Universal Caching

Ativ Joshi
School of Technology and Computer Science
Tata Institute of Fundamental Research
Mumbai 400 005, India
Email: ativ@cmi.ac.in

Abhishek Sinha
School of Technology and Computer Science
Tata Institute of Fundamental Research
Mumbai 400 005, India
Email: abhishek.sinha@tifr.res.in

***Abstract*—In learning theory, the performance of an online policy is commonly measured in terms of the *static regret* metric, which compares the cumulative loss of an online policy to that of an optimal benchmark in hindsight. In the definition of static regret, the action of the benchmark policy remains *fixed* throughout the time horizon. Naturally, the resulting regret bounds become loose in non-stationary settings where fixed actions often suffer from poor performance. In this paper, we investigate a stronger notion of regret minimization in the context of online caching. In particular, we allow the action of the benchmark at any round to be decided by a finite state machine containing any number of states. Popular caching policies, such as LRU and FIFO, belong to this class. Using ideas from the universal prediction literature in information theory, we propose an efficient online caching policy with a sub-linear regret bound. To the best of our knowledge, this is the first data-dependent regret bound known for the caching problem in the universal setting. We establish this result by combining a recently-proposed online caching policy with an incremental parsing algorithm, namely Lempel-Ziv '78. Our methods also yield a simpler learning-theoretic proof of the improved regret bound as opposed to the involved problem-specific combinatorial arguments used in the earlier works.**

I. Introduction and Related Work

We investigate the standard caching problem from an online learning perspective [1]–[5]. Consider a library consisting of N unit-sized files $\{1, 2, \ldots, N\} \equiv [N]$, and a cache of storage capacity C (typically $C \ll N$). The system evolves in discrete rounds. At the beginning of round t, an online caching policy π prefetches (possibly in a randomized fashion) a set of C files, denoted by the incidence vector $\boldsymbol{y}_t \in \{0,1\}^N$, where $||\boldsymbol{y}_t||_1 = C$. After that, the user requests a file, which is denoted by the incidence vector $\boldsymbol{x}_t \in \{0,1\}^N$, such that $||\boldsymbol{x}_t||_1 = 1$ (see Figure 1)[1]. The file request sequence $\{x_t\}_{t\geq 1}$ could be adversarial. In the case of a *cache-hit*, which occurs when the requested file is present in the cache, the policy receives a unit reward. In the complementary event of a *cache-miss*, the policy receives zero rewards. For simplicity, we do not charge any cost for file downloads (see [5] for a model with download cost). Thus, the reward accrued by the policy at round t is given by $\langle \boldsymbol{x}_t, \boldsymbol{y}_t \rangle$. The goal of the caching policy is to achieve a hit rate close to that of an optimal offline finite-state prefetcher (FSP) described next.

Definition 1 (Finite State Prefetcher (FSP) [1])**.** An FSP is described by a quintuple $(\mathcal{S}, [N], g, f, s_0)$, where $\mathcal{S}$ is a finite set of states, $[N]$ is a library of N files, $g : \mathcal{S} \times [N] \to \mathcal{S}$ is the state transition function, $f : \mathcal{S} \to [N]^C$ is a possibly randomized prefetching policy that caches a set of C files depending on the current state, and s_0 is the initial state. The components of an FSP without the prefetcher function f is known as a Finite State Machine (FSM).

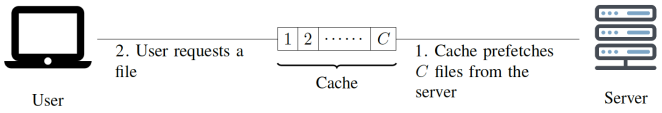


Fig. 1: Setup for the caching problem

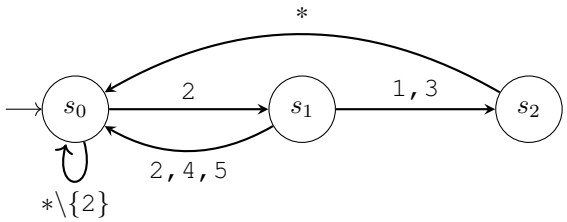
Let $x_1, x_2, \ldots$ be an N-ary sequence denoting the file requests. On round t, an FSP $\hat{\pi}$, which is currently at state s_t, first prefetches (possibly randomly) a set of C files given by $\hat{\boldsymbol{y}}_t(\hat{\pi}) = f(s_t)$, observes the file request x_t for round t, incurs cache hits/misses, and then finally changes its state to $s_{t+1} = g(s_t, x_t)$. The reward obtained by an FSP $\hat{\pi}$ at round t is given by the inner-product $\langle \boldsymbol{x}_t, \hat{\boldsymbol{y}}_t \rangle$. Denote the set of all FSPs containing at most s states by $\mathcal{G}_s$ and define the set of all FSPs by $\mathcal{G} = \cup_{s=1}^{\infty} \mathcal{G}_s$[2]. Informally, our objective is to design an online caching policy π that performs as well as the best FSP in hindsight that knows the entire file request sequence a priori. Quantitatively, our goal is to design an online caching policy π that attains a sublinear bound uniformly for all file request sequences for the regret metric $\mathcal{R}_T^{\pi}$ defined below:

$$\mathcal{R}_T^{\pi} = \sup_{\{\boldsymbol{x}_t\}_{t\geq 1}} \left(\max_{\hat{\pi}\in\mathcal{G}} \sum_{t=1}^{T} \langle \boldsymbol{x}_t, \hat{\boldsymbol{y}}_t(\hat{\pi}) \rangle - \sum_{t=1}^{T} \langle \boldsymbol{x}_t, \boldsymbol{y}_t(\pi) \rangle \right). \quad (1)$$

A brief discussion on the benchmark class $\mathcal{G}$ used in the performance metric (1) is in order. In the case of the standard static regret minimization problems, the action of the offline benchmark $\hat{\boldsymbol{y}}_t$ remains constant throughout the entire time horizon of interest [6]. A number of recent papers studied the static regret minimization problem in the context of caching and proposed efficient online policies achieving sublinear regret [2]–[5], [7], [8]. However, in terms of the absolute performance (total number of cache hits), these policies may perform poorly in "non-stationary" settings where the best

[1]We will be using the (one-hot encoded) vectorized symbols $\boldsymbol{x}_t \in \{0,1\}^N$ and the corresponding scalars $x_t \in [N]$ interchangeably throughout the paper.

[2]To be precise, the class $\mathcal{G}$ is parameterized by the numbers N and C. Since these parameters will be clear from the context, we drop the parameters to avoid cluttering the notations.



(a) State transition function $g(s, x)$ of the FSM.

States (s) \ Symbol (x)	1	2	3	4	5
s_0	0	4	0	0	1
s_1	1	0	2	1	0
s_2	0	0	0	1	2

(b) Frequency $N_T(s, x)$.

