Learning to Cache and Caching to Learn: Regret Analysis of Caching Algorithms

Archana Bura, Desik Rengarajan, Dileep Kalathil, Srinivas Shakkottai, and Jean-Francois Chamberland-Tremblay

Abstract—Crucial performance metrics of a caching algorithm include its ability to quickly and accurately learn a popularity distribution of requests. However, a majority of work on analytical performance analysis focuses on hit probability after an asymptotically large time has elapsed. We consider an online learning viewpoint, and characterize the "regret" in terms of the finite time difference between the hits achieved by a candidate caching algorithm with respect to a genie-aided scheme that places the most popular items in the cache. We first consider the Full Observation regime wherein all requests are seen by the cache. We show that the Least Frequently Used (LFU) algorithm is able to achieve order optimal regret, which is matched by an efficient counting algorithm design that we call LFU-Lite. We then consider the Partial Observation regime wherein only requests for items currently cached are seen by the cache, making it similar to an online learning problem related to the multi-armed bandit problem. We show how approaching this "caching bandit" using traditional approaches yields either high complexity or regret, but a simple algorithm design that exploits the structure of the distribution can ensure order optimal regret. We conclude by illustrating our insights using numerical simulations.

I. INTRODUCTION

Caching is a fundamental aspect of content distribution. Since it is often the case that the same content item is requested by multiple clients over some timescale, replicating and storing content in near proximity to the requesting clients over that timescale can both reduce latency at the clients, as well as enable more efficient usage of network and server resources. Indeed, this is the motivation for a variety of cache eviction policies such as Least Recently Used (LRU), First In First Out (FIFO), RANDOM, CLIMB [1], Least Frequently Used (LFU) [2] etc., all of which attempt to answer a basic question: suppose that you are aware of the timescale of change of popularity, what are the right content items to store?

Viewed from this angle, the problem of caching is simply that there is some underlying unknown popularity distribution (that could change with time) over a library of content items, and the goal of a caching algorithm is to quickly learn which items are most popular and place them in a location that minimizes client latencies. Taking this viewpoint of "caching equals fast online learning of an unknown probability distribution," it is clear that it is not sufficient for a caching algorithm to learn a fixed popularity distribution accurately, *it must also learn it quickly* in order to track the changes on popularity that might happen frequently.

Most work on the performance analysis of caching algorithms has focused on the stationary (long term) hit probabilities under a fixed request distribution. However, such

Authors are with the Department of Electrical and Computer Engineering, Texas A&M University, Texas, USA. Email of the corresponding author: dileep.kalathil@tamu.edu
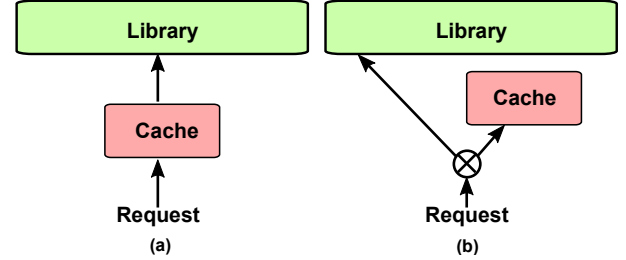
Fig. 1. Request forwarding with (a) Full observation, and (b) Partial observation at the cache.

an approach does not account for the fact that request distributions change with time, and finite time performance is a crucial metric. Suppose that all content items are of the same size, a cache can hold C content items, and the request process consists of independent draws (called the Independent Reference Model (IRM)). Then a genie-aided algorithm that is aware of the underlying popularity distribution would place the top C most popular items in the cache to maximize the hit probability. Yet, any pragmatic caching algorithm needs to learn the popularity distribution as requests arrive, and determine what to cache. The *regret* suffered is the difference in the number of cache hits between the two algorithms. How does the regret scale with the number of requests seen?

The goal of this work is to conduct a systematic analysis of caching from the perspective of regret, with idea that a low-regret algorithm implies fast and accurate learning in finite time, and hence should be usable in a setting where the popularity distribution changes with time. We consider two different caching paradigms, illustrated in Figure 1. In both paradigms, there is a single cache of size C and a much larger library of items of size L. In the first setup that we call the *full observation* model, all requests are first directed towards the cache, and then forwarded to the library if there is a cache miss. Here, the cache sees all requests and can learn the underlying popularity distribution with full observation of the requests. The second model is more akin to a data center with multiple nodes, wherein requests may be forwarded via cache routing towards a node that has the item. We call this case as the *partial observation* model, since the cache only sees requests for items currently cached in it, while the other requests are directly forwarded to the library. Thus, items must be cached in an exploratory manner in order to learn their popularities, and whether they are worth retaining. Can we design regret optimal algorithms that apply to each of these cases?

Main Results

In our analytical model, we consider a system in which one request arrives at each discrete time unit, i.e., the total

number of requests is the same as the elapsed time T. We begin with insight that, under the full observation regime, the empirical frequency is a sufficient statistic of all information on the popularity distribution received thus far. Furthermore, the appropriate use of this estimate is to choose the top C most frequent items to cache. This approach is identical to the LFU, since it evicts the item with the least empirical frequency at each time. Our first result is to show that LFU has an $O(1)$ regret, not only with respect to time T, but also with respect to library size.

While LFU is known to attain high hit rates, it suffers from the fact that the number of counters is the same as the library size, since every request must be counted. This is clearly prohibitive, and has given rise to approximations such as W-LFU [3], which only keeps counts within a moving window of requests, and TinyLFU [2] that uses a sketch for approximate counting. Our next result is to show that these approximations never entirely eliminate the error in estimating the popularity distribution, leading to the worst possible regret of $\Omega(T)$.

We then propose a varient of LFU that we term LFU-Lite, under which we use a moving window of requests to decide whether or not a particular item appears to be popular enough to be counted accurately. Thus, we maintain a counter bank, and only count those content items that meet a threshold frequency in any window of requests thus far. The counter bank size grows in a concave manner with time, and we find its expected size to ensure $O(1)$ regret for a target time T. Thus, given a time constant of change in popularity, we can decide on the ideal number of counters.

We next consider the partial observation regime, wherein the cache can only see requests of items currently cached in it. We relate this problem to that of the classical multi-armed bandit under which actions must be taken to learn the value of pulling the different arms. We first consider an algorithm that builds up the correct posterior probabilities given the requests seen thus far, and caches the most frequent items in a sample of this posterior distribution. Although its empirical performance is excellent, maintaining the full posterior sampling (FPS) quickly becomes prohibitively difficult.

We then consider an algorithm that simply conducts a marginal posterior sampling (MPS) by updating counts only for the items that are in the cache. Here, counts of hits and misses are awarded to the appropriate cached item, but a miss (which manifests itself as no request being made to the cache) is not used to update the posterior distribution of items not in the cache. Clearly, we are not using the information effectively, and this is reflected in the regret scaling as $O(\log T)$. This result is similar to earlier work [4].

We then ask whether we can exploit the structure of the problem to do better? In particular, suppose that we know that requests will follow a certain probability distribution (e.g., Zipf), although we do not know the ranking of items (i.e., we do not know which one is the most popular etc.). We develop a Structured Information (SI) algorithm that considers this information about the distribution to reduce the regret to $O(1)$. We also describe a "Lite" version of the SI algorithm similar to LFU-Lite to reduce the number of counters.

We first verify our analytical results via numerical simula-tions conducted using an IRM model drawn from Zipf distribu-tions with different parameters of library and cache sizes. We also find that Lite-type schemes appear to empirically perform even better than predicted by the analytical results.

We then construct versions of the algorithms that are capable of following a changing popularity distribution by simply "for-getting" counts, which takes the form of periodically halving the counts in the counters. The expectation is that a low regret algorithm, augmented with such a forgetting rule with an appropriately chosen periodicity should be able to track a moving popularity distribution accurately. We conduct trace-based simulations using (non-stationary) data sets obtained from IBM and YouTube, and compare hit performance against the ubiquitous LRU algorithm. We show that the LFU variants outperform LRU, and that incorporating forgetting enhances their hit-rates.

Since the amount of change over time in the existing traces is low, we stress test our algorithms by creating a synthetic trace that has higher changes in popularity over time. Again, we show that the versions of our algorithms that incorporate forgetting are able to track such changing distributions, and are still able to outperform LRU, which builds a case for their eventual adoption.

Related Work

Existing analytical studies of caching algorithms largely follow the IRM model, with the focus being on closed form results of the stationary hit probabilities of LRU, FIFO, RANDOM, and CLIMB [5], [1], [6], [7]. The expressions are often hard to compute for large caches, and approximations have been proposed for larger cache sizes [8]. Of particular interest is the Time-To-Live (TTL) approximation [9], [10], [11], [12] that associates each cached item with a lifetime after which it is evicted. Appropriate choice of this lifetime enables the accurate approximation of different caching schemes [11].

Recent work on performance analysis of caching algorithms has focused on the online learning aspect. For instance, [13] propose TTL-based schemes to show that a desired hit rate can be achieved under non-stationary arrivals. Other work such as [14] characterize the mixing times of several simple caching schemes such as LRU, CLIMB, k-LRU etc. with the goal of identifying their learning errors as a function of time. However, the algorithms studied all have stationary error (they never learn perfectly) and so regret in our context would be $\Omega(T)$.

