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

Archana Bura, Desik Rengarajan, Dileep Kalathil, *Senior Member, IEEE*, Srinivas Shakkottai, *Senior Member, IEEE*, and Jean-Francois Chamberland, *Senior Member, IEEE*

***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.**

***Index Terms*— Caching algorithms, online learning, multi armed bandits.**

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.

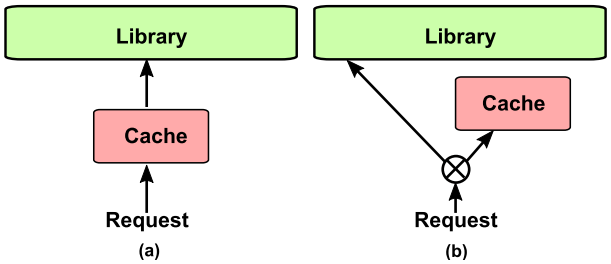

Fig. 1. Request forwarding with (a) full observation, and (b) partial observation at the cache.

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 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? We have two main themes in this project that are illustrated in the conceptual settings of Figure 1. Here, we have shown two fundamentally different learning architectures using a single cache, and a much larger "library," which is a remote (possibly distributed) database that contains all content in the system. In both, there is a request process that is exogenous, and learning involves using this process to determine which items should be cached. Our learning themes are as follows.

A. Learning to Cache With Full Observations

One possibility is that there is a hierarchy of caches, with requests passing from one to the next, and eventually to the library. Applications of this model include hierarchical

Manuscript received September 7, 2020; revised April 14, 2021 and July 7, 2021; accepted July 18, 2021; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor K. Jagannathan. This work was supported in part by the National Science Foundation under Grant CNS-1955696, Grant CRII-CPS-1850206, Grant NSF-Intel CNS-1719384, Grant ARO W911NF-19-1-0367, and Grant ARO W911NF-19-2-0243. *(Corresponding author: Archana Bura.)*

The authors are with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843 USA (e-mail: archanabura@tamu.edu; desik.rengarajan@tamu.edu; dileep.kalathil@tamu.edu; sshakkot@tamu.edu; chmbrlnd@tamu.edu).

Digital Object Identifier 10.1109/TNET.2021.3105880

systems, such as chip-level L1, L2 caches leading up to the main memory or storage [3], and sequences of content caches beginning close to a user and leading up to a data center [4]. In the representative scenario where there is just a single caching element in the hierarchy, it would see all requests as illustrated in Fig. 1 (a). A cache hit would mean that the item can be serviced from the cache, while a miss would mean that the request would be forwarded to the library. Since the cache sees all requests, whether they result in hits or misses, the caching algorithm simply needs to quickly and efficiently determine the top items to cache.

B. Caching to Learn With Partial Observations

In a cache routing approach, requests for content are routed towards caches that are believed to possess the content (or along a default path without any information). Such routing might be done via a name server such as DNS redirect, and a cache only sees requests that are directed towards it. In turn, this depends on whether the cache has the content in question. Applications of this model include caching at cellular base stations [5], and content dissemination using a Global Name Service [6] (similar to DNS) such as Mobility-First [7]. The representative single caching element scenario is shown in Fig. 1 (b), and we refer to it as the *partial observation* regime.

In a typical cache network, requests are seen at multiple caches, but request information is either not aggregated, or only partially. Specifically, only request summaries might be disseminated over time. The availability of such summaries implies that the content provider often has a good idea of the nature of the arrival distribution, for instance the Zipf parameter that it follows and the timescale at which changes are observed. However, each cache does not see the hits and misses of the other caches in realtime, nor does it know the identity of the popular items in advance of the fact. Thus, *explicit caching actions* must be taken in order for a cache to learn what is popular. Hence, we have a regime in which structural information about the distribution could be known, but individual caches are not up to date on each other's hits and misses as they occur, and need to learn popularity as the arrival process changes, corresponding to the partial observation case.

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. Can we design regret optimal algorithms that apply to each of our learning paradigms?

1) 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 obtained on the popularity distribution received thus far. Here, there is no exploration problem, and the goal is to simply exploit the observations received via estimating the empirical frequencies. Hence, 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. It can also be shown that the regret of LRU provably high. Intuitively, LRU does not learn the true popularity of the requests, rather it keeps a track of the recently arrived requests. Thus it suffers a constant regret at each time step, resulting in $\Omega(T)$ cumulative regret.

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 [8], 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 variant 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 (MAB) under which actions must be taken to learn the value of pulling the different arms. Hence, explicit exploration actions are needed in this regime. 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 [5].

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 ones are the most popular). 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 simulations conducted using an IRM model drawn from Zipf distributions 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 "forgetting" 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. We further incorporate an online change detection mechanism into our algorithms to detect the changes in popularity on the fly. We show via a synthetic non-stationary trace that the online change detection scheme, when combined with the simple forgetting rule makes our algorithms robust to the changes in the popularity under non-stationary traffic.

2) 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 [1], [9]–[11]. The expressions are often hard to compute for large caches, and approximations have been proposed for larger cache sizes [12]. Of particular interest is the Time-To-Live (TTL) approximation [13]–[16] 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 [15].

Recent work on performance analysis of caching algorithms has focused on the online learning aspect. For instance, [17] propose TTL-based schemes to show that a desired hit rate can be achieved under non-stationary arrivals. Other work such as [18] 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 $\Omega(T)$.

An alternative approach is taken in [19], [20], where the request arrival process is taken to be adversarial. These works present asymptotic and non-asymptotic regret lower bounds respectively, and show that a coded and an uncoded policy respectively achieve this bound. As in many algorithm design and analysis settings, the adversarial model and the stochastic (Bayesian) model produce significantly different results that are not directly comparable. For instance, [20] shows that the LFU algorithm incurs an $\Omega(T)$ regret in the adversarial setting, i.e., the bound suggests poor performance. In contrast, in the stochastic arrival setting, we show among other results that the LFU algorithm will achieve the best possible regret of $O(1)$ in the full observation regime. Our results are supported through empirical trace-based (non-adversarial) simulations.

Information Centric caching has gained much recent interest, and is particularly relevant to edge wireless networks. Joint caching and routing is studied in [21] 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 [5], 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 [22] 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 [23], which enabled a simple implementation while achieving the optimal regret. The posterior sampling approach, first proposed by Thompson in [24], has recently been shown to attain optimal regret [25]. For a detailed survey, we point to a monograph [26] and a recent book [27]. Another line of works in bandits is related to the best-arm identification [28], [29], which can be considered as a pure exploration problem. In our manuscript, the full observation setting does not need to perform exploration. The exploration vs. exploitation trade off naturally arises in the partial observation regime, hence, in that theme, we follow approaches inspired from multi-armed bandit literature. Although there are similarities, the basic approaches and theoretical guarantees provided by MAB and the best arm identification problems are different.