	$f^*(s)$
s_0	{2,5}
s_1	{1,3}
s_2	{4,5}

(c) Optimal prefetcher function $f^*(s)$

Fig. 2: This figure illustrates the computation of the optimal offline prefetcher for a given 3-state FSM for the 5-ary input sequence of length $T = 12$ given by $(2, 1, 5, 2, 3, 5, 2, 4, 5, 2, 3, 4)$. We assume that the size of the cache is $C = 2$. The state transition function $g(\cdot)$ is shown in part (a) of the figure. The variable $N_T(s, x), x \in [5]$, denotes the number of times the file x was requested while the FSM was visiting state s. For the given request sequence, the sequence of states visited by the FSM is given by $(s_0, s_1, s_2, s_0, s_1, s_2, s_0, s_1, s_0, s_0, s_1, s_2)$. Upon counting the frequency of the requests at each state, it is easy to see that the optimal prefetcher for the given FSM is $f^*(s_0) = \{2, 5\}$, $f^*(s_1) = \{1, 3\}$ and $f^*(s_2) = \{4, 5\}$. The fraction of cache misses conceded by the optimal offline FSP is 0.083.

offline static cache configuration has a poor hit rate. In the online learning literature, several generalizations of the static regret metric have been proposed to quantify the performance of policies in non-stationary environments. For example, the *Tracking Regret* metric allows changing the benchmark a fixed number of times within a given time horizon [9]–[11]. The *Adaptive Regret* metric compares the performance of an online policy over any arbitrary sub-interval $[s, e] \subseteq [T]$ with the best static policy $\pi^*[s, e]$ for that sub-interval [12]–[15]. In *Dynamic Regret*, the benchmarks are allowed to vary slowly with time, subject to certain regularity constraints [14], [16], [17]. The FSP benchmark considered in the paper includes a rich class of comparators, which arises naturally in many contexts. In Section VII-C of the Appendix, we show that popular caching policies with optimal competitive ratios, such as LRU and FIFO, belong to this class.

The seminal paper [18] considers a special case of the regret minimization problem (1), which, in our setup, corresponds to a library of size $N = 2$ and a cache of capacity $C = 1$. In this context, the authors proposed an efficient universal prefetching policy by utilizing the Lempel-Ziv incremental parser [19]. Follow this up, the paper [1] considered the online caching problem with arbitrary values for N and C, and proposed a universal caching policy achieving a sublinear regret. One of the key contributions of [1] is the design of a new prefetching policy that is competitive against a single state FSP. In this paper, we give a tighter *data-dependent* regret bound by utilizing a recent online learning policy obtained by combining the standard HEDGE algorithm with Madow's sampling [6], [20]–[22]. These improved bounds are obtained by using general learning-theoretic techniques, as opposed to the involved combinatorial arguments employed in [1]. We also mention the paper [23] which proposes a universal caching policy that is constant-factor optimal in the stochastic setting.

II. CHARACTERIZATION OF FINITE-STATE PREFETCHERS

Before designing online policies, we first characterize the offline performance of the FSPs. In particular, we show that with almost no loss of generality, our attention can be restricted to a sub-class of FSPs, known as *Markov Prefetchers*.

Characterization of the Optimal Offline Prefetcher: Assume that an FSM $\mathcal{M}$ is run with the file request sequence x_1^T. In this case, the optimal offline prefetcher f^*, that maximizes the cumulative hits, is easy to determine (see Figure 2 for an illustration). Let the variable $N_T(s, i)$ denote the number of times the i^{th} file was requested while the FSM was visiting the state s. Let the set $\mathcal{A}_s$ denotes the most frequently-requested collection of C files while the FSM was on the state s. Since the set of the prefetched file depends only on the current state of the FSP, the optimal offline prefetcher function for $\mathcal{M}$ is given by $f^*(s) = \mathcal{A}_s$. We now recall a special sub-class of FSPs, known as *Markov Prefetchers*, that plays a central role in universal caching.

Definition 2 (k^{th}-order Markov Prefetcher [18])**.** A k^{th} order Markov Prefetcher is a sub-class of FSPs with N^k states, where the state at round t is given by the k-tuple of the previous k file requests, *i.e.,* $s_t = (x_{t-1}, x_{t-2}, \ldots, x_{t-k})$. The state transition function $g(\cdot)$ is defined naturally using a shift operator.

For any given file request sequence x_1^T, let $\tilde{\pi}_S(x_1^T)$ denote the maximum fraction of cache hits [3] achieved by any FSP containing at most S states and $\tilde{\mu}_k(x_1^T)$ denote the maximum fraction of cache hits achieved by a k^{th} order Markov prefetcher. The following result, which is a generalization of [18, Theorem 2] shows that the Markov Prefetchers are asymptotically optimal in the class of FSPs.

Theorem 1. *The hit rate of a Markovian prefetcher of a sufficiently large order k exceeds the hit rate of any FSP with a fixed (S) number of states (up to a vanishingly small term). In particular, for any file request sequence x^T, we have*

$$\tilde{\pi}_S(x^T) - \tilde{\mu}_k(x^T) \leq \min\left(1 - C/N, \sqrt{\frac{\ln S}{2(k+1)}}\right). \quad (2)$$

Please refer to Section VII-A of the Appendix for the proof of Theorem 1. The proof closely follows the arguments for the binary case given in [18]. The message conveyed by Theorem 1 is that, in order to be competitive with *any* FSM with a finite number of states S, an online policy only needs to be competitive with respect to a Markov prefetcher of a sufficiently

[3]For notational conveniences, we work with cache hits rather than cache misses as in [18]. We use the tilde symbol on the top of the variables to emphasize that they represent offline quantities.

large order ($k \gg \ln(S)$). The latter problem can be handled using techniques from the online learning theory, which we discuss in the following section.

III. An Online Caching Policy that is Competitive against all Finite-State Prefetchers

As the first step towards designing a universal caching policy, we propose a basic online prefetcher that is competitive against the optimal offline single-state (*i.e.,* zeroth-order Markov) prefetcher, where the action of the comparator remains fixed throughout. Subsequently, we show how to extend the proposed prefetcher to compete against multi-state FSPs. We use the classic *Prediction with Expert advice* framework [6] to design our basic online prefetching policy. This is in sharp contrast with the paper [1], which proposes a problem-specific basic prefetching policy and carries out its analysis using an involved combinatorial method.

A. Prediction with Expert advice and Online Caching

For the sake of completeness, we first briefly review the framework of *Prediction with Expert Advice*. Assume that there is a set of M experts. Consider a two-player sequential game played between the learner and an adversary described next. At each round t, the adversary selects a reward value $r_{ti} \in [0,1]$ for each expert $i \in [M]$. At the same time (without knowing the rewards for the current round), the learner samples an expert randomly according to a probability distribution $\boldsymbol{p}_t$ and accrues the expected reward $\langle \boldsymbol{p}_t, \boldsymbol{r}_t \rangle$. The objective of the learner is to achieve a small regret (1) with respect to the best expert in hindsight. Many variants of the above problem have been studied in the literature and multiple different online policies achieving sublinear regret for this problem are known [24], [25].

