Caching Performance under Constraints on Cache Updates

Chinmaya Venkatesh
Dept of Electrical Engineering
Indian Institute of Technology, Madras
Chennai, India
chinmaya.venkatesh@gmail.com

Avhishek Chatterjee
Dept of Electrical Engineering
Indian Institute of Technology, Madras
Chennai, India
avhishek@ee.iitm.ac.in

Sharayu Moharir
Dept of Electrical Engineering
Indian Institute of Technology, Bombay
Mumbai, India
sharayu.moharir@gmail.com

***Abstract*—Caching is a fundamental component of content delivery systems such as cloud services and edge networks. Most existing analytical studies of caching assume that the cache can be updated after every file request; however, this assumption is impractical in real systems due to computational constraints. In this paper, we study an online caching framework in which cache updates occur only at prescribed intervals, resulting in bunched request arrivals between successive updates. We analyze the performance of the Follow-The-Perturbed-Leader (FTPL) algorithm under such constraints while incorporating computation cost into the regret metric. For various combinations of adversarial and stochastic request and arrival processes, we derive upper bounds on regret as a function of the cache update frequency. Our results show that both the regret and the total computation cost can be made sublinear in the time horizon with appropriately chosen update intervals. In particular, significantly tighter bounds are obtained when arrivals follow stochastic processes such as the Poisson distribution.**

***Index Terms*—Caching; Computational Cost; Regret**

I. Introduction

Caching plays a critical role in the efficient delivery of online video, cloud computing, and other cloud-based services. Motivated by the non-stationarity and arbitrary triggering of content popularity, online caching under adversarial request sequences has been studied extensively, with the Follow-The-Perturbed-Leader (FTPL) algorithm shown to achieve optimal regret when the cache is updated after every request [2], [8]. Classical caching policies such as LRU and LFU [9], [10] operate under stationarity assumptions and do not provide regret guarantees against adversarial sequences. Existing caching policies update the cache after every file request, which is practically infeasible due to computational overhead. Executing the caching algorithm at specific, pre-determined epochs, which are not necessarily equispaced, is more practical.

The work closest to ours is that of Faticanti and Neglia [5], who study optimistic caching under batched requests, where each batch contains a fixed number of arrivals. Our model differs in a fundamental way: cache refreshes occur at prescribed *time intervals* rather than after a fixed count of requests, so the number of arrivals per update interval is itself stochastic or adversarial. Moreover, we explicitly incorporate computation cost into the regret metric, a dimension absent from prior caching analyses, and we characterize the joint tradeoff between caching performance and computational overhead for FTPL as a function of the update interval.

Our work also connects to the broader literature on batched and lazy-update online learning. Joulani et al. [7] and Cesa-Bianchi et al. [3] study regret under delayed or batched feedback, but focus on feedback latency rather than hard computational budgets or time-interval-based update constraints. Lazy regularization methods in online convex optimization [11] defer parameter updates for sparsity and efficiency, but the motivation is algorithmic convenience rather than a cost-constrained update budget. Finally, switching-cost regret minimization [1], [4] penalizes changes to the decision variable between rounds. This is structurally distinct from our computation cost, which penalizes each execution of the caching algorithm regardless of whether the cached file set changes. By unifying the caching regret and computation cost into a single modified regret metric, our framework provides a principled basis for designing cache update intervals that are simultaneously efficient in performance and computation.

A. Our Contributions

Our main contributions are summarized below.

- We introduce a caching model with *constrained update frequency and computation cost*, to capture practical limitations of real systems where running the caching algorithm after every request is computationally infeasible.
- We introduce a *modified regret*, defined as the caching regret plus a weighted computation cost incurred by cache updates.
- We analyze the performance of the Follow-The-Perturbed-Leader (FTPL) algorithm for both fixed and adaptive update intervals and characterize conditions under which the modified regret remains sublinear in the time horizon. These results show how appropriately chosen update intervals can reduce computation while maintaining favorable regret guarantees.

II. The Model

The cache receives file requests from users according to a temporal process. Each request is characterized by an arrival time and the identity of the requested file. The arrival time is a real number, while the requested file belongs to a library of

size L. As in prior caching works, we consider a cache with capacity $\mathcal{C} \ll L$.