Information Centric caching has gained much recent inter-est, and is particularly relevant to edge wireless networks. Joint caching and routing is studied in [15] where the objective is to show asymptotic accuracy of the placements, rather than finite time performance that we focus on. Closest to our ideas on the partial observation model is work such as [4], which draws a parallel between bandit algorithms and caching under this setting. However, the algorithms considered are in the manner of the traditional Multi-Armed Bandit (MAB) approach that does not account for problem structure, and hence can only attain $O(\log T)$ regret.

With regard to the MAB problem, Lai and Robbins [16] showed in seminal work the $\Omega(\log T)$ regret lower bound

pertaining to any online learning algorithm. An index based algorithm using the upper confidence idea (UCB1 algorithm) was proposed in [17], which enabled a simple implementation while achieving the optimal regret. The the posterior sampling approach, first proposed by Thompson in [18], has recently been shown to attain optimal regret [19]. For a detailed survey, we point to a monograph [20] and a recent book [21].

Much work also exists on the empirical performance evaluation of caching algorithms using traces gathered from different applications. While several discover fundamental insights [22], [23], [24], our goal in this work is on analytical performance guarantees, and we do not provide a comprehensive review.

II. System Model

We consider the optimal cache content placement problem in a communication network. The library, which is the set of all files, is denoted by $\mathcal{L} = \{1, \ldots, L\}$. We assume for expositional simplicity that all files are of the same size, and that the cache has a capacity of C, i.e., it can store C files at a given time. We denote the popularity of the files by the profile $\mu = (\mu_1, \mu_2, \ldots, \mu_L)$, with $\sum_i \mu_1 = 1$. Without loss of generality, we assume that $\mu_1 > \mu_2 > \cdots > \mu_L$. Let $x(t) \in \mathcal{L}$ be the file request received at time t. We assume that requests are generated independently according to the popularity profile μ, i.e., $\mathbb{P}(x(t) = i) = \mu_i$.

Let $C(t)$ be set of files placed in the cache by the caching algorithm at time t. We say that cache gets a *hit* if $x(t) \in C(t)$ and a *miss* if $x(t) \notin C(t)$. The goal of the caching algorithm is to maximize the expected cumulative hits over time, $\mathbb{E}[\sum_{t=1}^{T} \mathbb{1}\{x(t) \in C(t)\}]$, where the expectation is over all the random choices on $C(t)$ made by the caching algorithm, and the randomness over the requests $x(t)$. Clearly, if popularity distribution μ is known, the optimal caching policy is to place the most popular items in the cache all the time, i.e., $C^*(t) = \mathcal{C}$, where $\mathcal{C} = \{1, 2, \ldots, C\}$. However, in most real world applications, the popularity distribution is unknown to the caching algorithm apriori. So the goal of a caching algorithm is to learn the popularity distribution from the sequential observations, and to place files in the cache by optimally using this available observation at each time in order to maximize the expected cumulative hits.

In the multi-armed bandits literature, it is common to characterize the performance of an online learning algorithm using the metric of *regret*. Regret is defined as the performance loss of the algorithm as compared to the optimal algorithm with complete information. Since $C^*(t) = \mathcal{C}$, the cumulative regret of a caching algorithm after T time steps is defined as

$$R(T) = \sum_{t=1}^{T} \mathbb{1}\{x(t) \in \mathcal{C}\} - \mathbb{1}\{x(t) \in C(t)\}. \quad (1)$$

Let $s(t)$ be the observation available to the caching algorithm at time t and let $h(t) = (s(1), \ldots, s(t-1))$ be the history of observations until time t. The *optimal caching problem* is defined as the problem of finding a policy $\pi(\cdot)$ that maps $h(t)$ to $C(t)$, i.e., $C(t) = \pi(h(t))$, in order to minimize the expected cumulative regret, $\mathbb{E}[R(T)]$.

The choice of the caching policy will clearly depend on the nature of the sequential observations available to it. We consider two different observation structures that are most common in communication networks.

1) **Full Observation:** In the full observation structure, we assume that the caching algorithm is able to observe the file request at each time, i.e., $s(t) = x(t)$. In the setup of a cache and library, this regime corresponds to all requests being sent to the cache, which can then forward the request to the library in case of a miss.

2) **Partial Observation:** In the partial observation structure, the caching algorithm can observe the request only in the case of a hit, i.e., only if the requested item is in the cache already. More precisely, we define $s(t) = x(t)\mathbb{1}\{x(t) \in C(t)\}$ under this observation structure. In the case of a miss, $s(t) = 0$. In the setup of a cache and library, this regime corresponds to the context of information centric caching, wherein requests are forwarded to the cache only if the corresponding content is cached.

We propose different caching algorithms to address the optimal caching problem under these two observation structures.

III. Caching with Full Observation

We first consider the full observation structure where the caching algorithm can observe every file request. Our focus is on a class of algorithms following Least Frequently Used (LFU) eviction, since it uses cumulative statistics of all received requests (unlike other popular algorithms such as Least Recently Used (LRU)), and so is likely to have low regret.

A. LFU Algorithm

At each time t, the LFU algorithm selects the top C requested files until time t and places them in the cache. More precisely, LFU algorithm maintains an empirical estimate of the popularity distribution, denoted by $\hat{\mu}(t) = (\hat{\mu}_1(t), \ldots, \hat{\mu}_L(t))$, defined as

$$\hat{\mu}_i(t) = \frac{1}{t} \sum_{\tau=1}^{t} \mathbb{1}\{x(\tau) = i\}, \quad \forall i \in \mathcal{L}.$$

The files to be placed in cache at time $t+1$, $C_{\text{LFU}}(t+1)$, is then selected as

$$C_{\text{LFU}}(t+1) = \arg \max_C \ (\hat{\mu}_1(t), \ldots, \hat{\mu}_L(t))$$

where $\arg \max_C$ indicates the indices of the top C elements of the vector $\hat{\mu}(t)$.

We now present the finite time performance guarantee for the LFU algorithm.

Theorem 1. *The LFU algorithm has an expected regret of $O(1)$. More precisely,*

$$\mathbb{E}[R(T)] < \min \left(\frac{16}{\Delta_{\min}^2}, \frac{4C(L-C)}{\Delta_{\min}} \right)$$

where $\Delta_{\min} = \mu_C - \mu_{C+1}$.

Remark 1. We note that both terms of the regret upper bound are distribution dependent, i.e., they depend on $\Delta_{\min}$s. Roughly, if $LC < 1/\Delta_{\min}$, then the second term dominates.

We will use the following Lemma for proving Theorem 1.

Lemma 2.
$$\mathbb{P}(\max_i |\hat{\mu}_i(t) - \mu_i| > \epsilon) \leq 2e^{-t\epsilon^2/2}$$

Proof. The proof follows from the Dvoretzky-Kiefer-Wolfowitz inequality [25].

Let $F_i = \sum_{j=1}^{i} \mu_j$ and $\hat{F}_i(t) = \sum_{j=1}^{i} \hat{\mu}_j(t)$ for $i \in \mathcal{L}$. Define $F_i = \hat{F}_i(t) = 0$ for $i < 0$ and $F_i = \hat{F}_i(t) = 1$ for $i > L$. Then, from the Dvoretzky-Kiefer-Wolfowitz inequality [25], for any $\epsilon > 0$, we have
$$\mathbb{P}(\max_i |\hat{F}_i(t) - F_i| > \epsilon) \leq 2e^{-2t\epsilon^2}$$

Now, $\{\max_i |\hat{F}_i(t) - F_i| > \epsilon/2\}$
$$= \{\max_i |\hat{F}_i(t) - F_i| + \max_i |\hat{F}_{i-1}(t) - F_{i-1}| > \epsilon\}$$
$$\supset \{\max_i |(\hat{F}_i(t) - F_i) - (\hat{F}_{i-1}(t) - F_{i-1})| > \epsilon\}$$
$$= \{\max_i |\hat{\mu}_i(t) - \mu_i| > \epsilon\}$$
So, $\mathbb{P}(\max_i |\hat{\mu}_i(t) - \mu_i| > \epsilon) \leq \mathbb{P}(\max_i |\hat{F}_i(t) - F_i| > \epsilon/2)$
$$\leq 2e^{-t\epsilon^2/2}$$
$\square$

We now present the proof of Theorem 1.

Proof. We denote $C_{\text{LFU}}(t)$ just as $C(t)$ for notational convenience. We first show that if $\max_i |\hat{\mu}_i(t) - \mu_i| < \Delta_{\min}/2$, then $C(t) = \mathcal{C}$. Indeed, if $\max_i |\hat{\mu}_i(t) - \mu_i| < \Delta_{\min}/2$, for any $j \in \mathcal{C}$ and for any $k \in \mathcal{L} \setminus \mathcal{C}$,
$$\hat{\mu}_j(t) \geq \mu_j - \Delta_{\min}/2 \geq \mu_C - \Delta_{\min}/2$$
$$\geq \mu_{C+1} + \Delta_{\min}/2 \geq \mu_k + \Delta_{\min}/2 \geq \hat{\mu}_k(t)$$
and hence $C(t) = \mathcal{C}$. Now,
$$\mathbb{E}[R(T)] = \mathbb{E}[\sum_{t=1}^{T} \mathbb{1}\{x(t) \in \mathcal{C}\} - \mathbb{1}\{x(t) \in C(t)\}]$$
$$\leq \mathbb{E}[\sum_{t=1}^{T} \mathbb{1}\{C(t) \neq \mathcal{C}\}] = \sum_{t=1}^{T} \mathbb{P}(C(t) \neq \mathcal{C})$$
$$\leq \sum_{t=1}^{T} \mathbb{P}(\max_i |\hat{\mu}_i(t) - \mu_i| \geq \Delta_{\min}/2)$$
$$\leq \sum_{t=1}^{T} 2e^{-t\Delta_{\min}^2/8} \leq \frac{16}{\Delta_{\min}^2}. \tag{2}$$

