Optimistic Online Caching for Batched Requests

Francescomaria Faticanti, Giovanni Neglia

Optimistic Online Caching for Batched Requests

Francescomaria Faticanti
Inria, Université Côte d'Azur

Giovanni Neglia
Inria, Université Côte d'Azur

***Abstract*—In this paper we study online caching problems where predictions of future requests, e.g., provided by a machine learning model, are available. We consider different optimistic caching policies which are based on the Follow-The-Regularized-Leader algorithm and enjoy strong theoretical guarantees in terms of regret. These new policies have a higher computational cost than classic ones like LRU, LFU, as each update of the cache state requires to solve a constrained optimization problem. We study then their performance when the cache is updated less frequently in order to amortize the update cost over time or over multiple requests.**

***Index Terms*—caching, online optimization, predictions, batched requests**

I. Introduction

Caching systems represent one of the most deeply studied research areas that span from the design of CPU hardware to the development of caching services in cloud computing, e.g., elastic caching systems for cloud and edge [1], [2]. The main objective of such systems is to reduce specific costs for the users, the network operator or the caching service 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 [3]–[5]. 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 [6].

Online caching has been studied in the online convex optimization (OCO) framework [7] starting from the work [3]. In this setting, the main objective is to design algorithms that minimize the *regret*, i.e., the difference between the cost incurred by the proposed solution and the cost of the optimal offline static solution that has complete knowledge of future requests over a fixed time horizon. Later contributions analyzed other online learning algorithms [8] and provided new lower bounds on the regret [4].

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 [9], [10]. Online algorithms that rely on such predictions are called *optimistic* [11], [12]. References [11], [13] provide example of optimistic online algorithms based on the Follow-The-Regularized-Leader (FTRL) and Online Mirror Descent (OMD) frameworks [7]. Mhaisen et al. [12] 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 operates on single requests requiring the cache to be updated each time a new file request is received. These updates are computationally very expensive, as they require to solve a constrained optimization problem, and can limit the applicability of online caching policies. To amortize the update cost over time and over multiple requests a *batched* approach can be adopted, where 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 [8]. We stress that the batched approach does not cause any additional delay for the user.

In this paper, we propose 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 [12] and prove that it still enjoys sublinear regret.
2) We introduce a new optimistic batched caching policy based on the per-component-based algorithm in [11].
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 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. Section II introduces the system model and the problem description. In Section III we describe the optimistic caching framework and we present the main algorithms that take into account predictions: the one presented in [12] and the one we propose. Section IV 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 V. Finally, Section VI concludes the paper.

II. System Description and Problem Formulation

A. System Model

We consider the same system's setting described in [8]. The system receives requests for equal-size files in the catalog

This research was supported in part by Inria under the exploratory action MAMMALS

$\mathcal{N} = \{1, 2, \ldots, N\}$. 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. The local cache has finite capacity $k \in \{1, \ldots, N\}$, and it can store arbitrary fractions of files from the catalog as in [3], [12], [14]. 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 $t = 1, \ldots, T$ 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

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

For some results we will rely on the following additional assumption:

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

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 the output of a ML model such as a neural network. Such prediction models can be similar to those used in streaming services like Netflix to provide recommendations to users on the basis of their history view [9]. 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 $\tilde{g}_{t+1,i} = -w_i \tilde{r}_{t+1,i}$.

B. Online Caching Problem

We can fit our caching problem in the *Online Convex Optimization* (OCO) framework [15], [16], 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 $f_{\mathbf{r}_t}(\mathbf{x}_t)$, 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 $((\mathbf{x}_1, r_1), \ldots, (\mathbf{x}_t, r_t))$, 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

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

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 $(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_t, \ldots)$, we denote their aggregate sum up to time t as $\mathbf{y}_{1:t} \triangleq \sum_{s=1}^{t} \mathbf{y}_s$.