Much work also exists on the empirical performance evaluation of caching algorithms using traces gathered from different applications. While several discover fundamental insights [30]–[32], 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 $C(t)$ denote the set of files placed in the cache by the caching algorithm at time t. We say that the 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 the randomness in the requests and the ensuing choices on $C(t)$ made by the caching algorithm. Clearly, if popularity distribution μ is known, the optimal caching policy is to place the most popular items in the cache at all times, 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 a priori. So the goal of a caching algorithm is to learn the popularity distribution (or part of it) from the sequential observations, and to place files in the cache by judiciously using the available information at each time in order to maximize the expected cumulative hits.

In the literature on multi-armed bandits, it is common to characterize the performance of an online learning algorithm using the metric of *regret*, which is defined as the performance loss of the algorithm as compared to the optimal strategy 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 π that maps $h(t)$ to $C(t)$, i.e., $C(t) = \pi_t(h(t))$, 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.

Below, 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. As indicated earlier, this is purely an exploitation problem, since every request is seen at the cache, and so all hits and misses are known regardless of the cached items. This regime can be compared to a multi-armed bandit in which the reward of every arm is revealed irrespective of the arm that is pulled, i.e., no exploration is needed.

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, the LFU algorithm maintains an empirical estimate of the popularity distribution, which we denote by $\hat{\mu}(t) = (\hat{\mu}_1(t), \ldots, \hat{\mu}_L(t))$. It is 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 collection of files to be placed in the cache at time $t+1$, $C_{\text{LFU}}(t+1)$, is then equal to

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

where $\arg\max_C$ indicates the indices of the top C elements of the vector $\hat{\mu}(t)$. Having established these notions, we present below the finite time performance guarantee for the LFU algorithm.

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

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

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 $\Delta_{\min}$. Roughly, if $LC < 1/\Delta_{\min}$, then the second term dominates.

We will use the following Lemma for proving Theorem 1.

Lemma 2: For $\epsilon > 0$, we have

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

The lemma is obtained through an application of the Dvoretzky-Kiefer-Wolfowitz inequality [33]. We omit the proof due to page limitation.

We proceed with the proof of Theorem 1.

Proof: We denote $C_{\text{LFU}}(t)$ just as $C(t)$ for notational convenience. We first argue 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 $k \in \mathcal{L} \setminus \mathcal{C}$,

$$\begin{aligned}\hat{\mu}_j(t) &\geq \mu_j - \Delta_{\min}/2 \geq \mu_C - \Delta_{\min}/2 \\ &\geq \mu_{C+1} + \Delta_{\min}/2 \geq \mu_k + \Delta_{\min}/2 \geq \hat{\mu}_k(t)\end{aligned}$$

and hence $C(t) = \mathcal{C}$. The expected regret can then be bounded, with

$$\begin{aligned}\mathbb{E}[R(T)] &= \mathbb{E}\left[\sum_{t=1}^{T} \mathbb{1}\{x(t) \in \mathcal{C}\} - \mathbb{1}\{x(t) \in C(t)\}\right] \\ &\leq \mathbb{E}\left[\sum_{t=1}^{T} \mathbb{1}\{C(t) \neq \mathcal{C}\}\right] = \sum_{t=1}^{T} \mathbb{P}(C(t) \neq \mathcal{C}) \\ &\leq \sum_{t=1}^{T} \mathbb{P}\left(\max_i |\hat{\mu}_i(t) - \mu_i| \geq \Delta_{\min}/2\right) \\ &\leq \sum_{t=1}^{T} 2e^{-t\Delta_{\min}^2/8} \leq \int_{t=0}^{\infty} 2e^{-t\Delta_{\min}^2/8} \leq \frac{16}{\Delta_{\min}^2}. \quad (2)\end{aligned}$$

We can also upper bound $\mathbb{E}[R(T)]$ using a different approach, to show the trade off between $\Delta_{\min}^2$ and $\Delta_{\min}$, L, and C. This approach makes use of Hoeffding's inequality.

$$\begin{aligned}&\mathbb{E}[R(T)] \\ &= \mathbb{E}[\sum_{t=1}^{T} \mathbb{1}\{x(t) \in \mathcal{C}\} - \mathbb{1}\{x(t) \in C(t)\}] \\ &= \mathbb{E}\left[\sum_{t=1}^{T} \mathbb{E}[\mathbb{1}\{x(t) \in \mathcal{C}\}|C(t)] - \mathbb{E}[\mathbb{1}\{x(t) \in C(t)\}|C(t)]\right] \\ &= \mathbb{E}\left[\sum_{t=1}^{T}\left(\sum_{j\in\mathcal{C}} \mu_j - \sum_{k\in C(t)} \mu_k\right)\right] \quad (3) \\ &\leq \mathbb{E}\left[\sum_{t=1}^{T}\sum_{j=1}^{C}\sum_{k=C+1}^{L} \Delta_{j,k}\mathbb{1}\{j \notin C(t), k \in C(t)\}\right] \quad (4)\end{aligned}$$

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

Using the Hoeffding inequality [34], 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. □

B. WLFU Algorithm

LFU achieves a regret of $O(1)$, but its implementation is expensive in terms of memory requirements. This cost arises because LFU maintains a popularity estimate for each item in the library ($\hat{\mu}_i(t)$), and the library size L is extremely large for most practical applications. Typically, allocating memory to maintain the popularity distribution estimate for the whole library is impractical.

There are many approaches proposed to address this issue [2], [8]. 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.

In this article, we consider the Window-LFU (WLFU) algorithm [8] that has been proposed as way to overcome the expensive memory requirement of 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 within 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 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: This result can be established by finding a lower bound on the probability that cache does not match the most likely items. From the proof of Theorem 1 (c.f. (3)), we have

$$\mathbb{E}[R(T)]=\mathbb{E}\left[\sum_{t=1}^{T}\mathbb{1}\{x(t)\in\mathcal{C}\}-\mathbb{1}\{x(t)\in C(t)\}\right]$$
$$=\sum_{t=1}^{T}\left(\sum_{j\in\mathcal{C}}\mu_j-\mathbb{E}\left[\mathbb{1}\{x(t)\in C(t)\}\right]\right)$$
$$=\sum_{t=1}^{T}\mathbb{E}\left[\sum_{j\in\mathcal{C}\setminus C(t)}\mu_j-\sum_{k\in C(t)\setminus\mathcal{C}}\mu_k\right]$$
$$\geq\sum_{t=1}^{T}\mathbb{E}\left[\sum_{k\in C(t)\setminus\mathcal{C}}(\mu_C-\mu_k)\right]$$
$$=\sum_{t=1}^{T}\mathbb{E}\left[\sum_{k\in\mathcal{L}\setminus\mathcal{C}}(\mu_C-\mu_k)\mathbb{1}\{k\in C(t)\}\right]$$
$$=\sum_{t=1}^{T}\sum_{k\in\mathcal{L}\setminus\mathcal{C}}(\mu_C-\mu_k)\mathbb{P}(k\in C(t))$$
$$\geq\sum_{t=1}^{T}(\mu_C-\mu_{C+1})\mathbb{P}(C+1\in C(t)), \tag{7}$$

where the last inequality follows by focusing on a sub-event. Given that the probability of item $C+1$ in non-zero, we can establish the desired lower bound using window $W[t-w,t)$,

$$\mathbb{P}(C+1\in C(t))\geq\mathbb{P}(\{x(\tau)=C+1:\tau\in[t-w,t)\})$$
$$=(\mu_{C+1})^w.$$