The request arrival times and requested files are not known to the cache a priori. The request process may be stochastic or adversarial, and we study both settings. In the stochastic case, the statistics of the request process are also unknown to the cache.

In much of the caching literature, the standard model assumes a discrete-time setting in which exactly one request arrives in each epoch, from either a stochastic or arbitrary distribution. The cache is updated after each request arrival. This is typically infeasible in practice.

To capture these practical constraints, we consider caching in the presence of *bunched arrivals*. Instead of executing the caching algorithm after every request arrival, the algorithm is run after a prescribed duration of time.

Specifically, time is divided into update intervals indexed by $k = 1, 2, \ldots$. The duration of slot k is τ_k file request slots, where the duration of a file request slot is fixed at s. The caching algorithm is executed at the end of each update interval, and the cache is updated according to the algorithm output. We distinguish between update intervals and file request slots: in file request slot t, N_t requests arrive. The time horizon of interest consists of T file request slots, and the total number of cache refreshes is T_B, where $\sum_{k=1}^{T_B} \tau_k = T$.

The update interval durations τ_k are not necessarily identical; in fact, allowing them to vary over time can improve caching performance in certain scenarios.

Finally, in file request slot t, the number of arriving requests is denoted by N_t. This quantity may be stochastic or adversarial and may depend on the update interval active at time t. An illustration of the same is shown in Figure 1.

The file requested in the ith request during file request slot t is denoted by $x_{t,i}$, where $x_{t,i} \in \{0,1\}^L$. The set of files stored in the cache during slot t is denoted by $C(t)$. For $t \geq 2$, let X_t denote the cumulative requests up to time $t-1$, given by $X_t = \sum_{l=1}^{t-1} \sum_{i=1}^{N_l} x_{l,i}$.

Let y_t represent the cache state at the end of file request slot t. The component $y_{t,i} = 1$ if file i is cached and 0 otherwise. The cache capacity $\mathcal{C}$ imposes the constraint $\sum_i y_{t,i} = \mathcal{C}, \quad \forall t$.

File requests and arrivals: We consider two types of file request processes: *adversarial* and *stochastic*, as in the existing literature.

In the stochastic setting, each request $x_{t,i}$ is independently drawn from a distribution $\mu = (\mu_1, \mu_2, \ldots, \mu_L)$, where $\mathbb{P}(x_{t,i} = l) = \mu_l$ for $l \in \{1, \ldots, L\}$. Without loss of generality, assume $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_L$. The distribution μ is unknown to the caching policy.

In the adversarial setting, the request sequence $\{x_{t,i}\}$ need not follow any statistical distribution and can be chosen arbitrarily by an external agent prior to the start of the time horizon.

As mentioned before, in our model, cache updates occur at update interval k of duration τ_k, while N_t denotes the number of arrivals in file request slot t. We study the impact of both stochastic and adversarial N_t on caching performance.

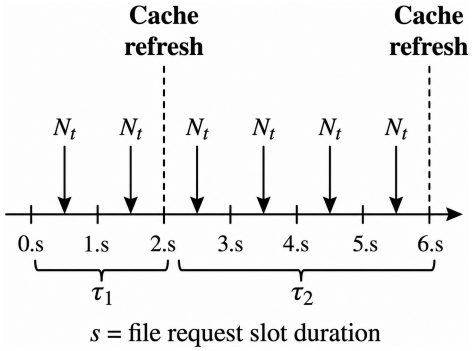

Fig. 1. An illustration of the practical setup with file requests and cache computation is shown above

Policies: At the beginning of file request slot $t = 1, 2, \ldots$, a caching policy π uses the history up to time $t-1$, $h(t) = \{x_{u,i} : u \leq t-1,\ 1 \leq i \leq N_u\}$, to determine the cache configuration $C(t)$, i.e., $C(t) = \pi(h(t))$. At $t = 1$, the cache is initialized by selecting $\mathcal{C}$ files uniformly at random from L.

In the caching literature, the Follow-The-Perturbed-Leader (FTPL) algorithm and its variants have been shown to perform well under both stochastic and adversarial arrivals. A linear loss function $\tilde{g}_t^T x_t = \tilde{g}_t(i_t)$ is chosen to incur a cost of 1 unit when there is a *cache miss* and 0 when there is a *cache hit*.