III. Optimistic Caching

As highlighted in [12], 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 [11], [17]. The classic *Follow-The-Leader* (FTL) algorithm [18] 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

Algorithm 1: `Optimistic Online Caching`

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

1 **for** $t = 1, \dots, T$ **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 $((\mathbf{x}_1, \mathbf{r}_1), \dots, (\mathbf{x}_t, \mathbf{r}_t))$ according to (3).

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

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

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).

A. Optimistic Bipartite Caching (OBC)

The first algorithm is called *Optimistic Bipartite Caching* (OBC) and was introduced in [12] for a bipartite caching system with a single request at each time-slot. OBC adopts as proximal regularizer

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

with the following parameters

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

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 [11]: 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.

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

B. Per-Coordinate Optimistic Caching (PCOC)

Mohri et al. [11, 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:

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

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]

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

IV. 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.

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

We extend the regret bound in [12, Theorem 1] to the case of batched requests, but we also improve the coefficients taking into account the capacity constraint.

Theorem IV.1. *The regret of OBC is bounded as follows:*

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

Proof. We start from the inequality in [11, Theorem 1],

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

Substituting the regularization functions we obtain

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

In our case, as highlighted in [3], the Euclidean diameter of $\mathcal{X}$ is upper bounded by Δ

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

Hence, introducing the bound in (10), we obtain

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

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

[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.

Theorem IV.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,*

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

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

B. Regret bound of PCOC

The following proof follows the steps in [11, 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 IV.2. *The regret of PCOC is bounded as follows*

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

Proof. From [11, Theorem3], applying the regularization function defined in (6) and the norms defined in (7), we obtain

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

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 [19, Lemma 3.5] to $\sum_{t=1}^{T} \|\mathbf{g}_t - \tilde{\mathbf{g}}\|_{(t),\star}^2$ 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,*

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

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

C. 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 [20], the algorithm is executed every time unit, and per-file costs equal 1. In this case, $g_{t,i} \sim Poisson(\lambda p_i)$ 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 OBC bound, we obtain

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

For the PCOC bound, we obtain

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

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.

D. 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 IV.1 (the same analysis can be carried out on the bound in Theorem IV.2). We consider a time interval of duration Θ time units and that requests are batched every τ time units. We analyse the expected value of such bound and compare it against a general batched-requests approach where the caching decisions are taken every τ among an overall time interval of

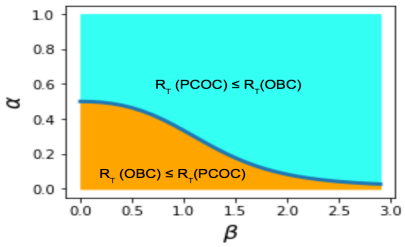


Fig. 1: OBC vs PCOC, different regimes for the regret as a function of the Zipf exponent (β) and the relative cache size ($k = \alpha N$).

Θ time units where a single requests is available at each time. Looking at the expected value of the regret bound we have:

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

In this case we have $g_{t,i} \sim Poisson(\lambda_i \tau)$. For the predictions $\tilde{g}_{t,i}$ we consider two options: i) they can 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, let $C = 2\sqrt{2\min\{k, N-k\}}$. We have

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

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

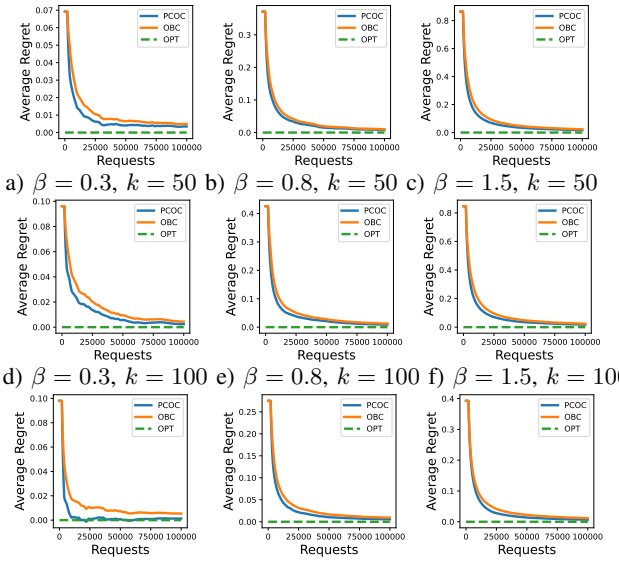
In the second case, for $t > 1$, $\tilde{g}_{t,i} = g_{t-1,i} = n_{t,i}(\tau) \sim Poisson(\lambda_i\tau)$, where $n_{t,i}(\tau)$ is the number of arrivals within the interval $[(t-2)\tau, (t-1)\tau]$. The initial prediction is given by $\tilde{g}_{1,i} = \frac{n_i(\tau_0)}{\tau_0}$, where τ_0 is a first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