Combining this result with (7), we get expression

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

which has order T. Since the cost per stage is bounded, we obtain the statement of the theorem. □

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

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

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

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

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)

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

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

$$\begin{aligned}
&\mathbb{E}[\sum_{t=1}^{T}\sum_{j=1}^{C}\sum_{k=C+1}^{L} \Delta_{j,k}\mathbb{1}\{\hat{\mu}_j(t) - \mu_j \leq -\Delta_{j,k}/2\}] \\
&= \mathbb{E}[\sum_{j=1}^{C}\sum_{k=C+1}^{L} \mathbb{E}[\sum_{t=1}^{T} \Delta_{j,k}\mathbb{1}\{\hat{\mu}_j(t) - \mu_j \leq -\Delta_{j,k}/2\}|t_j]] \\
&\leq \mathbb{E}[\sum_{j=1}^{C}\sum_{k=C+1}^{L} \Delta_{j,k}(t_j + \mathbb{E}[\sum_{t=t_j}^{T} \mathbb{1}\{\hat{\mu}_j(t) - \mu_j \\
&\leq -\Delta_{j,k}/2\}|t_j])] \\
&\leq \mathbb{E}[\sum_{j=1}^{C}\sum_{k=C+1}^{L} \Delta_{j,k}(t_j + \sum_{t=t_j}^{T} \mathbb{P}(\hat{\mu}_j(t) - \mu_j \leq -\Delta_{j,k}/2|t_j))] \\
&\leq \mathbb{E}[\sum_{j=1}^{C}\sum_{k=C+1}^{L} \Delta_{j,k}(t_j + \sum_{t=t_j}^{T} e^{-(t-t_j)\Delta_{j,k}^2/2}] \\
&\leq \sum_{j=1}^{C}\sum_{k=C+1}^{L} (\Delta_{j,k}\mathbb{E}[t_j] + \frac{2}{\Delta_{j,k}}). \qquad (11)
\end{aligned}$$

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

$$\begin{aligned}
&\mathbb{E}[\sum_{t=1}^{T}\sum_{j=1}^{C}\sum_{k=C+1}^{L} \Delta_{j,k}\mathbb{1}\{\hat{\mu}_k(t) - \mu_k > \Delta_{j,k}/2\}] \\
&= \mathbb{E}[\sum_{j=1}^{C}\sum_{k=C+1}^{L} \mathbb{E}[\sum_{t=1}^{T} \Delta_{j,k}\mathbb{1}\{\hat{\mu}_k(t) - \mu_k > \Delta_{j,k}/2\}|t_k]] \\
&= \mathbb{E}[\sum_{j=1}^{C}\sum_{k=C+1}^{L} \Delta_{j,k} \sum_{t=t_k}^{T} \mathbb{P}(\hat{\mu}_k(t) - \mu_k > \Delta_{j,k}/2|t_k)] \\
&\leq \mathbb{E}[\sum_{j=1}^{C}\sum_{k=C+1}^{L} \Delta_{j,k} \sum_{t=t_k}^{T} e^{-(t-t_k)\Delta_{j,k}^2/2}] \\
&\leq \sum_{j=1}^{C}\sum_{k=C+1}^{L} \frac{2}{\Delta_{j,k}}. \qquad (12)
\end{aligned}$$

Combining (11) and (12) we obtain

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

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

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

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

Combining (15) and (14), we obtain

$$\begin{aligned}
\mathbb{E}[R(T)] &\leq \sum_{j=1}^{C}\sum_{k=C+1}^{L} \frac{w\Delta_{j,k}}{p_j} + \sum_{j=1}^{C}\sum_{k=C+1}^{L} \frac{4}{\Delta_{j,k}} \\
&\leq \frac{C(L-C)w}{p_{\min}} + \frac{4C(L-C)}{\Delta_{\min}}. \qquad (15)
\end{aligned}$$

□

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.

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

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. □

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 $w_1 \geq w_2$. 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 $i \notin \mathcal{C}$, $p_i(w_1) \leq p_i(w_2)$. 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, $(1-p_i(w_2))^{\lceil t/w_2 \rceil} \leq (1-p_i(w_1))^{\lceil t/w_2 \rceil} \leq (1-p_i(w_1))^{\lceil t/w_1 \rceil}$. So, $1-(1-p_i(w_2))^{\lceil t/w_2 \rceil} \geq 1-(1-p_i(w_1))^{\lceil t/w_1 \rceil}$. 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 IV-C.

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 in the cache. Hence, the caching algorithm has to perform active *exploration* by placing a file in the cache sufficiently often 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 estimate of each file in the library can be improved after each time step due to full visibility of all the requests.

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.

We formulate the caching problem as a multi-player multi-armed bandit problem, in which the content placement in the cache is viewed equivalent to arm pulls. The request for an item is considered as its reward, which is a $\{0,1\}$ random variable, sampled from the popularity vector. We call this formulation as "Caching Bandits". Unlike in the multi-armed bandit problem, the rewards in the caching bandit problem are not independent, as the request for one item in the cache indicate that there is no request for the other items.

A. Caching Bandit With Full Posterior Sampling

Posterior sampling based algorithms for MAB [25], [35] typically use a Beta prior (with Bernoulli likelihood) or Gaussian prior (with Gaussian likelihood) in order to exploit 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 $\mu = (\mu_1, \ldots, \mu_L)$, parametrized by $\alpha = (\alpha_1, \ldots, \alpha_L)$. 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. In case of a miss, we code $s(t) = 0$. Given the current parameter $\alpha(t) = \alpha$, we can show that the posterior distribution in case of a miss can be computed as

$$\begin{aligned} f_{t+1}(\mu | s(t) = 0) &\propto \mathbb{P}(s(t) = 0 | \mu) f_t(\mu;\alpha) \\ &= \frac{1}{\sum_{i=1}^{L} \alpha_i} \sum_{j \notin C(t)} \alpha_j f_t(\mu; \alpha + e_j). \end{aligned}$$

Algorithm 2 CB-FPS Algorithm

Initialize the prior distribution f_0
for $t = 1, \ldots, T$ **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, the posterior update in case of a miss is a combination of $(L-C)$ Dirichlet priors from the previous step. With the first miss, the algorithm needs to store a set of size $(L-C)$ consisting of Dirichlet parameters. With each miss, parameter sets of size $(L-C)$ need to be stored, one such set for each of the parameters at the previous step. Thus, at the t^{th} miss, the number of parameters that we need to store are $(L-C)^t$, growing exponentially in t. Hence, as the number of misses increases, the memory required to store these parameters will increase exponentially, rendering the full posterior update algorithm infeasible from an implementation perspective.

