section 3 introduces the system model and the problem description. In Section 4 we describe the optimistic caching framework and we present the main algorithms that take into account predictions: the one presented in [15] and the one we propose. Section 5 presents an analysis of the regret bounds achieved by the two algorithms and a comparison between the single-request operation and the batched one. Experimental results are presented in Section 6. Finally, Section 7 concludes the paper.

2. Related Work

Caching optimization problems have been deeply studied in the literature both on the offline and on the online perspective [17]. Several works have explored the offline static allocation of files under the assumption of knowing the requests [18, 19, 20]. On the online perspective, online caching policies based on gradient methods have been studied under the assumption of stochastic requests [21, 22]. In these works, the proposed algorithms have been evaluated under various performance metrics. We consider adversarial requests, i.e. the requests are thought as they are generated by an adversary trying to deteriorate the system's performance, and the regret as the main performance metric following the recent regret-based research on caching [7, 8, 23, 24, 11]. In this context, the main goal is to design algorithms with sublinear regret with respect to the time horizon leading to algorithms that behave as the optimal static solution in hindsight on average. Such online policies are called *no-regret* algorithms [6].

Adversarial requests are considered in caching since Sleator and Tarjan's paper [25] through the *competitive ratio* metric. However, as proved in [26], algorithms that ensure constant competitive ratio do not necessarily guarantee sublinear regret.

The main optimization framework adopted in this paper is the Online Convex Optimization (OCO). It was first introduced by Zinkevich [27] showing that the projected gradient descent achieves sublinear regret bounds in the online setting. The works from Paschos et al. [6, 17] were the first to apply the OCO framework to caching problems providing no-regret algorithms for the online caching problem. Bhattacharjee et al. [7] extended the work from Paschos et al. showing tighter lower bounds for the regret and proposing new online caching policies for the networked scenario based on the Follow-The-Perturbed-Leader (FTPL) algorithm. In our case, we consider the single-cache scenario and analyse the framework of the Follow-The-Regularized-Leader (FTRL) that has been proved one of the most promising algorithms for taking into account predictions in the online learning setting [14]. Indeed, as shown in [28], the optimistic version of FTRL benefits more from the use of predictions with respect to the optimistic FTPL.

The combination of predictions and caching has recently drawn attention given the significant usage of machine learning (ML) models for the computation of such predictions. The idea of exploiting predictions in the decision process has lead to the design of so called *optimistic* online algorithms. Some works have already incorporated predictions in stochastic optimization [29, 30] assuming the requests and system perturbations to be stationary. In our work we do not make any assumption on the quality of the predictions that can be also thought as generated by and adversary. Mohri et al. [14] studied the regret performance of FTRL algorithms in adversarial settings including the predictions proving sublinear regret bounds. To the best of the authors' knowledge, Mhaisen et al. [15] have been the first to apply optimistic online algorithms in the caching framework under adversarial settings. They proposed FTRL-based algorithms that, at each new request, update the cache state based on the previous incurred costs and the prediction for the next request. However, such algorithms imply the application of computationally-expensive operations, such as the projection on the domain set of the cache states [31, 32], at each new request. To amortize the computational cost over time we propose to collect a *batch* of requests before deciding the new cache state, leading to less frequent updates of the cache state. Theoretical analysis confirm that the size of such a batch does not affect the regret guarantees of the presented algorithms. A batched approach in caching has been presented in [31] but without taking into account predictions for future requests. Other optimistic online algorithms for caching are proposed in [28]. However, the proposed policies update the cache state at each new request, and the files are entirely stored in the cache, whilst, in line with recent works [6, 15], we assume that the cache can store arbitrary fraction of files.

The novelty of this work is in studying the performance of optimistic version of FTRL-based algorithms dealing with batches of requests. The account of batched requests reinforces also the use of predictions in the optimization process. It is reasonable indeed, when the predictions come from ML models, to involve a set of possible future requests in the predictions rather than a single future request. We show that the optimistic online batched algorithms introduced in this work present the

best performance in terms of final miss-ratio and computational cost with respect the most practical and implemented caching policies.

3. System Description and Problem Formulation

3.1. System Model

We consider the same system setting described in [11]. We consider a catalog $\mathcal{N} = \{1, 2, \dots, N\}$ with N equal-size files. File requests are served by a single local cache or by a remote server. In particular, a request for a file $i \in \mathcal{N}$ can be served by the cache for free or by a remote server incurring a per-file dependent cost $w_i \in \mathbb{R}^+$ (more details about our cost model below). This cost can be related to the time needed to retrieve the file from a remote server, or be a monetary cost due to the utilisation of a third-party infrastructure for the file retrieval. We do not make any assumption on the requests arrival process, i.e., we analyse the system in an adversarial online setting where the requests can be thought as generated by an adversary trying to deteriorate system's performance.

Cache State. We assume the cache can store arbitrary fractions of files from the catalog (as in [33, 6, 15]) up to a maximum of k complete files, where $k \in \{1, 2, \dots, N\}$. We denote as $x_{t,i} \in [0, 1]$ the fraction of file i stored in the cache at time t. The cache state, at time t, is then represented by the vector $\mathbf{x}_t = [x_{t,i}]_{i \in \mathcal{N}}$ belonging to the set

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

The set $\mathcal{X}$ is the capped simplex defined by the capacity constraint of the local cache. It is sometimes convenient to express the cache capacity as a fraction of the catalog size, i.e., $k = \alpha N$, where $\alpha \in [0, 1]$.

Cache Updates. Caching decisions are taken after batches (potentially of different sizes) of requests have been served. Formally, at each time-slot $t = 1, \dots, T$ the system collects R_t requests from the users and then it may updates the cache state. The request process can then be represented as a sequence of vectors $\mathbf{r}_t = (r_{t,i} \in \mathbb{N} : i \in \mathcal{N})$, $\forall t$, where $r_{t,i}$ denotes the number of requests for file i in the t-th timeslot. The request process belongs then to the set

$$\mathcal{R} = \left\{ \mathbf{r}_t \in \mathbb{N}^N, t = 1, \dots, T | \sum_{i \in \mathcal{N}} r_{t,i} = R_t \right\}.$$

For some results we will rely on the following additional assumption (already proposed in [11]):

Assumption 1. Every batch contains the same number of requests (i.e., $R_t = R$ for all $t \in 1, \dots, T$) and the number of requests for each file within the batch is bounded by h (i.e., $r_n^t \in \{0, \dots, h\}$).

Cost Function. For each new batch of requests $\mathbf{r}_t$ the system pays a cost proportional to the missing fraction $(1 - x_{t,i})$ for each file $i \in \mathcal{N}$ from the local cache. More formally:

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

The sum is weighted by the cost w_i and by the number of times $r_{t,i}$ file i is requested in the batch $\mathbf{r}_t$.

Predictions. Predictions for the next batch of requests can be provided by a ML model such as a neural network. The ML model may output an estimate of the number of requests for each file in the next time-slot, or equivalently an estimate of the current popularity of each file (the probability that a request is for that file). The model can be similar to those used in streaming services like Netflix to provide recommendations to users on the basis of their view history [12]. ML regression models to predict file requests for caching applications have been proposed in [34, 35, 36]. An optimistic caching algorithm can exploit such predictions in order to update the cache content. In this paper, we assume that the predictor provides an estimate for the number of requests for each file in the next time-slot. We indicate with $\tilde{r}_{t+1,i}$ the prediction of the number of requests for file i at time $t+1$. It is then possible to directly estimate the gradient of the cost function in that time-slot. More formally, we denote by $\tilde{\mathbf{g}}_{t+1}$ the prediction of $\mathbf{g}_{t+1} = \nabla f_{\mathbf{r}_{t+1}}(\mathbf{x}_{t+1})$, the gradient of the cost function at time $t+1$, where $\tilde{g}_{t+1,i} = -w_i \tilde{r}_{t+1,i}$. We are going to show that the presented algorithms present good performance even in cases of low-quality estimations of the predictions.