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

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 $m^2 = \frac{\lambda_i\tau_0}{\tau_0^2}$. 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 $\tau = \frac{\tau_0}{2}$.



a) $\beta = 0.3$, $k = 50$ b) $\beta = 0.8$, $k = 50$ c) $\beta = 1.5$, $k = 50$
d) $\beta = 0.3$, $k = 100$ e) $\beta = 0.8$, $k = 100$ f) $\beta = 1.5$, $k = 100$
g) $\beta = 0.3$, $k = 600$ h) $\beta = 0.8$, $k = 600$ i) $\beta = 1.5$, $k = 600$

Fig. 2: PCOC vs. OBC

V. Numerical Results

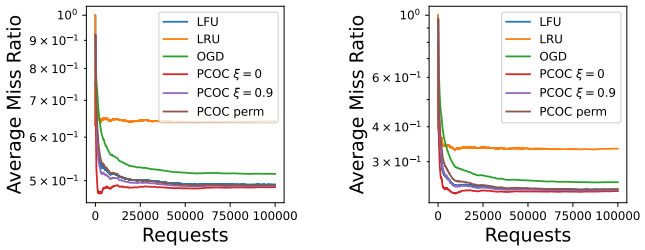
A. Experimental Settings

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 $\beta \in \{0.3, 0.8, 0.9, 1.2, 1.5\}$ from a catalog of $N = 1000$ files. 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 $R \in \{100, 2000\}$ 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 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 [21]. 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.

Predictions. We implemented two kinds of predictions: i) the first ones are generated according to $\tilde{\mathbf{g}}_t = (1-\xi)\mathbf{g}_t + \xi\frac{R}{N}$, with $\xi \in [0, 1]$; ii) the second ones are generated as random permutations of the original gradients. The first case interpolates between perfect predictions (for $\xi = 0$) and a situation where all files appear equally popular (for $\xi = 1$). In the second case, files' future popularities are arbitrarily ranked.

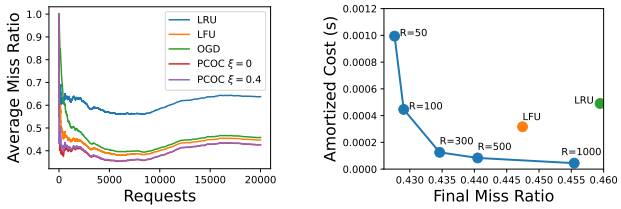
Metrics. We compare all the algorithms with respect three metrics: i) the *Average Miss Ratio*, i.e., the total cost over the first t iterations, normalized by Rt; ii) the *Time Average Regret* over the first t iterations, and iii) the *Amortized Cost*, i.e., the average computational time per request.

Baselines. We compare OBC and PCOC presented in Section III against classical online algorithms such as LFU, LRU, and OGD [3].