We next upper bound $\mathbb{E}[R(T)]$ in a different manner.
$$\mathbb{E}[R(T)] = \mathbb{E}[\sum_{t=1}^{T} \mathbb{1}\{x(t) \in \mathcal{C}\} - \mathbb{1}\{x(t) \in C(t)\}]$$
$$= \mathbb{E}[\sum_{t=1}^{T} \mathbb{E}[\mathbb{1}\{x(t) \in \mathcal{C}\}|\mathcal{C}] - \mathbb{E}[\mathbb{1}\{x(t) \in C(t)\}|C(t)]$$

$$= \mathbb{E}[\sum_{t=1}^{T}(\sum_{j \in \mathcal{C}} \mu_j - \sum_{k \in C(t)} \mu_k)] \tag{3}$$

$$\leq \mathbb{E}[\sum_{t=1}^{T}\sum_{j=1}^{C}\sum_{k=C+1}^{L} \Delta_{j,k} \mathbb{1}\{j \notin C(t), k \in C(t)\}] \tag{4}$$

$$\leq \mathbb{E}[\sum_{t=1}^{T}\sum_{j=1}^{C}\sum_{k=C+1}^{L} \Delta_{j,k} \mathbb{1}(\hat{\mu}_k(t) > \hat{\mu}_j(t))]$$

$$\leq \mathbb{E}[\sum_{t=1}^{T}\sum_{j=1}^{C}\sum_{k=C+1}^{L} \Delta_{j,k} (\mathbb{1}\{\hat{\mu}_j(t) - \mu_j \leq -\Delta_{j,k}/2\}$$
$$+ \mathbb{1}\{\hat{\mu}_k(t) - \mu_k > \Delta_{j,k}/2\})] \tag{5}$$

Using the Hoeffding inequality [26], we obtain
$$\mathbb{P}(\hat{\mu}_j(t) - \mu_j \leq -\Delta_{j,k}/2) \leq e^{-t\Delta_{j,k}^2/2},$$
$$\mathbb{P}(\hat{\mu}_k(t) - \mu_k > \Delta_{j,k}/2) \leq e^{-t\Delta_{j,k}^2/2}$$

Now, continuing form (6) and by taking expectation inside the summation, we obtain

$$\mathbb{E}[R(T)] \leq \sum_{j=1}^{C}\sum_{k=C+1}^{L}\sum_{t=1}^{T} \Delta_{j,k} 2e^{-t\Delta_{j,k}^2/2}$$
$$\leq \sum_{j=1}^{C}\sum_{k=C+1}^{L} \frac{4}{\Delta_{j,k}} \leq \frac{4C(L-C)}{\Delta_{\min}} \tag{6}$$

Combining (2) and (6), we obtain the desired result. $\square$

B. WLFU Algorithm

We showed that the LFU achieves a regret of $O(1)$. However, the implementation of the LFU is expensive in terms of the memory required, because it maintains the popularity estimate for each item in the library ($\hat{\mu}_i(t)$), and the library size L is extremely large for most practical applications. Hence, allocating memory to maintain the popularity distribution estimate for the whole library is impractical.

There are many approaches proposed to address this issue [3], [2]. However, most approaches rely on heuristics-based approximations of the empirical estimate, often with a tight pre-determined constraint on the memory. This leads to non-optimal use of the available information, and could result in poor performance of the corresponding algorithms.

Here, we consider the Window-LFU (WLFU) algorithm [3] that has been proposed as way to overcome the expensive memory requirement of the LFU. WLFU employs a sliding window approach. At each time t, the algorithm keeps track of only the past w file requests. This is equivalent to maintaining a time window from $t - w$ to t, denoted by $W[t - w, t]$. Caching decisions are made based on the file requests that appeared in this window. In particular, the items to be placed in the cache at time t, $C_{WLFU}(t)$, are the top C files with maximum appearances in the window $W[t - w, t]$.

We now show that the expected cumulative regret incurred by WLFU increases linearly in time ($\Omega(T)$), as opposed to the constant regret ($O(1)$) of the standard LFU. Since $\Omega(T)$ is the worst possible regret for any learning algorithm, it

5

suggests that in practice there will occasionally be arbitrarily bad sample paths with many misses.

Theorem 3. *Under the WLFU algorithm,* $\mathbb{E}[R(T)] = \Omega(T)$.

Proof. From the proof of Theorem 1 (c.f. (3)), we have,

$$
\begin{aligned}
\mathbb{E}[R(T)] &= \mathbb{E}[\sum_{t=1}^{T}(\sum_{j \in \mathcal{C}} \mu_j - \sum_{k \in C(t)} \mu_k)] \\
&= \mathbb{E}[\sum_{t=1}^{T}(\sum_{j \in \mathcal{C} \backslash C(t)} \mu_j - \sum_{k \in C(t) \backslash \mathcal{C}} \mu_k)] \\
&\geq \mathbb{E}[\sum_{t=1}^{T} \sum_{k \in C(t) \backslash \mathcal{C}} (\mu_C - \mu_k)] \\
&= \mathbb{E}[\sum_{t=1}^{T} \sum_{k \in \mathcal{L} \backslash \mathcal{C}} (\mu_C - \mu_k) \mathbb{1}\{k \in C(t)\}] \\
&= \sum_{k \in \mathcal{L} \backslash \mathcal{C}} (\mu_C - \mu_k) \mathbb{E}[\sum_{t=1}^{T} \mathbb{1}\{k \in C(t)\}] \\
&= \sum_{k \in \mathcal{L} \backslash \mathcal{C}} (\mu_C - \mu_k) \sum_{t=1}^{T} \mathbb{P}(k \in C(t)) \\
&\geq (\mu_C - \mu_{C+1}) \sum_{t=1}^{T} \mathbb{P}(C+1 \in C(t)) \quad (7)
\end{aligned}
$$

We have, $\mathbb{P}(C+1 \in C(t))$
$\geq \mathbb{P}(\text{all requests in } W[t, t-w] \text{ are } C+1) = (\mu_{C+1})^w$

Now, continuing from (7),

$$
\mathbb{E}[R(T)] \geq (\mu_C - \mu_{C+1})(\mu_{C+1})^w T
$$

$\square$

C. LFU-Lite Algorithm

We now propose a new scheme that we call the LFU-Lite algorithm. Unlike the LFU algorithm, LFU-Lite algorithm does not maintain an estimate of the popularity for each item in the library. Instead, it maintains the popularity estimate only for a subset of the items that it has observed. This approach significantly reduces the memory required as compared to the standard LFU implementation. At the same time, we show that the LFU-Lite achieves an $O(1)$ regret similar to that of the LFU, and thus has a superior performance compared to WLFU which suffers an $\Omega(T)$ regret.

We achieve this 'best of both' performance by a clever combination of a window based approach to decide the items to maintain an estimate, and by maintaining a separate *counter bank* to keep track of these estimates. At each time t, LFU-Lite selects the top C items with maximum appearances in the window of observation $W[t-w, t]$. We denote this set of files as $A(t)$. Let $B(t-1)$ be the set of items in the counter bank at the beginning of t. Then, if any item $j \in A(t)$ is not present in $B(t-1)$, it is added to the counter bank, and the counter bank is updated to $B(t)$. Once an item is placed in the counter bank, it is never removed from the counter bank.

LFU-Lite maintains an estimate of the popularity of each item in the counter bank. The popularity estimate of item $i \in B(t)$, $\hat{\mu}_i(t)$, is defined as

$$
\hat{\mu}_i(t) = \frac{1}{(t - t_i)} \sum_{\tau=t_i+1}^{t} \mathbb{1}\{x(t) \in B(t)\} \quad (8)
$$

where t_i is the time at which the item i has been added to the counter bank. The item to be placed in the cache at time t, $C_{\text{LL}}(t)$, is then selected as

$$
C_{\text{LL}}(t) = \arg \max_C (\hat{\mu}_j(t), j \in B(t))
$$

Description of the LFU-Lite is also given in Algorithm 1.

Algorithm 1 LFU-Lite

---

for $t = 1, \ldots, T$ **do**
  Observe $x(t)$
  Select $A(t)$, the top C files with maximum appearances in the window $W[(t-w)_+, t]$
  for Each $j \in A(t)$ **do**
    if ($j \in A(t)$ is not in $B(t-1)$) **then**
      $t_j \leftarrow t$
      Add file j into $B(t)$
    end if
  end for
  Select the files $C_{\text{LL}}(t) = \arg \max_C (\hat{\mu}_j(t), j \in B(t))$ and place them in the cache
end for

We now present the performance guarantee for the LFU-Lite algorithm.

Theorem 4. *The expected regret under the LFU-Lite algorithm is*

$$
\mathbb{E}[R(T)] \leq \frac{C(L-C)w}{p_{\min}} + \frac{4C(L-C)}{\Delta_{\min}},
$$

where $\Delta_{\min} = \mu_C - \mu_{C+1}$, $p_{\min} = \sum_{n=\mu_{C+1}w+1}^{w} \binom{w}{n} \mu_C^n (1 - \mu_C)^{w-n}$.

