Learning to Caching Under the Partial-feedback Regime

Qingsong Liu* and Yaoyu Zhang*

***Abstract*—We consider the caching problem in an online learning perspective, i.e., no model assumptions and prior knowledge for the file request sequence. Our goal is to design an efficient online caching policy with minimal regret, i.e, minimizing the total number of cache-miss with respect to the best static configuration in hindsight. Previous studies such as Follow-The-Perturbed-Leader (FTPL) caching policy, have provided some near-optimal results, but their theoretical performance guarantees only valid for the regime wherein all arrival requests could be seen by the cache, which is not the case in some practical scenarios like caching at cellular base station, content dissemination via DNS, etc. Hence our work study the partial-feedback regime wherein only requests for currently cached files are seen by the cache, which is more challenging and has not been studied before in the online learning perspective. We propose an online caching policy combining the FTPL with a novel popularity estimation procedure called Geometric Resampling (GR), and show that it yields the first sublinear regret guarantee in this regime. We also conduct numerical experiments to validate the theoretical guarantees of our caching policy.**

***Index Terms*—Online learning, Online caching policies, Quality of service (QoS)**

I. Introduction

With the perpetual growth of Internet traffic fueled by new services such as AR/VR [8], caching policies that learn fast to maximize cache hits can mitigate the increasing costs of information transportation and improve the quality of service (QoS) [7]. The core idea of caching has been widely-adopted in many diverse scenarios such as improving CPU paging performance through L1/L2 caches [12], implementing Web-caching through Content Distribution Networks [2], [27], [28], and realizing low-latency wireless video transmission through Femtocaching [31].

There is a variety of caching policies in the literature. When the entire file request sequence is known in advance, the MIN policy [34] is an optimal offline caching policy. Among the online policies, the Least Frequently Used policy (LFU) [17], the Least Recently Used policy (LRU) [14], [17], the FIFO policy [9], and the Least Recently Sent (LRS) policy [25] have been studied extensively. [14] analyze the performance of the LRU policy for stationary/stochastic file request sequence (i.e., the file requests are i.i.d. or the file popularity is stationary). [25] showed that their LRS policy could outperform the LRU policy under a Markovian assumption of the request sequence. [11] introduced a unified framework to analyze these popular caching policies again under the stationary request sequence. And [19] shows that some of them augmented with a machine-learned oracle could guarantee low competitive ratios. On a different line of work, the studies [15], [16], [20], [21] derived information-theoretic lower bounds and efficient caching policies to facilitate the delivery of cached files to users in a bandwidth-efficient manner. Depending on the applicable scenarios, these caching policies could be divided into two categories: coded-caching policies (e.g, LRS) and uncoded-caching policies (LRU, LFU, FIFO, etc). In uncoded-caching, complete files are cached. In coded-caching, the original files are first encoded via a class of rateless erasure codes, e.g., Raptor code [18], [33], and then some of the resulting coded symbols are cached. Our work focus on the uncoded-caching problem since it is more common in actual scenarios, while code-caching is mostly used in video applications [35].

Despite there are many caching policies developed in the literature. The performance guarantee of almost them, however, depends largely on some prior assumptions (or knowledge) about the generative model of the file request sequence [30]. While in practice, the statistical characteristics of the file popularity are usually unknown in advance [24] and even time-varying /non-stationary due to the frequent addition of new content to the library, mobility of the users, etc. Thus, even though many different caching policies are well-adopted today, the performance of caching strategies in an online learning perspective has not been fully studied to date. This prompts us to study the caching problem from an online learning perspective, i.e., with no assumptions and prior statistical knowledge on the file request sequence.

There are several works on caching problem along the line of online learning. For uncoded-caching in an online learning point-of-view, [24] invested several classical caching policies including LRU, LFU, and FIFO and proven that they perform poorly (suffer from linear ($O(T)$) regrets) when there is no obvious statistical characteristics on file request sequence or the file request popularity is non-stationary/time-varying, i.e., non-stationary request model. The authors in [4] proposed an online caching policy called Follow-The-Perturbed-Leader (FTPL), and showed that it can guarantee an $O(\sqrt{T})$ regret for non-stationary request model. [22] extends FTPL and the corresponding analysis into the setting that taking into account the switching costs and also obtained the regret bound of $O(\sqrt{T})$. Of cause, their results also provide the worst-case regret guarantee for stochastic/stationary request model, since it is a special case of non-stationary request model. There is also another line of research [10], [23], [24], [29] that studies the coded-caching in the online learning perspective.

*Both authors contributed equally to this work, and are with the Institute for Interdisciplinary Information Sciences, Tsinghua University, China. Contacts {liu-qs19, yaoyu-zh19}@mails.tsinghua.edu.cn.

It is worth noting that all the studies have been listed including well-known classic caching policies only valid for the regime wherein all requests would send to the cache whether or not the requested file is in the cache, i.e., full-feedback regime. However, this is not the case in some practical scenarios wherein the cache can only see the requests for files currently cached. For example, when routing requests via DNS redirect, the cache only see the requests which are directed towards it. We refer to this case as the partial-feedback regime. Besides, in some typical cache networks consisting of multiple caches, since the request information is either not aggregated or only partially, each cache does not see the hits and misses of the other caches in real time, corresponding to the partial-feedback regime. Applications of the partial-feedback regime include caching at cellular base stations [5], content dissemination using a Global Name Service [32] (e.g., DNS), etc. To the best of our knowledge, there is no work on the caching problem from an online-learning point-of-view under the partial-feedback regime. The above discussions inspire us to put forward the following question: *can we design an efficient online caching policy that works provably (near-)optimal for the partial-feedback regime?*

A. Contributions

In the process of answering the above question, our contributions are summarized as follows:

- Under the partial-feedback regime, we propose an online caching policy called FTPL+GR, as it combines the FTPL policy with a novel popularity estimation procedure: Geometric Resampling (GR), and show that it guarantees a sublinear regret for any request model. To the best of our knowledge, we are the first to study the online caching problem under the partial-feedback regime from an online learning point-of-view. We also show that FTPL+GR does not sacrifice much in running time compared to FTPL.
- We provide a thorough theoretical analysis for FTPL+GR. We presents various important properties of our estimation procedure that is independent of caching policies, which may be independently interesting.
- We also conduct numerical experiments to validate the theoretical guarantees of FTPL+GR.

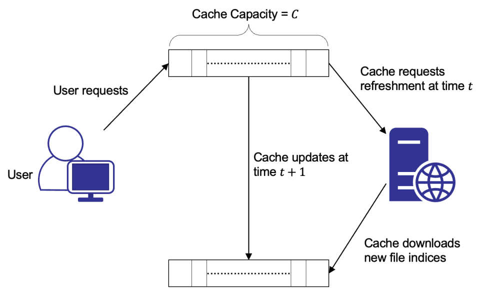
Cache Capacity = C
User requests
Cache requests refreshment at time t
User
Cache updates at time $t+1$
Cache downloads new file indices

Fig. 1. The setup of online caching

II. System Model

Here we describe a simplified abstraction of a caching system. Assume that a set of N unique files (with equal size) is available in a remote server. A local cache of limited storage capacity can hold up to C files at a time where $C < N$. In many real-world scenarios, the capacity of cache is much smaller than the total library size (i.e., $C \ll N$). Time is slotted such that at most one file is requested from the group of users at a time [1]. We use the vector $\boldsymbol{x}(t) \in \{0,1\}^N$ to represent the file request at time t, where $x_f(t) = 1$ if and only if the f-th file is requested by users at time t, i.e., one-hot encoding.

In our paper, we do not make any statistical assumptions or have any prior knowledge on the file request sequence $\{\boldsymbol{x}(t)\}$. Thus, the file request sequence may be generated from the stationary model or non-stationary model which is typical for internet traffic with transient content popularity.