We present the CB-FPS in Algorithm 2. At each time t, the algorithm takes a sample $\hat{\mu}(t)$ according to the current

posterior $f_t(\cdot)$. It places the top C items in the order of decreasing $\hat{\mu}(t)$ in the cache.

In Section IV-C, 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 Dirchlet 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 $t=1,\ldots,T$ **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 $x(t) \in C_{\text{MPS}}(t)$ **then**
    $\alpha_{x(t)}(t+1) \leftarrow \alpha_{x(t)}(t)+1$
    $\beta_i(t+1) \leftarrow \beta_i(t)+1, \forall i \in C_{\text{MPS}}(t), i \neq x(t)$
  end if
end for

The CB-MPS algorithm generates samples for each item from an independent beta distribution, and places the C items with the largest samples into the cache. The algorithm then updates independent beta posteriors for each item in the library. However, it updates the posterior for only those items currently in the cache. The posteriors for all the other items remain the same.

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

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

It is clear that CB-MPS algorithm disregards the inherent structure in the Caching Bandit formulation, by choosing not to update the non-cache items. Note that, in the caching formulation, the popularity vector has inter dependent components (it is a probability distribution), a cache miss means that one of the non-cache items is surely requested. Hence, the CB-MPS algorithm views the caching bandit problem almost like a multi player multi armed bandit problem, regardless of the additional structure imposed by the caching bandit. But we emphasize that this loss of information in the CB-MPS algorithm is unavoidable due to the partial observation structure limiting the observations to the cached items. Note that the beta posterior in CB-MPS will be corrupted if we update the missed requests for the non cache items in any heuristic way.

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 [36] can be used with small modifications to show the above result.

C. Caching Bandit With Structural Information

Even though the CB-MPS 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 [37], which can be considered as a special case with $C=1$. CB-SI algorithm is given in Algorithm 4.

The algorithm maintains an empirical estimate for each item, and places into cache the items whose empirical estimate crosses a certain threshold decided by μ_C, and $\Delta_{\min}$. If there are not enough items that cross this threshold, the algorithm samples the rest of the items without replacement, according to the probability inversely proportional to the square of items' gap from the C^{th} most popular item. The intuition is that with the knowledge of the threshold, one can reduce the amount of exploration needed, and hence reduce the regret. To see this, once an item's empirical estimate is more than $\mu_C - \frac{\Delta_{\min}}{2}$, we do not need to explore other items to resolve the uncertainty about the item in question, as this item belongs to the C most popular items. We still need to explore to observe the other popular items. In this way, the algorithm reduces the exploration of less popular items, while exploiting the information accrued so far about the popular items, by the knowledge of the threshold. As shown in theorem below, this directed exploration suffers only a constant regret.

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

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

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.

$$\begin{aligned}
&\mathbb{E}[R(T)] \\
&\leq \mathbb{E}\Big[\sum_{t=1}^{T}\sum_{i=1}^{C}\sum_{j=C+1}^{L} \Delta_{j,k}\mathbb{1}\{i \notin C(t), j \in C(t)\}\Big] \\
&\leq C\ \mathbb{E}\Big[\sum_{t=1}^{T}\sum_{j=C+1}^{L} \mathbb{1}\{j \in C(t)\}\Big] = C\sum_{j=C+1}^{L}\sum_{t=1}^{T}\mathbb{P}(j \in C(t)) \\
&= C\sum_{j=C+1}^{L}\sum_{t=1}^{T}\big(\mathbb{P}(\hat{\mu}_j(n_j(t)) > \mu_C - \Delta_j/2, j \in C(t)) \\
&\quad + \mathbb{P}(\hat{\mu}_j(n_j(t)) \leq \mu_C - \Delta_j/2, j \in C(t))\big). \qquad (17)
\end{aligned}$$

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

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

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

    Sample $C - |A(t)|$ elements from the set $\mathcal{L} \setminus A(t)$ according to the probability $p_i(t)$. Denote these elements as $B(t)$
    $C_{\text{SI}}(t) = A(t) \cup B(t)$
    $Z_t \leftarrow 2$
  end if
  Place the files $C_{\text{SI}}(t)$ 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

inequality holds.

$$\begin{aligned}
&\sum_{t=1}^{T} \mathbb{P}(\hat{\mu}_j(n_j(t)) > \mu_C - \Delta_j/2, j \in C(t)) \\
&\quad \leq \sum_{t=1}^{T} \mathbb{P}(\hat{\mu}_j(n_j(t)) > \mu_j + \Delta_j/2, j \in C(t)) \\
&\quad \leq \sum_{t=1}^{T} \mathbb{P}(\hat{\mu}_j(t) > \mu_j + \Delta_j/2) \overset{(a)}{\leq} \sum_{t=1}^{T} e^{-\Delta_j^2 t/2} \leq \frac{2}{\Delta_j^2},
\end{aligned} \tag{18}$$

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 j^{th} 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 $Z_t = 2$.

$$\begin{aligned}
\mathbb{P}(\hat{\mu}_j(n_j(t)) &\leq \mu_C - \frac{\Delta_j}{2}, j \in C(t)) \\
&\leq \mathbb{P}(\hat{\mu}_j(n_j(t)) \leq \mu_C - \frac{\Delta}{2}, j \in C(t)) \\
&\overset{(b)}{=} \mathbb{P}(\hat{\mu}_j(n_j(t)) \leq \mu_C - \frac{\Delta}{2}, j \in C(t), Z_t = 2) \\
&\overset{(c)}{=} \mathbb{P}(j \in C(t) | \hat{\mu}_j(n_j(t)) \leq \mu_C - \frac{\Delta}{2}, Z_t = 2) \\
&\quad \mathbb{P}(\hat{\mu}_j(n_j(t)) \leq \mu_C - \frac{\Delta}{2}, Z_t = 2) \\
&= p_{j,t} \mathbb{P}\left(\hat{\mu}_j(n_j(t)) \leq \mu_C - \frac{\Delta}{2}, Z_t = 2\right) \\
&= \mathbb{E}\left[p_{j,t} \mathbb{1}\{\hat{\mu}_j(n_j(t)) \leq \mu_C - \frac{\Delta}{2}, Z_t = 2\}\right] \\
&= \mathbb{E}\left[\frac{p_{j,t}}{p_{i,t}} p_{i,t} \mathbb{1}\{\hat{\mu}_j(n_j(t)) \leq \mu_C - \frac{\Delta}{2}, Z_t = 2\}\right],
\end{aligned}$$