3.2. Online Caching Problem

We can fit our caching problem in the *Online Convex Optimization* (OCO) framework [27, 37], where a learner (in our case the caching system) has to take a decision $\mathbf{x}_t$ from a convex set $\mathcal{X}$ at each time slot t before the adversary selects the cost function $f_{\mathbf{r}_t}$, i.e., the learner changes the cache state before experiencing the cost. Hence, the main objective is to devise a caching policy $\mathcal{A}$ that, at each time-slot t, computes the cache state $\mathbf{x}_{t+1}$ for the next time-slot given the current cache state $\mathbf{x}_t$, the whole history up to time t $((\mathbf{x}_1, r_1), \dots, (\mathbf{x}_t, r_t))$, and possibly the predictions for the next time-slot. As it is common in online learning, the main performance metric for the caching policy $\mathcal{A}$ is the regret defined as

$$R_T(\mathcal{A}) = \sup_{\{\mathbf{r}_1, \dots, \mathbf{r}_T\}} \left\{ \sum_{t=1}^{T} f_{\mathbf{r}_t}(\mathbf{x}_t) - \sum_{t=1}^{T} f_{\mathbf{r}_t}(\mathbf{x}^\star) \right\}. \tag{2}$$

This function denotes the difference between the total cost obtained by the online policy $\mathcal{A}$ over a time horizon T, and the total cost of the best caching state $\mathbf{x}^\star$ in hindsight , i.e., $\mathbf{x}^\star = \arg\min_{x \in \mathcal{X}} \sum_{t=1}^{T} f_{\mathbf{r}_t}(\mathbf{x})$. The supremum in (2) indicates an adversarial setting for the regret definition, i.e., the regret is measured against an adversary that generates requests trying to deteriorate the performance of the caching system. The main goal in this setting is to design a caching policy $\mathcal{A}$ that achieves sublinear regret, $R_T(\mathcal{A}) = o(T)$. This ensures a zero average regret as T grows implying that the designed policy behaves on average as the optimal static one.

In what follows, given a sequence of vectors $(\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_t, \dots)$, we denote their aggregate sum up to time t as $\mathbf{y}_{1:t} \triangleq \sum_{s=1}^{t} \mathbf{y}_s$.

Algorithm 1: `Optimistic Online Caching`

Input: $\mathcal{N}, k, x_1 \in \mathcal{X}$

1 **for** $t = 1, \ldots, T$ **do**
2 Receive the batch of requests $\mathbf{r}_t$;
3 Incur cost $f_{\mathbf{r}_t}(\mathbf{x}_t)$;
4 Receive the new prediction $\tilde{\mathbf{g}}_{t+1}$;
5 Compute $\mathbf{x}_{t+1}$ taking into account $\tilde{\mathbf{g}}_{t+1}$ and the history $((\mathbf{x}_1, \mathbf{r}_1), \ldots, (\mathbf{x}_t, \mathbf{r}_t))$ according to (3).

4. Optimistic Caching

As highlighted in [15], an optimistic caching policy can exploit, at each time-slot t, predictions for the requests at time $t+1$ in order to compute the caching state $\mathbf{x}_{t+1}$. The general scheme for optimistic online caching is described in Algorithm 1. Given an initial feasible solution $\mathbf{x}_1 \in \mathcal{X}$, the cache operates at each time-slot t as follows: i) the new batch of requests $\mathbf{r}_t$ is revealed; ii) based on the current cache state $\mathbf{x}_t$, the cache incurs the cost $f_{\mathbf{r}_t}(\mathbf{x}_t)$; iii) the cache receives the prediction $\tilde{\mathbf{g}}_{t+1}$ for the next time-slot, and iv) based on such predictions and on all the history up to time t $((\mathbf{x}_1, r_1), \ldots, (\mathbf{x}_t, \mathbf{r}_t))$, it computes the next cache state $\mathbf{x}_{t+1}$.

In the OCO literature, algorithms exploiting predictions are usually variants of the *Follow-The-Regularized-Leader* (FTRL) algorithm [38, 14]. The classic *Follow-The-Leader* (FTL) algorithm [39] greedily selects the next state in order to minimize the aggregate cost over the past, i.e.,

$$\mathbf{x}_{t+1} := \arg\min_{\mathbf{x}\in\mathcal{X}} \sum_{s=1}^{t} f_{\mathbf{r}_s}(\mathbf{x}) = \arg\min_{\mathbf{x}\in\mathcal{X}} \mathbf{g}_{1:t}^{\intercal}\mathbf{x},$$

where the last equality follows from the linearity of the cost functions. The linearity of the problem leads FTL to commit to store entirely some files (i.e., $\mathbf{x}_{t+1} \in \{0,1\}^N$), but this can be exploited by the adversary and leads to a linear regret. The FTRL algorithm improves the performance of FTL by adding a non-linear proximal regularization term, which leads to more cautious updates.[1] Let $r_t(\mathbf{x})$ be the regularization function used at time t (to be specified later). The FTRL algorithm's update step is given by

$$\mathbf{x}_{t+1} := \arg\min_{\mathbf{x}\in\mathcal{X}} \left\{ r_{1:t}(\mathbf{x}) + (\mathbf{g}_{1:t} + \tilde{\mathbf{g}}_{t+1})^{\top}\mathbf{x} \right\}. \tag{3}$$

As we are going to see, the function to minimize in (3) is a quadratic function. The Problem 3 can then be solved through popular solvers like `CVX`, but the presence of the constraint $\mathbf{x} \in \mathcal{X}$ makes the update a potentially expensive operation, motivating the batched operation we propose.

In what follows, we describe two particular FTRL instances applied to our caching problem. The two instances differ by the specific regularization function used in (3) for updating the cache state (line 5 of Algorithm 1).

[1] A regularizer is proximal if $\arg\min_{\mathbf{x}\in\mathcal{X}} r_t(\mathbf{x}) = \mathbf{x}_t$.

4.1. Optimistic Bipartite Caching (OBC)

The first algorithm is called *Optimistic Bipartite Caching* (OBC) and was introduced in [15] for a bipartite caching system with a single request at each time-slot. OBC adopts as proximal regularizer

$$r_t(\mathbf{x}) = \frac{\sigma_t}{2}\|\mathbf{x} - \mathbf{x}_t\|^2, t \geq 1, \tag{4}$$

with the following parameters

$$\sigma_t = \sigma(\sqrt{h_{1:t}} - \sqrt{h_{1:t-1}}), \quad \text{where} \quad h_t = \|\mathbf{g}_t - \tilde{\mathbf{g}}_t\|^2, \tag{5}$$

and $\sigma \geq 0$. The regularizer $r_{1:t}(\mathbf{x})$ is 1-strongly convex with respect to the norm $\|\mathbf{x}\|_{(t)} = \sqrt{\sigma_{1:t}}\|\mathbf{x}\|$ whose dual norm we denote by $\|\mathbf{x}\|_{(t),\star}$. The regularizer depends on the Euclidean distance between the actual gradient $\mathbf{g}_t$ and the predicted one $\tilde{\mathbf{g}}_t$. Qualitatively, if predictions are very accurate, $r_{1:t}(\mathbf{x})$ is small and then the update in (3) will focus on minimizing the (predicted) aggregate cost $(\mathbf{g}_{1:t} + \tilde{\mathbf{g}}_{t+1})^{\intercal}\mathbf{x}$. On the contrary, if predictions are not accurate, the regularizer will lead to more cautious updates. The regularization function can then be interpreted as an implicit adaptive learning rate [14]: as gradient predictions become more accurate the algorithm *accelerates* towards the minimum of the aggregate cost $(\mathbf{g}_{1:t} + \tilde{\mathbf{g}}_{t+1})^{\intercal}\mathbf{x}$.