An online caching policy caches C files before the request for that slot arrives. We use the vector $\boldsymbol{y}(t) \in \{0,1\}^N$ to represent the cache configuration at time t, whose i-th component $y_i(t)$ denotes whether the i-th file is cached at time t or not. $\boldsymbol{y}(t)$ may be randomized selected and may depend on the caching configurations and file requests up to time $t-1$. The set of all admissible caching configurations $\mathcal{Y}$, which respects the capacity constraint, is given below:

$$\mathcal{Y} = \{\boldsymbol{y} \in \{0,1\}^N, \ ||\boldsymbol{y}||_1 \leq C\}. \tag{1}$$

At any time t, if the requested file is present in the cache (cache-hit), i.e., $\langle \boldsymbol{x}(t), \boldsymbol{y}(t) \rangle = 1$, the request is promptly served by the cache. Otherwise, the request is forwarded to the remote server and accrues cache-miss. We aim to design a online caching policy that maximizes the expected accumulative cache-hits over the time-horizon T, i.e.,

$$\mathbf{E}[\sum_{t=1}^{T} \langle \boldsymbol{y}(t), \boldsymbol{x}(t) \rangle], \tag{2}$$

where the expectation is taken w.r.t. the randomness of the requests and policy's internal randomness. See Figure 1 for a scheme. As stated before, in most real scenarios the popularity distribution of requests is unknown to the caching policy a priori and even time-varying. So the caching policy is aiming to learn and track the popularity distribution (or part of it) from the sequential observations, and cache files by wisely using the available information at each time in order to maximize the expected cumulative hits.

In the online learning literature, it is standard to measure the performance of an online policy in terms of regret, which is defined as the difference between the total cache-hits experienced by the online policy over a time horizon T and that of the best fixed policy with complete information. Mathematically,

$$\text{Regret}(T) = \max_{\boldsymbol{y} \in \mathcal{Y}} \sum_{t=1}^{T} \langle \boldsymbol{y}, \boldsymbol{x}(t) \rangle - \mathbf{E}[\sum_{t=1}^{T} \langle \boldsymbol{y}(t), \boldsymbol{x}(t) \rangle]. \tag{3}$$

Note that minimizing the regret (3) is equivalent to maximizing (2), i.e., accumulated cache-hits. And our goal is to find a caching policy that ensures the average performance gap with respect to the best fixed configuration with hindsight diminish as T grows, i.e., $\text{Regret}(T)/T \to 0$. This implies that the online caching policy achieves a sublinear regret $o(T)$ and learns to adapt the cache configuration without any prior knowledge about the request model.

[1]Our algorithms design and the corresponding analysis can be easily adapted to the case with at most K requests.

Our paper considers the partial-feedback regime that has many scenarios in computer systems and communication networks. We mathematically show the difference between it and full-feedback regime in the following, and illustrate them conceptually in the Figure 2.

- **Full-feedback regime**: At each time t, the cache is able to observe the file request $\boldsymbol{x}(t)$, regardless of the choice of $\boldsymbol{y}(t)$. This regime corresponds to all requests being sent to the cache.
- **Partial-feedback regime**: In this case, the caching policy could observe the request only in the case that cache-hit happens, i.e., only if the requested item is cached already. Mathematically, the cache can only observe the vector $(x_1(t)y_1(t), ..., x_N(t)y_N(t))$ at time t. This regime corresponds to the context of information centric caching, wherein requests are forwarded to the cache only if the corresponding content is cached.

In this sequel, we develop an online caching policy based on the FTPL framework for the partial-feedback regime. To address the challenge of unobservable file request when cache-miss happens, we combine the FTPL policy with a novel popularity estimation procedure called Geometric Resampling (GR), and confirm its sublinear regret guarantee for any request models.

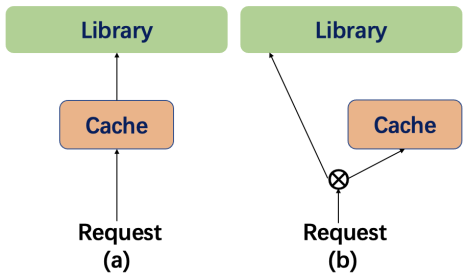

Fig. 2. (a) Request forwarding in the full-feedback regime; (b) Request forwarding in the partial-feedback regime.

III. Overview of our caching policy

Our online caching policy builds on the framework of FTPL, which is the state-of-the-art online universal caching policy for full-feedback regime. To facilitate our caching policy design, we first introduce the FTPL caching policy in details.

A. Preliminary: FTPL caching policy

At each time t, the Follow the Perturbed Leader (FTPL) caching policy, first introduced in [4], caches C files according to the following principle:

$$\boldsymbol{y}(t) = \arg\max_{\boldsymbol{y}\in\mathcal{Y}} \langle \boldsymbol{y}, \boldsymbol{X}(t) + \eta_t \boldsymbol{\gamma}(t)\rangle, \tag{4}$$

where $\boldsymbol{X}(t) = \sum_{\tau=0}^{t-1} \boldsymbol{x}(\tau)$ is the cumulative count vector until time t, $\boldsymbol{\gamma}(t)$ is a sampled vector of standard Gaussian random variables at time t, and $\eta_t = k\sqrt{t}$ is the learning rate. Specifically, at every time t, FTPL policy adds a scaled version of the sampled random variables to the current cumulative count vector $\boldsymbol{X}(t)$ to obtain a perturbed cumulative count vector, then caches t files having the highest perturbed positive cumulative count at time t. Note that the update rule of FTPL caching policy could be rewritten as

$$\boldsymbol{y}(t) = \arg\max_{\boldsymbol{y}\in\mathcal{Y}} \langle \boldsymbol{y}, \boldsymbol{X}(t) + \eta_t \boldsymbol{\gamma}(t)\rangle = \arg\max_{\boldsymbol{y}\in\mathcal{Y}} \langle \boldsymbol{y}, \frac{\boldsymbol{X}(t) + \eta_t \boldsymbol{\gamma}(t)}{t}\rangle$$
$$= \arg\max_{\boldsymbol{y}\in\mathcal{Y}} \langle \boldsymbol{y}, \frac{1}{t}\sum_{\tau=0}^{t-1} \boldsymbol{x}(\tau) + \frac{k\sqrt{t}}{t}\boldsymbol{\gamma}(t)\rangle$$
$$= \arg\max_{\leq C} (\hat{\mu}_1(t), ..., \hat{\mu}_N(t)),$$

where $\hat{\mu}_i(t) = \frac{1}{t}\sum_{\tau=0}^{t-1} \mathbf{I}\{x_i(\tau) = 1\} + k\frac{1}{\sqrt{t}}\gamma_i(t)$. Thus, FTPL policy could also be viewed as selecting files with the Top-C highest perturbed empirical estimate of popularity (frequency) at each time, while LFU loads the C most frequent files without adding perturbation. Intuitively, LFU cannot adapt to the non-stationary request sequence as it does not add any perturbation to the frequency estimate, i.e., just do exploitation without exploration. The perturbation term makes FTPL do exploration to adapt into the non-stationary request sequence, and gradually decreases over time to make FTPL behave as closely to LFU in the long run, i.e., be optimistic in the case of stationary request sequence.

We remark that the computational-complexity of FTPL is essentially the same as choosing the top-C elements with maximum compound value. Therefore, the worst-case running time of FTPL is $O(N\log N) = \tilde{O}(N)$.

B. FTPL with Geometric Resampling (GR)

In this subsection, we present our caching policy under the partial-feedback regime. As described before, in this regime the caching policy can only access the vector $(x_1(t)y_1(t), ..., x_N(t)y_N(t))$ at time t. To address this challenge, our main idea is to perform an estimation of file request in the event of cache-miss to learn and track the file popularity better in order to improve the frequency of cache-hits. Specifically, we use some estimators to estimate $\boldsymbol{x}(t)$ when it is not available to the cache. This process is in sharp contrast to the full-feedback regime in which the caching policy could improve the popularity estimate of each file after each time due to full visibility of all the requests. Intuitively, the inaccurate estimation is costly as the caching policy would place sub-optimal files in the cache and incur more regret.