However, these studies assume that the cache is updated after every request [2], [8]. Since FTPL is a well-studied algorithm, we aim to characterize its performance in the more realistic setting where cache updates occur only at prescribed intervals.

FTPL is used to determine which files are stored in the cache at each update interval. The loss $\tilde{g}_t(i_t)$ is appropriately modified to equal the total number of *cache misses* in each refresh epoch, and can at most be N_t i.e.,

$$\tilde{g}_t(i_t) = \sum_{i=1}^{N_t} \mathbf{1}\{x(t,i) \notin C(t)\}$$

For $\mathcal{C} = 1$, the algorithm (Section 5.5.3 in [6]) is given below.

Algorithm 1 FTPL for prediction from expert advice

Input: $\eta > 0$
Draw n exponentially distributed variables $n(i) \sim e^{-\eta x}$
Let $x_1 = \arg\min_{e_i \in \Delta_n} \{-e_i^T n\}$
for $t = 1$ to T **do**
    Predict using expert i_t such that $\hat{x}_t = e_{i_t}$
    Observe the loss vector and incur loss $\tilde{g}_t^T \hat{x}_t = \tilde{g}_t(i_t)$
    Update (w.l.o.g. choose $\hat{x}_{t+1}$ to be a cache configuration)

$$\hat{x}_{t+1} = \arg\min_{x \in \Delta_n} \sum_{s=1}^{t} \tilde{g}_s^T x - n^T x$$

end for

The perturbation $n(i)$ follows a one-sided negative exponential distribution,

$$\mathbb{P}(n(i) \leq x) = 1 - e^{-\eta x}.$$

We analyze the case $\mathcal{C} \geq 1$ only for Setting 3 (described below), where file requests are stochastic, and modify the expressions accordingly. For the remainder of the paper, we restrict attention to $\mathcal{C} = 1$.

Performance Metric: The reward obtained by a caching policy for a file request is 1 if the requested file is in the cache and 0 otherwise. The total reward over a time horizon is the sum of rewards across all requests.

Policies are evaluated through the regret relative to the optimal stationary policy. The regret up to time T is defined as the difference between the total reward accumulated by the optimal stationary policy and that obtained by the caching policy over the same horizon.

Regret: We consider only static regret, i.e., the contents cached by the optimal stationary policy remain fixed over time. For a policy π and request sequence $\{\{x_{t,i}\}_{i=1}^{N_t}\}_{t=1}^{T}$, the regret up to time T under adversarial file requests is

$$R_A^{\pi}(\{\{x_{t,i}\}_{i=1}^{N_t}\}_{t=1}^{T}, T) = \sup_{y \in Y} \langle y, X_{T+1} \rangle - \sum_{t=1}^{T} \sum_{i=1}^{N_t} \mathbb{E}\langle y_t, x_{t,i} \rangle,$$

where the expectation is with respect to any randomness introduced by the policy and Y denotes the set of feasible cache configurations. The worst-case regret is

$$R_A^{\pi}(T) = \sup_{\{\{x_{t,i}\}_{i=1}^{N_t}\}_{t=1}^{T}} R_A^{\pi}(\{\{x_{t,i}\}_{i=1}^{N_t}\}_{t=1}^{T}, T).$$

For stochastic file requests, the regret under policy π is

$$R_S^{\pi}(T) = \mathbb{E}\left[\sum_{t=1}^{T} \sum_{i=1}^{N_t} \mathbf{1}\{x(t,i) \in C^*\} - \mathbf{1}\{x(t,i) \in C(t)\}\right],$$

where C^* denotes the optimal cache configuration, which is well defined in the stochastic setting.

Existing caching literature focuses on minimizing regret under different request patterns. An important practical consideration, however, is the cost associated with executing the caching algorithm. We model this as the **computation cost**, incurred whenever the cache configuration is recomputed. To incorporate this aspect, we consider a **modified regret** defined as the caching regret plus a weighted computation cost over the horizon.

Specifically, the cache incurs a cost of $\mathcal{K}$ units each time the caching algorithm is executed. Under this model, the modified regret defined above is a natural metric, since the optimal stationary policy incurs only a single computation of cost $\mathcal{K}$.