In the next section, we present theoretical guarantees on the OBC's regret for the batched setting considered in this paper.

4.2. Per-Coordinate Optimistic Caching (PCOC)

Mohri et al. [14, Corollary 2] proposed an FTRL algorithm where the regularization function decomposes over the coordinates and thus the acceleration occurs on a per-coordinate basis. In this case, if gradient predictions are more accurate on certain coordinates, the algorithm will accelerate the convergence of such coordinates. Here we present a generalization of this algorithm, called *Per-Coordinate Optimistic Caching* (PCOC), which introduces a generic parameter σ in the definition of the regularization function:

$$r_t(\mathbf{x}) = \sum_{i=1}^{N}\sum_{s=1}^{t} \frac{\sigma_{t,i}}{2}(x_i - x_{s,i})^2, \tag{6}$$

where $\sigma_{t,i} = \sigma(\Delta_{t,i} - \Delta_{t-1,i})$, and $\Delta_{s,i} = \sqrt{\sum_{a=1}^{s}(g_{a,i} - \tilde{g}_{a,i})^2}$. The function $r_{0:t}(\mathbf{x})$ is 1-strongly convex with respect to[2]

$$\|\mathbf{x}\|_{(t)}^2 = \sum_{i=1}^{N} \sigma_{1:t,i}x_i^2, \quad \text{with} \quad \|\mathbf{x}\|_{(t),*}^2 = \sum_{i=1}^{N} \frac{x_i^2}{\sigma_{1:t,i}}. \tag{7}$$

5. Performance Analysis

Here we prove theoretical guarantees for the regret bounds of the algorithms presented in the previous section in the case of a single cache and multiple requests at each time-slot. We show that both algorithms enjoy sublinear regrets even if gradient predictions are inaccurate.

[2] With some abuse of notation we use the same symbols (resp. $\|\cdot\|_{(t)}$ and $\|\cdot\|_{(t),*}$) to denote the norms and the dual norms for OBC and PCOC. The interpretation of the symbols should be clear from the context.

5.1. Regret bound of OBC with single cache and R requests

We extend the regret bound in [15, Theorem 1] to the case of batched requests, but we also improve the coefficients taking into account the capacity constraint.

Theorem 5.1. *The regret of OBC is bounded as follows:*

$$R_T(OBC) \leq 2\sqrt{2\min\{k, N-k\} \cdot \sum_{t=1}^{T} \|\mathbf{g}_t - \tilde{\mathbf{g}}_t\|^2}. \tag{8}$$

Proof. We start from the inequality in [14, Theorem 1],

$$R_T \leq r_{1:T}(\mathbf{x}^\star) + \sum_{t=1}^{T} \|\mathbf{g}_t - \tilde{\mathbf{g}}_t\|^2_{(t),\star}, \quad \forall \mathbf{x}^\star \in \mathcal{X}. \tag{9}$$

Substituting the regularization functions we obtain

$$R_T \leq \frac{\sigma}{2} \sum_{t=1}^{T} (\sqrt{h_{1:t}} - \sqrt{h_{1:t-1}}) \|\mathbf{x}^\star - \mathbf{x}_t\|^2 + \sum_{t=1}^{T} \frac{h_t}{\sigma\sqrt{h_{1:t}}} \tag{10}$$

In our case, as highlighted in [6], the Euclidean diameter of $\mathcal{X}$ is upper bounded by Δ

$$\|\mathbf{x} - \mathbf{x}_t\|^2 \leq \Delta^2 \triangleq \min\{2k, 2(N-k)\}, \forall \mathbf{x}, \mathbf{x}_t \in \mathcal{X}. \tag{11}$$

Introducing Δ in (10), and using [40, Lemma 3.5] it follows

$$\begin{aligned} R_T &\leq \frac{\sigma}{2}\Delta^2 \sum_{t=1}^{T} (\sqrt{h_{1:t}} - \sqrt{h_{1:t-1}}) + \sum_{t=1}^{T} \frac{h_t}{\sigma\sqrt{h_{1:t}}} \\ &\leq \frac{\sigma}{4}\Delta^2 \sqrt{h_{1:T}} + \frac{2}{\sigma}\sqrt{h_{1:T}} = (\frac{\sigma}{2}\Delta^2 + \frac{2}{\sigma})\sqrt{h_{1:T}}. \end{aligned} \tag{12}$$

Setting $\sigma = 2/\Delta$ we obtain the desired bound. □

Theorem 5.1 shows that the regret bound depends on the cache size, and on the accuracy in the predictions. The algorithm enjoys a zero regret if the cache is able to store the complete catalog, i.e., $k = N$, or if predictions are perfect, i.e., $\tilde{g}_t = g_t$. On the other hand, even if predictions are imperfect, OBC may guarantee sublinear regret, as shown by the following corollary.

Corollary 1. *Under Assumption 1,*

$$R_T \leq 2\|w\|_\infty \sqrt{2\min\{k, N-k\}TRh} = O(\sqrt{T}). \tag{13}$$

The proof easily follows from $\|\tilde{\mathbf{g}}_t - \mathbf{g}_t\|^2 \leq \|w\|^2_\infty Rh$ under *Assumption 1*.

5.2. Regret bound of PCOC

The following proof follows the steps in [14, Corollary 2], introducing the adjustable parameter $\sigma \geq 0$ in the definition of the regularizer 6 and taking into account that $x_i \in [0, 1]$ for our caching application.

Theorem 5.2. *The regret of PCOC is bounded as follows*

$$R_T(PCOC) \leq 2\sum_{i=1}^{N} \sqrt{\sum_{t=1}^{T} (g_{t,i} - \tilde{g}_{t,i})^2}. \tag{14}$$

Proof. From [14, Theorem 3], applying the regularization function defined in (6) and the norms defined in (7), we obtain

$$\begin{aligned} R_T &\leq \frac{\sigma}{2} \sum_{i=1}^{N} \sum_{s=1}^{T} (\Delta_{s,i} - \Delta_{s-1,i})(x_i - x_{s,i})^2 + \sum_{t=1}^{T} \|\mathbf{g}_t - \tilde{\mathbf{g}}\|^2_{(t),\star} \\ &\overset{(a)}{\leq} \frac{\sigma}{2} \sum_{i=1}^{N} \sqrt{\sum_{t=1}^{T} (g_{t,i} - \tilde{g}_{t,i})^2} + \sum_{t=1}^{T} \|\mathbf{g}_t - \tilde{\mathbf{g}}\|^2_{(t),\star} \\ &\overset{(b)}{\leq} \frac{\sigma}{2} \sum_{i=1}^{N} \sqrt{\sum_{t=1}^{T} (g_{t,i} - \tilde{g}_{t,i})^2} + \frac{2}{\sigma} \sum_{i=1}^{N} \sqrt{\sum_{t=1}^{T} (g_{t,i} - \tilde{g}_{t,i})^2} \\ &= \left(\frac{\sigma}{2} + \frac{2}{\sigma}\right) \sum_{i=1}^{N} \sqrt{\sum_{t=1}^{T} (g_{t,i} - \tilde{g}_{t,i})^2}, \end{aligned} \tag{15}$$

where (a) follows from $(x_i - x_{s,i})^2 \leq 1$ and the results of the telescopic sum $\sum_{s=1}^{t} \Delta_{s,i} - \Delta_{s-1,i}$, and (b) from the application of [40, Lemma 3.5] to $\sum_{t=1}^{T} \|\mathbf{g}_t - \tilde{\mathbf{g}}\|^2_{(t),\star}$ once the definition of dual norm in (7) has been applied. For the minimization of the regret bound we can set $\sigma = 2$. □

Similar to OBC, PCOC has zero regret under perfect predictions, and sublinear regret under *Assumption 1*.