Before introducing our estimator, we remark that FTPL is essentially a randomized algorithm mapping histories to probability distributions over all cache configurations, whose randomness is brought by the random perturbation terms. Hence, at each time t, the FTPL policy could be viewed as the procedure of specifying a distribution $\boldsymbol{p}_t$ over all cache configurations in $\mathcal{Y}$ and selecting one cache configuration $\boldsymbol{S}$ according to $\boldsymbol{p}_t$, i.e., $p_t(\boldsymbol{S}) = \Pr\{\boldsymbol{y}(t) = \boldsymbol{S}|\mathcal{F}_{t-1}\}$, where $\mathcal{F}_{t-1}$ is the history of the caches's observations and used perturbation vectors up to time $t-1$. And we have that $p_{t,i} = \Pr\{y_i(t) = 1|\mathcal{F}_{t-1}\}$. When the request incurs a cache-miss, i.e., $\boldsymbol{x}(t)$ is not observable, it is commonplace to consider importance-weighted estimator to estimate it of the form:

$$\hat{x}_i(t) = \frac{x_i(t)}{p_{t,i}}\mathbf{I}\{y_i(t) = 1\}, \ \forall i \in [N]. \tag{5}$$

It is very easy to show that $\hat{x}_i(t)$ is an unbiased estimate of $x_i(t)$, i.e., $\mathrm{E}[\hat{x}_i(t)|\mathcal{F}_{t-1}] = x_i(t)$ when $p_{t,i} > 0$, otherwise

Algorithm 1 Geometric Resampling (GR) procedure

```
1: The policy draws $\boldsymbol{y}(t) \sim \boldsymbol{p}_t$
2: for $k = 1...M$ do
3:     draw $\boldsymbol{y}^k(t) \sim \boldsymbol{p}_t$
4:     for $i = 1...N$ do
5:         $K_{t,i} = \min(\{k : y_i^k(t) = 1\})$
6:         $//K_{t,i} = \min(\{k : y_i^k(t) = 1\} \cup \{M\})$ (capping)
7:     end for
8: end for
```

$\mathrm{E}[\hat{x}_i(t)|\mathcal{F}_{t-1}] = 0 \leq x_i(t)$. If the probabilities $p_{t,i}$ are readily available, the estimates (5) can be computed efficiently. However, this is not the case for FTPL. In particular, the probabilities $p_{t,i}$ cannot be specified implicitly (i.e., cannot be expressed in closed form), and thus the importance weights are not readily available for FTPL.

To overcome this difficulty, we propose a novel estimation procedure called Geometric Resampling, which is the most critical element in our technique. Using this procedure, we can compute $\hat{x}_i(t)$ efficiently even when $p_{t,i}$ is not available to the cache. Our estimation procedure dubbed GR is based on the simple observation that, even though $p_{t,i}$ might not be computable in closed form, one can simply generate a geometric random variable with expectation $1/p_{t,i}$ by repeated sampling from $\boldsymbol{p}_t$. Specifically, we propose the procedure in Algorithm 1 to be executed in time t.

Observe that $K_{t,i}$ generated in this way is a geometrically distributed random variable given $\boldsymbol{y}(t)$ and $\mathcal{F}_{t-1}$. Consequently, we have that $\mathrm{E}[K_{t,i}|\mathcal{F}_{t-1}, \boldsymbol{y}(t)] = 1/p_{t,i}$. We use this property to construct the estimates as follows,

$$\hat{x}_i(t) = x_i(t)\mathbf{I}\{y_i(t) = 1\}K_{t,i}, \ \forall i \in [N]. \tag{6}$$

We can easily show that (6) is still an unbiased estimate of $x_i(t)$ whenever $p_{t,i} > 0$ since

$$\begin{aligned}\mathrm{E}[\hat{x}_i(t)|\mathcal{F}_{t-1}] &= \sum_n p_{t,n}\mathrm{E}[\hat{x}_i(t)|\mathcal{F}_{t-1}, y_n(t) = 1] \\ &= p_{t,i}\mathrm{E}[x_i(t)K_{t,i}|\mathcal{F}_{t-1}, y_i(t) = 1] \\ &= p_{t,i}x_i(t)\mathrm{E}[K_{t,i}|\mathcal{F}_{t-1}, y_i(t) = 1] = x_i(t),\end{aligned}$$

and (6) produces $\hat{x}_i(t) = 0$ whenever $p_{t,i} = 0$, giving $\mathrm{E}[\hat{x}_i(t)|\mathcal{F}_{t-1}]$ for such i and t. Besides, the above estimation procedure indeed applies to the partial-feedback regime for the reason that the estimates (6) could be rewritten as

$$\hat{x}_i(t) = [x_i(t)y_i(t)]K_{t,i}, \ \forall i \in [N], \tag{7}$$

which matches the case that the cache can only observes the vector $(x_1(t)y_1(t), ..., x_N(t)y_N(t))$ in this regime. One practical concern with the above sampling procedure is that its worst-case running time is unbounded. Specifically, the actual number of samples might be much larger. To address this concern, we cap off the number of samples at some finite $M > 0$ (See line 6 in Algorithm 1). While this capping obviously introduces some bias, we will show later that for appropriate values of M, this bias does not hurt the performance too much, i.e., does not change the order of regret guarantee.

Now we illustrate the FTPL+GR, Follow-the-Perturbed-Leader with Geometric Resampling, in Algorithm 2. As we

Algorithm 2 FTPL with GR. Denote $\boldsymbol{a} \circ \boldsymbol{b}$ as the element-wise product of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$: $(\boldsymbol{a} \circ b)_i = a_i b_i$.

```
1: Cache capacity $C$, Geometric Resampling's capping $M$
2: Initialize: $\boldsymbol{X}(1) \leftarrow 0$
3: for round $t = 1...T$ do
4:     Sample: $\boldsymbol{\gamma}(t) \sim Exp(1)$
5:     $\eta_t \leftarrow \sqrt{t}$
6:     $\boldsymbol{y}(t) \leftarrow \arg\max_{\boldsymbol{y}\in\mathcal{Y}} \langle \boldsymbol{y}, \boldsymbol{X}(t) + \eta_t\boldsymbol{\gamma}(t)\rangle$
7:     Cache files according to $\boldsymbol{y}(t)$
8:     $\boldsymbol{K} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{y}(t)$
9:     for $n = 1, \cdots, M$ do
10:        $\boldsymbol{K} = \boldsymbol{K} + \boldsymbol{r}$
11:        Sample: $\boldsymbol{\gamma}' \sim Exp(1)$
12:        $\boldsymbol{y}' \leftarrow \arg\max_{\boldsymbol{y}\in\mathcal{Y}} \langle \boldsymbol{y}, \boldsymbol{X}(t) + \eta_t\boldsymbol{\gamma}'\rangle$
13:        $\boldsymbol{r} = \boldsymbol{r} \circ \boldsymbol{y}'$
14:        if $\boldsymbol{r} = \boldsymbol{0}$ then
15:            break
16:        end if
17:    end for
18: end for
19: User requests a file corresponding to request vector $\boldsymbol{x}(t)$
20: $\boldsymbol{X}(t+1) \leftarrow \boldsymbol{X}(t) + \boldsymbol{K} \circ \boldsymbol{x}(t) \circ \boldsymbol{y}(t)$
```

mentioned before, the distribution $\boldsymbol{p}_t$, while implicitly specified by $\boldsymbol{\gamma}(t)$ and $\boldsymbol{X}(t)$, cannot normally be expressed in closed form for FTPL. However, sampling the cache configurations can be carried out by drawing additional perturbation vectors independently from the same distribution as $\boldsymbol{\gamma}(t)$ and then solving a Top-C selecting problem. We emphasize that the above additional cache configurations are never actually adopted by the policy, but are only necessary for constructing the estimates.