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

Fig. 3: PCOC vs. Classic Policies



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

Fig. 4: Akamai Trace

B. Results

PCOC vs. OBC. We compare the two algorithms for different capacities, i.e., $k \in \{50, 100, 600\}$ and different exponents of the Zipf distribution, i.e., $\beta \in \{0.3, 0.8, 1.5\}$ with $R = 2000$. As showed in Figure 2, the gap between the performance of the two solutions is broader when $\beta = 0.3$, and the difference between the two algorithms becomes significant for higher values of α. This confirms the results of Figure 1 where the difference between the two regrets becomes more evident as the cache size increases. 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. Classic Policies. Figures 3a and 3b show the performance of PCOC against classical online algorithms in cases where $\beta = 0.9$, and $\beta = 1.2$, with $R = 100$. 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 4 shows the performance of PCOC on the Akamai trace. Figure 4a) 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. Finally, in Figure 4b), we compare different versions of PCOC that updates the local cache every $R \in \{50, 100, 300, 500, 1000\}$ requests. In this experiment $k = 10$. The amortized cost vanishes as the value of R increases (since the number of projections performed in the optimization process diminishes) at a cost of higher miss ratio. However, this confirms the applicability of such a batched method with less frequent updates since 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.

VI. 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-component based solution (PCOC) outperforms classic caching policies 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. Future works will study optimistic versions of other online algorithms, e.g., the Online Mirror Descent, applied to the caching framework.

References

[1] D. Carra, G. Neglia, and P. Michiardi, "Elastic provisioning of cloud caches: A cost-aware ttl approach," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1283–1296, 2020.

[2] N. Carlsson and D. Eager, "Worst-case bounds and optimized cache on mth request cache insertion policies under elastic conditions," *Performance Evaluation*, vol. 127, pp. 70–92, 2018.

[3] G. S. Paschos *et al.*, "Learning to cache with no regrets," in *IEEE INFOCOM*, 2019, pp. 235–243.

[4] R. Bhattacharjee *et al.*, "Fundamental limits on the regret of online network-caching," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 4, no. 2, pp. 1–31, 2020.

[5] Y. Li, T. Si Salem, G. Neglia, and S. Ioannidis, "Online caching networks with adversarial guarantees," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 5, no. 3, pp. 1–39, 2021.

[6] M. Leconte *et al.*, "Placing dynamic content in caches with small population," in *IEEE INFOCOM*, 2016, pp. 1–9.

[7] S. Shalev-Shwartz *et al.*, "Online learning and online convex optimization," *Foundations and Trends in Machine Learning*, vol. 4, no. 2, 2012.

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

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

[10] S. S. Khanal *et al.*, "A systematic review: machine learning based recommendation systems for e-learning," *Education and Information Technologies*, vol. 25, no. 4, 2020.

[11] M. Mohri and S. Yang, "Accelerating online convex optimization via adaptive prediction," in *AISTAS*. PMLR, 2016, pp. 848–856.

[12] N. Mhaisen, G. Iosifidis, and D. Leith, "Online caching with optimistic learning," *arXiv preprint arXiv:2202.10590*, 2022.

[13] S. Rakhlin and K. Sridharan, "Optimization, learning, and games with predictable sequences," *NeurIPS*, vol. 26, 2013.

[14] N. Golrezaei *et al.*, "Femtocaching and device-to-device collaboration: A new architecture for wireless video distribution," *IEEE Communications Magazine*, vol. 51, no. 4, pp. 142–149, 2013.

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

[16] E. Hazan *et al.*, "Introduction to online convex optimization," *Foundations and Trends® in Optimization*, vol. 2, no. 3-4, pp. 157–325, 2016.

[17] H. B. McMahan, "A survey of algorithms and analysis for adaptive online learning," *The Journal of Machine Learning Research*, 2017.

[18] N. Littlestone and M. K. Warmuth, "The weighted majority algorithm," *Information and computation*, vol. 108, no. 2, pp. 212–261, 1994.

[19] P. Auer, N. Cesa-Bianchi, and C. Gentile, "Adaptive and self-confident on-line learning algorithms," *Journal of Computer and System Sciences*, vol. 64, no. 1, pp. 48–75, 2002.

[20] R. Fagin, "Asymptotic miss ratios over independent references," *Journal of Computer and System Sciences*, vol. 14, no. 2, pp. 222–250, 1977.

[21] G. Neglia *et al.*, "Access-time-aware cache algorithms," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, vol. 2, no. 4, pp. 1–29, 2017.