Corollary 2. *Under Assumption 1,*

$$R_T \leq 2Nh\|w\| \sqrt{T} = O(\sqrt{T}). \tag{16}$$

The proof follows from $(g_{t,i} - \tilde{g}_{t,i})^2 \leq \|w\|^2 h^2$ under *Assumption 1*.

5.3. Comparison between the two regret bounds

We compare the two bounds presented above in two specific scenarios for the prediction error: i) a constant error on each component of the gradient, and ii) a prediction error proportional to the popularity of the files in the catalog.

In the first case, OBC presents a better bound with respect to the one obtained by PCOC. In fact, say that $|g_{t,i} - \tilde{g}_{t,i}| = \epsilon$ for each i and t, then $R_T(OBC) = 2\sqrt{2\min\{k, N-k\}NT\epsilon^2} \leq 2N\sqrt{T\epsilon^2} = R_T(PCOC)$.

In the second case, PCOC may perform better because it specifically takes into account the heterogeneity of the prediction error across the components. We deviate here from the adversarial request model and consider that 1) requests arrive according to a Poisson process with rate λ, and 2) a request is for file i with probability p_i independently from the past [41]. Moreover, we assume the algorithm is executed every time unit, and per-file costs equal 1. In this case, $g_{t,i} \sim \text{Poisson}(\lambda p_i)$ for each $i \in \mathcal{N}$. We compute the expected value of the bounds in (8) and in (14), assuming that the cache can store a fraction α of the catalog ($k = \alpha N$), and $\tilde{g}_{t,i} = \lambda p_i$, i.e., we have a perfect predictors for the expected number of future requests. For the

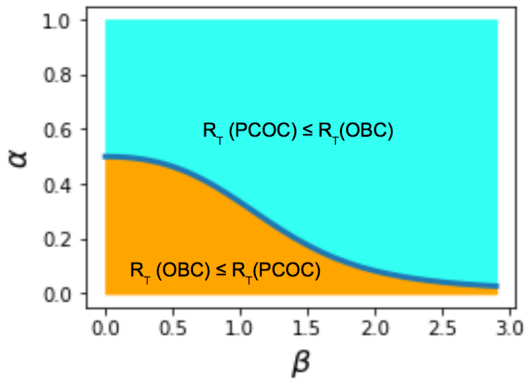


Figure 1: OBC vs PCOC, different regimes for the regret as a function of the Zipf exponent (β) and the relative cache size ($k = \alpha N$).

OBC bound, we obtain

$$\mathbb{E}\left[2\sqrt{2\alpha N\sum_{t=1}^{T}\sum_{i=1}^{N}(g_{t,i}-\tilde{g}_{t,i})^2}\right] \leq$$
$$\leq 2\sqrt{2\alpha N\sum_{t=1}^{T}\sum_{i=1}^{N}\mathbb{E}[(g_{t,i}-\tilde{g}_{t,i})^2]} =$$
$$= 2\sqrt{2\alpha N\sum_{t=1}^{T}\sum_{i=1}^{N}\lambda p_i} = 2\sqrt{2\alpha\lambda NT} \quad . \tag{17}$$

For the PCOC bound, we obtain

$$\mathbb{E}\left[\sum_{i=1}^{N}\sqrt{\sum_{t=1}^{T}(g_{t,i}-\tilde{g}_{t,i})^2}\right] \leq \sum_{i=1}^{N}\sqrt{\sum_{t=1}^{T}\mathbb{E}[(g_{t,i}-\tilde{g}_{t,i})^2]}$$
$$= 2\sum_{i=1}^{N}\sqrt{\sum_{t=1}^{T}\lambda p_i} = 2\sum_{i=1}^{N}\sqrt{T\lambda p_i} \quad . \tag{18}$$

Comparing the two bounds (18) and (17), we find that (18) is a smaller than (17) when $\alpha \geq \left(\sum_{i=1}^{N}\sqrt{p_i}\right)^2/(2N\sum_{i=1}^{N}p_i)$. If p_i obeys to a Zipf law with exponent β, we can numerically find from the inequality the minimum value of α such that the bound of (18) is tighter. In Figure 1 we can notice that the threshold for α decreases as β increases. In the case of a uniform popularity distribution ($\beta = 0$), OBC outperforms PCOC unless the cache can store at least half of the catalog. As the popularity distribution becomes more skewed, PCOC is expected to perform better than OBC in terms of regret bound, but for very small caches.

5.4. Batch Selection

We maintain the Poisson assumption about the request arrival process and evaluate what is the effect of requests batching on the regret, focusing on the bound in Theorem 5.1 (the same analysis can be carried out on the bound in Theorem 5.2). We analyse the expected value of such bound in a general batched-requests setting where the caching decisions are taken every τ among an overall time interval of Θ time units where a single request is available at each time. Looking at the expected value of the regret bound we have:

$$\mathbb{E}\left[R_{\Theta/\tau}\right] \leq \mathbb{E}\left[C\sqrt{\sum_{t=1}^{\Theta/\tau}\sum_{i=1}^{N}(g_{t,i}-\tilde{g}_{t,i})^2}\right], \tag{19}$$

where $C \triangleq 2\sqrt{2\min\{k, N-k\}}$. In this case we have $g_{t,i} \sim \text{Poisson}(\lambda_i\tau)$. For the predictions $\tilde{g}_{t,i}$ we consider two options: i) they coincide with the expected number of future requests, or ii) they coincide with the requests seen during the previous times-lots.

In the first case we have

$$\mathbb{E}\left[C\sqrt{\sum_{t=1}^{\Theta/\tau}\sum_{i=1}^{N}(g_{t,i}-\tilde{g}_{t,i})^2}\right] \overset{(a)}{\leq} C\sqrt{\sum_{t=1}^{\Theta/\tau}\sum_{i=1}^{N}\mathbb{E}\left[(g_{t,i}-\tilde{g}_{t,i})^2\right]} =$$
$$= C\sqrt{\sum_{t=1}^{\Theta/\tau}\sum_{i=1}^{N}\text{Var}(g_{t,i})} = C\sqrt{\sum_{t=1}^{\Theta/\tau}\sum_{i=1}^{N}\lambda_i\tau} = C\sqrt{\Theta\sum_{i=1}^{N}\lambda_i}, \tag{20}$$

where (a) follows from Jensen's inequality. The right hand side of (20) suggests that batching has no effect on the algorithm's regret.