One of the most fundamental algorithms for the experts problem is Hedge (also known as Exponential Weights). Let the vector $\boldsymbol{R}_{t-1} = \sum_{\tau=1}^{t-1} \boldsymbol{r}_t$ denote the cumulative rewards of all experts up to round $t-1$. At round t, the Hedge policy chooses the distribution $p_{t,i}^{\text{Hedge}} \propto \exp(\eta R_{t-1})$ for some fixed learning rate $\eta > 0$. It is well-known that the Hedge policy achieves the following regret bound [24]:

$$\max_{i\in[M]} R_{T,i} - \sum_{t=1}^{T} \langle \boldsymbol{p}_t^{\text{Hedge}}, \boldsymbol{r}_t \rangle \leq \frac{\ln M}{\eta} + \eta \sum_{t=1}^{T}\sum_{i=1}^{M} p_{t,i} l_{t,i}^2, \quad (3)$$

where $l_{t,i} = 1 - r_{t,i}$ is the loss of the i^{th} expert at round t.

Connection to the Caching problem: The problem of designing an online prefetcher that competes against a static benchmark can be straightforwardly reduced to the previous experts framework. For this purpose, define an instance of the experts problem with $M = \binom{N}{C}$ experts, each corresponding to a subset of C files. Let the reward $r_{t,i}$ accrued by the i^{th} subset at round t be equal to 1 if the i^{th} expert (which corresponds to a particular subset of C files) contains the file x_t requested at round t. Else, the value of $r_{t,i}$ is set to zero. A simple but computationally inefficient online caching policy can be obtained by using the Hedge policy on the experts problem defined above. However, a major issue with this naive reduction is that, apparently, it needs to maintain an exponentially large probability vector $\boldsymbol{p}_t$ (with $\binom{N}{C}$ components) at every round t, which is clearly computationally infeasible. In a recent paper, we proposed the Sage policy, which gives a near-linear time implementation of the Hedge policy in this context [20, Algorithm 3]. We now review the Sage policy and show how it can be used in the context of Universal Caching.

B. The Sage Framework for Online Caching [20]

The Sage framework, proposed in [20, Algorithm 1], gives a generic meta-policy that yields an efficient implementation of the Hedge policy by using randomized sampling and exploiting the linearity of the reward function. In particular, we observe that in the online caching problem, the reward accrued by the learner at any round depends only on the *marginal* inclusion probabilities of each file, and not on their joint distribution. Hence, any online learning policy, that yields the same marginal inclusion probabilities as the Hedge policy, achieves the same regret as the Hedge policy. It is inconsequential whether the joint inclusion probabilities are the same or different for these two policies. Based on the above simple observation, the Sage meta-policy works as follows. (a) First, it efficiently computes the marginal file inclusion probabilities induced by the Hedge policy by exploiting the linearity of the reward function. (b) Then it efficiently samples a subset of C files without replacement consistent with the marginals computed in the previous step. In the following, we outline how the above two steps can be carried out efficiently.

a) Efficient computation of the marginal inclusion probabilities: Let the expert S correspond to the subset S of files (with $|S| = C$). The Hedge policy assigns the following probability mass to the expert S at round t:

$$p_t(S) = \frac{w_{t-1}(S)}{\sum_{S'\subseteq[N]:|S'|=C} w_{t-1}(S')}, \quad (4)$$

where $w_{t-1}(S) \equiv \exp(\eta R_{t-1}(S))$, s.t. $R_{t-1}(S) \equiv \sum_{\tau=1}^{t-1} \mathbb{1}(x_\tau \in S)$ denotes the cumulative (offline) cache hits accrued by the subset S up to round t, and η is the learning rate. Consequently, the marginal inclusion probability for the i^{th} file is given by:

$$p_t(i) = \frac{w_{t-1}(i) \sum_{S\subseteq[N]\setminus\{i\}:|S|=C-1} w_{t-1}(S)}{\sum_{S'\subseteq[N]:|S'|=C} w_{t-1}(S')}. \quad (5)$$

In the above, we have defined $w_{t-1}(i) \equiv \exp(\eta R_{t-1}(i))$, where $R_{t-1}(i) \equiv \sum_{\tau=1}^{t-1} \mathbb{1}(x_\tau = i)$ denotes the total number of times the i^{th} file was requested up to time $t-1$. Both the numerator and the denominator in the probability expression (5) have exponentially many terms and are non-trivial to compute directly. A key observation made in [20] is that both the numerator and denominator can be expressed in terms of certain *elementary symmetric polynomials* (ESP), which can be efficiently evaluated. To see this, define the vectors $\boldsymbol{w}_t = (w_t(i))_{i\in[N]}$ and $\boldsymbol{w}_{-i,t} = (w_t(i))_{i\in[N]\setminus\{i\}}$. Let $\boldsymbol{e}_k(\cdot)$ denote the ESP of order k, defined as follows:

$$e_k(\boldsymbol{w}) = \sum_{I\subseteq[N],|I|=k} \prod_{j\in I} w_j.$$

With the above definitions in place, the probability term given in (5) can be expressed in terms of ESPs as:

$$p_t(i) = \frac{w_t(i)\boldsymbol{e}_{C-1}(\boldsymbol{w}_{-i,t})}{\boldsymbol{e}_C(\boldsymbol{w}_t)}. \tag{6}$$

It is known that any ESP of order k with N variables can be computed efficiently in $\tilde{O}(N)$ time using FFT-based polynomial multiplication methods [26], [27].

b) Sampling without replacement according to a prescribed set of inclusion probabilities: Consider the problem of efficiently sampling a subset of C items without replacement from a universe of N items, where the i^{th} item is included in the sampled subset with a prescribed probability $p_i \in [0,1], 1 \leq i \leq N$. In other words, if the set S is sampled with probability $\mathbb{P}(S)$, then it is required that $\sum_{S:i\in S,|S|=k} \mathbb{P}(S) = p_i, \forall i \in [N]$. Given that the inclusion probabilities satisfy the necessary and sufficient condition $\sum_{i=1}^{N} p_i = k$, the sampling problem can be efficiently solved using Madow's systematic sampling procedure given below [21].

Algorithm 1 Madow's sampling

Input: Set $[N]$, size of the sampled set C, probability $\boldsymbol{p}$.
Output: A random set S containing C elements s.t. $\mathbb{P}(i \in S) = p_i, \forall i$.

1: Let $P_0 = 0$ and $P_i = P_{i-1} + p_i, \forall i \in [N]$
2: Sample a uniform random variable $U \in [0,1]$.
3: $S \leftarrow \emptyset$
4: **for** $i \leftarrow 0$ to $k-1$ **do**
5: Select element j if $P_{j-1} \leq U + i \leq P_j$
6: $S \leftarrow S \cup \{j\}$
7: **end for**
8: **return** S

Combining part (a) and (b), the overall SAGE caching policy is summarized in Algorithm 2.