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,

$$\begin{aligned}
&\leq \mathbb{E}\left[\frac{|\mu_C - \hat{\mu}_i(n_i(t))|^2}{(\frac{\Delta}{2})^2} p_{i,t} 1\{\hat{\mu}_j(n_j(t)) \leq \mu_C - \frac{\Delta}{2}, Z_t = 2\}\right] \\
&\leq \frac{4}{\Delta^2} \mathbb{E}\left[|\mu_C - \hat{\mu}_i(n_i(t))|^2 \; p_{i,t} \mathbb{1}\{Z_t = 2\}\right] \\
&\leq \frac{4}{\Delta^2} \mathbb{E}\Big[|\mu_C - \hat{\mu}_i(n_i(t))|^2 \\
&\qquad \mathbb{P}\left(i \in C(t) | \hat{\mu}_i(n_i(t)) \leq \mu_C - \frac{\Delta}{2}, Z_t = 2\right)\Big] \\
&= \frac{4}{\Delta^2} \mathbb{E}\Big[|\mu_C - \hat{\mu}_i(n_i(t))|^2 \\
&\qquad \mathbb{E}\left[\mathbb{1}\{i \in C(t)\} | \hat{\mu}_i(n_i(t)) < \mu_C - \frac{\Delta}{2}, Z_t = 2\right]\Big] \\
&= \frac{4}{\Delta^2} \mathbb{E}\Big[|\mu_C - \hat{\mu}_i(n_i(t))|^2 \\
&\qquad \mathbb{E}\Big[\mathbb{1}\{i \in C(t), \hat{\mu}_i(n_i(t)) < \mu_C - \frac{\Delta}{2}\} | \\
&\qquad\quad \{\hat{\mu}_i(n_i(t)) < \mu_C - \frac{\Delta}{2}, Z_t = 2\}\Big]\Big] \\
&= \frac{4}{\Delta^2} \mathbb{E}\Big[|\mu_C - \hat{\mu}_i(n_i(t))|^2 \\
&\qquad \mathbb{1}\{i \in C(t), \hat{\mu}_i(n_i(t)) < \mu_C - \frac{\Delta}{2}\}\Big].
\end{aligned} \tag{19}$$

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.

$$\begin{aligned}
&\sum_{t=1}^{T} \mathbb{E}\left[|\mu_C - \hat{\mu}_i(n_i(t))|^2 1\{\hat{\mu}_i(n_i(t)) < \mu_C - \frac{\Delta}{2}, i \in C(t)\}\right] \\
&\leq \sum_{t=1}^{T} \mathbb{E}\left[|\mu_C - \hat{\mu}_i(t)|^2 1\{|\hat{\mu}_i(t) - \mu_C| > \frac{\Delta}{2}\}\right] \\
&= \sum_{t=1}^{T} \int_0^\infty \mathbb{P}(|\mu_C - \hat{\mu}_i(t)|^2 1\{|\hat{\mu}_i(t) - \mu_C| > \frac{\Delta}{2}\} \geq x) dx \\
&= \sum_{t=1}^{T} \int_0^\infty \mathbb{P}(|\mu_C - \hat{\mu}_i(t)|^2 1\{|\hat{\mu}_i(t) - \mu_C|^2 > \frac{\Delta^2}{4}\} \geq x) dx \\
&= \sum_{t=1}^{T} \Bigg[\int_0^{\frac{\Delta^2}{4}} \mathbb{P}\left(|\mu_C - \hat{\mu}_i(t)|^2 \right. \\
&\qquad \left. 1\{|\hat{\mu}_i(t) - \mu_C|^2 > \frac{\Delta^2}{4}\} \geq x\right) dx \\
&\quad + \int_{\frac{\Delta^2}{4}}^\infty \mathbb{P}(|\mu_C - \hat{\mu}_i(t)|^2 1\{|\hat{\mu}_i(t) - \mu_C|^2 > \frac{\Delta^2}{4}\} \geq x) dx\Bigg]
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{t=1}^{T} \left[\int_0^{\frac{\Delta^2}{4}} \mathbb{P}(|\mu_C - \hat{\mu}_i(t)|^2 \geq x, |\hat{\mu}_i(t) - \mu_C|^2 > \frac{\Delta^2}{4}) dx \right. \\
&\quad \left. + \int_{\frac{\Delta^2}{4}}^{\infty} \mathbb{P}(|\mu_C - \hat{\mu}_i(t)|^2 1\{|\hat{\mu}_i(t) - \mu_C|^2 > \frac{\Delta^2}{4}\} \geq x) dx \right] \\
&= \sum_{t=1}^{T} \left[\int_0^{\frac{\Delta^2}{4}} \mathbb{P}\left(|\hat{\mu}_i(t) - \mu_C|^2 > \frac{\Delta^2}{4} \right) dx \right. \\
&\quad \left. + \int_{\frac{\Delta^2}{4}}^{\infty} \mathbb{P}\left(|\mu_C - \hat{\mu}_i(t)|^2 1\{|\hat{\mu}_i(t) - \mu_C|^2 > \frac{\Delta^2}{4}\} \geq x \right) dx \right] \\
&= \sum_{t=1}^{T} \left[\frac{\Delta^2}{4} \mathbb{P}\left(|\hat{\mu}_i(t) - \mu_C|^2 > \frac{\Delta^2}{4} \right) + \right. \\
&\quad \left. \int_{\frac{\Delta^2}{4}}^{\infty} \mathbb{P}\left(|\mu_C - \hat{\mu}_i(t)|^2 1\{|\hat{\mu}_i(t) - \mu_C|^2 > \frac{\Delta^2}{4}\} \geq x \right) dx \right] \\
&= \sum_{t=1}^{T} \left[2\frac{\Delta^2}{4} e^{-\frac{t\Delta^2}{8}} + \int_{\frac{\Delta^2}{4}}^{\infty} \Pr\{|\mu_C - \hat{\mu}_i(t)|^2 \geq x\} dx \right] \\
&\leq 4 + \frac{32}{\Delta^2} \exp(-\frac{\Delta^2}{8}). \qquad (20)
\end{aligned}
$$

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.

We now compare the performance of CB-MPS and CB-SI via a lower bound argument. The known lower bound on the regret of a classical multi-player multi-armed bandit is $\Omega(\log T)$, and the Thompson sampling algorithm is known to achieve this. As discussed previously, the CB-MPS algorithm is equivalent to the Thompson sampling algorithm for the multi-player multi-armed bandit formulation. Clearly, the multi-player multi-armed bandit is at least as hard as the classical multi-armed bandit problem and will have a regret $\Omega(\log T)$. Thus, we argue that the performance of CB-MPS is worse than CB-SI, which achieves $O(1)$ regret.