Policies that update the cache after every request incur linear modified regret due to the linear computation cost. Our goal is to determine whether FTPL can achieve sublinear modified regret.

Alternatively, one could study the original regret while separately ensuring that computation cost remains sublinear. However, whenever the modified regret is sublinear, both the original regret and the computation cost are also sublinear. Hence, the **modified regret** (referred to as **regret** henceforth) provides a convenient and practical performance metric.

We study different combinations of the arrival process N_t and the file request process within each bunch:

1) **Setting 1:** *Arbitrary bunched arrivals, arbitrary file requests, zero computation cost*. Here both arrivals and file requests may be adversarial. Computation cost is ignored and periodic refresh is considered, i.e., $\tau_k = 1$.
2) **Setting 2:** *Arbitrary bunched arrivals, arbitrary file requests, non-zero computation cost*. Each execution of the FTPL algorithm incurs cost $\mathcal{K}$. The objective is to optimize both the cache configuration and the number of computations. Two refresh strategies are considered:
   a) Fixed update interval: $\tau_k = \tau(T)$ determined by the horizon T and slot duration s.
   b) Adaptive update interval: $\tau_k(t)$ increases as a predetermined function of time t.

   We also consider a special case where the arrival process is stochastic.
3) **Setting 3:** *Arbitrary bunched arrivals, stochastic file requests, non-zero computation cost*. Restricting file requests to be stochastic allows tighter regret bounds due to the well-defined request distribution.

III. Results

In summary, our main theoretical contribution lies in quantifying the effect of infrequent cache updates on the regret of the well known FTPL caching policy. We characterize the regret behavior in terms of the cache update frequency under different scenarios. In particular, we show that both regret and the total computation cost can be made sub-linear in the length of the time horizon. Note that in the existing literature on caching, the computation cost scales linearly since the cache is updated for each new file request.

Theorem 1. *[Setting 1] When the bunched arrivals are arbitrary, and so are the file requests, in the absence of computation cost, the regret of FTPL policy can be upper-bounded as:*

$$R^{FTPL(\eta=T^{-\alpha})}(T) \leq \mathcal{O}(T^{\alpha})$$

for each $\alpha = 1 - \frac{\beta}{2}$, *when* N_t *is* $\mathcal{O}(T^{\frac{1}{2}(1-\beta)})$ *for each* $t \leq T$ *and* $\beta \in (0, 1)$.

The above theorem extends to the case when N_t is a random variable. The result of Theorem 1 also extends to stochastic N_t if $\mathbb{E}[N_t^2]$ is $\mathcal{O}(T^{1-\beta})$.

This theorem says that we can obtain a regret that grows sub-linearly with the time horizon of interest T, as long as N_t does not exceed $\sqrt{T}$ for each t. In fact, it is possible to obtain a regret upper bound close to $\sqrt{T}$, if $\mathbb{E}[N_t^2]$ is sub-polynomial in T, i.e. $\mathbb{E}[N_t^2]$ is $\mathcal{O}(T^{\delta})$ for any $\delta > 0$.

This implies that if FTPL is run per sub-polynomial (in time horizon) number of arrivals instead of every arrival, the standard square root regret guarantee of FTPL still holds [6].

This implies that the performance of FTPL is robust up to sub-polynomial bunching of arrivals. In the next sections, we study what happens to FTPL's performance when there is non-zero cache computation cost, which, as discussed above, is an issue that all caches face.

Setting 2, Case 1: The total number of times computation is performed is $\frac{T}{\tau(T)}$. The net computation cost is $\mathcal{K}\frac{T}{\tau(T)}$. Hence, the total computation cost is sublinear as long as $\tau(T)$ grows as $T^{1-\delta}$ for some $\delta > 0$.

Theorem 2. *[Setting 2, Case 1] When the bunched arrivals are arbitrary, and so are the file requests, in the presence of computation cost and fixed τ_k we have:*

$$R^{FTPL(\eta=T^{-\alpha})}(T, \mathcal{K}, \tau(T)) \leq \mathcal{O}(T^{\alpha})$$

for $\alpha = 1 - \frac{\beta}{3}$, when N_t is $\mathcal{O}(T^{\frac{1}{2}(1-\beta)})$ for each $t \leq T$ and $\beta \in (0, 1)$, and $\tau(T) = T^{\gamma}$ where $\gamma = \frac{\beta}{3}$.