In the second case, for $t > 1$, $\tilde{g}_{t,i} = g_{t-1,i} = n_{t,i}(\tau) \sim \text{Poisson}(\lambda_i\tau)$, where $n_{t,i}(\tau)$ is the number of arrivals within the interval $[(t-2)\tau, (t-1)\tau]$. The initial prediction is given by $\tilde{g}_{1,i} = \frac{n_i(\tau_0)}{\tau_0}$, where τ_0 is a first warm-up interval. Looking at the expectation of $(g_{t,i} - \tilde{g}_{t,i})^2$, we have

$$\mathbb{E}[(g_{t,i}-\tilde{g}_{t,i})^2] =$$
$$\mathbb{E}[(g_{t,i}-\tilde{g}_{t,i}-\mathbb{E}[g_{t,i}]+\mathbb{E}[g_{t,i}]-\mathbb{E}[\tilde{g}_{t,i}]+\mathbb{E}[\tilde{g}_{t,i}])^2] =$$
$$\text{Var}(g_{t,i}) + \text{Var}(\tilde{g}_{t,i}) + (\mathbb{E}[g_{t,i}]-\mathbb{E}[\tilde{g}_{t,i}])^2 =$$
$$= \begin{cases} 2\lambda_i\tau, & t > 1 \\ \lambda_i\tau + (\frac{\tau}{\tau_0})^2\lambda_i\tau_0, & t = 1. \end{cases} \tag{21}$$

Summing all the terms over N and Θ/τ, we obtain

$$\sum_{i=1}^{N}\sum_{t=1}^{\Theta/\tau}\mathbb{E}[(g_{t,i}-\tilde{g}_{t,i})^2] = \sum_{i=1}^{N}\frac{\Theta-\tau}{\tau}2\lambda_i\tau + \lambda_i\tau + m^2\tau^2, \tag{22}$$

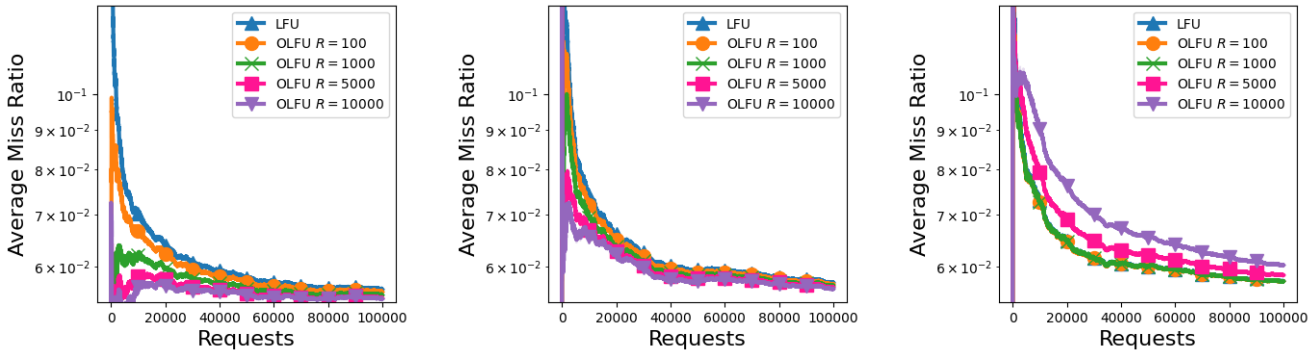
where $m^2 \triangleq \frac{\lambda_i\tau_0}{\tau_0^2}$. Under these predictions, there is indeed an optimal timescale τ^* for batching, that is $\tau^* = \min\{\frac{\tau_0}{2}, \Theta\}$. Hence, in case of a good initial prediction (large τ_0) we should select $\tau = \Theta$. Otherwise, in case of a less accurate initial prediction we should choose a smaller value $\tau = \frac{\tau_0}{2}$.

6. Numerical Results

6.1. Experimental Settings

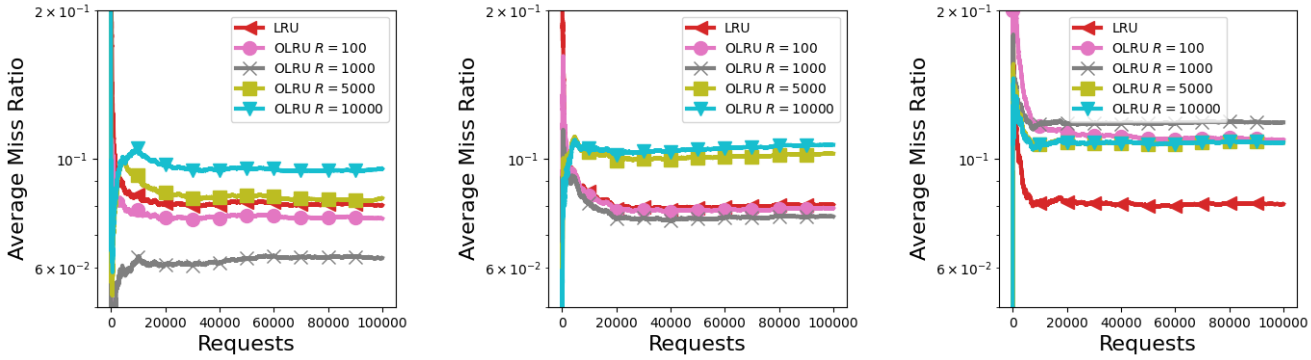
6.1.1. Datasets

We evaluated the presented approaches on both synthetic and real traces. For the synthetic case, we generated stationary synthetic traces where individual file requests are generated i.i.d. according to a Zipf distribution with parameter



(a) $\pi = 0$ (b) $\pi = 0.3$ (c) $\pi = 0.9$

Figure 2: Average Miss Ratio of OLFU vs. LFU, Zipf distribution with $\beta = 1.5$ and $k = 100$.



(a) $\pi = 0$ (b) $\pi = 0.3$ (c) $\pi = 0.9$

Figure 3: Average Miss Ratio of OLRU vs. LRU, Zipf distribution with $\beta = 1.5$ and $k = 100$.

$\beta \in \{0.8, 1.2, 1.5\}$ from a catalog of $N = 1000$ files. The values used for the exponent of the Zipf distribution are in line with the estimates reported in the literature for various kinds of systems [42]. We evaluate the studied solutions against state-of-the-art algorithms over a horizon of $I = 10^5$ requests. *Batched* algorithms have a constant batch size, i.e., $R_t = R$ with $R \in \{100, 1000, 2000, 5000, 10000\}$ for synthetic traces, and $R \in \{10, 50, 100, 300, 1000\}$ for the real trace. The cache size k varies in $\{10, 50, 100, 600\}$. The cache capacity and the batch sizes chosen for the experimentations are in line with the one selected in other theoretical studies on online caching policies such as [31]. The real trace counts $2 \cdot 10^4$ requests for the $N = 10^3$ most popular files as measured at a given server in Akamai CDN provider [43]. In all the experiments we set $w_i = 1, \forall i \in \mathcal{N}$, the cost in (1) corresponds then to the total number of misses. In Figures 2, 3, 4, 5, and 7b we report the average and the 95% confidence interval for the metric interest as computed over 30 different runs. The sequence of file requests is the same across different runs, but predictions are independently generated at each run. We observe that 95% confidence intervals are often too narrow to be observable.

6.1.2. *Predictions*

As stated before, the predictions for the algorithms could be the output of a prediction ML model, such as a neural network, that forecasts the number of requests for each file in the next time-slot. Such predictions can be more or less accurate. In order to show the robustness of the considered algorithms to the quality of the predictions, we have generated the predictions in such a way their accuracy with respect to the actual requests can be tuned varying the values of single parameters.

Similarly to what done in related work [15, 28], we consider some stylized predictors, which have the advantage of exhibiting a tunable accuracy or to better represent an adversarial behaviour. In particular, we have considered three types of predictions:

- *Type 1* predictions are generated according to $\tilde{\mathbf{g}}_{t+1} = (1 - \xi)\mathbf{g}_{t+1} + \xi \frac{R}{N}$, with $\xi \in [0, 1]$;
- *Type 2* predictions are the same as in [28]: each predicted request is attributed to a file selected uniformly at random with probability π and otherwise it coincides with an actual future request;
- *Type 3* predictions are random permutations of the correct gradients.

The first type interpolates between perfect predictions (for $\xi = 0$) and a non-informative situation where all files are predicted to be equally popular (for $\xi = 1$). The second type is similar, but exhibits more variability in the request process. In particular, the parameter π plays a role similar to the parameter ξ with $\pi = 0$ corresponding to perfect requests and $\pi = 1$ to an IRM model where all files are equally popular. Finally, the third type

corresponds to a more adversarial setting where predictions are misleading as the random permutation leads to attribute the actual future number of requests for a given file i to a randomly selected file j.