V. Simulations

In this section, we start by conducting simulations with requests generated under the IRM model to verify our insights on regret obtained in the earlier sections. In each instance, we present results averaged over ten runs of the algorithm under test. 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. We also test an online change detection mechanism, along with the forgetting rule, under non-stationary request arrival process.

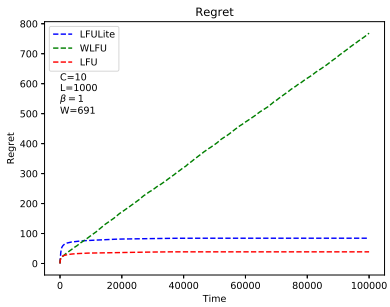

Fig. 2. Regret of LFU, WLFU, LFU-Lite.

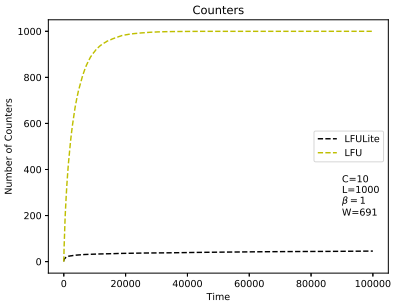

Fig. 3. Growth of counters for LFU, LFU-Lite.

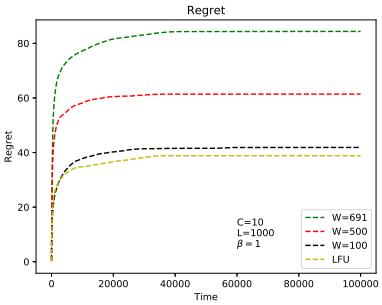

Fig. 4. Regret of LFU-Lite for varying W.

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 $C = 10, L = 1000, W = C^2 \log L$ [8] 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

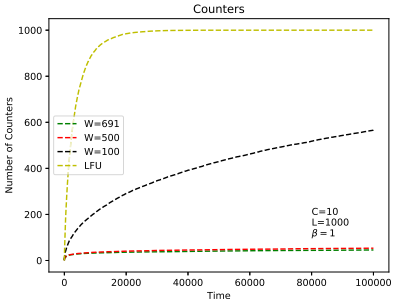

Fig. 5. Growth of counters for varying W.

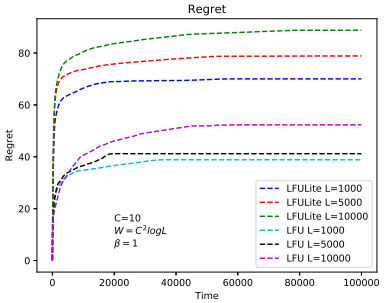

Fig. 6. Regret of LFU-Lite for varying L.

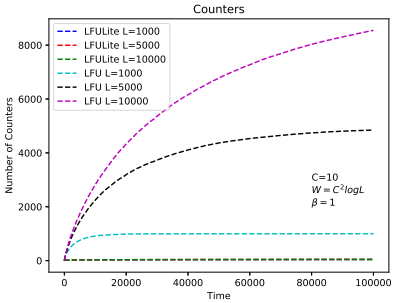

Fig. 7. Growth of counters for varying L.

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 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.

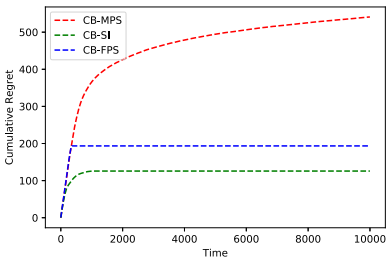

Fig. 8. Cumulative regret performance comparison for $L = 100$, $C = 1, \beta = 2$.

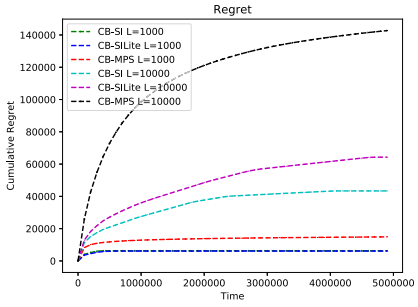

Fig. 9. Regret of CB-SI, CB-SILite, CB-MPS for $C = 10, \beta = 1$.

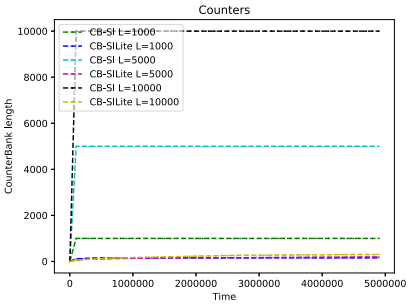

Fig. 10. Growth of counters for varying L.

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 [38]. 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 [39]. 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

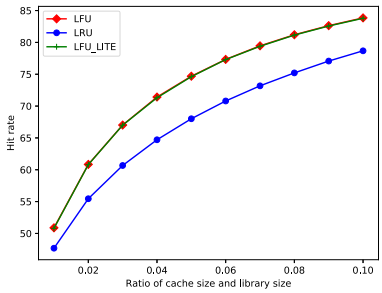

Fig. 11. Hit rates for full observation for IBM trace.

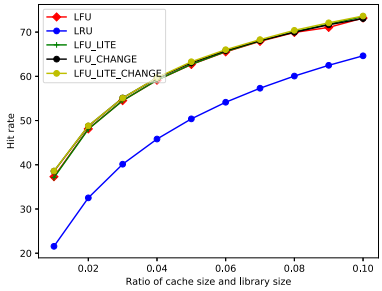

Fig. 12. Hit rates for full observation for YouTube trace.

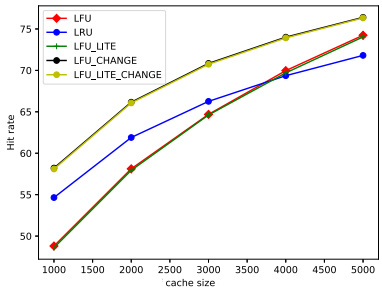

Fig. 13. Hit rates for full observation for change trace.

TABLE I

COUNTER BANK SIZE FOR LFULITE, FULL OBSERVATION

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%

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 [8]. 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.

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

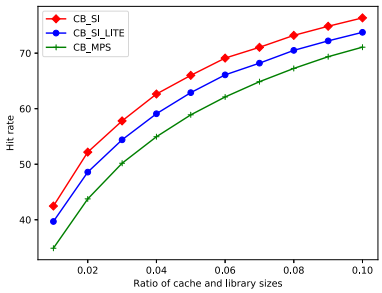

Fig. 14. Hit rates for partial observation, IBM trace.