Proof. For each item $i \in \mathcal{L}$, $\hat{\mu}_i(t)$ is defined as in (8) for $t > t_i$. Here, we also define $\hat{\mu}_i(t) = 0$ for $t \leq t_i$, before item i enters the counter bank. We note that this is only a proof approach and doesn't influence the implementation of the algorithm. Now, from (4)

$$
\begin{aligned}
\mathbb{E}[R(T)] &\leq \mathbb{E}[\sum_{t=1}^{T} \sum_{j=1}^{C} \sum_{k=C+1}^{L} \Delta_{j,k} \mathbb{1}\{j \notin C(t), k \in C(t)\}] \\
&\leq \mathbb{E}[\sum_{t=1}^{T} \sum_{j=1}^{C} \sum_{k=C+1}^{L} \Delta_{j,k} \mathbb{1}(\hat{\mu}_k(t) > \hat{\mu}_j(t))] \quad (9) \\
&\leq \mathbb{E}[\sum_{t=1}^{T} \sum_{j=1}^{C} \sum_{k=C+1}^{L} \Delta_{j,k} (\mathbb{1}\{\hat{\mu}_j(t) - \mu_j \leq -\Delta_{j,k}/2\} \\
&\qquad\qquad + \mathbb{1}\{\hat{\mu}_k(t) - \mu_k > \Delta_{j,k}/2\})]. \quad (10)
\end{aligned}
$$

Note that the LFU-Lite algorithm incurs a regret at time t if an item $j \in \mathcal{C}$ is not present in the counter bank $B(t)$. This is taken into account in the above expression (c.f. (9)) by defining $\hat{\mu}_j(t) = 0$ for $j \notin B(t)$.

The first term in (10) can be bounded as

$$\mathbb{E}[\sum_{t=1}^{T}\sum_{j=1}^{C}\sum_{k=C+1}^{L}\Delta_{j,k}\mathbb{1}\{\hat{\mu}_j(t)-\mu_j\le -\Delta_{j,k}/2\}]$$

$$=\mathbb{E}[\sum_{j=1}^{C}\sum_{k=C+1}^{L}\mathbb{E}[\sum_{t=1}^{T}\Delta_{j,k}\mathbb{1}\{\hat{\mu}_j(t)-\mu_j\le -\Delta_{j,k}/2\}|t_j]]$$

$$\le\mathbb{E}[\sum_{j=1}^{C}\sum_{k=C+1}^{L}\Delta_{j,k}(t_j+\mathbb{E}[\sum_{t=t_j}^{T}\mathbb{1}\{\hat{\mu}_j(t)-\mu_j\le -\Delta_{j,k}/2\}|t_j])]$$

$$\le\mathbb{E}[\sum_{j=1}^{C}\sum_{k=C+1}^{L}\Delta_{j,k}(t_j+\sum_{t=t_j}^{T}\mathbb{P}(\hat{\mu}_j(t)-\mu_j\le -\Delta_{j,k}/2|t_j))]$$

$$\le\mathbb{E}[\sum_{j=1}^{C}\sum_{k=C+1}^{L}\Delta_{j,k}(t_j+\sum_{t=t_j}^{T}e^{-(t-t_j)\Delta_{j,k}^2/2}]$$

$$\le\sum_{j=1}^{C}\sum_{k=C+1}^{L}(\Delta_{j,k}\mathbb{E}[t_j]+\frac{2}{\Delta_{j,k}}). \qquad (11)$$

Similarly, the second term in (10) can be bounded as

$$\mathbb{E}[\sum_{t=1}^{T}\sum_{j=1}^{C}\sum_{k=C+1}^{L}\Delta_{j,k}\mathbb{1}\{\hat{\mu}_k(t)-\mu_k > \Delta_{j,k}/2\}]$$

$$=\mathbb{E}[\sum_{j=1}^{C}\sum_{k=C+1}^{L}\mathbb{E}[\sum_{t=1}^{T}\Delta_{j,k}\mathbb{1}\{\hat{\mu}_k(t)-\mu_k > \Delta_{j,k}/2\}|t_k]]$$

$$=\mathbb{E}[\sum_{j=1}^{C}\sum_{k=C+1}^{L}\Delta_{j,k}\sum_{t=t_k}^{T}\mathbb{P}(\hat{\mu}_k(t)-\mu_k > \Delta_{j,k}/2|t_k)]$$

$$\le\mathbb{E}[\sum_{j=1}^{C}\sum_{k=C+1}^{L}\Delta_{j,k}\sum_{t=t_k}^{T}e^{-(t-t_k)\Delta_{j,k}^2/2}]$$

$$\le\sum_{j=1}^{C}\sum_{k=C+1}^{L}\frac{2}{\Delta_{j,k}}. \qquad (12)$$

Combining (11) and (12) we obtain

$$\mathbb{E}[R(T)]\le\sum_{j=1}^{C}\sum_{k=C+1}^{L}\Delta_{j,k}\mathbb{E}[t_j]+\sum_{j=1}^{C}\sum_{k=C+1}^{L}\frac{4}{\Delta_{j,k}} \qquad (13)$$

It only then remains to bound $\mathbb{E}[t_j]$ for $j\in\mathcal{C}$, which can easily be shown to satisfy

$$\mathbb{E}[t_j]\le\sum_{t=1}^{\infty}(1-p_j)^{\lceil t/w\rceil}\le\sum_{k=1}^{\infty}w(1-p_j)^k\le w/p_j. \qquad (14)$$

where p_j is the probability that item j is selected in a given window.

Combining (15) and (14), we obtain

$$\mathbb{E}[R(T)]\le\sum_{j=1}^{C}\sum_{k=C+1}^{L}\frac{w\Delta_{j,k}}{p_j}+\sum_{j=1}^{C}\sum_{k=C+1}^{L}\frac{4}{\Delta_{j,k}}$$

$$\le\frac{C(L-C)w}{p_{\min}}+\frac{4C(L-C)}{\Delta_{\min}}. \qquad (15)$$

$\square$

Proposition 5. *The growth of the expected size of the counter bank as a function of time is concave.*

Proof. Let $\bar{p}_j^t$ be the probability with which file i enters the counter bank by time t. Note that $\bar{p}_i^t = 1-(1-p_i)^{\lceil t/w\rceil}$. Where $p_i = \sum_{n=\mu_{C+1}w+1}^{w}\binom{w}{n}\mu_i^n(1-\mu_i)^{w-n}$ is the probability that item i enters the counter bank in any given window.

$$\mathbb{E}[B(t)]=\mathbb{E}[\sum_{i=1}^{L}\mathbb{1}_{\{i\in B(t)\}}]=\sum_{i=1}^{L}\bar{p}_i^t=\sum_{i=1}^{L}1-(1-p_i)^{\lceil t/w\rceil} \qquad (16)$$

Observe that $1-(1-p_i)^{\lceil t/w\rceil}$ is concave in t and $E[B(t)]$ is a sum of L concave functions, and is hence concave. $\square$

Remark 2. Intuitively, the counter bank will keep the counts only for more popular items. It is also straight forward to show that the expected size of the counter bank decreases with the window length w. To see this, consider two different window length w_1 and w_2 such that $w_1\ge w_2$. Let $p_i(w)$ be the probability that item i enters the counter bank in any given window when the window of length is w. Then, for $i\notin\mathcal{C}$, $p_i(w_1)\le p_i(w_2)$. Intuitively, larger window length leads to more observations and hence to smaller probability of observing item $i\notin\mathcal{C}$ more than the threshold $\mu_{C+1}w$. Now, $(1-p_i(w_2))^{\lceil t/w_2\rceil}\le(1-p_i(w_1))^{\lceil t/w_2\rceil}\le(1-p_i(w_1))^{\lceil t/w_1\rceil}$. So, $1-(1-p_i(w_2))^{\lceil t/w_2\rceil}\ge 1-(1-p_i(w_1))^{\lceil t/w_1\rceil}$. Hence, the contribution of $i\notin\mathcal{C}$ to the expected size of the counter bank according (16) is smaller for larger window length. The exact dependence of $\mathbb{E}[B(t)]$ on w is cumbersome to characterize. We, however, illustrate this through extensive simulations in Section V.

IV. CACHING WITH PARTIAL OBSERVATION

We now consider the problem of optimal caching under the partial observation regime. As described earlier, here the algorithm can observe a file request only if the requested file is already in the cache. Hence, the caching algorithm has to perform an active *exploration* by placing a file in the cache for a sufficiently long time to learn its popularity, in order to decide if that file belongs to the set of the most popular files. This procedure is in sharp contrast to the full observation structure where the popularity of each file in the library can be learned (improved) after each time step.

However, the exploration is costly because the algorithm incurs regret every time that a sub-optimal file is placed in the cache for exploration. Hence, the algorithm also has to perform an active *exploitation*, i.e., place the most popular items according to the current estimate in the cache. The optimal *exploration vs exploitation* trade-off for minimizing the regret is at the core of most online learning algorithms. The Multi-Armed Bandit (MAB) model is a canonical formalism for this class of problems. Here, there are multiple arms (actions) that yield random rewards independently over time, with the (unknown) mean of arm i being μ_i. The objective is to learn the mean reward of each arm by exploration and maximize cumulative reward by exploitation.

A. Caching Bandit with Full Posterior Sampling

Posterior sampling based algorithms for MAB [19], [27] typically use a Beta prior (with Bernoulli likelihood) or

Gaussian prior (with Gaussian likelihood) in order to exploit the the conjugate pair property of the prior and likelihood (reward) distributions. Hence, the posterior at any time will have the same form as the prior distribution, albeit with different parameters. This provides a computationally tractable and memory efficient way to keep track of the posterior distribution evolution. However, in the optimal caching problem, the unknown popularity vector μ has interdependent components through the constraint $\sum_i \mu_i = 1$. Hence, standard prior distributions like Beta will not be able to capture the full posterior evolution in the caching problem.

