Universal Caching

Ativ Joshi and Abhishek Sinha

Tata Institute of Fundamental Research, Mumbai, India

ITW 2022

Introduction

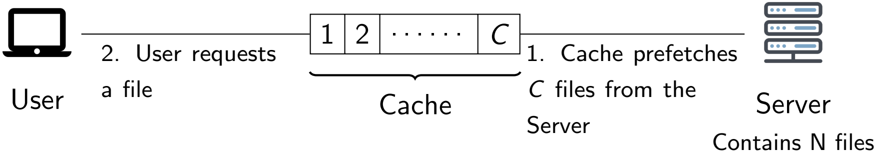


Figure: Setup of the caching problem

Introduction

- There are N files, out of which C files needs to be prefetched into the cache.
- At each timestep t, an online caching policy π pre-fetches a set of C files, denoted by the vector $\boldsymbol{y}_t \in \{0,1\}^N$, where $||\boldsymbol{y}_t||_1 = C$.
- At the same time, the user requests a file, denoted by the vector $\boldsymbol{x}_t \in \{0,1\}^N$, such that $||\boldsymbol{x}_t||_1 = 1$. The reward at round t can be expressed as $\langle \boldsymbol{x}_t, \boldsymbol{y}_t \rangle$.

Finite-State Regret

- Our objective is to design an algorithm that is competitive against a dynamic benchmark. Formally, we want to minimize the FS-Regret defined below

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

where $\mathcal{G}$ is the set of all FSPs.

FSP & FSM

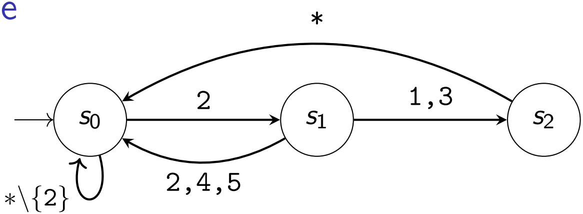
Definition 1 (Finite State Prefetcher (FSP))

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 the set of alphabets corresponding to N files
- $g : \mathcal{S} \times [N] \to \mathcal{S}$ is a state transition function
- $f : \mathcal{S} \to [N]^C$ is a possibly randomized prefetching strategy
- s_0 is the initial state

The components of an FSP without the prefetcher f form a Finite State Machine (FSM).

Example



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

Symbol (x)

States (s)	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 prediction function $f^*(s)$

Figure: Offline optimal policy for a given 3-state FSP for the 5-ary input sequence of length $T = 12$ given by $(2, 1, 5, 2, 3, 5, 2, 4, 5, 2, 3, 4)$ and $C = 2$.

Why FSPs?

- FSP can easily capture the repetitive patterns in the input requests.
- Widely deployed policies with a finite competitive ratio, such as LRU and FIFO belong to the class of Finite State Prefetchers.

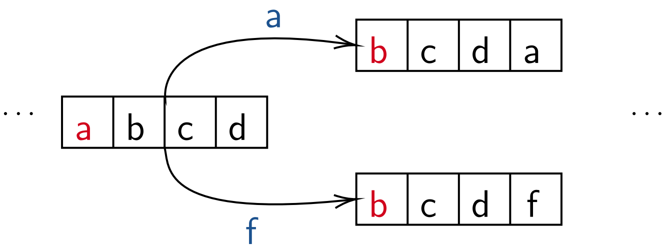


Figure: LRU as an FSP

Offline Performance Characterization

Definition 2 (k^{th}-order Markov Prefetcher)

A k^{th} order Markov Prefetcher is a special class of FSP with N^k states, where the state at round t is given by the k-tuple of the previous k file requests, *i.e.*, $s_t = (x_{t-1}, x_{t-2}, \ldots, x_{t-k})$.