Algorithm 2 Online Caching with the SAGE framework

Input: $w(i) = 1, \forall i \in [N]$, learning rate $\eta > 0$.
Output: A subset of C cached files at every round

1: **for** $t = 1, 2, \ldots T$ **do**
2: $w(i) \leftarrow \exp(\eta \mathbb{1}\{x_{t-1} = i\}) w(i), \forall i \in [N]$.
3: $p(i) \leftarrow \frac{w(i)\boldsymbol{e}_{C-1}(\boldsymbol{w}_{-i})}{\boldsymbol{e}_C(\boldsymbol{w})}, \forall i \in [N]$. ▷ *Efficient evaluation using FFT*
4: Sample a set of C files, with marginal inclusion probabilities $\boldsymbol{p}$ computed as above, using Madow's sampling (Algorithm 1).
5: **end for**

Static regret bound for the SAGE *policy:* Recall that the quantity $\tilde{\pi}_1(x^T)$ denotes the hit rate achieved by the optimal offline FSP containing a single state. By tuning the learning rate η adaptively, the HEDGE policy achieves the following data-dependent regret bound [20, Eqn. (14)]:

$$T(\tilde{\pi}_1 - \pi^{\text{HEDGE}}) \leq \sqrt{2Cl_T^* \ln(Ne/C)} + C\ln(Ne/C), \tag{7}$$

where $l_T^* \equiv T - T\tilde{\pi}_1(x^T)$ is the cumulative number of cache misses incurred by the optimal offline caching configuration in hindsight. From the above discussion, it is clear that the SAGE policy also achieves the regret bound (7). Since $l_T^* \leq T$, Eqn. (7) trivially yields a sublinear $O(\sqrt{T})$ static regret bound for the online caching problem. Hence, our regret bound (7) improves upon the previous $O(NC^2 \log T\sqrt{T})$ regret bound of the prefetcher (referred to as "P_1") proposed by [1, Theorem 1, Lemma 3]. More importantly, Eqn. (7) gives what is known as a "small-loss" bound [28]. In particular, for any request sequence for which the optimal static offline policy concedes a small number of cache-misses (*i.e.,* $l_T^* \ll T$), Eqn. (7) provides a much tighter bound. We will exploit the small-loss bound in our subsequent analysis.

C. Augmenting the SAGE policy with a Markovian Prefetcher

Equation (7) gives an upper-bound on the static regret for the SAGE caching policy against all offline static prefetchers where the action of the benchmark policy does not change with time. Now consider any given FSM $\mathcal{M}$ containing S number of states. For each state $s \in \mathcal{S}$, let $\boldsymbol{x}_s$ be the subsequence of the original file requests obtained by aggregating the requests when the FSM $\mathcal{M}$ was visiting the state s. Upon running a separate copy of the SAGE policy for each state of the given FSM $\mathcal{M}$, we obtain the following regret bound:

$$\begin{aligned}
\mathcal{R}_T &= T(\tilde{\pi}_S^{\mathcal{M}}(x^T) - \pi_S^{\text{SAGE}}(x^T)) \\
&= T\sum_{s=1}^{S} \left(\tilde{\pi}_1^{\mathcal{M}}(\boldsymbol{x}_s) - \pi_1^{\text{SAGE}}(\boldsymbol{x}_s)\right) \\
&\overset{(a)}{\leq} \sum_{s=1}^{S} \sqrt{2Cl_{T,s}^* \ln(Ne/C)} + CS\ln(Ne/C) \\
&\overset{(b)}{\leq} \sqrt{2CSL_{T,S}^* \ln(Ne/C)} + CS\ln(Ne/C),
\end{aligned} \tag{8}$$

where $l_{T,s}^*$ denotes the total number of cache misses in the state s incurred by the optimal offline single-state prefetcher and $L_{T,S}^* \equiv \sum_{s=1}^{S} l_{T,s}^*$. In the above, inequality (a) follows from Eqn. (7) applied to each of the S states of the FSM $\mathcal{M}$ separately, and inequality (b) follows from an application of Jensen's inequality. Specializing the bound (8) to a k^{th} order Markov-prefetcher containing $S = N^k$ many states, we obtain

$$\begin{aligned}
T(\tilde{\mu}_k(x^T) - \pi_k^{\text{SAGE}}(x^T)) \leq \sqrt{2N^k CL_{T,k}^* \ln\frac{Ne}{C}} \\
+ N^k C \ln\frac{Ne}{C},
\end{aligned} \tag{9}$$

where $L_{T,k}^*$ denotes the minimum number of cache misses incurred by the optimal k^{th} order Markovian prefetcher for the file request sequence x^T. Note that the cumulative cache misses $L_{T,k}^*$ could be much smaller than the horizon-length T for many "regular" request sequences. Hence, Theorem 2 gives a new and tighter adaptive regret bound compared to the previously-known weaker $O(\sqrt{T})$ bound given by [18, Eqn. (24)].

EXAMPLE 1: Consider a "regular" request sequence x^T generated by an m^{th}-order Markovian FSM. By taking $k \geq m$,

we can ensure that $L^*_{T,k}(x^T) = 0$ for this sequence. Hence, in this case, the first term on the RHS of the bound (9) vanishes, resulting in $O(1)$ regret.

Combining the regret bound (8) with Theorem 1, we have the following guarantee against any FSM containing S many states:

Theorem 2. *For any file request sequence $\boldsymbol{x}^T$, the regret of the k^{th} order Markovian FSM running a separate copy of the* SAGE *caching policy on each state, compared to an optimal offline FSP containing at most S states, is upper-bounded as:*

$$\mathcal{R}_T \equiv T(\tilde{\pi}_S(x^T) - \pi_k^{\text{SAGE}}(x^T)) \leq T \min\left(1 - C/N, \sqrt{\frac{\ln S}{2(k+1)}}\right) + \sqrt{2N^k C L^*_{T,k} \ln \frac{Ne}{C}} + N^k C \ln \frac{Ne}{C}.$$

EXAMPLE 2: Consider a file request sequence $\boldsymbol{x}_Q^T$ generated by any FSM containing at most Q states. The FSM needs not be Markovian (c.f. EXAMPLE 1). Refer to Section VIII-A of the Appendix for details on the request sequence generation. Combining Theorem 1 and Theorem 2, we have:

$$\begin{aligned} &\text{Expected fraction of cache misses conceded by the} \\ &\text{SAGE policy used with a } k^{\text{th}} \text{ order Markovian FSM} \\ \leq\ & \min\left(1 - \frac{C}{N}, \sqrt{\frac{\ln Q}{2(k+1)}}\right) + \\ & \sqrt{\frac{2N^k C}{T} \ln \frac{Ne}{C} \min\left(1 - \frac{C}{N}, \sqrt{\frac{\ln Q}{2(k+1)}}\right)} \\ & + \frac{N^k C}{T} \ln \frac{Ne}{C}, \end{aligned} \tag{10}$$

where we have used the fact that an optimal FSM with Q many states incurs *zero* cache misses for the request sequence $\boldsymbol{x}_Q^T$. If the value of Q is known (however, the structure of the FSM remains unknown), the optimal order k^* of the Markovian prefetcher minimizing the upper bound in Eqn. (10) can be computed using calculus. From Eqn. (10), it also follows that for any fixed value of Q, the expected fraction of cache-misses can be made approach to zero at the rate of $O(T^{-1/2})$ by taking $k \gg \ln Q$. See Section VIII for the numerical results.