We use a Dirichilet prior on the popularity distribution $\mu = (\mu_1, \dots, \mu_L)$, parametrized by $\alpha = (\alpha_1, \dots, \alpha_L)$. More precisely,

$$f_0(\mu; \alpha) = \frac{1}{B(\alpha)} \prod_{i=1}^{L} \mu_i^{\alpha_i - 1}, \text{ where, } B(\alpha) = \frac{\prod_{i=1}^{L} \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{L} \alpha_i)}$$

and $\Gamma(\cdot)$ is the Gamma function.

Let f_t be the posterior distribution at time t with parameter $\alpha(t)$. The posterior is updated according to the observed information $s(t)$. In the case of a hit, the file request $x(t)$ is observed and $s(t) = x(t)$. It is easy to see that the correct posterior update is $\alpha(t) = \alpha(t) + e_{x(t)}$, where $e_{x(t)}$ is the unit vector with non-zero element at index $x(t)$.

The posterior update is complex in the case of the cache miss. Given the current parameter $\alpha(t) = \alpha$, the probability of a cache miss is given by

$$\mathbb{P}(s(t) = 0) = \sum_{j \notin C(t)} \Pr(x(t) = j)$$
$$= \sum_{j \notin C(t)} \int_S \mu_j \frac{\prod_{i=1}^{K} \mu_i^{\alpha_i - 1}}{B(\alpha)} dP = \frac{\sum_{j \notin C(t)} \alpha_j}{\sum_{i=1}^{L} \alpha_i}$$

Then we can show that the posterior distribution can be computed as

$$f(\mu | s(t) = 0) = \mathbb{P}(s(t) = 0 | \mu) f(\mu; \alpha) / \mathbb{P}(s(t) = 0)$$
$$= \frac{1}{\sum_{j \notin C(t)} \alpha_j} \sum_{j \notin C(t)} \alpha_j f(\mu; \alpha + e_j)$$

Algorithm 2 CB-FPS Algorithm

Initialize the prior distribution f_0
for $t = 1, \dots, T$ **do**
    Sample $\hat{\mu}(t) \sim f_t(\cdot)$
    Select $C_{\text{FPS}}(t) = \arg \max_C \hat{\mu}(t)$
    Receive the observation $s(t)$
    Update the posterior $f_{t+1}(\mu) \propto \mathbb{P}(s(t) | \mu) f_t(\mu)$
end for

Hence, with each miss, the algorithm needs to store a set of size $(L - C)$ consisting of Dirichlet parameters. As the number of misses increases, the memory required to store these parameters will increase exponentially. Hence, the full posterior update algorithm is infeasible from an implementation perspective.

We present the CB-FPS in Algorithm 2. In Section V, we will see that a Monte Carlo version of this algorithm can be implemented for small values of L and C, which seems to achieve an $O(1)$ regret. A rigorous proof that shows such regret, even in some special cases and by neglecting computational tractability, is an interesting open problem.

B. Caching Bandit with Marginal Posterior Sampling

We now propose an algorithm that only performs a marginal posterior update. Instead of maintaining a Dirichlet prior for the popularity vector μ, we use a Beta prior for the popularity of each *individual* item μ_i. The CB-MPS is described in Algorithm 3.

Algorithm 3 CB-MPS Algorithm

Initialize $\alpha_i(0) = 1, \beta_i(0) = 1, \forall i \in \mathcal{L}$.
for $t = 1, \dots, T$ **do**
    Generate samples $\hat{\mu}_i(t) \sim \text{Beta}(\alpha_i(t), \beta_i(t))$
    $C_{\text{MPS}}(t) \leftarrow \arg \max_C \hat{\mu}(t)$
    if $x(t) \in C(t)$ **then**
        $\alpha_{x(t)}(t+1) \leftarrow \alpha_{x(t)}(t) + 1$
        $\beta_i(t+1) \leftarrow \beta_i(t) + 1, \forall i \in C(t), i \neq x(t)$
    end if
end for

We now provide a performance guarantee for the CB-MPS algorithm.

Theorem 6. *Under marginal posterior sampling algorithm,* $\mathbb{E}[R(T)] = O((L - C)C \log T)$.

We omit the proof of this theorem because the analysis is similar to that of multi-payer multi-armed bandit algorithm. In particular, the posterior sampling method proposed in [28] can be used with small modifications to show the above result.

C. Caching Bandit with Structural Information

Even though the MPU algorithm is easy to implement, it suffers an $O(\log T)$ regret, which is much worse than the $O(1)$ regret incurred by LFU and LFU-Lite. This is due to the partial observation structure that limits the rate of learning. We now propose an algorithm that we call Caching Bandit with Structural Information (CB-SI). We show that with a minimal assumption on the availability of the structural information about the popularity distribution, CB-SI can achieve an $O(1)$ regret even in the partial observation regime.

We assume that the algorithm knows the value of μ_C and $\Delta_{\min}$, the popularity value of the Cth most popular item and the optimality gap. Note that we do not assume knowledge of the identity of the Cth most popular file. We note that our proof approach follows the techniques developed in [29], which can be considered as a special case with $C = 1$. CB-SI algorithm is given in Algorithm 4.

Theorem 7. *The expected cumulative regret of CB-SI Algorithm is,*

$$\mathbb{E}[R(T)] \leq C \sum_{j \in \mathcal{L} \setminus \mathcal{C}} \left(\frac{2}{\Delta^2} + \frac{4}{\Delta^2} [4 + \frac{32}{\Delta^2} \exp(-\frac{\Delta^2}{8})] \right),$$

Algorithm 4 CB-SI Algorithm

Initialize $\alpha_i(0) = 0, \beta_i(0) = 0, \hat{\mu}_i(0) = 1/L, \forall i \in \mathcal{L}$.
Initialize $n_i(t) = 0, \forall i \in \mathcal{L}$
for $t = 1, \ldots, T$ **do**
    Compute the set $A(t) = \{i \in \mathcal{L} : \hat{\mu}_i(n_i(t)) \geq \mu_C - \frac{\Delta}{2}\}$
    if $|A(t)| \geq C$ **then**
        $C_{SI}(t) = \arg\max_C \hat{\mu}_j(n_j(t), j \in A(t))$
        $Z_t \leftarrow 1$
    else
        For each $i \in \mathcal{L} \setminus A(t)$, compute

$$p_i(t) = c/(\mu_C - \hat{\mu}_i(n_i(t)))^2$$
$$c = \sum_{i \in \mathcal{L} \setminus C(t)} 1/(\mu_C - \hat{\mu}_i(n_i(t)))^2$$

        Sample $C - |A(t)|$ elements from the set $\mathcal{L} \setminus A(t)$ according to the probability $p_i(t)$. Denote these elements as $B(t)$
        $C_{SI}(t) = A(t) \cup B(t)$
        $Z_t \leftarrow 2$
    end if
    Place the files $C_{SI}(t)$ in the cache
    if $x(t) \in C(t)$ **then**
        $\alpha_{x(t)}(t+1) \leftarrow \alpha_{x(t)}(t) + 1$
        $\beta_i(t+1) \leftarrow \beta_i(t) + 1, \forall i \in C(t), i \neq x(t)$
    end if
    $n_i(t+1) = \alpha_i(t+1) + \beta_i(t+1)$
    $\hat{\mu}_i(n_i(t+1)) = \alpha_i(t+1)/n_i(t+1)$
end for

---

where $\Delta = \mu_C - \mu_{C+1}$.

Proof. We denote $\Delta_j = \mu_C - \mu_j$. In this proof, we will show that the expected regret of CB-SI algorithm is bounded above by a constant. From the proof of Theorem 1 (c.f. (6)), we get that the expected regret is bounded as below.

$$\mathbb{E}[R(T)] \leq \mathbb{E}[\sum_{t=1}^{T} \sum_{i=1}^{C} \sum_{j=C+1}^{L} \Delta_{j,k} \mathbb{1}\{i \notin C(t), j \in C(t)\}]$$

$$\leq C\, \mathbb{E}[\sum_{t=1}^{T} \sum_{j=C+1}^{L} \mathbb{1}\{j \in C(t)\}] = C \sum_{j=C+1}^{L} \sum_{t=1}^{T} \mathbb{P}(j \in C(t))$$

$$= C \sum_{j=C+1}^{L} \sum_{t=1}^{T} (\mathbb{P}(\hat{\mu}_j(n_j(t)) > \mu_C - \Delta_j/2, j \in C(t))$$

$$+ \mathbb{P}(\hat{\mu}_j(n_j(t)) \leq \mu_C - \Delta_j/2, j \in C(t))). \quad (17)$$

We address each term in the above summation separately. First, observe that for any $j \in \{C+1 \ldots L\}$, the following inequality holds.

$$\sum_{t=1}^{T} \mathbb{P}(\hat{\mu}_j(n_j(t)) > \mu_C - \Delta_j/2, j \in C(t))$$

$$\leq \sum_{t=1}^{T} \mathbb{P}(\hat{\mu}_j(n_j(t)) > \mu_j + \Delta_j/2, j \in C(t))$$

$$\leq \sum_{t=1}^{T} \mathbb{P}(\hat{\mu}_j(t) > \mu_j + \Delta_j/2) \overset{(a)}{\leq} \sum_{t=1}^{T} e^{-\Delta_j^2 t/2} \leq \frac{2}{\Delta_j^2}, \quad (18)$$

where the inequality (a) follows from Hoeffding's inequality. For bounding the second term in (17), we use the policy definition. Since $\Delta_j \geq \Delta$, the first inequality follows trivially. The equality (b) follows from the fact that when the mean estimate of the jth item is smaller than $\mu_C - \frac{\Delta}{2}$, the only means by which it can enter the cache is through exploration part of the algorithm, which is denoted by $Z_t = 2$.