If we consider the case when N_t are random variables, which is similar to the case we discuss right after Theorem 1, the same sub-linear regret bound of Theorem 2 also holds given $\mathbb{E}\left[(max\{N_t, t \in k\text{th cache update interval}\})^2\right]$ is $\mathcal{O}(T^{1-\beta})$ for each $k \leq \frac{T}{\tau(T)}$.

Theorem 2 implies that FTPL has a sublinear **modified** regret, which means that it has sublinear (original) regret without computation cost as well as a sublinear total computation cost. However, the regret bound is worse than $\sqrt{T}$ when compared to the zero-cost setting. In fact, as long as N_t is sub-polynomial in T, the regret bound is $T^{\frac{2}{3}}$. This is expected since constraining the computation cost constrains the frequency of updates by FTPL.

It would be interesting to understand what truly causes the worsening of the regret bound. In particular, which of the following two has a larger impact on the regret when computation cost is considered: the adversarial arrivals or the adversarial file requests? To answer this we study a special sub-case, where arrivals of requests are according to a Poisson process, but the file requests they bring are arbitrary.

Special sub-case: Stochastic bunched arrivals, Arbitrary file requests with non zero computation cost

Theorem 3. *When arrivals follow a Poisson distribution with mean λ, a tighter bound holds: Let $\eta = (\frac{\log n}{\lambda T})^{\frac{2}{3}}$ and $\tau(T) = (\frac{T}{\lambda^2 \log n})^{\frac{1}{3}}$. Then, we have:*

$$\mathbb{E}[\sum_{k=1}^{\frac{T}{\tau(T)}} \tilde{g}_k^T(\hat{x_k} - x^*)] \leq \mathcal{O}((\lambda T\sqrt{\log n})^{\frac{2}{3}})$$

This result says that if the arrivals are according to Poisson distribution, FTPL offers $T^{\frac{2}{3}}$ regret even when there is a constraint on the computation frequency, as long as the number of allowed computation scales no slower than $T^{\frac{2}{3}}$. Notice how this result does not require N_t to be sub-polynomial in T. Thus, this implies that arbitrary arrivals, and not arbitrary file requests, are the main cause behind the worsening of the regret bound for FTPL in the presence of a constraint on the frequency of cache computations.

Setting 2, Case 2: In particular, we explore the possibility of the update interval growing as an arithmetic progression, where $\tau_k = \tau(1 + (k-1)d)$. This indirectly restricts the number of cache computations over a duration T to $\sqrt{T}$ while ensuring that the frequency of cache computation reduces as $\frac{1}{\sqrt{T}}$ with time.

Theorem 4. *[Setting 2, Case 2] When the bunched arrivals are arbitrary, and so are the file requests, in the presence of computation cost and adaptive [linearly increasing] τ_k we have:*

$$R^{FTPL(\eta=T^{\beta-1})}(T, \mathcal{K}, \tau_k) \leq \mathcal{O}(T^{1-\beta})$$

when N_t is $\mathcal{O}(T^{\frac{1}{4}-\beta})$ for each $t \leq T$ and $\beta \in (0, \frac{1}{4})$, and $\tau_k = \tau(1 + (k-1)d)$ for each $k \leq T_B$.

Comparing with Theorem 2 at the same regret scaling, i.e. $\beta = \frac{3}{4}$ in Theorem 2 and $\beta = \frac{1}{4}$ in Theorem 4, we observe that the total number of computations for fixed τ_k is $T^{\frac{3}{4}}$, whereas that for AP τ_k is $\sqrt{T}$. Also, by looking at the conditions in Theorem 2 it is clear that for fixed τ_k, the total number of cache computations cannot be smaller than $T^{\frac{2}{3}}$ if we want sublinear regret whereas that for AP τ_k can be as small as $\sqrt{T}$. Thus, AP τ_k offers lower number of cache computations at the same regret performance. However, there is a caveat. In Theorem 4 we require a tighter bound on N_t.

