`LeadCache`: Regret-Optimal Caching in Networks

Debjit Paria
Department of Computer Science
Chennai Mathematical Institute
Chennai 603103, India
debjit.paria1999@gmail.com

Abhishek Sinha
Department of Electrical Engineering
Indian Institute of Technology Madras
Chennai 600036, India
abhishek.sinha@ee.iitm.ac.in

Abstract

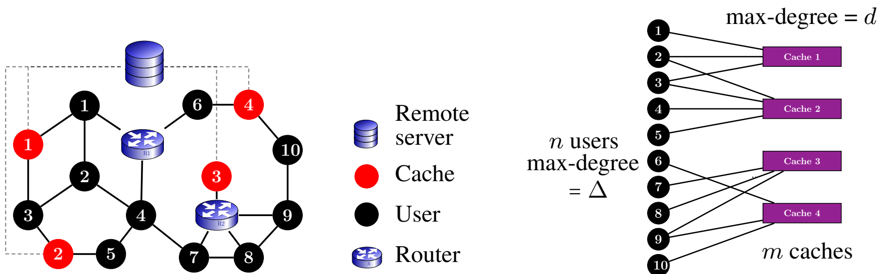
We consider a set-valued online prediction problem in the context of network caching. Assume that users are connected to a number of caches via a bipartite network. At any time slot, each user requests some file chosen from a large catalog. A user's request is met if the requested file is cached in at least one of the caches connected to the user. The objective is to predict and optimally store the files on the caches to maximize the total number of cache-hits. We propose `LeadCache` - an online caching policy based on the Follow-the-Perturbed-Leader paradigm. We show that the policy is regret-optimal up to a factor of $\tilde{O}(n^{3/8})$, where n is the number of users. We implement the policy by designing a new linear-time Pipage rounding algorithm. With an additional Strong-Law-type assumption, we show that the total number of file fetches under `LeadCache` remains almost surely finite. Additionally, we derive a tight regret lower bound using results from graph coloring. Our conclusion is that the proposed learning-based caching policy decisively outperforms the classical policies both theoretically and empirically.

1 Introduction

We consider an online structured learning problem, called Bipartite Caching, which lies at the heart of many large-scale internet applications, including Content Distribution Networks (CDN) and Cloud Computing. Formally, a set $\mathcal{I}$ of n users is connected to a set $\mathcal{J}$ of m caches via a bipartite network $G(\mathcal{I} \cup \mathcal{J}, E)$. Each cache is connected to at most d users, and each user is connected to at most Δ caches (see Figure 1 (b)). There is a catalog consisting of N unique files, and each of the m caches can host at most C files at a time where $C < N$. The system evolves on a slotted timescale. Each of the n users may request any file from the catalog at each time slot. The file request sequence could be dictated by an adversary. Given the storage capacity constraints, an online caching policy determines which files are to be cached on which caches at each slot before the file requests for that slot arrive. The objective is to maximize the total number of *hits* by the unknown incoming requests by coordinating the caching decisions among multiple caches in an online fashion. Technically, the Bipartite Caching problem is a strict generalization of the online $\boldsymbol{k}$**-sets** problem, which has been studied extensively in the learning literature [Koolen et al., 2010, Cohen and Hazan, 2015]. However, unlike the $\boldsymbol{k}$**-sets** problem, which predicts a single subset at a time, in this problem, we are interested in sequentially predicting multiple subsets, each corresponding to one of the caches. The interaction among the caches through the non-linear reward function makes this problem challenging.

The Bipartite Caching problem can be thought of as a simplified abstraction of the more general Network Caching problem, which is central to the commercial CDNs, such as Akamai [Nygren et al., 2010], Amazon Web Services (AWS), and Microsoft Azure [Paschos et al., 2020]. Here one is provided with an arbitrary graph $\mathcal{G}(V, E)$, a set of users $\mathcal{I} \subseteq V$, and a set of caches $\mathcal{J} \subseteq V$. A user can retrieve a file from a cache only if the cache hosts the requested file. If the i^{th} user retrieves the requested file from the j^{th} cache, the user receives a reward of $r_{ij} \geq 0$ for that slot. If the requested

Preprint. Under review.