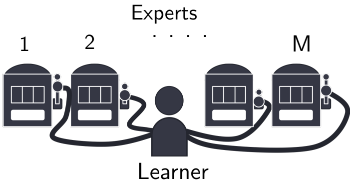
Let $\tilde{\pi}_S(x_1^T)$ and $\tilde{\mu}_k(x_1^T)$ denote the offline fractional hitrates of an S-state FSP and order-k Markov Prefetcher for a given sequence x_1^T.

Theorem 1 (MP vs FSP)

The hit rate of a Markov prefetcher of a sufficiently large order exceeds the hit rate of any FSP (up to a vanishingly small term). In particular, for any file request sequence x^T, we have:

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

Online Caching Policy for a Single State: HEDGE



- At each round t, experts make prediction. The learner chooses an expert k with probability $p_{t,k}$. The adversary gives a reward $r_{t,i}$ to every expert. Expected reward of the learner is $\langle \boldsymbol{p}_t, \boldsymbol{r}_t \rangle$.
- Hedge samples an expert i with probability $p_{t,i} \propto \exp(\eta \sum_{\tau=1}^{t-1} r_{t,i})$.
- A naive approach would be to run HEDGE on $M = \binom{N}{C}$ meta-experts. Obviously, this is computationally intractable.

The SAGE Framework (Mukhopadhyay et al. [1])

- The SAGE algorithm gives an efficient implementation of the HEDGE policy using randomized sampling and exploiting the linearity of the reward function.
- In the online caching problem, the reward depends only on the *marginal* inclusion probabilities of each file. Formally, SAGE works as follows:
  - Efficiently computes the marginal file inclusion probabilities induced by HEDGE.
  - Efficiently sample a subset of C files without replacement consistent with these marginals.

Sage: Computing the Marginals

- The marginal inclusion probability for the i^{th} file is given by:

$$p_t(i) = \frac{w_{t-1}(i) \sum_{S \subseteq [N] \setminus \{i\} : |S| = C-1} w_{t-1}(S)}{\sum_{S' \subseteq [N] : |S'| = C} w_{t-1}(S')}, \tag{3}$$

  where $w_t(S) = \Pi_{i \in S} w_t(i)$, $w_t(i) \equiv \exp(\eta R_t(i))$ and $R_t(i)$ is the total number of times file i was requested up to time t.

- Both the numerator and denominator can be expressed in terms of certain *elementary symmetric polynomials* (ESP), which can be efficiently evaluated in $\tilde{O}(N)$ time using FFT-based polynomial multiplication methods.

SAGE: Madow's Sampling

- We want to sample C out of N files such that each file is sampled with probability p_i. Given that $\sum_{i=1}^{N} p_i = C$, the files can be sampled using the following procedure :

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

SAGE: Madow's Sampling

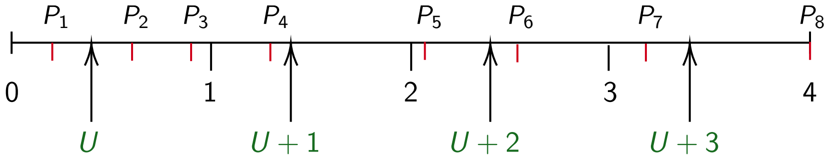


Figure: Example of Madow's Sampling. Out of 8 items, the 4 which will be selected are $\{2, 5, 6, 8\}$.

SAGE: Regret Bound

- The SAGE algorithm gives a "small-loss" bound on the static regret:

$$T(\tilde{\pi}_1 - \pi^{\text{HEDGE}}) \leq \sqrt{2Cl_T^* \ln(Ne/C)} + C \ln(Ne/C), \qquad (4)$$

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.

$\textsc{Sage}$ with FSM

- Consider any given S-state FSM. Let $\boldsymbol{x}_s$ be the sequence of file requests corresponding to the state s.
- Upon running a separate copy of the $\textsc{Sage}$ policy for each state of the given FSM with the request sequence $\boldsymbol{x}_s, s \in \mathcal{S}$, we obtain the following regret bound:

$$T(\tilde{\pi}_S(x^T) - \pi_S^{\textsc{Sage}}(x^T)) \leq \sqrt{2CSL^*_{T,S} \ln(Ne/C)} + CS \ln(Ne/C) \tag{5}$$

where $L^*_{T,S} \equiv \sum_{s=1}^{S} l^*_{T,s}$.