$$\mathbb{P}(\hat{\mu}_j(n_j(t)) \leq \mu_C - \frac{\Delta_j}{2}, j \in C(t))$$

$$\leq \mathbb{P}(\hat{\mu}_j(n_j(t)) \leq \mu_C - \frac{\Delta}{2}, j \in C(t))$$

$$\overset{(b)}{=} \mathbb{P}(\hat{\mu}_j(n_j(t)) \leq \mu_C - \frac{\Delta}{2}, j \in C(t), Z_t = 2)$$

$$\overset{(c)}{=} \mathbb{P}(j \in C(t) | \hat{\mu}_j(n_j(t)) \leq \mu_C - \frac{\Delta}{2}, Z_t = 2)$$

$$\mathbb{P}(\hat{\mu}_j(n_j(t)) \leq \mu_C - \frac{\Delta}{2}, Z_t = 2)$$

$$= p_{j,t} \mathbb{P}\left(\hat{\mu}_j(n_j(t)) \leq \mu_C - \frac{\Delta}{2}, Z_t = 2\right)$$

$$= \mathbb{E}\left[p_{j,t} \mathbb{1}\{\hat{\mu}_j(n_j(t)) \leq \mu_C - \frac{\Delta}{2}, Z_t = 2\}\right]$$

$$= \mathbb{E}\left[\frac{p_{j,t}}{p_{i,t}} p_{i,t} \mathbb{1}\{\hat{\mu}_j(n_j(t)) \leq \mu_C - \frac{\Delta}{2}, Z_t = 2\}\right],$$

for any $i \in \mathcal{C}$. Note that, in the equality (c), we used the definition of $p_{j,t}$. Now, substituting the value for the sampling probability $p_{i,t}$, we obtain,

$$\leq \mathbb{E}\left[\frac{|\mu_C - \hat{\mu}_i(n_i(t))|^2}{(\frac{\Delta}{2})^2} p_{i,t} \mathbb{1}\{\hat{\mu}_j(n_j(t)) \leq \mu_C - \frac{\Delta}{2}, Z_t = 2\}\right]$$

$$\leq \frac{4}{\Delta^2} \mathbb{E}\left[|\mu_C - \hat{\mu}_i(n_i(t))|^2 p_{i,t} \mathbb{1}\{Z_t = 2\}\right]$$

$$\leq \frac{4}{\Delta^2} \mathbb{E}\left[|\mu_C - \hat{\mu}_i(n_i(t))|^2 \right.$$
$$\left. \mathbb{P}\left(i \in C(t) | \hat{\mu}_i(n_i(t)) \leq \mu_C - \frac{\Delta}{2}, Z_t = 2\right)\right]$$

$$= \frac{4}{\Delta^2} \mathbb{E}\left[|\mu_C - \hat{\mu}_i(n_i(t))|^2 \right.$$
$$\left. \mathbb{E}\left[\mathbb{1}\{i \in C(t)\} | \hat{\mu}_i(n_i(t)) < \mu_C - \frac{\Delta}{2}, Z_t = 2\right]\right]$$

$$= \frac{4}{\Delta^2} \mathbb{E}\left[|\mu_C - \hat{\mu}_i(n_i(t))|^2 \right.$$
$$\mathbb{E}\left[\mathbb{1}\{i \in C(t), \hat{\mu}_i(n_i(t)) < \mu_C - \frac{\Delta}{2}\}| \right.$$
$$\left.\left. \{\hat{\mu}_i(n_i(t)) < \mu_C - \frac{\Delta}{2}, Z_t = 2\}\right]\right]$$

$$= \frac{4}{\Delta^2} \mathbb{E}\left[|\mu_C - \hat{\mu}_i(n_i(t))|^2 \right.$$
$$\left. \mathbb{1}\{i \in C(t), \hat{\mu}_i(n_i(t)) < \mu_C - \frac{\Delta}{2}\}\right]. \quad (19)$$

Here, the inequalities follow from the properties of conditional expectation. Now, we obtain a bound for the second term in

(17), using (19), as below.

$$\sum_{t=1}^{T} \mathbb{E}\left[|\mu_C - \hat{\mu}_i(n_i(t))|^2 1\{\hat{\mu}_i(n_i(t)) < \mu_C - \frac{\Delta}{2}, i \in C(t)\}\right]$$

$$\leq \sum_{t=1}^{T} \mathbb{E}\left[|\mu_C - \hat{\mu}_i(t)|^2 1\{|\hat{\mu}_i(t) - \mu_C| > \frac{\Delta}{2}\}\right]$$

$$= \sum_{t=1}^{T} \int_0^{\infty} \mathbb{P}(|\mu_C - \hat{\mu}_i(t)|^2 1\{|\hat{\mu}_i(t) - \mu_C| > \frac{\Delta}{2}\} \geq x)dx$$

$$= \sum_{t=1}^{T} \int_0^{\infty} \mathbb{P}(|\mu_C - \hat{\mu}_i(t)|^2 1\{|\hat{\mu}_i(t) - \mu_C|^2 > \frac{\Delta^2}{4}\} \geq x)dx$$

$$= \sum_{t=1}^{T} \left[\int_0^{\frac{\Delta^2}{4}} \mathbb{P}\left(|\mu_C - \hat{\mu}_i(t)|^2 \right.\right.$$

$$1\{|\hat{\mu}_i(t) - \mu_C|^2 > \frac{\Delta^2}{4}\} \geq x\right) dx$$

$$\left. + \int_{\frac{\Delta^2}{4}}^{\infty} \mathbb{P}(|\mu_C - \hat{\mu}_i(t)|^2 1\{|\hat{\mu}_i(t) - \mu_C|^2 > \frac{\Delta^2}{4}\} \geq x)dx\right]$$

$$= \sum_{t=1}^{T} \left[\int_0^{\frac{\Delta^2}{4}} \mathbb{P}(|\mu_C - \hat{\mu}_i(t)|^2 \geq x, |\hat{\mu}_i(t) - \mu_C|^2 > \frac{\Delta^2}{4})dx\right.$$

$$\left. + \int_{\frac{\Delta^2}{4}}^{\infty} \mathbb{P}(|\mu_C - \hat{\mu}_i(t)|^2 1\{|\hat{\mu}_i(t) - \mu_C|^2 > \frac{\Delta^2}{4}\} \geq x)dx\right]$$

$$= \sum_{t=1}^{T} \left[\int_0^{\frac{\Delta^2}{4}} \mathbb{P}\left(|\hat{\mu}_i(t) - \mu_C|^2 > \frac{\Delta^2}{4}\right) dx\right.$$

$$\left. + \int_{\frac{\Delta^2}{4}}^{\infty} \mathbb{P}\left(|\mu_C - \hat{\mu}_i(t)|^2 1\{|\hat{\mu}_i(t) - \mu_C|^2 > \frac{\Delta^2}{4}\} \geq x\right) dx\right]$$

$$= \sum_{t=1}^{T} \left[\frac{\Delta^2}{4}\mathbb{P}\left(|\hat{\mu}_i(t) - \mu_C|^2 > \frac{\Delta^2}{4}\right) + \right.$$

$$\left. \int_{\frac{\Delta^2}{4}}^{\infty} \mathbb{P}\left(|\mu_C - \hat{\mu}_i(t)|^2 1\{|\hat{\mu}_i(t) - \mu_C|^2 > \frac{\Delta^2}{4}\} \geq x\right) dx\right]$$

$$= \sum_{t=1}^{T} \left[2\frac{\Delta^2}{4}e^{-\frac{t\Delta^2}{8}} + \int_{\frac{\Delta^2}{4}}^{\infty} \Pr\{|\mu_C - \hat{\mu}_i(t)|^2 \geq x\}dx\right]$$

$$\leq 4 + \frac{32}{\Delta^2}\exp(-\frac{\Delta^2}{8}) \tag{20}$$

Combining equations (17),(19),(18),(20), we observe:

$$\mathbb{E}[R(T)] \leq C \sum_{j \in \mathcal{L} \setminus \mathcal{C}} \left[\frac{2}{\Delta^2} + \frac{4}{\Delta^2}[4 + \frac{32}{\Delta^2}\exp(-\frac{\Delta^2}{8})]\right]$$

□

Remark 8. *We introduce another version of CB-SI algorithm, which is similar in spirit to LFULite. Following a similar rule to LFULite, we maintain a window of the W past observations, and at each time, the C most frequently requested items in the window are added to the counter bank, if those items are not already present in it. The mean estimates of CB-SI are calculated only for items in the counter bank. We call this algorithm as CB-SILite. In Section V-B, we will observe that CB-SILite drastically reduces the number of counters needed to give a similar hit performance to CB-SI.*

V. SIMULATIONS

In this section, we start by conducting simulations with requests generated under the IRM model to verify the insights on regret obtained in the earlier sections. We then use two data traces to compare the performance of our proposed algorithms when exposed to a non-stationary arrival process. Since these requests change with time, we modify the algorithms to "forget" counts, by halving the counts at a fixed periodicity. In the full observation regime, we also compare the performance against LRU, which is widely deployed and implicitly has a finite memory (i.e., it automatically "forgets"). We also further explore the reaction of our approaches to non-stationary requests by creating a synthetic trace that exhibits changes at a faster timescale than the data traces.