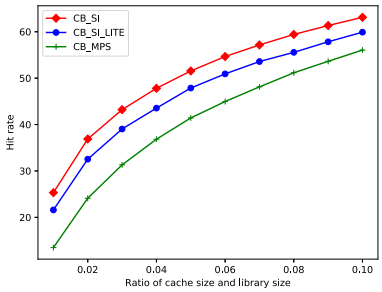

Fig. 15. Hit rates for partial observation, YouTube trace.

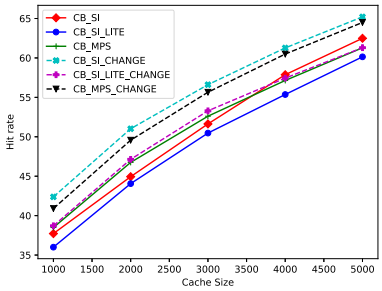

Fig. 16. Hit rates for partial observation, change trace.

TABLE II

COUNTER BANK SIZE FOR CB-SILITE, PARTIAL 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%

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-

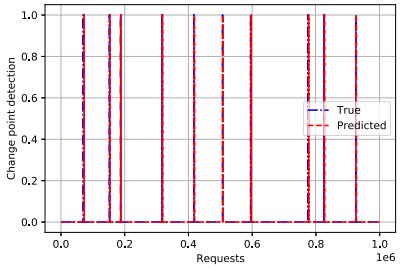

Fig. 17. Change point detection.

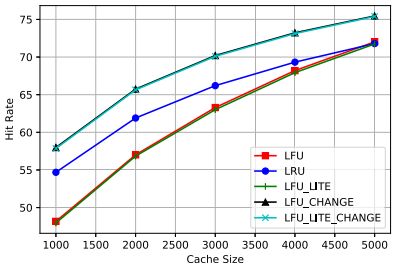

Fig. 18. Hit rates for full observation model.

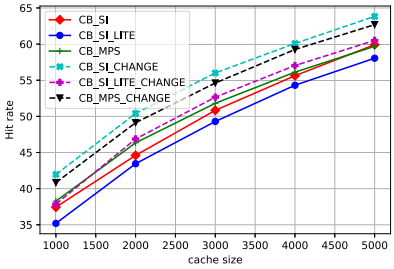

Fig. 19. Hit rates for partial observation model.

SILite uses small counter bank for all the traces as shown in see Table II.

C. Online Change Detection for Non-Stationary Requests

In this section, we use an online change detection algorithm to find if there is any change in the request arrival distribution. We augment our caching algorithms with this online change detection mechanism to adapt it to the scenarios where the requests are non-stationary. We devise this mechanism in such a way that it works independent of the specific caching algorithm we use.

The detection mechanism is based on a two-window paradigm proposed in [40]. It maintains a reference window, and a current window. The current window slides forward with each incoming data point, and the reference window is updated whenever a change is detected. Our scheme compares requests in the reference window to the requests in the current window. We measure the total variation distance between the empirical distribution of requests in the reference window and of that in the current window. If this distance is greater than a threshold, a change detection is announced. When a change is detected, it signals the caching algorithm, which employs a heuristic to reduce its counters by half. This reduction ensures that the algorithms reflect the changes in the request distribution.

To evaluate the performance of our approach, as in the previous subsections, we first create a synthetic trace of 1 million requests with a library size of 50000. These requests are generated by sampling a Zipf distribution, with parameter 1. We then induce non-stationarity in this trace by cycling the popularities at the change points, i.e., at each change point, we cyclically left-shift the components of the popularity vector by 500 positions. We follow the same procedure at all the 10 randomly generated change points in the trace. In this way, we ensure that the resulting trace contains requests with changing distribution at random points in the trace, modeling a real world non-stationary behavior.

Figure 17 illustrates the performance of our change point detection scheme. We note that the scheme indeed is able to detect changes accurately. Figure 18 illustrates the performance of our algorithms with the change detection mechanism in the full information setting. We observe that LFU and LFU-Lite (with the heuristic) outperform all the other algorithms in all cases. Similarly, Figure 19 illustrates the performance of our algorithms for partial information setting, where we observe that CB-SI with the embedded CHANGE heuristic performs the best.

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. We have also augmented our algorithms with an online change detection mechanism, independent to the proposed algorithms. This approach enhanced our algorithms to detect changes in the distributions on the fly. When we enabled the algorithms with this approach along with a simple forgetting rule, we were able to achieve a good empirical performance under non-stationary traffic.

References

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

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

[3] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Amsterdam, The Netherlands: Elsevier, 2011.

[4] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "A hierarchical internet object cache," in *Proc. USENIX Annu. Tech. Conf.*, 1996, pp. 153–164.

[5] P. Blasco and D. Gunduz, "Learning-based optimization of cache content in a small cell base station," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2014, pp. 1897–1903.

[6] A. Sharma, X. Tie, H. Uppal, A. Venkataramani, D. Westbrook, and A. Yadav, "A global name service for a highly mobile internetwork," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 247–258, 2015.

[7] A. Venkataramani, J. Kurose, D. Raychaudhuri, K. Nagaraja, M. Mao, and S. Banerjee, "MobilityFirst: A mobility-centric and trustworthy internet architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 74–80, Jul. 2014.