(a) (b)

Figure 1: Reduction of the Network Caching problem (a) to the Bipartite Caching problem (b). In this schematic, we assumed that a cache, located within two hops, is reachable to a user.

file is not hosted in any of the caches reachable to the user, the user receives zero rewards for that slot. The goal of a network caching policy is to dynamically place files on the caches so that cumulative rewards obtained by all users is maximized. Clearly, the Network Caching problem reduces to the Bipartite Caching problem when the rewards are restricted to the set $\{0,1\}$. It turns out that the algorithms presented in this paper can be extended to the general Network Caching problem as well.

1.1 Problem Formulation

Let us denote the file requested by the i^{th} user by the one-hot encoded N-dimensional vector $\boldsymbol{x}_t^i$. In other words, $\boldsymbol{x}_{tf}^i = 1$ if the i^{th} user requests the file $f \in [N]$ at slot t, and is zero otherwise. Since a user may request at most one file per slot, we have: $\sum_{f=1}^N \boldsymbol{x}_{tf}^i \leq 1, \quad \forall i \in \mathcal{I}, \forall t$. An online caching policy decides the file placements on the caches at slot t based on the information available so far. Unlike the classical caching algorithms, such as LRU, LFU, FIFO, Marker, which fetches a file immediately upon a cache-miss, we do not enforce this constraint, as this leads to linear regret [Bhattacharjee et al., 2020]. The set of files placed on the j^{th} cache at time t is represented by the N-dimensional incidence vector $\boldsymbol{y}_t^j \in \{0,1\}^N$. In other words, $\boldsymbol{y}_{tf}^j = 1$ if the j^{th} cache hosts the file $f \in [N]$ at time t, and is zero otherwise. Due to the limited cache capacity, the following constraint must be satisfied at each time slot t: $\sum_{f=1}^N \boldsymbol{y}_{tf}^j \leq C, \quad \forall j \in \mathcal{J}$.

The set of all admissible caching configurations, denoted by $\mathcal{Y} \subseteq \{0,1\}^{Nm}$, is dictated by the capacity constraint. Hence, in principle, the policy is allowed to replace all elements of the caches at every slot, incurring a potentially huge fetching/downloading cost over an interval. However, in Section 3, we show that the rate of the file fetchings to the caches under the proposed `LeadCache` policy goes to zero almost surely under very mild assumptions on the request sequence.

The i^{th} user gets a *cache-hit* at slot t if and only if *anyone* of the caches connected to the i^{th} user hosts the requested file at slot t. In the case of a cache-hit, the user obtains a unit reward. On the other hand, in the case of a *cache-miss*, the user does not obtain any reward for that slot. Hence, for a given request vector $\boldsymbol{x}_t = \left(\boldsymbol{x}_t^i, i \in \mathcal{I}\right)$ from all users and the cache configuration vector $\boldsymbol{y}_t = \left(\boldsymbol{y}_t^j, j \in \mathcal{J}\right)$ of all caches, the total reward $q(\boldsymbol{x}_t, \boldsymbol{y}_t)$ obtained by the users at time t is

$$q(\boldsymbol{x}_t, \boldsymbol{y}_t) \equiv \sum_{i \in \mathcal{I}} \boldsymbol{x}_t^i \cdot \min\left\{\mathbf{1}_{N\times 1}, \left(\sum_{j \in \partial^+(i)} \boldsymbol{y}_t^j\right)\right\}, \tag{1}$$

where $\boldsymbol{a} \cdot \boldsymbol{b}$ denotes the inner-product of the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, $\mathbf{1}_{N\times 1}$ denotes the N-dimensional all-one column vector, the set $\partial^+(i)$ denotes the set of all caches connected to the i^{th} user, and the "min" operator is applied component wise. The total reward $Q(T)$ accrued in a time-horizon of length T is obtained by summing-up the slot-wise rewards, *i.e.,* $Q(T) = \sum_{t=1}^T q(\boldsymbol{x}_t, \boldsymbol{y}_t)$. Following the standard practice in the online learning literature, we measure the performance of any online policy π using the notion of static regret $R^\pi(T)$, defined as the maximum difference in the cumulative rewards obtained by the optimal fixed caching-configuration in the hindsight and that of the online policy π, *i.e.,*

$$R^\pi(T) \stackrel{\text{(def.)}}{=} \sup_{\{\boldsymbol{x}_t\}_{t=1}^T} \left(\sum_{t=1}^T q(\boldsymbol{x}_t, \boldsymbol{y}^*) - \sum_{t=1}^T q(\boldsymbol{x}_t, \boldsymbol{y}_t^\pi)\right), \tag{2}$$