Similar to previous sections, we argue that when N_t is a random variable, we can extend the same calculations as long as we have $\mathbb{E}\left[(max\{N_t, t \in k\text{th cache update interval}\})^2\right]$ is $\mathcal{O}(T^{\frac{1}{2}-2\beta})$ for each $k \leq T_B$, for the same sub-linear bound result to hold. The structure of the argument would be the same as observed in *Theorems 1 and 2*.

As we move to the final setting, we are interested in the bunched arrivals with N_t possibly random, but the requested files follow an unknown (to the cache) stochastic distribution. Moreover, we relax the condition $\mathcal{C} = 1$ to any finite $\mathcal{C}$ and investigate if FTPL is able to achieve sublinear regret guarantees while ensuring sublinear total computation cost.

In accordance with the stochastic file request assumption, each request $x_{t,i}$ is independently chosen from a probability distribution $\mu = (\mu_1, \mu_2, \ldots, \mu_L)$, where the probability of $x_{t,i}$ being l is denoted as $\mathbb{P}(x_{t,i} = l) = \mu_l$ for each $l \in L$. Without loss of generality, we can assume that $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_L$.

Before we start, we define some notations. We use $\mu_{\mathcal{C}}$ to denote the sum $\sum_{i=1}^{\mathcal{C}} \mu_i$. We also define $\Delta_{j,k}$ as $\mu_j - \mu_k$ for each $j, k \in L$. The mathematical definition of regret for the stochastic file requests case is as described in Section 2.

Theorem 5. *[Setting 3] When the bunched arrivals are arbitrary, and the file requests are stochastic, in the presence*

of computation cost and arbitrary τ_k we have:

$$R_S^{FTPL\left(\eta_k=\alpha\sqrt{\sum_{l=1}^{k}\overline{N}_l}\right)}(T) \leq \beta_1 \sum_{k=1}^{T_B} \overline{A}_1 (\prod_{n=1}^{k-1} \overline{B}_1 + \overline{C}_1) + \mathcal{K} T_B$$

where $\overline{N}_k$ is the number of requests in the kth cache update interval, i.e. $\overline{N}_k = \sum_{(k-1)\tau(T)+1}^{k\tau(T)} N_t$, $\beta_1 = 2\mathcal{C}(L-\mathcal{C})\mu_{C^*}$ $\beta_2 = \Delta_{min}^2/8$ and $\beta_3 = \Delta_{min}^2/32\alpha^2$. Δ_{min} is taken to be $\min_{j\in C^*, k\notin C^*}\{\Delta_{j,k}\}$. and $\overline{A}_1 = \overline{N}_k, \overline{B}_1 = \mathbb{E}\left[e^{-\beta_2 \overline{N}_n}\right]$ and $\overline{C}_1 = \mathbb{E}\left[e^{-\beta_3 \overline{N}_n}\right]$

This theorem gives a general result, which forms a basis for deriving stronger regret bounds via well-established results on Moment Generating Functions. We maintain that the zero computation cost case can just as easily be studied by nullifying the cost effect. We emphasize the fact that the above result caters to the general cache size $\mathcal{C}$ case, which evidently makes an appearance in the value of the bound.

In particular, when $N_i \sim$ Poisson(λ), $\overline{N}_k$ is $\sim$ Poisson($\lambda\tau_k$). $\mathbb{E}\left[e^{-N_i\Delta_{min}^2/8}\right] = M(-\Delta_{min}^2/8)$, where M is the Moment Generating function $\exp[\lambda(e^t-1)]$ for a Poisson random variable. For the sake of notational brevity, we let $\mathbb{E}\left[e^{-N_i\Delta_{min}^2/8}\right] = \exp\left[\lambda(e^{-\Delta_{min}^2/8}-1)\right] = c_1$ and,

$\mathbb{E}\left[e^{-N_i\Delta_{min}^2/32\alpha^2}\right] = \exp\left[\lambda(e^{-\Delta_{min}^2/32\alpha^2}-1)\right] = c_2$. We may observe $c_1 < 1$, $c_2 < 1$. Consequently, for any $p > 0$, $c_1^p < 1$ and $c_2^p < 1$.