Perturbation term modification. It is worthing note that in standard FTPL policy (4), the cache usually loads less than C files, as the perturbed cumulative counts may be negative caused by the Gaussian distributed perturbation. This inadequate caching would lead to potential performance degradation. To overcome this issue, in our FTPL+GR policy, we adopt the standard Exponentially distributed instead of Gaussian distributed sample vector as the perturbation term in FTPL-component, which is an another novelty of our policy. The motivation behind this is mainly technical and is simply meant to ensure the positivity of the perturbed cumulative counts and the full of cache, while maintaining the same property of exponential decline as the Gaussian distribution. We refer the FTPL policy that uses the Gaussian distributed perturbation and Exponentially distributed perturbation as FTPL(G) and FTPL(E), respectively. In the later, we empirically show that FTPL(E) indeed has a better cache-performance than FTPL(G).

IV. Performance guarantees of FTPL(E)+GR

Now we are ready to state the theoretical performances of FTPL(E)+GR. First, we give the regret guarantee of it for any request model (the-worst-case).

Theorem 1 *Our caching policy FTPL(E) with GR ensures that:*

$$\textit{Regret}(T) \le \sqrt{T}(C + 4CN + C\log N/C) + \frac{NT}{eM}. \quad (8)$$

In particular, with $M = O(\sqrt{T})$ we have

$$\textit{Regret}(T) \le \sqrt{T}(C + 4CN + C\log N/C + N/e). \quad (9)$$

Based on the above result, we can see that our FTPL(E)+GR caching policy achieves a sublinear regret guarantee of the order $O(\sqrt{T})$ in the worst-case. We note that the bigger M is, the smaller regret is, which is intuitive as the estimate bias decreases when M increases.

Running Time of FTPL(E)+GR. Let us now turn our attention to computational issues. As mentioned before, the running time of FTPL, i.e., solving the combinatorial optimization problem of finding (exactly the Top-C selecting problem) is at most $O(N\log N)$. Naturally, our estimation procedure multiplies these calculations by the number of samples taken in each round. While terminating the estimation procedure after M samples helps in controlling the running time with high-probability, observe that the naive bound of MT on the number of samples becomes way too large when setting M as suggested by Theorem 1. The next theorem shows that the expected amortized samples taken by our estimation procedure remains as low as $O(N)$ even for large values of M.

Theorem 2 *Let N_t denote the number of samples taken by our estimation procedure at time t, then we have*

$$E[N_t] \le N, \quad (10)$$

and for any $\delta > 0$,

$$\sum_{t=1}^{T} N_t \le (e-1)NT + M\log\frac{1}{\delta} \quad (11)$$

holds with probability at least $1-\delta$.

Proof *Note that for any t, we have*

$$N_t = \max_{i:y_i(t)=1} K_{t,i} = \max_{i\in[N]} y_i(t)K_{t,i} \le \sum_{i=1}^{N} y_i(t)K_{t,i}. \quad (12)$$

Combine the above result with the fact that $E[K_{t,i}|\mathcal{F}_{t-1}, y_i(t)] \le 1/E[y_i(t)|\mathcal{F}_{t-1}]$, we obtain $E[N_t] \le N$. For the second statement, define $S_t = N_t - E[N_t|\mathcal{F}_{t-1}]$, then we have

$$\begin{aligned}&\textit{Var}[S_t|\mathcal{F}_{t-1}] = E[(N_t - E[N_t|\mathcal{F}_{t-1}])^2|\mathcal{F}_{t-1}] = E[N_t^2|\mathcal{F}_{t-1}] \\ &\overset{(a)}{\le} E[(\sum_{i=1}^{N} y_i(t)K_{t,i})^2|\mathcal{F}_{t-1}] \overset{(b)}{\le} \sum_{i=1}^{N}\min\{\frac{1}{p_{t,i}}, M\} \le NM,\end{aligned}$$

where (a) holds due to (12)*; (b) comes from the fact that $E[K_{t,i}^2|\mathcal{F}_{t-1}] = \frac{2-p_{t,i}}{p_{t,i}^2}$. Since S_t is a martingale-difference sequence with respect to $\mathcal{F}_t$ and $S_t \le M$, we complete the proof of the second statement by applying Freedman's inequality [3] (Lemma 7 in our online technical report [1]) with $B = M$ and $\sum_T \le NMT$.*

Therefore, when setting $M = O(\sqrt{T})$ as suggested by Theorem 1, it holds with high probability that the amortized running time of FTPL(E)+GR is at most $O((N + \sqrt{N/T})N\log N) \approx \tilde{O}(N^2)$ when T is large. We emphasize that $\tilde{O}(N^2)$ is the upper bound of amortized running time of FTPL(E)+GR. In fact, we empirically find that even if T is very large and even goes to infinity, setting M to a constant (e.g., $O(\sqrt{N})$) which is independent of T does not result in much increase in regret (decrease in hit-ratio).

V. Analysis

Now we give the proof of Theorem 1. To facilitate the understanding, we will present statements with respect to the estimation procedure and the learning part (FTPL(E)-component) separately.

As we set M to a finite value to bound the computational-complexity of the sampling procedure, we start analyzing the bias of GR estimates introduced by the early termination of GR. As noted earlier, $\boldsymbol{p}(t)$ is the probability distribution over all cache configurations introduced by our algorithm at time t, and $p_{t,i} = \Pr\{y_i(t) = 1|\mathcal{F}_{t-1}\}$. We first give an explicit expression on the expectation of the GR estimates generated by our sampling procedure.

Lemma 1 *For all t, the request estimates* (6) *satisfy*

$$E[\hat{x}_i(t)|\mathcal{F}_{t-1}] = \Big(1 - (1-p_{t,i})^M\Big)x_i(t),\ \forall i \in [N]. \quad (13)$$

Proof *To begin, we observe that*

$$\begin{aligned}&E[K_{t,i}|\mathcal{F}_{t-1}] \\ &= \sum_{k=1}^{\infty} k(1-p_{t,i})^{k-1}p_{t,i} - \sum_{k=M}^{\infty}(k-M)(1-p_{t,i})^{k-1}p_{t,i} \\ &= \sum_{k=1}^{\infty} k(1-p_{t,i})^{k-1}p_{t,i} \\ &\quad - (1-p_{t,i})^M\sum_{k=M}^{\infty}(k-M)(1-p_{t,i})^{k-M-1}p_{t,i} \\ &= (1-(1-p_{t,i})^M)\sum_{k=1}^{\infty}k(1-p_{t,i})^{k-1}p_{t,i} = \frac{(1-(1-p_{t,i})^M)}{p_{t,i}}.\end{aligned}$$

Combine the above inequality with the fact that $E[\hat{x}_i(t)|\mathcal{F}_{t-1}] = p_{t,i}x_i(t)E[K_{t,i}|\mathcal{F}_{t-1}]$ then we complete the proof.

We also have the following lemma which gives the important properties of our GR estimates (6).