6.1.3. Online Algorithms

We compare OBC and PCOC presented in Section 4 against classical online algorithms such as LFU, LRU, and OGD [6]. Furthermore, we designed and implemented optimistic version of LFU and LRU.

Optimistic Least Frequently Used (OLFU). The algorithm takes into account predictions for the next requests but updates the cache state at each new requests according to the LFU eviction policy. At the beginning of each batch of requests, OLFU increases the frequency of each file within the predictions for the next batch of R requests. In the face of a new request, the algorithm i) updates the cache state using LFU with the updated frequencies; ii) checks if the file request was in the predicted batch: if it was not, OLFU increases the frequency for that file and decreases the frequency of a random file from the catalog different from the requested one. At the end of each batch the frequencies of OLFU and the ones computed by a classic LFU policy are equal.

Optimistic Least Recently Used (OLRU). This policy considers the predictions for the next R requests and consider the files within the batch as the most recently requested. For each file $i \in \mathcal{N}$, the algorithm keeps a counter, namely *last-time-requested*, indicating the last time file i has been requested. In particular, given a batch of predicted requests, OLRU sets the *last-time-requested* counter of all those predicted files to the current time. In the face of a new request, the algorithm updates the cache using LRU, i.e., evicting the least recently used file from the cache according to the counters updated through the predictions.

6.1.4. Metrics

We evaluate all the algorithms according to three metrics:
i) The *Average Miss Ratio* is a common metric used to evaluate a caching policy. It is the ratio of the number of file misses over the total number of requests up to time-slot t. In terms of the cost function introduced in (1) (we need to set $w_i = 1$ for each $i \in \mathcal{N}$), it can be written as:

$$\frac{1}{Rt}\sum_{s=0}^{t} f_{\mathbf{r}_s}(\mathbf{x}_s) = \frac{1}{Rt}\sum_{s=0}^{t}\sum_{i=1}^{N} r_{s,i}(1 - x_{s,i}).$$

ii) The *Time Average Regret* is the average regret per time-slot, that is the regret experienced up to time-slot t divided by t:

$$\frac{1}{t}\left(\sum_{s=0}^{t} f_{\mathbf{r}_s}(\mathbf{x}_s) - \sum_{s=0}^{t} f_{\mathbf{r}_s}(\mathbf{x}^\star)\right).$$

iii) The *Amortized Cost* is the average time required by each algorithm to update the cache after each new request as measured by running the algorithm on an Intel-Core i7-5650U 2.2 GHz with 8 GB of RAM.

6.2. Results

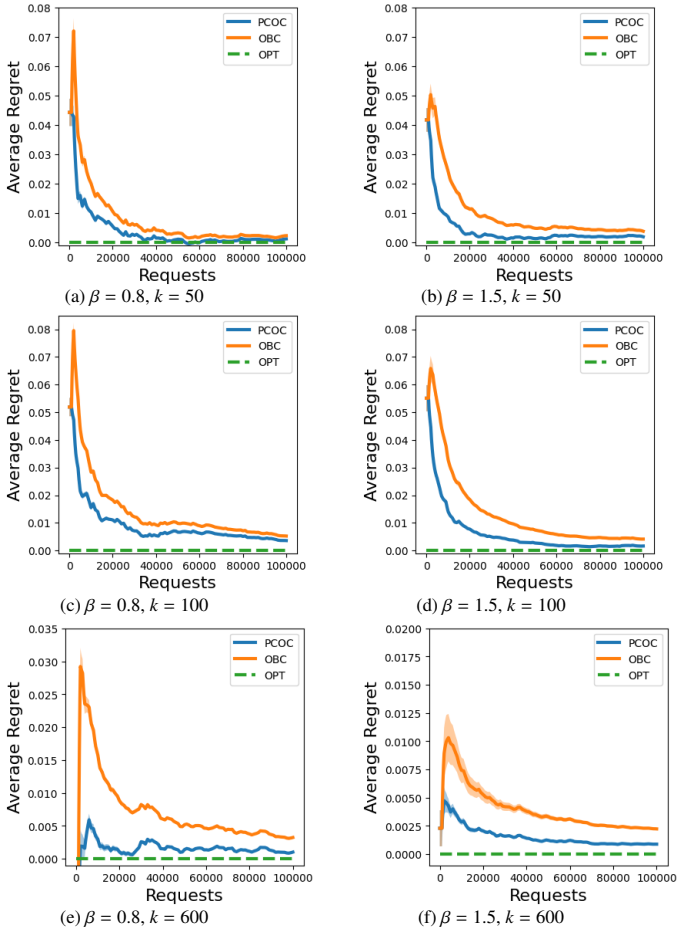
First of all we compare the optimistic versions of LFU and LRU with respect to their classical versions. Afterwards, we focus on the Follow-The-Regularized-Leader-based algorithms evaluating their performance in terms of average regret. Consequently, we compare PCOC with respect to OLFU and classical policies. Finally, we evaluate the optimistic versions of the presented algorithms on the Akamai trace showing also the trade-off between the final missing-ratio and the amortized cost varying the batch size.

OLFU vs. LFU. Figure 2 compares OLFU against LFU for increasing batch sizes and for different accuracy of *Type 2* predictions with files requested according to a Zipf distribution with $\beta = 1.5$, and cache size $k = 100$. We can observe the sensitivity OLFU with respect to the batch size as the predictions become less accurate. Indeed, in case of perfect predictions (Figure 2a), all the versions of OLFU with different batch sizes reach a better miss-ratio than vanilla LFU, as expected. As the accuracy of the predictions decreases, the performance of OLFU starts to deteriorate (Figure 2b) until the case with very inaccurate predictions (Figure 2c), where the versions of OLFU with larger batch size are outperformed by LFU. This is due to the amount of incorrect information brought by the perturbed predictions as the batch size increases. Hence, reasonably, the algorithms' performance deteriorates as the accuracy of the predictions decreases, however, it is worth noting that even in the worst case of Figure 2c, the versions of OLFU with $R = 100$ and $R = 1000$ perform as well as LFU.

OLRU vs. LRU. Figure 3 shows the behaviour of OLRU in the same settings of Figure 2. While OLFU performance improves with the batch size under perfect predictions, OLRU's performance first improves and then degrades (Figure 3a) as the batch size increases. This is due to the fact that the largest the batch, the more the files requested in the batch are equally like to be evicted. In this case, OLRU tends to act as the RANDOM eviction policy [44]. Indeed, the average miss ratio values showed in Figure 3a reflect the ones that can be computed for LRU and RANDOM from equation (1) and the equation before Proposition 4.1 in [44], respectively, with $k = 100$ and $\beta = 1.5$. Then, for less accurate predictions (Figure 3b and Figure 3c), the trend is inverted, i.e., the performance of smaller batch sizes degrades. Reasonably, also in this case, the *Average Miss Ratio* increases as the accuracy of the predictions decreases.

PCOC vs. OBC. We compare the two algorithms for different cache capacities, i.e., $k \in \{50, 100, 600\}$ and different exponents of the Zipf distribution, i.e., $\beta \in \{0.8, 1.5\}$ with $R = 1000$ with predictions of *Type 2* ($\pi = 0.5$). As showed in Figure 4 the difference between the two algorithms becomes significant as the values of α and β increase. This is in line with the results shown in Figure 1 where the difference between the two regrets becomes more evident for higher values of the cache size and the Zipf's exponent. In particular when $k = 600$, i.e., the cache can store at least half of the catalog, PCOC clearly outperforms OBC for all the values of β.