In the following section, we design a universal caching policy that achieves a sublinear $O\big((\log T)^{-1/2}\big)$ regret bound for all file request sequences against any FSP containing *unknown and arbitrarily many states S*.

IV. A UNIVERSAL CACHING POLICY

In Theorem (2), we are free to choose the order k of the Markovian FSM as a function of the (known) horizon-length T. Since the number of states S in the benchmark comparator could be arbitrarily large, it is clear that in order to achieve asymptotically zero regret (normalized w.r.t. the horizon length T), the order k of the Markovian prefetcher needs to be increased accordingly with T. By setting $N^k = O(\frac{T}{\log T})$ in Theorem 2, we obtain the following bound on the regret against *all* FSPs having arbitrarily many states: $\frac{\mathcal{R}_T}{T} \leq O(\frac{1}{\sqrt{\log T}})$.

Efficient implementation using Lempel-Ziv (LZ) parsing

Similar to the binary prediction problem considered in [18], we can use incremental parsing algorithms, such as Lempel-Ziv'78 [19] to adaptively increment the order of the Markovian Prefetcher when the horizon length T is not known a priori [1], [29]. However, instead of constructing a binary parse tree as in [18], we build an N-ary tree with the SAGE policy running at each node. In particular, the LZ parsing algorithm parses the N-ary request sequence into distinct phrases such that each phrase is the shortest phrase that is not a previously parsed phrase. In the parse tree, each new phrase corresponds to a leaf in the tree. The parsing proceeds as follows: the LZ tree is initialized with a root node and N leaves. The current tree is used to create the next phrase by following the path from the root to leave according to the consecutive file requests. Once a leaf node is reached, the tree is extended by making the leaf an internal node, and adding N offsprings to the tree and then moving to the root of the tree. Each node of the tree now corresponds to a state of the Markovian prefetcher and runs a separate instance of the SAGE policy. A classical result, established in [30, Theorem 2], tells that the number of nodes in an N-ary LZ tree generated by an arbitrary sequence of length T grows sub-linearly as $c(T) = O(\frac{T \log N}{\log T})$. Hence, for any fixed k, the fraction of file requests made on a node with depth less than k vanishes asymptotically. Hence, the expected fraction of cache hits π^{LZ} achieved by the LZ prefetcher is asymptotically lower bounded by that of a k^{th} order Markovian FSP containing $N^k \approx c(T)$ states up to a sublinear regret term. The following theorem makes this statement precise.

Theorem 3. *For any integer $k \geq 0$, the regret of the* LZ *prefetcher w.r.t. an offline k^{th} order Markovian prefetcher can be upper-bounded as:*

$$\mathcal{R}_T \equiv T(\tilde{\mu}_k - \pi^{\text{LZ}}) \leq \delta(c(T), L_T^{*,LZ}) + kc(T),$$

where $c(T) \equiv O(\frac{T \log N}{\log T})$ *and* $\delta(B, l_T^*) \equiv \sqrt{2BCL_T^{*,LZ} \ln(Ne/C)} + CB \ln(Ne/C)$.

See Section VII-B of the Appendix for the proof.

V. CONCLUSION

In this paper, we proposed an efficient online universal caching policy that results in a sublinear regret against all finite-state prefetchers containing arbitrarily many states. We presented the first data-dependent regret bound for the universal caching problem by making use of the SAGE framework [20]. In the future, it will be interesting to extend these techniques to other online learning problems to design policies with improved regret guarantees. Furthermore, designing universal algorithms that also guarantee sublinear *dynamic* regret and take into account the file download costs will be of interest.

VI. ACKNOWLEDGMENT

This work was partly supported by a grant from the DST-NSF India-US collaborative research initiative under the TIH at the Indian Statistical Institute at Kolkata, India.

REFERENCES

[1] P. Krishnan and J. S. Vitter, "Optimal prediction for prefetching in the worst case," *SIAM Journal on Computing*, vol. 27, no. 6, pp. 1617–1636, 1998.

[2] 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, Jun. 2020. [Online]. Available: https://doi.org/10.1145/3392143

[3] G. S. Paschos, A. Destounis, L. Vigneri, and G. Iosifidis, "Learning to cache with no regrets," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 235–243.

[4] D. Paria and A. Sinha, "LeadCache: Regret-optimal caching in networks," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[5] S. Mukhopadhyay and A. Sinha, "Online caching with optimal switching regret," in *2021 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2021, pp. 1546–1551.

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

[7] G. Paschos, G. Iosifidis, G. Caire *et al.*, "Cache optimization models and algorithms," *Foundations and Trends® in Communications and Information Theory*, vol. 16, no. 3–4, pp. 156–345, 2020.

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

[9] M. Herbster and M. K. Warmuth, "Tracking the best expert," *Machine learning*, vol. 32, no. 2, pp. 151–178, 1998.

[10] N. Cesa-Bianchi, P. Gaillard, G. Lugosi, and G. Stoltz, "Mirror descent meets fixed share (and feels no regret)," *Advances in Neural Information Processing Systems*, vol. 25, 2012.

[11] L. Chen, Q. Yu, H. Lawrence, and A. Karbasi, "Minimax regret of switching-constrained online convex optimization: No phase transition," *Advances in Neural Information Processing Systems*, vol. 33, pp. 3477–3486, 2020.

[12] E. Hazan and C. Seshadhri, "Adaptive algorithms for online decision problems," in *Electronic colloquium on computational complexity (ECCC)*, vol. 14, no. 088, 2007.

[13] A. Daniely, A. Gonen, and S. Shalev-Shwartz, "Strongly adaptive online learning," in *International Conference on Machine Learning*. PMLR, 2015, pp. 1405–1411.

[14] L. Zhang, T. Yang, Z.-H. Zhou *et al.*, "Dynamic regret of strongly adaptive methods," in *International conference on machine learning*. PMLR, 2018, pp. 5882–5891.

[15] D. Adamskiy, W. M. Koolen, A. Chernov, and V. Vovk, "A closer look at adaptive regret," in *International Conference on Algorithmic Learning Theory*. Springer, 2012, pp. 290–304.

[16] O. Besbes, Y. Gur, and A. Zeevi, "Non-stationary stochastic optimization," *Operations research*, vol. 63, no. 5, pp. 1227–1244, 2015.