Theorem 2

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

$$T(\tilde{\pi}_S(x^T) - \pi_k^{\textsc{Sage}}(x^T)) \leq T \min(1 - C/N, \sqrt{A}) + \sqrt{2BL^*_{T,k}} + B$$

where $A = \frac{\ln S}{2(k+1)}$ and $B = N^k C \ln \frac{Ne}{C}$

Example

- Furthermore, for a request sequence $\boldsymbol{x}_Q^T$ generated by any FSM containing at most Q states, the expected number of cache misses conceded by the SAGE policy with a k^{th} order Markovian FSM can be upper bounded by

$$\leq A + \sqrt{2AB} + B$$

where $A = \left(1 - \frac{C}{N}, \sqrt{\frac{\ln Q}{2(k+1)}}\right)$ and $B = \frac{N^k C}{T} \ln \frac{Ne}{C}$.

Universal Caching Policy

- In Theorem 2, we are free to choose the order k of the Markovian prefetcher.
- Since the number of states S in the benchmark comparator could be arbitrarily large, to get asymptotically zero regret, we need to increase the order of the Markovian FSM with time.
- For this, we use an N-ary version of the LZ parsing tree and run SAGE on each of it's node.

Lempel-Ziv Tree

- The LZ parsing algorithm parses the N-ary request sequence into distinct phrases such that each phrase is the shortest phrase that is not previously parsed.
- 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 leaf according to the consecutive file requests.
    - Once a leaf node is reached, the tree is extended by making the leaf an internal node by adding N offsprings to the tree. Then we move back to the root of the tree.

Example

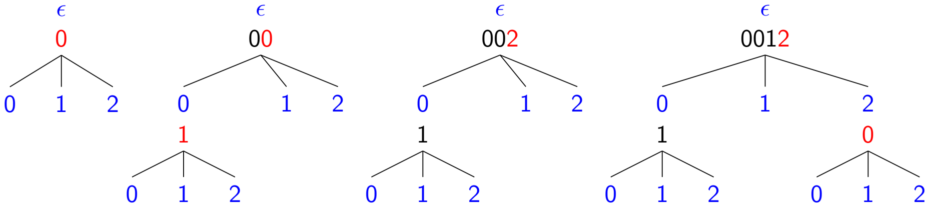


Figure: Evolution of LZ tree for $N = 3$ and input page request sequence 001220. The parsed phrases are $\{0, 01, 2, 20\}$. Each instance denotes the tree after parsing a phrase. The states are shown in blue. The requests at each state is shown in black, and the latest parsed phrase is shown in red.

Properties of LZ Tree

- The number of nodes in an N-ary LZ tree grows sub-linearly with T as $c(T) = O(\frac{T \log N}{\log T})$.
- 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.

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:

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

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

References

[1] Samrat Mukhopadhyay, Sourav Sahoo, and Abhishek Sinha. k-experts–online policies and fundamental limits. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2022.

[2] Meir Feder, Neri Merhav, and Michael Gutman. Universal prediction of individual sequences. *IEEE transactions on Information Theory*, 38(4):1258–1270, 1992.

[3] Samrat Mukhopadhyay, Sourav Sahoo, and Abhishek Sinha. k-experts - online policies and fundamental limits. *CoRR*, abs/2110.07881, 2021. URL https://arxiv.org/abs/2110.07881.

[4] Rajarshi Bhattacharjee, Subhankar Banerjee, and Abhishek Sinha. Fundamental limits on the regret of online network-caching. *Proc. ACM Meas. Anal. Comput. Syst.*, 4(2), June 2020. doi:10.1145/3392143. URL https://doi.org/10.1145/3392143.