To demonstrate the impact of the stochastic assumption on file requests, we introduce a geometrically growing τ_k i.e., τ_k grows as $\tau(1+r)^{k-1}$ considering case 2 of setting 2. Note that we still have $\sum_{k=1}^{T_B}\tau_k = T$, or more specifically, $T_B = \arg\min\{n : \sum_{k=1}^{n}\tau_k > T\}$. We may observe that $T_B = \mathcal{O}(\log \frac{1}{\tau} rT)$

Theorem 6. *[Setting 3] When the bunched arrivals are arbitrary, and the file requests are stochastic, in the presence of computation cost and adaptive [geometrically increasing] τ_k we have:*

$$R_S^{FTPL\left(\eta_k=\alpha\sqrt{\sum_{l=1}^{k}\overline{N}_l}\right)}(T, \mathcal{K}, \tau_k) \leq \mathcal{O}(\log \frac{1}{\tau} rT)$$

when $N_t \sim$ Poisson(λ), $\tau_k = \tau(1+r)^{k-1}$ for all $k \leq T_B$ and $r < \min\{\frac{1}{\langle c_1\rangle_\tau} - 1, \frac{1}{\langle c_2\rangle_\tau} - 1\}$ where $\langle c_1\rangle_\tau = c_1^\tau$ and $\langle c_2\rangle_\tau = c_2^\tau$

Theorem 6 implies that FTPL has a $\mathcal{O}(\log T)$ modified regret and $\mathcal{O}(\log T)$ computation cost when the file requests arrivals are Poisson, which is a significant improvement to previous settings. While this is expected, in most caching applications it is not possible to guarantee that file requests follow a stochastic model; therefore, the scenarios described in Theorems 2 and 4 are of broader practical interest.

IV. Conclusion

We analyzed online caching with constrained cache update frequencies motivated by computational limitations in practical systems. Using a modified regret metric that incorporates computation cost, we studied the performance of the FTPL algorithm under bunched request arrivals. Our results show that FTPL can achieve sublinear regret even when cache updates are performed infrequently, provided that the update intervals are chosen appropriately. Adaptive update schedules further reduce the computation overhead while maintaining favorable regret guarantees. When request arrivals follow stochastic processes such as Poisson distributions, significantly tighter regret bounds can be obtained. These results characterize the tradeoff between caching performance and computational overhead and provide insights for designing efficient cache update strategies in large-scale systems. Overall, the analysis highlights that the primary factor contributing to regret degradation under constrained updates is the adversarial nature of the arrival process rather than the distribution of file requests.

By initiating an investigation into the tradeoff between caching (cache miss) regret and computation cost, this work also opens several directions for future research. In particular, two immediate questions are the characterization of a lower bound on the modified regret and the design of a caching policy that attains this bound. Although FTPL achieves sublinear regret in the considered settings, it is not necessarily optimal when constraints on cache computation are imposed.

References

[1] L. Andrew, S. Barman, K. Ligett, M. Lin, A. Meyerson, A. Roytman, and A. Wierman. A tale of two metrics: Simultaneous bounds on competitiveness and regret. In *Proceedings of the 26th Annual Conference on Learning Theory (COLT)*, 2013.

[2] R. Bhattacharjee, S. Banerjee, and A. Sinha. Fundamental limits of online network-caching, 2020.

[3] N. Cesa-Bianchi, O. Dekel, and O. Shamir. Online learning with switching costs and other adaptive adversaries. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2013.

[4] O. Dekel, J. Ding, T. Koren, and Y. Peres. Bandits with switching costs: $T^{2/3}$ regret. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 459–467, 2014.

[5] F. Faticanti and G. Neglia. Optimistic online caching for batched requests. *Computer Networks*, 244:110341, 2024.

[6] Elad Hazan. Introduction to online convex optimization. *CoRR*, abs/1909.05207, 2019.

[7] P. Joulani, A. Gyorgy, and C. Szepesvári. Online learning under delayed feedback. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013.

[8] A. Kalai and S. Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3):291–307, 2005.

[9] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, S.H. Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. Lrfu: a spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Transactions on Computers*, 50(12):1352–1361, 2001.

[10] R.L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78–117, 1970.

[11] H. B. McMahan and M. Streeter. Delay-tolerant algorithms for asynchronous distributed online learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.