Lemma 2 *For all $\boldsymbol{S} \in \mathcal{Y}$, we have the following properties for GR estimates* (6)*,*

$$E[\langle \boldsymbol{S}, \hat{\boldsymbol{x}}(t)\rangle|\mathcal{F}_{t-1}] \le \langle \boldsymbol{S}, \boldsymbol{x}(t)\rangle, \quad (14)$$

$$E[\sum_{\boldsymbol{V}\in\mathcal{Y}} p_t(\boldsymbol{V})\langle \boldsymbol{V}, \hat{\boldsymbol{x}}(t)\rangle|\mathcal{F}_{t-1}] \ge \sum_{\boldsymbol{V}\in\mathcal{Y}} p_t(\boldsymbol{V})\langle \boldsymbol{V}, \boldsymbol{x}(t)\rangle - \frac{N}{eM}, \quad (15)$$

$$E[(\langle \boldsymbol{y}(t), \hat{\boldsymbol{x}}(t)\rangle)^2|\mathcal{F}_{t-1}] = \sum_{\boldsymbol{V}\in\mathcal{Y}} p_t(\boldsymbol{V})(\langle \boldsymbol{V}, \hat{\boldsymbol{x}}(t)\rangle)^2 \le 2CN. \quad (16)$$

The proof of Lemma 2 is given in the Appendix. In Lemma 2, (14) implies that any learning policy that relies on our GR estimates is optimistic in the perspective that the number of cache-hits of any fixed configuration will be underestimated in expectation. (15) ensures that the cache is not overly optimistic about its own performance. For the third property (16), we remark that the upper bound of conditional variance is CN for the standard (although usually not feasible) estimates (5), while it shows, somewhat surprisingly, that the variance of our estimates is no more than twice the variance of the standard estimates.

Remark 1 *We emphasize that Lemmas 1 and 2 remain valid no matter which randomization policy generates $\boldsymbol{y}(t)$.*

Next we give the key lemmas for analyzing the FTPL(E)-component of our policy. *We remark that the lemmas we will present are not specific to the GR estimates we used.* Our analysis is new to some respect in order to integrate our GR estimates. To achieve this, we borrow several ideas from [26] on FTL-style (Follow-the-Leader) methods and the proof of Corollary 4.5 in [6], and study the FTPL(E)-component of our policy via a virtual policy which uses a time-invariant perturbation vector and could peek one step into the future. Specifically, at time t, the virtual policy picks its cache configuration as

$$\hat{\boldsymbol{y}}(t) = \arg\max_{\boldsymbol{y}\in\mathcal{Y}} \langle \boldsymbol{y}, \hat{\boldsymbol{X}}(t+1) + \eta_{t+1}\hat{\boldsymbol{\gamma}}\rangle, \tag{17}$$

where $\hat{\gamma}$ is a time-invariant perturbation vector with the same distribution but independent as $\boldsymbol{\gamma}(1)$, and $\hat{\boldsymbol{X}}(t+1) = \sum_{\tau=1}^{t} \hat{\boldsymbol{x}}(\tau)$. It can be verify that, given $\mathcal{F}_t$, $\hat{\boldsymbol{y}}(t)$ and $\boldsymbol{y}(t+1)$ are conditionally independent and identically distributed. For convenience, we introduce the following notations:

$$p_{t,i} = \mathrm{E}[y_i(t)|\mathcal{F}_{t-1}] \qquad \hat{p}_{t,i} = \mathrm{E}[\hat{y}_i(t)|\mathcal{F}_t]$$
$$p_t(\boldsymbol{S}) = \Pr\{\boldsymbol{y}(t) = \boldsymbol{S}|\mathcal{F}_{t-1}\} \quad \hat{p}_t(\boldsymbol{S}) = \Pr\{\hat{\boldsymbol{y}}(t)) = \boldsymbol{S}|\mathcal{F}_t\}.$$

The following lemma bound the regret of virtual policy that picks the configuration sequence $\{\hat{\boldsymbol{y}}(t)\}_{t=1}^{T}$.

Lemma 3 *For any $\boldsymbol{S} \in \mathcal{Y}$, we have*

$$E[\sum_{t=1}^{T} \langle \hat{\boldsymbol{y}}(t), \hat{\boldsymbol{x}}(t)\rangle] - \sum_{t=1}^{T} \langle \boldsymbol{S}, \hat{\boldsymbol{x}}(t)\rangle$$
$$= \sum_{t=1}^{T}\sum_{\boldsymbol{V}\in\mathcal{Y}} \hat{p}_t(\boldsymbol{V})\langle \boldsymbol{V}, \hat{\boldsymbol{x}}(t)\rangle - \sum_{t=1}^{T} \langle \boldsymbol{S}, \hat{\boldsymbol{x}}(t)\rangle \leq \eta_T C(\log N/C + 1).$$

Then we are going to relate the performance of the virtual policy to the actual performance of FTPL(E)-component, which is shown in the next lemma.

Lemma 4 *For any t, we have*

$$\sum_{\boldsymbol{V}\in\mathcal{Y}} (p_t(\boldsymbol{V}) - \hat{p}_t(\boldsymbol{V}))\langle \boldsymbol{V}, \hat{\boldsymbol{x}}(t)\rangle \leq \frac{1}{\eta_t}\sum_{\boldsymbol{V}\in\mathcal{Y}} p_t(\boldsymbol{V})(\langle \boldsymbol{V}, \hat{\boldsymbol{x}}(t)\rangle)^2.$$

We give the full-proof of lemmas 3-4 in our online technical report [1] due to the space limit.

Now, everything is ready to prove Theorem 1, i.e., regret guarantee of FTPL(E) with GR for any request sequence. Put together Lemmas 2 (16), 3 and 4 gives

$$\begin{aligned}&\mathrm{E}[\sum_{t=1}^{T} (\sum_{\boldsymbol{V}\in\mathcal{Y}} p_t(\boldsymbol{V})\langle \boldsymbol{V}, \hat{\boldsymbol{x}}(t)\rangle - \langle \boldsymbol{S}, \hat{\boldsymbol{x}}(t)\rangle)]\\ &\leq \eta_T C\left(\log(N/C) + 1\right) + 2CN\sum_{t=1}^{T}\frac{1}{\eta_t},\ \forall \boldsymbol{S}\in\mathcal{Y}.\end{aligned} \tag{18}$$

Then according the fact that

$$\mathrm{E}[\sum_{t=1}^{T} \langle \boldsymbol{y}(t), \boldsymbol{x}(t)\rangle|\mathcal{F}_{t-1}] = \sum_{t=1}^{T}\sum_{\boldsymbol{V}\in\mathcal{Y}} p_t(\boldsymbol{V})\langle \boldsymbol{V}, \boldsymbol{x}(t)\rangle,$$

and Lemma 2 ((14), (15)), we have $\forall \boldsymbol{S} \in \mathcal{Y}$:

$$\begin{aligned}&\mathrm{E}[\sum_{t=1}^{T} \langle \boldsymbol{y}(t) - \boldsymbol{S}, \boldsymbol{x}(t)\rangle]\\ &\leq \mathrm{E}[\sum_{t=1}^{T} (\sum_{\boldsymbol{V}\in\mathcal{Y}} p_t(\boldsymbol{V})\langle \boldsymbol{V}, \hat{\boldsymbol{x}}(t)\rangle - \langle \boldsymbol{S}, \hat{\boldsymbol{x}}(t)\rangle)] + \frac{NT}{eM}.\end{aligned}$$

Put the above two inequalities together and use the inequality that $\sum_{t=1} 1/\sqrt{t} \leq 2\sqrt{T}$ then we complete the proof of Theorem 1.

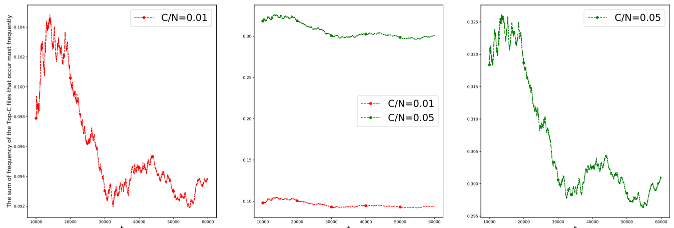

Fig. 3. The sum of the frequencies of Top-C files that occur most frequently in t rounds.

VI. Numerical results

In this section, we conduct numerical experiments to validate the theoretical performance of our caching policy. We first show that using Exponentially distributed instead of Gaussian distributed sample vector as the perturbation term in FTPL indeed improve the cache performance.