where $\boldsymbol{y}^*$ is the best static cache-configuration in the *hindsight* for the file request sequence $\{\boldsymbol{x}_t\}_{t=1}^T$, *i.e.,* $\boldsymbol{y}^* = \arg\max_{\boldsymbol{y}\in\mathcal{Y}} \sum_{t=1}^T q(\boldsymbol{x}_t, \boldsymbol{y})$. We assume that the file request sequence is generated by an *oblivious adversary*, *i.e.,* the entire request sequence $\{\boldsymbol{x}_t\}_{t\geq 1}$ is assumed to be fixed a priori. This is a mild assumption, because it is known from Cesa-Bianchi and Lugosi [2006] and Appendix B. of Suggala and Netrapalli [2020] that any algorithm that works for oblivious adversary also works for non-oblivious adversary, whose current actions may depend on past predictions of the algorithm. Note that the problem is *non-convex*, as we seek for integral cache allocations.

2 Background and Related Works

A canonical online learning model is Online Linear Optimization (OLO), which can be thought of as a repeated game played between a policy (also known as the forecaster) and an adversary [Cesa-Bianchi and Lugosi, 2006]. In this model, At every time slot t, the policy chooses an action vector $\boldsymbol{y}_t$ from a feasible action set $\mathcal{Y} \subseteq \mathbb{R}^d$. After that, the adversary reveals a reward vector $\boldsymbol{x}_t$ from a set $\mathcal{X} \subseteq \mathbb{R}^d$. The adversary is assumed to be *oblivious* in the sense that the reward vectors are determined before the game starts. At slot t, the policy receives a scalar reward given by the inner-product $q(\boldsymbol{x}_t, \boldsymbol{y}_t) \coloneqq \langle \boldsymbol{x}_t, \boldsymbol{y}_t \rangle$. A classic objective is to design a policy with minimal regret (viz. Eqn. (2)). Follow the Perturbed Leader (FTPL), is a well-known online policy for the OLO problem [Hannan, 1957]. At time slot t, the policy adds a random noise vector $\boldsymbol{\gamma}_t$ to the cumulative reward vector $\boldsymbol{X}_t = \sum_{\tau=1}^{t-1} \boldsymbol{x}_t$, and selects the action $\boldsymbol{y}_t = \arg\max_{\boldsymbol{y}\in\mathcal{Y}} \langle \boldsymbol{X}_t + \boldsymbol{\gamma}_t, \boldsymbol{y} \rangle$. The regret bound of the FTPL policy depends on the structure of the problem (*i.e.,* the sets $\mathcal{X}$ and $\mathcal{Y}$) and the distribution of the additive noise vector $\boldsymbol{\gamma}_t$ (*e.g.,* Gumbel [Cesa-Bianchi and Lugosi, 2006], Exponential [Kalai and Vempala, 2005], Gaussian [Cohen and Hazan, 2015, Devroye et al., 2015]). See Abernethy et al. [2016] for a unifying treatment of the FTPL policies through the lens of stochastic smoothing. Moreover, in addition to static regret, stronger forms of regret, such as strongly adaptive regret, have also been investigated in the literature. There now exist efficient Black-Box reductions that transform any standard low-regret algorithm to a strongly adaptive one [Daniely et al., 2015, Jun et al., 2017]. Hence, in this paper, we only consider the standard regret minimization problem.