A. IRM Simulations

1) Full Observation: We first conduct simulations for an IRM request process following a Zipf distribution with parameter β under the full observation setting. Figure 2 compares the regret suffered by LFU, WLFU and LFU-Lite for $C = 10, L = 1000, W = C^2 \log L$ [3] and $\beta = 1$. As expected, the regret suffered by the WLFU algorithm grows linearly with time, while LFU and LFU-Lite suffer a constant regret. Figure 3 shows the growth of the number counters used to keep an estimate of files. The merits of LFU-Lite are clearly seen here, as it uses approximately 35 counters to achieve a constant regret, while LFU uses 1000.

The growth of counters and the regret suffered by LFU-Lite depends on W, the window of observation. In Figures 4 and 5, we compare the growth of regret and counters with W for $L = 1000, C = 10, \beta = 1$. We see that the number of counters is essentially unchanged for a wide range of W, indicating a robustness to windowing as long as it is sufficiently large.

The key advantage of the LFU-Lite algorithm is that it suffers a constant regret, while keeping track of fewer items even for large library sizes. This is clearly seen in Figure 6 and 7. LFU-Lite only keeps track of approximately 45 items while the LFU algorithm keeps track of almost all items.

2) Partial Observation: Figure 8 shows the cumulative regret performance for CB-SI, CB-MPS and CB-FPS algorithms. We see that both FPS and SI versions have constant regret, whereas the MPS approach has increasing regret consistent with our analysis. While we are forced to keep L and C low in Figure 8 due to the complexity of the FPS approach, Figure 9 shows the cumulative regret performance for $L = 1000, 10000$, for CB-MPS, CB-SI and CB-SILite algorithms. As expected, the SI approach has constant regret. CB-SILite also suffers constant regret, while being a little worse compared to CB-SI. CB-MPS algorithm has logarithmic regret.

Finally, Figure 10 shows the number of counters used by CB-SI and CB-SILite algorithms for $L = 1000, 5000, 10000$, with $C = 10, \beta = 1$. The number of counters used by CB-SILite is very less even for large library sizes, compared to CB-SI.
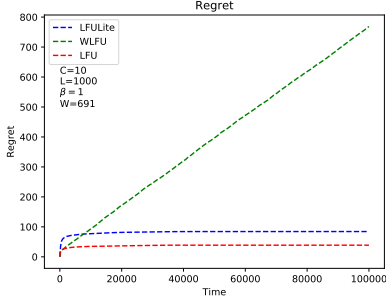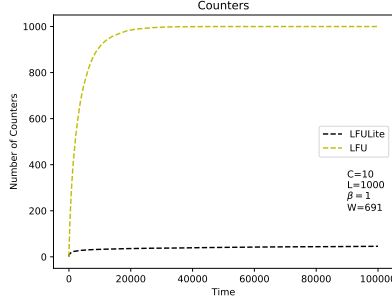
Fig. 2. Regret of LFU, WLFU, LFU-Lite

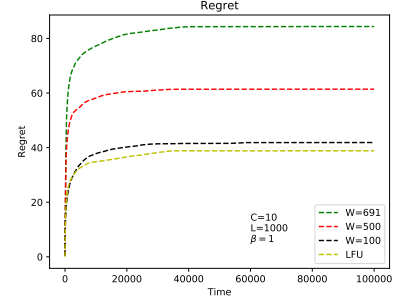Fig. 3. Growth of Counters for LFU, LFU-Lite
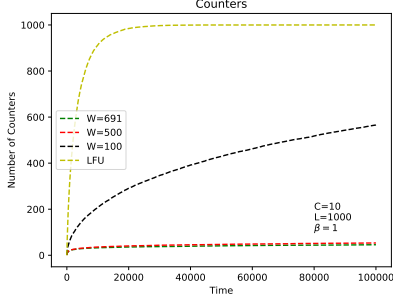

Fig. 4. Regret of LFU-Lite for varying W

Fig. 5. Growth of Counters for varying W
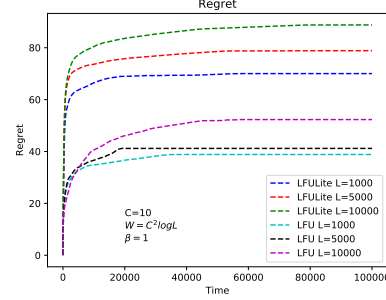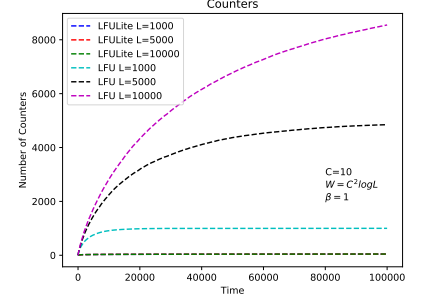

Fig. 6. Regret of LFU-Lite for varying L

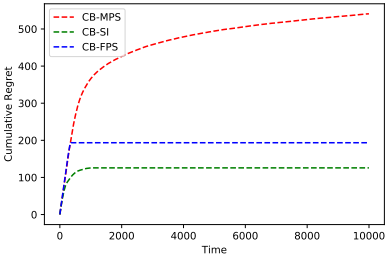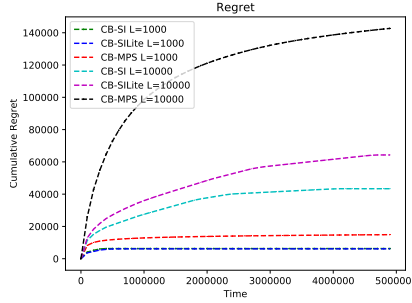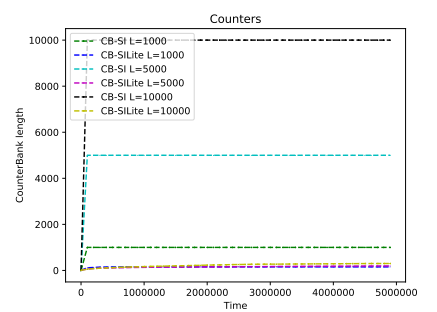Fig. 7. Growth of Counters for varying L

Fig. 8. Cumulative Regret performance comparison for $L = 100, C = 1, \beta = 2$

Fig. 9. Regret of CB-SI, CB-SILite, CB-MPS for $C = 10, \beta = 1$

Fig. 10. Growth of Counters for varying L

B. Trace-Based Simulations

We next conduct trace-based simulations using real world data. The description of the traces that we use in this work is given below.

1) **IBM trace:** This trace is obtained from [30]. It contains a total of 1 million requests to an IBM web server for a library of size 43857.

2) **YouTube trace:** This trace is obtained from [31]. It contains information about the requests made for 161085 newly created YouTube videos each week over 20 weeks. From the data, we compute the popularity distribution of the videos for each week, and obtain 50000 samples from each week's distribution. IN this manner, we create an access trace in which the request distribution changes over each set of 50000 requests. We run this trace for 1 million requests.

3) **Synthetic trace for changing popularity distribution:** We observe that the content popularities in the real world traces change quite slowly. Our goal is also to

understand the impact of non-stationarity in the request arrival process on the hit rate performance of the proposed algorithms. To obtain a reasonable amount of non-stationarity in the popularity of items, we generate a synthetic access trace that changes the popularities periodically. To create this trace, we use a Zipf distribution with parameter 1 to sample 1 million requests in the following manner. For every 100000 requests, we swap the probabilities of top 10000 items in the access distribution cyclically, in steps of 500. This approach results in considerable change in the distribution of the request arrivals for the top 10000 items in the library.

1) Full Observation: We compare the hit rates of LRU, LFU, and LFULite algorithms on the three traces. The size of the window for LFULite is chosen $O(C \log L)$, following the suggestions in [3]. Figure 11 shows the hit rates of LRU, LFU and LFULite on the IBM trace. We observe that LFU and LFULite outperform LRU. Moreover, LFULite gives the same performance as LFU, while using only a fraction of counters.
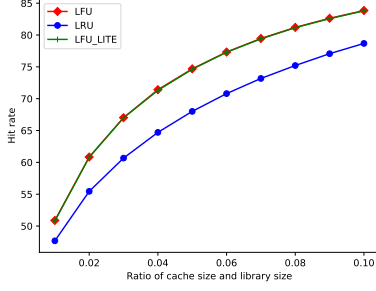
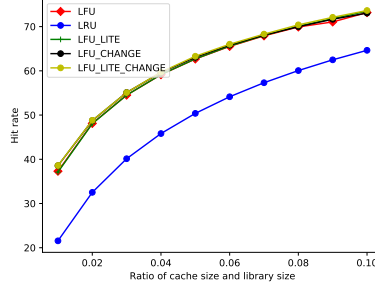Fig. 11. Hit rates for Full observation for IBM trace

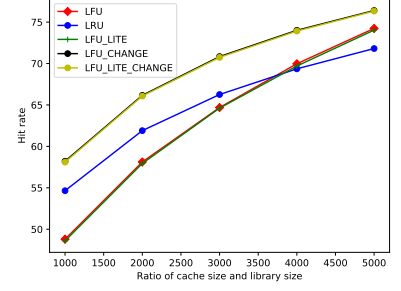Fig. 12. Hit rates for Full observation for YouTube trace