[17] A. Jadbabaie, A. Rakhlin, S. Shahrampour, and K. Sridharan, "Online optimization: Competing with dynamic comparators," in *Artificial Intelligence and Statistics*. PMLR, 2015, pp. 398–406.

[18] M. Feder, N. Merhav, and M. Gutman, "Universal prediction of individual sequences," *IEEE transactions on Information Theory*, vol. 38, no. 4, pp. 1258–1270, 1992.

[19] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE transactions on Information Theory*, vol. 24, no. 5, pp. 530–536, 1978.

[20] S. Mukhopadhyay, S. Sahoo, and A. Sinha, "k-experts-Online Policies and Fundamental Limits," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2022, pp. 342–365.

[21] W. G. Madow *et al.*, "On the theory of systematic sampling, ii," *The Annals of Mathematical Statistics*, vol. 20, no. 3, pp. 333–354, 1949.

[22] Y. Tillé, *Sampling algorithms*. Springer, 2006.

[23] G. Pandurangan and W. Szpankowski, "A universal online caching algorithm based on pattern matching," *Algorithmica*, vol. 57, no. 1, pp. 62–73, 2010.

[24] V. Vovk, "A game of prediction with expert advice," *Journal of Computer and System Sciences*, vol. 56, no. 2, pp. 153–173, 1998.

[25] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[26] A. Shpilka, "Lower bounds for small depth arithmetic and boolean circuits," 2001.

[27] V. Grolmusz, "Computing elementary symmetric polynomials with a subpolynomial number of multiplications," *SIAM Journal on Computing*, vol. 32, no. 6, pp. 1475–1487, 2003.

[28] T. Lykouris, K. Sridharan, and É. Tardos, "Small-loss bounds for online learning with partial information," in *Conference on Learning Theory*. PMLR, 2018, pp. 979–986.

[29] J. S. Vitter and P. Krishnan, "Optimal prefetching via data compression," *Journal of the ACM (JACM)*, vol. 43, no. 5, pp. 771–793, 1996.

[30] A. Lempel and J. Ziv, "On the complexity of finite sequences," *IEEE Transactions on information theory*, vol. 22, no. 1, pp. 75–81, 1976.

APPENDIX

VII. PROOFS

A. Proof of Theorem 1

Consider an FSP $\mathcal{M}$, containing S states, whose state transition function is given by g. For any integer $j \geq 1$, construct a new FSP $\tilde{\mathcal{M}}$ whose state at round t is given by $\tilde{s}_t = (s_t, x_{t-j}^{t-1})$. In other words, the states of the FSP $\tilde{\mathcal{M}}$ are constructed by juxtaposing the states of $\mathcal{M}$ and the k^{th} order Markov prefetcher. The transition function of $\tilde{\mathcal{M}}$ is naturally defined as $\tilde{g}((s_t, x_{t-j}^{t-1}), x_t) = (g(s_t), x_{t-j+1}^{t})$. Consequently, if both the FSPs $\mathcal{M}$ and $\tilde{\mathcal{M}}$ are fed with the same file request sequence $\boldsymbol{x}$, the current state of the FSP $\mathcal{M}$ can be read off from the first component of the current state of the FSP $\tilde{\mathcal{M}}$. Let $\tilde{\mu}_{j,S}(x^T)$ be the hit rate achieved by the new FSP $\tilde{\mathcal{M}}$ for the given file request sequence. Define a sequence of random variables $(X_1, X_2, \ldots)$, which are distributed according to the empirical probability measure induced by the consecutive terms of the given file request sequence x_1^T. In other words, for any natural number k, we define the joint distribution:

$$\begin{aligned} &\mathbb{P}\big((X_1, X_2, \ldots, X_k) = (z_1, z_2, \ldots, z_k)\big) \\ = \;& \frac{|\{q : (x_q, x_{q+1}, \ldots, x_{q+k-1}) = (z_1, z_2, \ldots, z_k)\}|}{T}. \end{aligned}$$

Recall that the quantity $\tilde{\mu}_j(x^T)$ denotes the cache hit rate achieved by the k^{th} order Markovian prefetcher. We first establish the following technical result.

Proposition 1. *For any file request sequence x^T, we have*

$$\tilde{\mu}_{j,S}(x^T) - \tilde{\mu}_j(x^T) \leq \min\Bigg(1 - C/N, \sqrt{\frac{\ln 2}{2}\big(H(X_{j+1}|X^j) - H(X_{j+1}|X^j, S)\big)}\Bigg).$$

Proof. Corresponding to the state $\tilde{s}_t \equiv (s_t, x_{t-j}^{t-1}) = (s, x^j)$ of the FSP $\tilde{\mathcal{M}}$, let $\tilde{p}_{s,x^j}(\cdot)$ be the conditional probability distribution for the next file request x_t. For any r.v. Z, let the quantity $\mathbb{E}_Z(\cdot)$ denote the expectation w.r.t. the empirical distribution of the r.v. Z. Since the optimal offline policy caches the most requested C files on any state, we have the following sequence of bounds

$$\begin{aligned} &\tilde{\mu}_{j,S}(x^T) - \tilde{\mu}_j(x^T) \\ = \;& \mathbb{E}_{S,X^j}\Bigg(\max_{A:|A|=C}\sum_{i\in A} p_{S,X^j}(X_{j+1} = i) - \\ & \max_{A:|A|=C}\sum_{i\in A} p_{X^j}(X_{j+1} = i)\Bigg) \\ \overset{(a)}{\leq} \;& \mathbb{E}_{S,X^j}\big(\max_{A:|A|=C}\sum_{i\in A}(p_{S,X^j}(i) - p_{X^j}(i))\big) \\ \leq \;& \mathbb{E}_{S,X^j}\big(\mathrm{TV}(p_{S,X^j}(X_{j+1}), p_{X^j}(X_{j+1}))\big) \\ \overset{(b)}{\leq} \;& \mathbb{E}_{S,X^j}\left(\sqrt{\frac{\ln 2}{2} D(P(X_{j+1}|S, X^j)||P(X_{j+1}|X^j))}\right) \\ \overset{(c)}{\leq} \;& \sqrt{\frac{\ln 2}{2} D(P(X_{j+1}|S, X^j)||P(X_{j+1}|X^j))} \\ = \;& \sqrt{\frac{\ln 2}{2}(H(X_{j+1}|X^j) - H(X_{j+1}|X^j, S))}. \end{aligned} \tag{11}$$

where (a) follows from the sub-additivity of the $\max(\cdot)$ function, (b) follows from Pinsker's inequality, and (c) follows from the concavity of the square root function and Jensen's inequality. Furthermore, we also have

$$\begin{aligned} &\tilde{\mu}_{j,S}(x^T) - \tilde{\mu}_j(x^T) \\ = \;& \mathbb{E}_{S,X^j}\Bigg(\max_{A:|A|=C}\sum_{i\in A} p_{S,X^j}(X_{j+1} = i) - \\ & \max_{A:|A|=C}\sum_{i\in A} p_{X^j}(X_{j+1} = i)\Bigg) \\ \leq \;& 1 - C/N. \end{aligned} \tag{12}$$