PCOC vs. OLFU. Figure 5 reports on the comparison between PCOC and OLF for different batch sizes and levels of



(a) $\beta = 0.8$, $k = 50$ (b) $\beta = 1.5$, $k = 50$

(c) $\beta = 0.8$, $k = 100$ (d) $\beta = 1.5$, $k = 100$

(e) $\beta = 0.8$, $k = 600$ (f) $\beta = 1.5$, $k = 600$

Figure 4: PCOC vs. OBC

accuracy in predictions of *Type 2*. For all the algorithms we set the initial cache state as $\mathbf{x}_0 := \arg\max_{x \in \mathcal{X}} \{\tilde{\mathbf{r}}_1^\top x\}$, i.e., we entirely store the files with the highest number of requests in the first predicted batch. Also in this case, the quality of the predictions impact the the *Average Miss Ratio* of the policies. We can observe that for high levels of accuracy in the predictions (Figure 5a and Figure 5b) PCOC outperforms OLFU for all the different batches. When the predictions have very low accuracy ($\pi = 0.9$) PCOC shows the same performance of OLFU for $R = 100$, however it still remains competitive reaching the convergence even for higher values of R.

PCOC vs. Classic Policies. Figures 6a and 6b show the performance of PCOC against classical online algorithms in cases where $\beta = 0.9$, and $\beta = 1.2$, with $R = 100$ with predicitons of *Type 1* and *Type 3*. We can notice the benefit of including predictions in the decision process looking at the lower miss ratio of PCOC against LFU. PCOC outperforms LFU even for a noisy factor ξ as large as 0.9 and it is still competitive with LFU when predictions are randomly scrambled. This confirms the advantage of the optimistic nature of such algorithms.

Akamai Trace. Figure 7 shows the performance of PCOC on the Akamai trace for $k = 10$ with predictions of *Type 1*. Figure 7a compares PCOC against OGD, LFU and LRU. The latter two policies take a decision at each file request, whilst PCOC and OGD updates the cache every $R = 10$ requests. Nevertheless, PCOC outperforms the classic policies. Furthermore, even in a non-stationary case, the predictions can help in reducing the miss ratio. Figure 7b shows the comparison between PCOC and OLFU for different batch sizes and with predictions of *Type 2* with $\pi = 0.3$. We can notice how the difference between the two policies becomes more evident in case of real trace even for higher batch sizes for PCOC. Finally, in Figure 7c, we compare different versions of PCOC that updates the local cache every $R \in \{50, 100, 300, 500, 1000\}$ requests. The amortized cost vanishes as the value of R increases (since the number of projections performed in the optimization process diminishes) at the cost of higher miss ratio. However, this confirms the applicability of such a batched method with less frequent updates since both the final miss ratio and the time complexity reached by PCOC with $R = 300$ and $R = 500$ are better than the performance achieved by the most used policies in practice such as LFU and LRU.

7. Conclusions

We presented online optimistic caching algorithms that enjoy sublinear regret in case of batched requests. First we studied the conditions where PCOC results to have a better regret with respect to OBC. Secondly, we showed that the per-coordinate based solution (PCOC) outperforms classic caching policies and their optimistic versions in different conditions. Finally, we showed that, over a real trace, a batched approach presents better performance in terms of final miss ratio and amortized cost compared to classical caching policies.

References

[1] F. Faticanti, G. Neglia, Optimistic Online Caching for Batched Requests, in: IEEE ICC, 2023, pp. 1–6.

[2] T.-C. Chiueh, P. Pradhan, Cache memory design for network processors, in: Proceedings Sixth International Symposium on High-Performance Computer Architecture. HPCA-6 (Cat. No. PR00550), IEEE, 2000, pp. 409–418.

[3] A. J. Smith, Design of CPU cache memories, Computer Science Division, University of California, 1987.

[4] D. Carra, G. Neglia, P. Michiardi, Elastic provisioning of cloud caches: A cost-aware ttl approach, IEEE/ACM Transactions on Networking 28 (3) (2020) 1283–1296.

[5] N. Carlsson, D. Eager, Worst-case bounds and optimized cache on mth request cache insertion policies under elastic conditions, Performance Evaluation 127 (2018) 70–92.

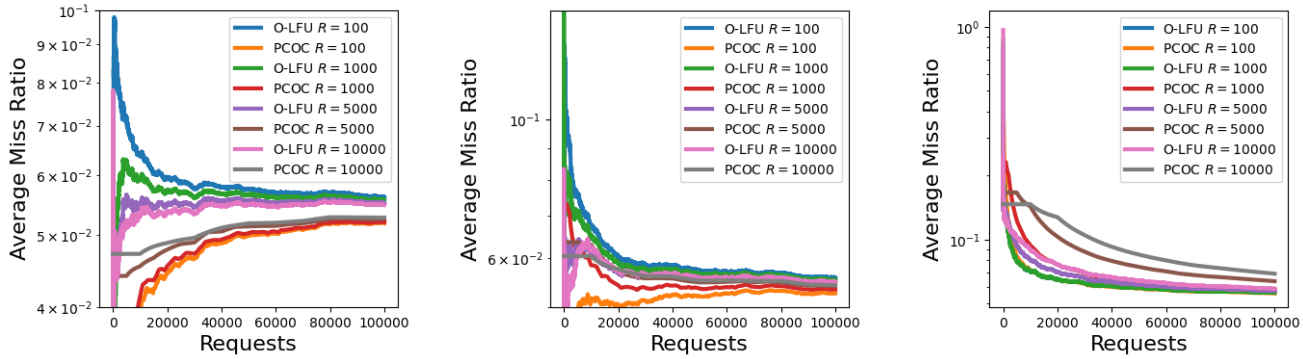
[6] G. S. Paschos, A. Destounis, L. Vigneri, G. Iosifidis, Learning to cache with no regrets, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, IEEE, 2019, pp. 235–243.

[7] R. Bhattacharjee, S. Banerjee, A. Sinha, Fundamental limits on the regret of online network-caching, Proceedings of the ACM on Measurement and Analysis of Computing Systems 4 (2) (2020) 1–31.

[8] Y. Li, T. Si Salem, G. Neglia, S. Ioannidis, Online caching networks with adversarial guarantees, Proceedings of the ACM on Measurement and Analysis of Computing Systems 5 (3) (2021) 1–39.

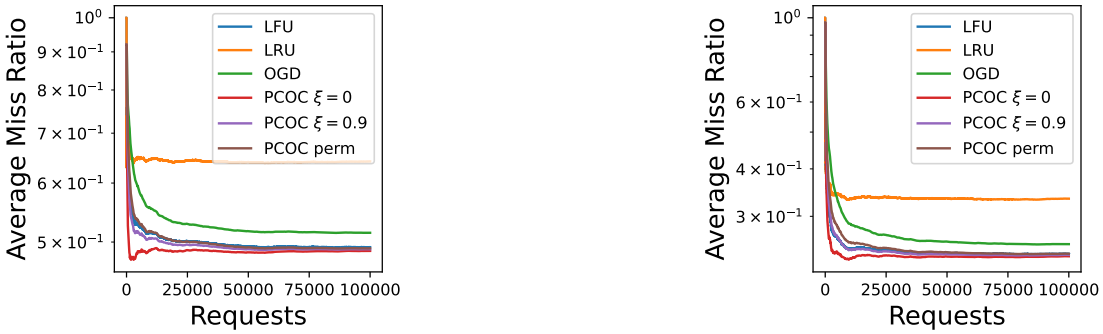
[9] M. Leconte, G. Paschos, L. Gkatzikis, M. Draief, S. Vassilaras, S. Chouvardas, Placing dynamic content in caches with small population, in: IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE, 2016, pp. 1–9.

[10] S. Shalev-Shwartz, Online learning and online convex optimization, Foundations and Trends in Machine Learning 4 (2) (2012).