A. The bonus of the exponentially distributed perturbation

To experimentally demonstrate the bonus of the exponentially distributed perturbation, our simulation is conducted on the non-stationary request sequence under the full-feedback regime. Specifically, our generated request sequence, or our used dataset is 'rating.csv' file of the MovieLens 20M dataset [13] whose popularities are gradually changing. It contains four columns: 'userID', 'movieID', 'rating', and 'timestamp'. To fit this dataset into our sequential request model, we extract data from the 'timestamp' and 'movieID' columns. The extracted 'timestamp' was used as arrival time-slot of the request, and 'movieID' as request index to the model.

In our experimental setup, we selected part of the dataset (about $10K$ requests) for our simulation. We set the cache capacity (C) to be 1% of the library size (N). To better present the non-stationary property of the dataset, we show in figure 3 how the sum of the frequencies of Top-C files with the highest frequency changes over time. The burr and shake on the curve could reflect the gradually changing popularities in our request sequence. Note that the top C files that appear most frequently are not fixed and the sum of frequency of the Top-C files that occur most frequently fluctuates around 10%. The baselines we used are LFU, LRU, FIFO, and FTPL(G). We do not compare [10], [23], [24], [29] as their policies only apply to the coded-caching. We plot the empirical results of

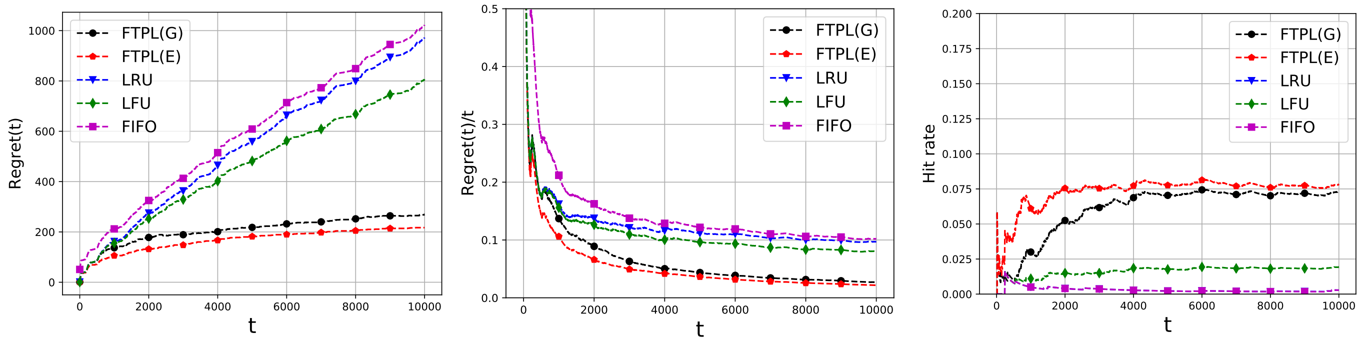

Fig. 4. Results for non-stationary request model under the full-feedback regime: (a) regret(t); (b) regret rate; (c) hit rate

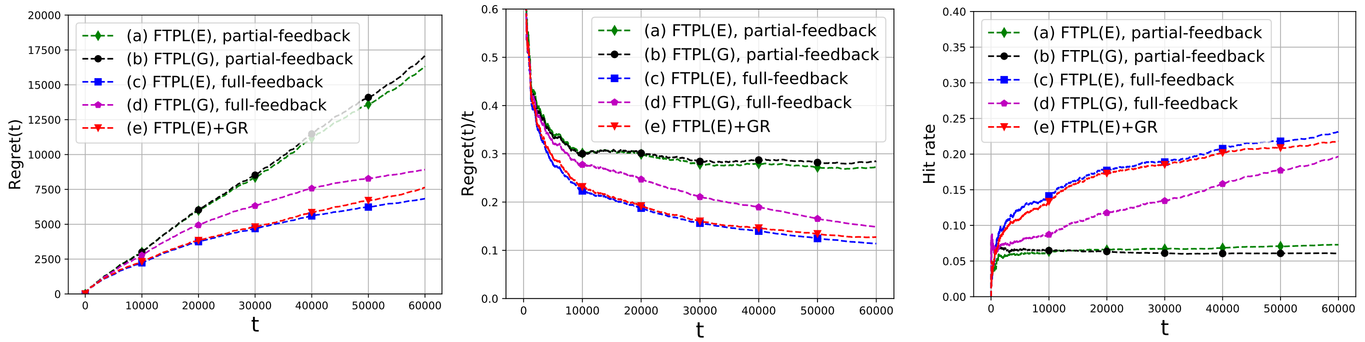

Fig. 5. Results for non-stationary request model under the partial-feedback regime: (a) regret(t); (b) regret rate; (c) hit rate

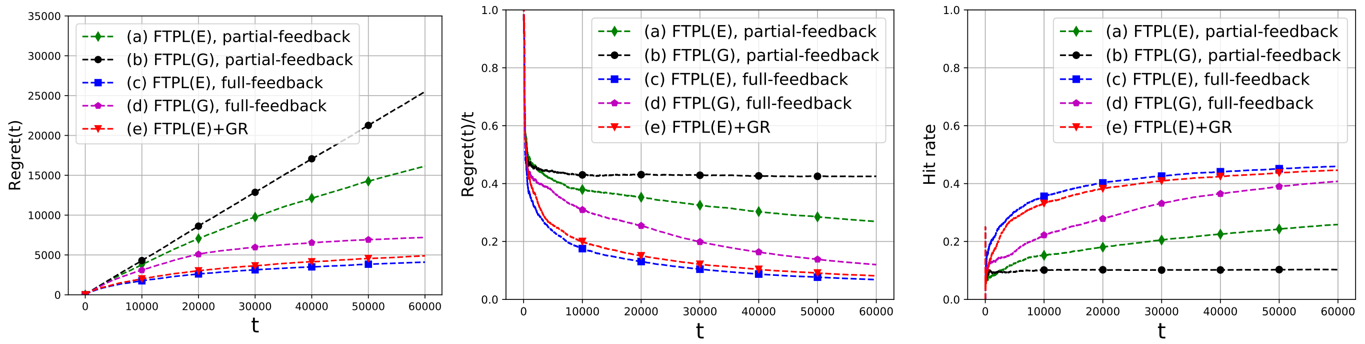

Fig. 6. Results for stationary request model under the partial-feedback regime: (a) regret(t); (b) regret rate; (c) hit rate

FTPL(E) and these baselines in Figure 4. From this figure we observe that FTPL(E) and FTPL(G) outperform FIFO, LRU, and LFU which incur linear regrets. It demonstrates that FTPL-based caching policies are superior to classic caching policies for non-stationary request sequence. And the performance of FTPL(E) is indeed better than that of FTPL(G), even though their trends are roughly the same. This confirms the advantages of our used Exponentially distributed perturbations. Besides, the hit-rate in a relatively large time-horizon ($T \geq 4 \cdot 10^3$) for FTPL(E) is $\sim 7.5\%$ which is only 25% worse than 10%.

B. Partial-feedback regime

We now present the empirical performance of FTPL(E)+GR under the regime of partial-feedback. We conduct numerical experiments for stationary request model and non-stationary request model, respectively, to validate the theoretical guarantees of FTPL(E)+GR.