Combining the bounds (11) and (12) completes the proof of Proposition 1. □

We now establish Theorem 1. Since the current state of the FSP $\mathcal{M}$ is a deterministic function of the current state of the FSP $\tilde{\mathcal{M}}$, it immediately follows that $\tilde{\pi}_S(x^T) \leq \tilde{\mu}_{j,S}(x^T)$ for any $j \geq 1$. By the same argument, we have $\tilde{\mu}_k(x^T) \geq \tilde{\mu}_j(x^T), \forall k \geq j$. Hence, we have

$$\begin{aligned} &\tilde{\pi}_S(x^T) - \tilde{\mu}_k(x^T) \\ \leq \;& \tilde{\mu}_{j,S}(x^T) - \tilde{\mu}_k(x^T) \\ \leq \;& \frac{1}{k+1}\sum_{j=0}^{k}[\tilde{\mu}_{j,S}(x^T) - \tilde{\mu}_j(x^T)] \\ \overset{(a)}{\leq} \;& \frac{1}{k+1}\sum_{j=0}^{k}\min\Bigg(1 - C/N, \\ & \sqrt{\frac{\ln 2}{2}(H(X_{j+1}|X^j) - H(X_{j+1}|S, X^j))}\Bigg) \\ \overset{(b)}{\leq} \;& \min\Bigg(1 - C/N, \\ & \sqrt{\frac{\ln 2}{2(k+1)}\big(\sum_{j=0}^{k} H(X_{j+1}|X^j) - H(X_{j+1}|S, X^j)\big)}\Bigg) \\ \overset{(c)}{=} \;& \min\Bigg(1 - C/N, \\ & \sqrt{\frac{\ln 2}{2(k+1)}\big(H(X^{k+1}) - H(X^{k+1}|S)\big)}\Bigg) \end{aligned}$$

$$= \min\left(1 - C/N, \sqrt{\frac{\ln 2}{2(k+1)} I(S; X^{k+1})}\right)$$
$$\overset{(d)}{\leq} \min\left(1 - C/N, \sqrt{\frac{\ln S}{2(k+1)}}\right),$$

where in (a), we have used Proposition 1 and in (b), we have Jensen's inequality twice (using the concavity of the $\min(\cdot)$ and the $\sqrt{(\cdot)}$ functions), in (c), we have used the chain rule for entropy, and in (d), we have trivially upper bounded the mutual information by $\log S$. ■

B. Proof of Theorem 3

Proof. Let $c(T)$ denote the total number of nodes in the LZ tree after time T and $L_T^{*,LZ}$ denote the number of cache misses by the optimal offline prefetching policy which follows the same tree growth process as the online LZ parsing algorithm (but could prefetch files different from that of the online policy). Using the bound (8) on each node of the LZ parse tree and applying Jensen's inequality, we get

$$T\pi^{\text{LZ}} + \delta(c(T), L_T^{*,LZ}) \geq \sum_{s=1}^{c(T)} \max_{\boldsymbol{y} \in \mathcal{Y}} \langle \boldsymbol{y}, \mathbf{X}_s(T) \rangle , \qquad (13)$$

where $\mathbf{X}_s(T)$ denotes the aggregate count vector of all requests made up to the time T while the LZ tree was visiting the state s. We now lower bound the RHS of the above inequality in terms of the total hits achieved by a k^{th}-order Markovian prefetcher. For any fixed $k \geq 0$, let J_1 be the set of states labeled by strings of length less than k and J_2 is the remaining set of states in the LZ tree. We can decompose the total cache hits as follows:

$$\begin{aligned} & \sum_{s=1}^{c(T)} \max_{\boldsymbol{y} \in \mathcal{Y}} \langle \boldsymbol{y}, \mathbf{X}_s(T) \rangle \\ = & \sum_{s \in J_1} \max_{\boldsymbol{y} \in \mathcal{Y}} \langle \boldsymbol{y}, \mathbf{X}_s(T) \rangle + \sum_{s \in J_2} \max_{\boldsymbol{y} \in \mathcal{Y}} \langle \boldsymbol{y}, \mathbf{X}_s(T) \rangle \\ \geq & \sum_{s \in J_2} \max_{\boldsymbol{y} \in \mathcal{Y}} \langle \boldsymbol{y}, \mathbf{X}_s(T) \rangle . \end{aligned} \qquad (14)$$

The states in J_2 form a refinement of an order-k Markovian prefetcher, since each state in J_2 is labeled by strings of length at least k [18]. Hence, the second term in (14) can be lower bounded by the number of cache hits accrued by the order-k Markovian prefetcher for the requests made on the states in J_2. Since the total number of parsed strings is $c(T)$ and only the first k requests of any parsed string are included in J_1, the total number of requests made on the bins in J_2 is at least $T - kc(T)$. Hence, the quantity (14) can be further lower bounded by $(T - kc(T))\hat{\mu}_k \geq T\hat{\mu}_k - kc(T)$, where $T\hat{\mu}_k$ is the total number of cache-hits by the order-k Markovian prefetcher over the entire input sequence. Substituting the lower bounds in Eqn. (13) we get

$$T\pi^{LZ} + \delta(c(T), L_T^*) \geq T\hat{\mu}_k - kc(T).$$

Rearranging the above, we get the final bound

$$\mathcal{R}_T = T(\hat{\mu}_k - \pi^{LZ}) \leq \delta(c(T), L_T^*) + kc(T).$$

□

C. LRU and FIFO are Finite State Prefetchers

In this section, we argue that LRU and FIFO caching policies belong to the class of the Finite State Prefetchers. To prove the claim, we need to construct FSPs that simulate the LRU and the FIFO policies, respectively.

Recall that LRU is a cache replacement policy that evicts the least-recently requested file from the cache to store the newly requested file that is not in the cache. Hence, the LRU policy caches the C most-recently requested files at all times.

LRU*:* Let each state $\sigma \equiv (\sigma_1, \ldots, \sigma_C)$ correspond to most-recently requested C distinct files ordered in the increasing order of the time-stamps of their latest requests. In other words, the file σ_C was requested most recently and the file σ_1 is the C^{th} most-recently requested file. The prefetching function for LRU is defined as

$$f^{\text{LRU}}(\sigma) = \sigma.$$

In other words, the prefetcher caches the C most-recently requested files at each state. Suppose, at the next round, the file $x \in [N]$ is requested. The state-transition function g is defined as:

$$g^{\text{LRU}}(\sigma, x) = \begin{cases} (\sigma_1, \sigma_2, \ldots \sigma_{i-1}, \sigma_{i+1} \ldots, \sigma_{C-1}, \sigma_C, x), \\ \quad \text{if } x \in \{\sigma_1, \ldots, \sigma_C\} \text{ and } x = \sigma_i, i \in N, \\ (\sigma_2, \sigma_3, \ldots, \sigma_{C-1}, \sigma_C, x), \text{ otherwise.} \end{cases}$$

It can be seen that the state transition function maintains the correct ordering of the files according to their latest requests at all times. The starting state s_0 can be selected arbitrarily. The total number of states in this construction is $|\mathcal{S}^{\text{LRU}}| = N(N-1)\ldots(N-C+1) \leq N^C$. This completes our description of LRU as an FSP.