[8] G. Karakostas and D. N. Serpanos, "Exploitation of different types of locality for web caches," in *Proc. 7th Int. Symp. Comput. Commun. (ISCC)*, Jul. 2002, pp. 207–212.
[9] W. F. King-III, "Analysis of demanding paging algorithms," in *Proc. IFIP Congr.* Amsterdam, The Netherlands: North-Holland, 1971, pp. 485–490.
[10] E. Gelenbe, "A unified approach to the evaluation of a class of replacement algorithms," *IEEE Trans. Comput.*, vol. C-22, no. 6, pp. 611–618, Jun. 1973.
[11] D. Starobinski and D. Tse, "Probabilistic methods for web caching," *Perform. Eval.*, vol. 46, nos. 2–3, pp. 125–137, Oct. 2001.
[12] E. J. Rosensweig, J. Kurose, and D. Towsley, "Approximate models for general cache networks," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.
[13] R. Fagin, "Asymptotic miss ratios over independent references," *J. Comput. Syst. Sci.*, vol. 14, no. 2, pp. 222–250, 1977.
[14] H. Che, Y. Tung, and Z. Wang, "Hierarchical web caching systems: Modeling, design and experimental results," *IEEE J. Sel.Areas Commun.*, vol. 20, no. 7, pp. 1305–1314, Sep. 2002.
[15] D. S. Berger, P. Gland, S. Singla, and F. Ciucu, "Exact analysis of TTL cache networks," *Perform. Eval.*, vol. 79, pp. 2–23, Sep. 2014.
[16] N. Gast and B. Van Houdt, "Asymptotically exact TTL-approximations of the cache replacement algorithms LRU(m) and h-LRU," in *Proc. 28th Int. Teletraffic Congr. (ITC)*, Sep. 2016, pp. 157–165.
[17] S. Basu, A. Sundarrajan, J. Ghaderi, S. Shakkottai, and R. Sitaraman, "Adaptive TTL-based caching for content delivery," *IEEE/ACM Trans. Netw.*, vol. 26, no. 3, pp. 1063–1077, Jun. 2018.
[18] J. Li, S. Shakkottai, J. C. S. Lui, and V. Subramanian, "Accurate learning or fast mixing? Dynamic adaptability of caching algorithms," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 6, pp. 1314–1330, Jun. 2018.
[19] G. S. Paschos, A. Destounis, L. Vigneri, and G. Iosifidis, "Learning to cache with no regrets," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2019, pp. 235–243.
[20] R. Bhattacharjee, S. Banerjee, and A. Sinha, "Fundamental limits on the regret of online network-caching," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 4, no. 2, pp. 1–31, Jun. 2020.
[21] S. Ioannidis and E. Yeh, "Jointly optimal routing and caching for arbitrary network topologies," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 6, pp. 1258–1275, Jun. 2018.
[22] T. L. Lai and H. Robbins, "Asymptotically efficient adaptive allocation rules," *Adv. Appl. Math.*, vol. 6, no. 1, pp. 4–22, Mar. 1985.
[23] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, no. 2, pp. 235–256, 2002.
[24] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, nos. 3–4, pp. 285–294, 1933.
[25] S. Agrawal and N. Goyal, "Further optimal regret bounds for Thompson sampling," in *Artificial Intelligence and Statistics*. Scottsdale, AZ, USA: PMLR, 2013, pp. 99–107.
[26] S. Bubeck and N. Cesa-Bianchi, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," *Found. Trends Mach. Learn.*, vol. 5, no. 1, pp. 1–122, 2012.
[27] N. Cesa-Bianchi and G. Lugosi, "Combinatorial bandits," *J. Comput. Syst. Sci.*, vol. 78, no. 5, pp. 1404–1422, Sep. 2012.
[28] K. Jamieson and R. Nowak, "Best-arm identification algorithms for multi-armed bandits in the fixed confidence setting," in *Proc. 48th Annu. Conf. Inf. Sci. Syst. (CISS)*, Mar. 2014, pp. 1–6.
[29] D. Shah, T. Choudhury, N. Karamchandani, and A. Gopalan, "Sequential mode estimation with Oracle queries," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 4, 2020, pp. 5644–5651.
[30] V. Martina, M. Garetto, and E. Leonardi, "A unified approach to the performance analysis of caching systems," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2014, pp. 2040–2048.
[31] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Watch global, cache local: Youtube network traffic at a campus network: Measurements and implications," *Proc. SPIE*, vol. 6818, Jan. 2008, Art. no. 681805.
[32] N. Megiddo and D. S. Modha, "ARC: A self-tuning, low overhead replacement cache," in *Proc. FAST*, 2003, pp. 115–130.
[33] A. Dvoretzky, J. Kiefer, and J. Wolfowitz, "Asymptotic minimax character of the sample distribution function and of the classical multinomial estimator," *Ann. Math. Statist.*, vol. 27, no. 3, pp. 642–669, 1956.
[34] W. Hoeffding, "Probability inequalities for sums of bounded random variables," in *The Collected Works Wassily Hoeffding*. New York, NY, USA: Springer, 1994, pp. 409–426.
[35] S. Agrawal and N. Goyal, "Thompson sampling for contextual bandits with linear payoffs," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 127–135.
[36] J. Komiyama, J. Honda, and H. Nakagawa, "Optimal regret analysis of Thompson sampling in stochastic multi-armed bandit problem with multiple plays," 2015, *arXiv:1506.00779*. [Online]. Available: http://arxiv.org/abs/1506.00779
[37] S. Bubeck, V. Perchet, and P. Rigollet, "Bounded regret in stochastic multi-armed bandits," in *Proc. Conf. Learn. Theory*, 2013, pp. 122–134.
[38] P. Zerfos, M. Srivatsa, H. Yu, D. Dennerline, H. Franke, and D. Agrawal, "Platform and applications for massive-scale streaming network analytics," *IBM J. Res. Develop.*, vol. 57, nos. 3–4, pp. 1–11, 2013.
[39] X. Cheng, C. Dale, and J. Liu, "Statistics and social network of Youtube videos," in *Proc. 16th Interntional Workshop Qual. Service*, Jun. 2008, pp. 229–238.
[40] D. Kifer, S. Ben-David, and J. Gehrke, "Detecting change in data streams," in *Proc. VLDB*, vol. 4. Toronto, ON, Canada, 2004, pp. 180–191.

Archana Bura is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Texas A&M University. Her research interests include reinforcement learning, optimization, and their applications to wireless networks.

Desik Rengarajan is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Texas A&M University. His research interests include reinforcement learning and game theory, with a focus on their application to the real world.

Dileep Kalathil (Senior Member, IEEE) received the Ph.D. degree from the University of Southern California (USC) in 2014. From 2014 to 2017, he was a Post-Doctoral Researcher with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX, USA. His research interests include reinforcement learning, with applications in communication networks, power systems, and intelligent transportation systems. He was a recipient of the NSF CAREER Award in 2021, the NSF CRII Award in 2019, the Best Ph.D. Dissertation Award from the Department of Electrical Engineering, USC, from 2014 to 2015, and the Best Academic Performance Award from the EE Department, IIT Madras, in 2008.

Srinivas Shakkottai (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana–Champaign in 2007. He was a Post-Doctoral Scholar in management science and engineering with Stanford University in 2007. He joined Texas A&M University in 2008, where he is currently a Professor of computer engineering with the Department of Electrical and Computer Engineering. His research interests include caching and content distribution, wireless networks, multi-agent learning and game theory, and network data collection and analytics. He was a recipient of the Defense Threat Reduction Agency Young Investigator Award (2009), the NSF Career Award (2012), and the Research Awards from Cisco (2008) and Google (2010). He also received an Outstanding Professor Award (2013), the Select Young Faculty Fellowship (2014), and the Engineering Genesis Award (2019) at Texas A&M University.

Jean-Francois Chamberland (Senior Member, IEEE) received the Ph.D. degree from the University of Illinois at Urbana–Champaign. He is currently a Professor with the Department of Electrical and Computer Engineering, Texas A&M University. His research interests include computing, information, and inference. He was a recipient of the IEEE Young Author Best Paper Award from the IEEE Signal Processing Society and the Faculty Early Career Development (CAREER) Award from the National Science Foundation. He served as an Associate Editor for the IEEE TRANSACTIONS ON INFORMATION THEORY from 2017 to 2020.