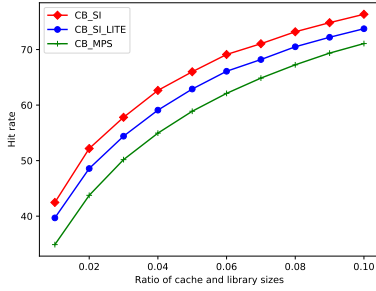Fig. 13. Hit rates for Full observation for Change trace

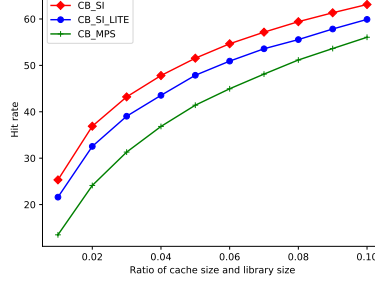Fig. 14. Hit rates for partial observation, IBM trace

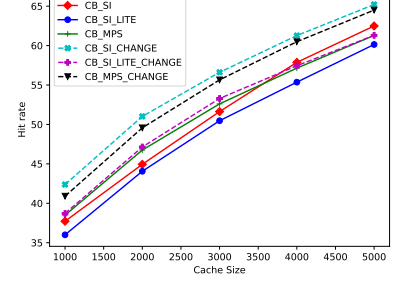Fig. 15. Hit rates for Partial observation, YouTube trace

Fig. 16. Hit rates for partial observation, Change trace

Cache Size	IBM	Youtube	Synthetic
2%	15.07%	13.25%	26.66%
4%	17.5%	23.14%	38.25%
6%	26.56%	30.40%	49.42%
8%	29.26%	36.33%	55.78%
10%	31.84%	42.52%	62.28%

TABLE I
COUNTER BANK SIZE FOR LFULITE, FULL OBSERVATION

Cache Size	IBM	Youtube	Synthetic
2%	8.39%	8.5%	8.4%
4%	13.24%	13.6%	14.09%
6%	16.51%	17.6%	18.22%
8%	19.39%	21.23%	22.12%
10%	22.21%	23.85%	25.88%

TABLE II
COUNTER BANK SIZE FOR CB-SILITE, PARTIAL OBSERVATION

(Table I)

For the YouTube trace (Figure 12), in addition to the three algorithms, we implement heuristic versions of LFU and LFULite that account for the change in distribution. We halve the counts of LFU and LFULite every 50000 requests. We observe that LFU and LFULite outperform LRU, while the change versions do slightly better. The small performance gain in LFUCHANGE and LFULiteCHANGE is due to the slowly varying popularities in the YouTube trace.

Next, we show the performance of all the five algorithms on the synthetic change trace (Figure 13). We observe that LRU dominates LFU and LFULite for small cache sizes, while LFU and LFULite outperforms LRU as the cache size grows. We also observe that the heuristic versions of LFU and LFULite outperforms LRU for all cache sizes.

2) Partial Observation: Figure 14 shows the hit rates for CB-SI,CB-SILite, and CB-MPS algorithms for the IBM trace. We observe that the CB-SI algorithm clearly outperforms CB-SILite and CB-MPS algorithms. Figure 15 shows the hit rate performance of these algorithms for the YouTube trace. Even here, the performances of CB-SI and CB-SILite are superior to the CB-MPS algorithm.

For the synthetic change trace, we also implement change versions of all the three algorithms by halving the counts periodically every 50000 requests. We notice that CB-SIChange outperforms all the other algorithms. We observe that CB-SILite uses small counter bank for all the traces as shown in see Table II.

VI. CONCLUSION

We considered the question of caching algorithm design and analysis from the perspective of online learning. We focused on algorithms that estimate popularity by maintaining counts of requests seen, in both the full and partial observation regimes. Our main findings were in the context of full observation, it is possible to follow this approach and obtain $O(1)$ regret using the simple LFU-Lite approach that only needs a small number of counters. In the context of partial observations, our finding using the CB-SI approach was that structure greatly enhances the learning ability of the caching algorithm, and is able to make up for incomplete observations to yield $O(1)$ regret. We verified these insights using both simulations and data traces. In particular, we showed that even if the request distribution changes with time,

our approach (enhanced with a simple "forgetting" rule) is able to outperform established algorithms such as LRU.

REFERENCES

[1] E. G. Coffman and P. J. Denning, *Operating Systems Theory*. Prentice-Hall Englewood Cliffs, NJ, 1973.

[2] G. Einziger, R. Friedman, and B. Manes, "TinyLFU: A highly efficient cache admission policy," *ACM Transactions on Storage (ToS)*, vol. 13, no. 4, p. 35, 2017.

[3] G. Karakostas and D. N. Serpanos, "Exploitation of different types of locality for web caches," in *Proceedings ISCC 2002 Seventh International Symposium on Computers and Communications*. IEEE, 2002, pp. 207–212.

[4] P. Blasco and D. Gündüz, "Learning-based optimization of cache content in a small cell base station," in *2014 IEEE International Conference on Communications (ICC)*. IEEE, 2014, pp. 1897–1903.

[5] W. F. King-III, "Analysis of Demanding Paging Algorithms," in *Proc. of IFIP Congress*, 1971.

[6] E. Gelenbe, "A Unified Approach to the Evaluation of a Class of Replacement Algorithms," *Computers, IEEE Transactions on*, 1973.

[7] D. Starobinski and D. Tse, "Probabilistic Methods for Web Caching," *Performance evaluation*, 2001.

[8] E. J. Rosensweig, J. Kurose, and D. Towsley, "Approximate Models for General Cache Networks," in *Proc. of INFOCOM*, 2010.

[9] R. Fagin, "Asymptotic Miss Ratios over Independent References," *Journal of Computer and System Sciences*, vol. 14, no. 2, pp. 222–250, 1977.

[10] H. Che, Y. Tung, and Z. Wang, "Hierarchical Web Caching Systems: Modeling, Design and Experimental Results," *Selected Areas in Communications, IEEE Journal on*, Sep 2002.

[11] D. S. Berger, P. Gland, S. Singla, and F. Ciucu, "Exact Analysis of TTL Cache Networks," *Performance Evaluation*, 2014.

[12] N. Gast and B. Van Houdt, "Asymptotically Exact TTL-Approximations of the Cache Replacement Algorithms LRU(m) and h-LRU," in *Proc. of ITC*, 2016.

[13] S. Basu, A. Sundarrajan, J. Ghaderi, S. Shakkottai, and R. Sitaraman, "Adaptive TTL-based caching for content delivery," *IEEE/ACM transactions on networking*, vol. 26, no. 3, pp. 1063–1077, 2018.

[14] J. Li, S. Shakkottai, J. C. Lui, and V. Subramanian, "Accurate learning or fast mixing? Dynamic adaptability of caching algorithms," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1314–1330, 2018.

[15] S. Ioannidis and E. Yeh, "Jointly optimal routing and caching for arbitrary network topologies," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1258–1275, 2018.

[16] T. L. Lai and H. Robbins, "Asymptotically efficient adaptive allocation rules," *Advances in applied mathematics*, vol. 6, no. 1, pp. 4–22, 1985.

[17] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[18] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, no. 3/4, pp. 285–294, 1933.

[19] S. Agrawal and N. Goyal, "Further optimal regret bounds for thompson sampling," in *Artificial intelligence and statistics*, 2013, pp. 99–107.

[20] S. Bubeck, N. Cesa-Bianchi *et al.*, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," *Foundations and Trends® in Machine Learning*, vol. 5, no. 1, pp. 1–122, 2012.

[21] N. Cesa-Bianchi and G. Lugosi, "Combinatorial bandits," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1404–1422, 2012.

[22] V. Martina, M. Garetto, and E. Leonardi, "A Unified Approach to the Performance Analysis of Caching Systems," in *Proc. of INFOCOM*, 2014.

[23] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Watch Global, Cache Local: YouTube Network Traffic at a Campus Network: Measurements and Implications," in *Electronic Imaging*, 2008.

[24] N. Megiddo and D. S. Modha, "ARC: A Self-Tuning, Low Overhead Replacement Cache." in *Proc. of FAST*, 2003.

[25] A. Dvoretzky, J. Kiefer, J. Wolfowitz *et al.*, "Asymptotic minimax character of the sample distribution function and of the classical multinomial estimator," *The Annals of Mathematical Statistics*, vol. 27, no. 3, pp. 642–669, 1956.

[26] W. Hoeffding, "Probability inequalities for sums of bounded random variables," in *The Collected Works of Wassily Hoeffding*. Springer, 1994, pp. 409–426.

[27] S. Agrawal and N. Goyal, "Thompson sampling for contextual bandits with linear payoffs," in *International Conference on Machine Learning*, 2013, pp. 127–135.

[28] J. Komiyama, J. Honda, and H. Nakagawa, "Optimal regret analysis of thompson sampling in stochastic multi-armed bandit problem with multiple plays," *arXiv preprint arXiv:1506.00779*, 2015.

[29] S. Bubeck, V. Perchet, and P. Rigollet, "Bounded regret in stochastic multi-armed bandits," in *Conference on Learning Theory*, 2013, pp. 122–134.

[30] P. Zerfos, M. Srivatsa, H. Yu, D. Dennerline, H. Franke, and D. Agrawal, "Platform and applications for massive-scale streaming network analytics," *IBM Journal of Research and Development*, vol. 57, no. 3/4, pp. 11–1, 2013.

[31] X. Cheng, C. Dale, and J. Liu, "Statistics and social network of youtube videos," in *2008 16th Interntional Workshop on Quality of Service*. IEEE, 2008, pp. 229–238.