Non-stationary request model. In order to model the non-stationary request sequence, we still used the MovieLens 20M dataset. To make the curve trends are more obvious under the partial-feedback regime, we choose larger time-horizon and larger ratio of C/N. Hence, we selected $60K$ requests of the dataset and set the cache capacity (C) to be 5% of the library size (N) in our experimental setting. From figure 3, we can see that the sum of frequency of the Top-C files that occur most frequently in the whole selected requests fluctuates around 31%. As we are the first work to study the online caching problem under the partial-feedback regime, we construct the following baselines for comparison: (a) FTPL that uses the Gaussian distributed perturbation and could observe all file requests, referred as ”FTPL(G), full-feedback”; (b) FTPL that uses the Exponentially distributed perturbation could observe all file requests, referred as ”FTPL(E), full-feedback”; (c) FTPL(G) that set $\boldsymbol{x}(t) = \mathbf{0}$ if request $\boldsymbol{x}(t)$ incurs hit-miss, referred as "FTPL(G), partial-feedback"; (d) FTPL(E) that set $\boldsymbol{x}(t) = \mathbf{0}$ if request $\boldsymbol{x}(t)$ incurs hit-miss, referred as "FTPL(E), partial-feedback". We do not compare classic caching policies (e.g., LFU, LRU, FIFO, etc) as they are designed for full-feedback regime and poor-performance achieving under non-stationary request sequence.

Figure 5 shows the performance of FTPL(E)+GR and all baselines. We can observe that FTPL(E)+GR indeed achieves a sublinear (about $O(\sqrt{T})$) regret guarantee and only incurs slight performance degradation compared with baseline "FTPL(E), partial-feedback", which is the best-case scenario under our experimental setup. Besides, we see that the hit-ratio guaranteed by the FTPL(E)+GR exceeds 20% when T is larger than $40K$, which is about 65% of the value 31%. Such performance is completely acceptable for the non-stationary request sequence and partial-feedback regime. Moreover, it is worth noting that FTPL(G) and FTPL(E) would incur linear regrets without any popularity estimation procedures. Thus, designing and combining appropriate estimation procedures is necessary for partial-feedback regime especially when request model is non-stationary.

Stationary request model. We also evaluate our FTPL(E)+GR under the Zipf distributed requests containing $N = 1244$ unique file indices. We set the cache capacity (C) to be 5% of the library size (N), and the sum of popularity of the most popular C files is approximately 52%. We use the same baselines in our experimental setup for non-stationary request model. We compare the performance of these baselines and FTPL(E)+GR in Figure 6. From this figure, we could conclude that our Geometric Resampling (GR) estimation procedure estimates the popularities correctly as FTPL(E)+GR achieves almost the same performance as FTPL(E) that could observe every request, and achieves a hit-rate of $\sim 50\%$ that is almost optimal. Surprisingly, we observe that FTPL(E)+GR has a better performance than FTPL(E) that could observe every request under the partial-feedback regime, which is a little counter-intuitive. The reason why this phenomenon happens is not only because our estimates are almost accurate under the stationary request popularity, but also because the perturbation terms that follow the exponential distribution are superior to those that follow the Gaussian distribution.

Running time of FTPL(E)+GR. We also measure the empirical runtime of FTPL(E)+GR in our simulations. We show the per-request processing time ratio of FTPL(E)+GR to FTPL(E) in Table I, i.e., the runtime ratio. As mentioned before, the worst-case amortized running time of FTPL(E)+GR is $\tilde{O}(N^2)$ when $M = O(\sqrt{T})$, which is at most $O(N)$ times that of FTPL(E). From Table I, we can observe that even setting $M = O(\sqrt{T})$, the runtime ratio of FTPL(E)+GR to FTPL(E) is far less than N which is somewhat surprisingly. This confirms the correctness and effectiveness of our Geometric Resampling (GR) estimation procedure. Interestingly, we also empirically find that when setting $M = O(\sqrt{N})$ or greater which is independent of T, the resulting performance degradation (increase of regret) is at most 18% both for our stationary and non-stationary dataset. Therefore, the regret upper bound in Theorem 1 might be further improved and we leave it as a future work.

TABLE I
THE RUNTIME RATIO OF FTPL(E)+GR AND FTPL(E)

Dataset	Request types (N)	Runtime ratio
ZIPF	1163	85.58
MovieLens	2160	100.92

VII. Conclusion

In this paper, we develop and analyze a novel online caching policy, FTPL+GR. To the best of our knowledge, FTPL+GR is the first online caching policy to achieve sublinear regret guarantee for any request model under the partial-feedback regime. We then conduct numerical experiments to validate the theoretical performance guarantee of our caching policy. For future work, it is a good direction to investigate sharper performance bounds for FTPL+GR or more advanced caching policy for partial-feedback regime. Moreover, developing an efficient and sublinear regret achieving online coded-caching policy for partial-feedback regime is also an open question.

Appendix

A. Proof of Lemma 2

Firstly, note that (14) could be derived immediately according to Lemma 1 as

$$\begin{aligned} &\mathrm{E}[\hat{x}_i(t)|\mathcal{F}_{t-1}] \le x_i(t),\ \forall i \in [N] \\ \Rightarrow\ &\mathrm{E}[\langle \hat{\boldsymbol{x}}(t), \boldsymbol{S}\rangle|\mathcal{F}_{t-1}] \le \langle \boldsymbol{x}(t), \boldsymbol{S}\rangle,\ \forall \boldsymbol{S} \in \mathcal{Y}. \end{aligned}$$

Also, by Lemma 1, we have

$$\begin{aligned} &\mathrm{E}[\sum_{\boldsymbol{V}\in\mathcal{Y}} p_t(\boldsymbol{V})\langle \boldsymbol{V}, \hat{\boldsymbol{x}}(t)\rangle|\mathcal{F}_{t-1}] \\ &= \sum_{i=1}^{N} p_{t,i}\mathrm{E}[\hat{x}_i(t)|\mathcal{F}_{t-1}] \\ &= \sum_{i=1}^{N} p_{t,i}(1-(1-p_{t,i})^M)x_i(t) \\ &= \sum_{i=1}^{N} p_{t,i}x_i(t) - \sum_{i=1}^{N} p_{t,i}(1-p_{t,i})^M x_i(t) \\ &\overset{(a)}{\ge} \sum_{i=1}^{N} p_{t,i}x_i(t) - \sum_{i=1}^{N} p_{t,i}e^{-Mp_{t,i}}x_i(t) \\ &\overset{(b)}{\ge} \sum_{i=1}^{N} p_{t,i}x_i(t) - \sum_{i=1}^{N} \frac{N}{eM}x_i(t) \\ &\overset{(c)}{\ge} \sum_{\boldsymbol{V}\in\mathcal{Y}} p_t(\boldsymbol{V})\langle \boldsymbol{V}, \boldsymbol{x}(t)\rangle - \frac{N}{eM}, \end{aligned}$$

where (a) holds due to $a(1-a)^M \le ae^{-Ma}$, $0 \le a \le 1$; (b) comes from the fact $f(a) = ae^{-aM}$ takes its maximum at $a = M^{-1}$; where (c) is due to $\sum_{i=1}^{N} p_{t,i}x_i(t) = \mathrm{E}[\sum_{t=1}^{T} \langle \hat{\boldsymbol{y}}(t), \boldsymbol{x}(t)\rangle|\mathcal{F}_{t-1}] = \sum_{t=1}^{T}\sum_{\boldsymbol{V}\in\mathcal{Y}} \hat{p}_t(\boldsymbol{V})\langle \boldsymbol{V}, \boldsymbol{x}(t)\rangle$. Then we complete the proof of (15). Lastly, the property (16) is proven as

$$\begin{aligned}
&\sum_{\boldsymbol{V}\in\mathcal{Y}} p_t(\boldsymbol{V})(\langle \boldsymbol{V}, \hat{\boldsymbol{x}}(t)\rangle)^2 \\
&= E[\sum_{i=1}^{N}\sum_{j=1}^{N}(y_j(t)\hat{x}_j(t))(y_i(t)\hat{x}_i(t))|\mathcal{F}_{t-1}] \\
&\overset{(a)}{\le} E[\sum_{i=1}^{N}\sum_{j=1}^{N}(y_j(t)K_{t,j}y_j(t)x_j(t))(y_i(t)K_{t,i}y_i(t)x_i(t))|\mathcal{F}_{t-1}] \\
&\overset{(b)}{\le} E[\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{K_{t,j}^2+K_{t,i}^2}{2}(y_j^2(t)x_j(t))(y_i^2(t)x_i(t))|\mathcal{F}_{t-1}] \\
&\overset{(c)}{\le} 2E[\sum_{i=1}^{N}\frac{1}{p_{t,i}^2}(y_i^2(t)x_i(t))\sum_{j=1}^{N}(y_j^2(t)x_j(t))|\mathcal{F}_{t-1}] \\
&\overset{(d)}{\le} 2E[\sum_{i=1}^{N}\frac{1}{p_{t,i}^2}y_i^2(t)C|\mathcal{F}_{t-1}] \le 2CN,
\end{aligned}$$