FIFO*:* Similar to the above construction, we now show that the FIFO caching policy can also be simulated by an FSP. Recall that FIFO is a cache replacement policy which, while inserting a newly requested file that is not in the cache, evicts the oldest file from the cache.

Let the state $\sigma \equiv (\sigma_1, \ldots, \sigma_C)$ correspond to the most-recently cached C distinct files ordered in the increasing order of the time-stamps of their insertion to the cache under the FIFO policy. In other words, the file σ_1 is the oldest file in the cache and the file σ_C is the newest file in the cache. The prefetching function is simply given by

$$f^{\text{FIFO}}(\sigma) = \sigma.$$

Suppose that at time t, the file $x \in [N]$ was requested. The state transition function is given by:

$$g^{\text{FIFO}}(\sigma, x) = \begin{cases} \sigma, \text{ if } x \in \{\sigma_1, \ldots, \sigma_C\}, \\ (\sigma_2, \sigma_3, \ldots, \sigma_{C-1}, \sigma_C, x), \text{ otherwise.} \end{cases}$$

Note that if the newly requested file is already in the cache, the state of the FSP does not change. On the other hand, when a non-cached file is requested, it is placed at the end of the state and the entire tuple is shifted to the left by one place and the

file σ_1 is evicted. This is precisely the FIFO policy. The total number of states in this construction is $N(N-1)\dots(N-C+1) \leq N^C$.

VIII. Experiments

In this section, we report some simulation results demonstrating the practical efficacy of the proposed universal caching policy [4]. In our simulations, we use a synthetic file request sequence generated by a randomly constructed Finite State Machine. The structure and the number of states of the FSM remain hidden from the online policies that we evaluate. The details of the setup and the simulation results are discussed below.

Algorithm 3 Synthetic File Request Generation

Input: Number of states Q, cache size C, transition function g, initial state s_0, arrays of files $A_s, |A_s| = C, \forall s \in \mathcal{S}$, horizon length T.

Output: Generated file request sequence $\boldsymbol{x}^T = \{x_1, \cdots, x_T\}$.

1: Initialize $s \leftarrow s_0$.
2: **for** $t \leftarrow 1$ to T **do**
3: Pick a file x_t uniformly at random from the set A_s.
4: $s \leftarrow g(x_t, s)$
5: **end for**
6: **return** x^T

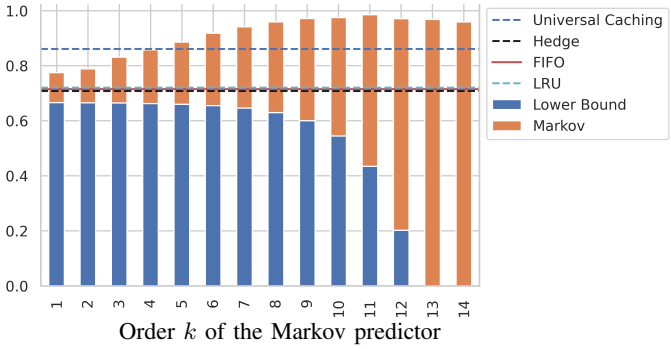
A. Synthetic Data Generation

We generate a synthetic file request sequence that can be perfectly predicted by some Finite State Predictor with zero cache misses. One such simple request generation scheme is outlined in Algorithm 3. In this scheme, we randomly construct an FSM $\mathcal{M}$ containing Q many states. For this, we initialize a random transition function $g(\cdot,\cdot)$ by generating a random $Q \times N$ matrix with entries in $\{1, \cdots, Q\}$. For each state $s \in \mathcal{S}$, we randomly sample an array A_s containing C files. We also randomly select an initial state s_0. Once constructed, we use the same FSM $\mathcal{M}$ for generating the entire file request sequence. To generate the request sequence, we start from the initial state s_0 and at each state s_t, we randomly pick a file x_t from the set A_{s_t} uniformly at random. Then we go to the next state $s_{t+1} = g(x_t, s_t)$ and repeat the process up to time T.

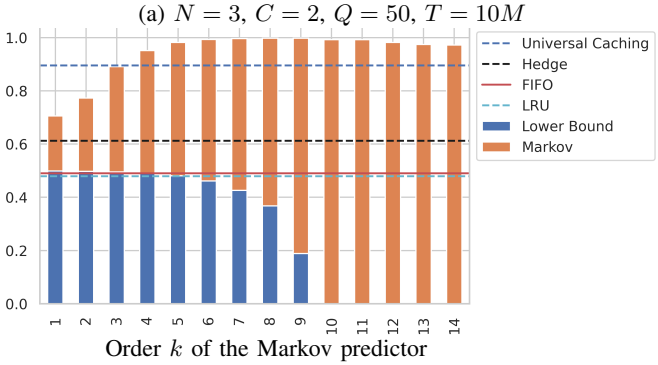
Note that the FSP characterized by the FSM $\mathcal{M}$ and the prediction function $f(s) = A_s$ predicts the generated request sequence with 100% accuracy.

B. Results and Discussion

The numerical simulation results are shown in Figure 3. The hit rate achieved by the k^{th} order FSM is shown by the saffron bars as a function of the order k. The blue bars represent the lower bound (10) on the hit rate of the k^{th} order Markov predictors. The horizontal dotted red line denotes the hit rate achieved by the SAGE policy. Finally, the horizontal dotted blue line denotes the hit rate achieved by the Universal Caching policy. From the plots, we see that both the universal caching policy and the k^{th} order Markovian FSP perform exceptionally well compared to the vanilla SAGE policy, which is competitive only against a static benchmark. Furthermore, the corresponding lower bound to the hit rate given by Eqn. (10) seems to be loose compared to the observed performance.



(a) $N = 3$, $C = 2$, $Q = 50$, $T = 10M$



(b) $N = 4$, $C = 2$, $Q = 50$, $T = 10M$

Fig. 3: Plots depicting the lower bounds (given by Eqn. (10)) and the actual hit rate achieved by different policies on the synthetic file request sequence generated a randomly constructed FSM for two different parameter settings. It appears that the bound in Eqn. (10) is quite conservative and the observed performance of the proposed universal policy far exceeds the lower bound. Furthermore, the universal caching policy and the k^{th} order FSP for a suitable value of k achieves very high hit rate compared to a vanilla SAGE policy.

[4] Code available at https://github.com/AtivJoshi/UniversalCaching