(a) $\pi = 0$ (b) $\pi = 0.3$ (c) $\pi = 0.9$

Figure 5: Average Miss Ratio of PCOC vs. OLFU



(a) $\beta = 0.9$, $k = 50$ (b) $\beta = 1.2$, $k = 50$

Figure 6: PCOC vs. Classic Policies

[11] T. S. Salem, G. Neglia, S. Ioannidis, No-regret caching via online mirror descent, in: IEEE ICC, 2021, pp. 1–6.

[12] C. A. Gomez-Uribe, N. Hunt, The netflix recommender system: Algorithms, business value, and innovation, ACM TMIS (2015).

[13] S. S. Khanal, P. Prasad, A. Alsadoon, A. Maag, A systematic review: machine learning based recommendation systems for e-learning, Education and Information Technologies 25 (2020) 2635–2664.

[14] M. Mohri, S. Yang, Accelerating online convex optimization via adaptive prediction, in: AISTAS, PMLR, 2016, pp. 848–856.

[15] N. Mhaisen, G. Iosifidis, D. Leith, Online caching with optimistic learning, in: 2022 IFIP Networking, IEEE, 2022, pp. 1–9.

[16] S. Rakhlin, K. Sridharan, Optimization, learning, and games with predictable sequences, NeurIPS 26 (2013).

[17] G. Paschos, G. Iosifidis, G. Caire, et al., Cache optimization models and algorithms, Foundations and Trends® in Communications and Information Theory 16 (3–4) (2020) 156–345.

[18] S. Borst, V. Gupta, A. Walid, Distributed caching algorithms for content distribution networks, in: 2010 Proceedings IEEE INFOCOM, IEEE, 2010, pp. 1–9.

[19] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, G. Caire, Femtocaching: Wireless content delivery through distributed caching helpers, IEEE Transactions on Information Theory 59 (12) (2013) 8402–8413.

[20] K. Poularakis, G. Iosifidis, V. Sourlas, L. Tassiulas, Exploiting caching and multicast for 5g wireless networks, IEEE Transactions on Wireless Communications 15 (4) (2016) 2995–3007.

[21] S. Ioannidis, L. Massoulie, A. Chaintreau, Distributed caching over heterogeneous mobile networks, in: Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems, 2010, pp. 311–322.

[22] S. Ioannidis, E. Yeh, Adaptive caching networks with optimality guarantees, ACM SIGMETRICS Performance Evaluation Review 44 (1) (2016) 113–124.

[23] D. Paria, A. Sinha, Leadcache: Regret-optimal caching in networks, Advances in Neural Information Processing Systems 34 (2021) 4435–4447.

[24] G. S. Paschos, A. Destounis, G. Iosifidis, Online convex optimization for caching networks, IEEE/ACM Transactions on Networking 28 (2) (2020) 625–638.

[25] D. D. Sleator, R. E. Tarjan, Amortized efficiency of list update and paging rules, Communications of the ACM 28 (2) (1985) 202–208.

[26] L. Andrew, S. Barman, K. Ligett, M. Lin, A. Meyerson, A. Roytman, A. Wierman, A tale of two metrics: Simultaneous bounds on competitiveness and regret, in: Conference on Learning Theory, PMLR, 2013, pp. 741–763.

[27] M. Zinkevich, Online convex programming and generalized infinitesimal gradient ascent, in: ICML 2003, 2003, pp. 928–936.

[28] N. Mhaisen, A. Sinha, G. Paschos, G. Iosifidis, Optimistic no-regret algorithms for discrete caching, Proceedings of the ACM on Measurement and Analysis of Computing Systems 6 (3) (2022) 1–28.

[29] K. Chen, L. Huang, Timely-throughput optimal scheduling with prediction, IEEE/ACM Transactions on Networking 26 (6) (2018) 2457–2470.

[30] X. Huang, S. Bian, X. Gao, W. Wu, Z. Shao, Y. Yang, J. C. Lui, Online vnf chaining and predictive scheduling: Optimality and trade-offs, IEEE/ACM Transactions on Networking 29 (4) (2021) 1867–1880.

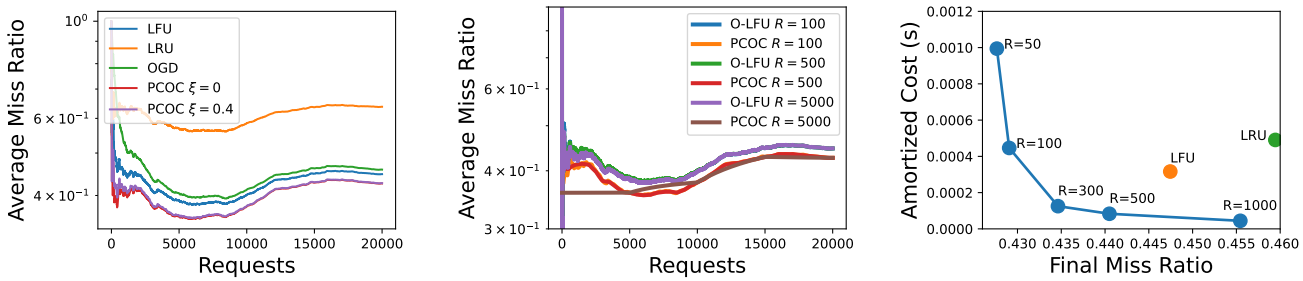
[31] T. Si Salem, G. Neglia, S. Ioannidis, No-regret caching via online mirror descent, ACM Transactions on Modeling and Performance Evaluation of Computing Systems 8 (4) (2023) 1–32.

[32] W. Wang, C. Lu, Projection onto the capped simplex, arXiv preprint arXiv:1503.01002 (2015).

[33] N. Golrezaei, A. F. Molisch, A. G. Dimakis, G. Caire, Femtocaching and device-to-device collaboration: A new architecture for wireless video distribution, IEEE Communications Magazine 51 (4) (2013) 142–149.

[34] V. Fedchenko, G. Neglia, B. Ribeiro, Feedforward neural networks for caching: N enough or too much?, acm sigmetrics performance evaluation review 46 (3) (2019) 139–142.

[35] M. Hashemi, K. Swersky, J. Smith, G. Ayers, H. Litz, J. Chang, C. Kozyrakis, P. Ranganathan, Learning memory access patterns, in: International Conference on Machine Learning, PMLR, 2018, pp. 1919–



(a) $R = 10, k = 10$ (b) $k = 10, \pi = 0.3$ (c) $k = 10, \xi = 0.4$

Figure 7: Akamai Trace

1928.

[36] P. K. Patra, M. Sahu, S. Mohapatra, R. K. Samantray, File access prediction using neural networks, IEEE transactions on neural networks 21 (6) (2010) 869–882.

[37] E. Hazan, Introduction to online convex optimization, Foundations and Trends® in Optimization 2 (3-4) (2016) 157–325.

[38] H. B. McMahan, A survey of algorithms and analysis for adaptive online learning, The Journal of Machine Learning Research (2017).

[39] N. Littlestone, M. K. Warmuth, The weighted majority algorithm, Information and computation 108 (2) (1994) 212–261.

[40] P. Auer, N. Cesa-Bianchi, C. Gentile, Adaptive and self-confident on-line learning algorithms, Journal of Computer and System Sciences 64 (1) (2002) 48–75.

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

[42] C. Fricker, P. Robert, J. Roberts, N. Sbihi, Impact of traffic mix on caching performance in a content-centric network, in: 2012 Proceedings IEEE INFOCOM Workshops, IEEE, 2012, pp. 310–315.

[43] G. Neglia, D. Carra, M. Feng, V. Janardhan, P. Michiardi, D. Tsigkari, Access-time-aware cache algorithms, ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS) 2 (4) (2017) 1–29.

[44] M. Garetto, E. Leonardi, V. Martina, A unified approach to the performance analysis of caching systems, ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS) 1 (3) (2016) 1–28.