where (a) holds by the definition of our estimates $\hat{\boldsymbol{x}}(t)$; (b) is due to $ab \le \frac{a^2+b^2}{2}$; (c) holds by the symmetry and the fact that $\mathrm{E}[K_{t,i}^2|\mathcal{F}_{t-1}] = \frac{2-p_{t,i}}{p_{t,i}^2} \le \frac{1}{p_{t,i}^2}$; (d) follows from the fact that $||\boldsymbol{y}(t)||_1 \le C$ and $y_i(t), x_i(t) \le 1,\ \forall i$; and (e) is because $E[y_i^2(t)|\mathcal{F}_{t-1}] \le E[y_i(t)|\mathcal{F}_{t-1}]E[y_i(t)|\mathcal{F}_{t-1}] = p_{t,i}^2$.

References

[1] Online report. https://cloud.tsinghua.edu.cn/f/31f38e94eb3e4da298cb/.

[2] Charu Aggarwal, Joel L Wolf, and Philip S. Yu. Caching on the world wide web. *IEEE Transactions on Knowledge and data Engineering*, 11(1):94–107, 1999.

[3] Alina Beygelzimer, John Langford, Lihong Li, Lev Reyzin, and Robert Schapire. Contextual bandit algorithms with supervised learning guarantees. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 19–26. JMLR Workshop and Conference Proceedings, 2011.

[4] Rajarshi Bhattacharjee, Subhankar Banerjee, and Abhishek Sinha. Fundamental limits on the regret of online network-caching. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 4(2):1–31, 2020.

[5] Pol Blasco and Deniz Gündüz. Learning-based optimization of cache content in a small cell base station. In *2014 IEEE International Conference on Communications (ICC)*, pages 1897–1903. IEEE, 2014.

[6] Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge university press, 2006.

[7] Jacob Chakareski. Vr/ar immersive communication: Caching, edge computing, and transmission trade-offs. In *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*, pages 36–41, 2017.

[8] Dimitris Chatzopoulos, Carlos Bermejo, Zhanpeng Huang, and Pan Hui. Mobile augmented reality survey: From where we are to where we go. *Ieee Access*, 5:6917–6950, 2017.

[9] Asit Dan and Don Towsley. An approximate analysis of the lru and fifo buffer replacement schemes. In *Proceedings of the 1990 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pages 143–152, 1990.

[10] Siqi Fan, I-Hong Hou, and Lotfi Benmohamed Van Sy Mai. Online service caching and routing at the edge with unknown arrivals. 2022.

[11] Philippe Flajolet, Daniele Gardy, and Loÿs Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Applied Mathematics*, 39(3):207–229, 1992.

[12] Per Brinch Hansen. *Operating system principles*. Prentice-Hall, Inc., 1973.

[13] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

[14] Predrag R Jelenković and Xiaozhu Kang. Characterizing the miss sequence of the lru cache. *ACM SIGMETRICS Performance Evaluation Review*, 36(2):119–121, 2008.

[15] Mingyue Ji, Giuseppe Caire, and Andreas F Molisch. Fundamental limits of caching in wireless d2d networks. *IEEE Transactions on Information Theory*, 62(2):849–869, 2015.

[16] Mingyue Ji, Ming Fai Wong, Antonia M Tulino, Jaime Llorca, Giuseppe Caire, Michelle Effros, and Michael Langberg. On the fundamental limits of caching in combination networks. In *2015 IEEE 16th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 695–699. IEEE, 2015.

[17] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. On the existence of a spectrum of policies that subsumes the least recently used (lru) and least frequently used (lfu) policies. In *Proceedings of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 134–143, 1999.

[18] Michael Luby, Amin Shokrollahi, Mark Watson, and Thomas Stockhammer. Raptor forward error correction scheme for object delivery. Technical report, 2007.

[19] Thodoris Lykouris and Sergei Vassilvtiskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning*, pages 3296–3305. PMLR, 2018.

[20] Mohammad Ali Maddah-Ali and Urs Niesen. Fundamental limits of caching. *IEEE Transactions on information theory*, 60(5):2856–2867, 2014.

[21] Mohammad Ali Maddah-Ali and Urs Niesen. Coding for caching: Fundamental limits and practical challenges. *IEEE Communications Magazine*, 54(8):23–29, 2016.

[22] Samrat Mukhopadhyay and Abhishek Sinha. Online caching with optimal switching regret. In *2021 IEEE International Symposium on Information Theory (ISIT)*, pages 1546–1551. IEEE, 2021.

[23] Georgios Paschos, George Iosifidis, Giuseppe Caire, et al. Cache optimization models and algorithms. *Foundations and Trends® in Communications and Information Theory*, 16(3–4):156–345, 2020.

[24] Georgios S Paschos, Apostolos Destounis, Luigi Vigneri, and George Iosifidis. Learning to cache with no regrets. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 235–243. IEEE, 2019.

[25] Ramtin Pedarsani, Mohammad Ali Maddah-Ali, and Urs Niesen. Online coded caching. *IEEE/ACM Transactions on Networking*, 24(2):836–845, 2015.

[26] Jan Poland. Fpl analysis for adaptive bandits. In *International Symposium on Stochastic Algorithms*, pages 58–69. Springer, 2005.

[27] Amr Rizk, Michael Zink, and Ramesh Sitaraman. Model-based design and analysis of cache hierarchies. In *2017 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 1–9. IEEE, 2017.

[28] Elisha J Rosensweig, Daniel S Menasche, and Jim Kurose. On the steady-state of cache networks. In *2013 Proceedings IEEE INFOCOM*, pages 863–871. IEEE, 2013.

[29] Tareq Si Salem, Giovanni Neglia, and Stratis Ioannidis. No-regret caching via online mirror descent. In *ICC 2021-IEEE International Conference on Communications*, pages 1–6. IEEE, 2021.

[30] Dimitrios N Serpanos, George Karakostas, and Wayne Hendrix Wolf. Effective caching of web objects using zipf's law. In *2000 IEEE International Conference on Multimedia and Expo. ICME2000. Proceedings. Latest Advances in the Fast Changing World of Multimedia (Cat. No. 00TH8532)*, volume 2, pages 727–730. IEEE, 2000.

[31] Karthikeyan Shanmugam, Negin Golrezaei, Alexandros G Dimakis, Andreas F Molisch, and Giuseppe Caire. Femtocaching: Wireless content delivery through distributed caching helpers. *IEEE Transactions on Information Theory*, 59(12):8402–8413, 2013.

[32] Abhigyan Sharma, Xiaozheng Tie, Hardeep Uppal, Arun Venkataramani, David Westbrook, and Aditya Yadav. A global name service for a highly mobile internetwork. *ACM SIGCOMM Computer Communication Review*, 44(4):247–258, 2014.

[33] Amin Shokrollahi. Raptor codes. *IEEE transactions on information theory*, 52(6):2551–2567, 2006.

[34] Benjamin Van Roy. A short proof of optimality for the min cache replacement algorithm. *Information processing letters*, 102(2-3):72–73, 2007.

[35] Chuan Wu and Baochun Li. rstream: resilient and optimal peer-to-peer streaming with rateless codes. *IEEE Transactions on Parallel and Distributed Systems*, 19(1):77–92, 2007.