A large number of papers on caching proceeds by assuming some stochastic model for the file request sequence, *e.g.,* Independent Reference Model (IRM) and Shot Noise Model (SNM) Traverso et al. [2013]. For adversarial request sequences, classic algorithms such as MIN, LRU, LFU, FIFO aim to minimize the competitive ratio [Van Roy, 2007, Lee et al., 1999, Dan and Towsley, 1990]. However, since the competitive ratio metric is multiplicative in nature, there can be a large performance gap in the hit-ratio of a competitively optimal policy and the optimal offline policy. To provide a robust *additive* performance guarantee, the caching problem has been recently investigated through the lens of regret minimization. Recall that, in the online $\boldsymbol{k}$**-sets** problem, which arises in the context of online ad display and personalized news recommendation, the goal of the learner is to predict a subset of cardinality k to maximize the hits [Cohen and Hazan, 2015]. Clearly, the single cache problem is identical to the $\boldsymbol{k}$**-sets** problem. Daniely and Mansour [2019] considered the problem of minimizing the regret plus the switching cost for a single cache. The authors proposed a variant of the celebrated exponential weight algorithm [Littlestone and Warmuth, 1994, Freund and Schapire, 1997] that ensures the minimum competitive ratio and a small but sub-optimal regret. Moreover, in the adversarial setting, the switching cost of their proposed policy grows as the square-root the length of the time-horizon. The Bipartite Caching model was first proposed in a pioneering paper by Shanmugam et al. [2013], where they considered a stochastic version of the problem with known popularity distribution of the files. Paschos et al. [2019] proposed an Online Gradient Ascent (OGA)-based Bipartite Caching policy which allows caching a fraction of the Maximum Distance Separable (MDS)-coded files. Closely related to this paper is the recent work by Bhattacharjee et al. [2020], where the authors designed a regret-optimal single-cache policy and a Bipartite Caching policy for fountain-coded files. However, the fundamental problem of designing a regret-optimal uncoded caching policy for the Bipartite Caching problem was left open due to its difficulty.

Why standard approaches fail: A straightforward way to formulate the Bipartite Caching problem is to pose it as an instance of the `Experts` problem [Cesa-Bianchi and Lugosi, 2006] with each of the possible $\binom{N}{k}^m$ cache configurations as experts. However, this approach is computationally infeasible due to the huge number of experts. Another possible way to learn a non-linear reward function is to linearize it via the first-order Taylor's series approximation [Orabona, 2019]. However,

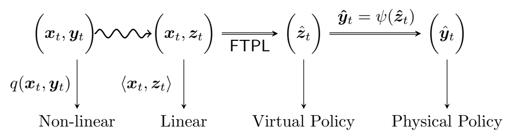


Figure 2: Reduction pipeline illustrating the translation of the Bipartite Caching problem with a non-linear reward function to an online learning problem with a linear reward function.

the non-differentiability of the reward function (1) and the integrality constraint of the allocation vectors precludes this approach in this problem. In view of the above challenges, we present our main technical contributions below.

3 Main Results

1. FTPL for a non-linear non-convex problem: We propose `LeadCache`, a network caching policy based on the Follow the Perturbed Leader paradigm. The *non-linearity* of the reward function and the non-convexity of the feasible set (due to the integrality of cache allocations) pose a major challenge in using the generic FTPL framework [Abernethy et al., 2016]. To circumvent this difficulty, we switch to a *virtual action* domain $\mathcal{Z}$ where the reward function is linear. We use an anytime version of the FTPL policy for designing a virtual policy for the linearized virtual learning problem. Finally, we translate the virtual policy back to the original action domain $\mathcal{Y}$ with the help of a mapping $\psi : \mathcal{Z} \to \mathcal{Y}$, which is obtained by solving a combinatorial optimization problem. The overall reduction pipeline is summarized in Figure 2. Our technique is general, leads to a *dimension-free* regret bound, and may be used in other problems with a similar structure.

2. New Rounding Technique: The problem P that translates the virtual actions to the physical caching actions in the above scheme turns out to be an NP-hard Integer Linear Program. As our second independent contribution, we design a linear-time Pipage rounding technique for this problem Ageev and Sviridenko [2004]. Incidentally, our rounding technique substantially improves upon a computationally complex rounding algorithm proposed by Shanmugam et al. [2013] in the context of caching with i.i.d. requests. It is worth mentioning that improving upon the rounding algorithm by Shanmugam et al. [2013] has been recognized as one of the most outstanding challenges in large-scale network caching Paschos et al. [2018]. Our rounding algorithm runs in linear time and yields the same approximation guarantee as that of Shanmugam et al. [2013].

3. Bounding the Switching Cost: As our third contribution, we show that if the file requests are generated by a stochastic process satisfying an additional mild Strong Law-type property then the caching configuration under the `LeadCache` policy *almost surely* converges to the corresponding optimal configuration *in finite time*. As new file fetches to the cache from the remote server consumes bandwidth, this result implies that the proposed policy offers the best of both worlds - (1) sub-linear regret for adversarial requests, and (2) zero rate download for "stochastically regular" requests.

4. New Regret Lower Bound: As our final contribution, we derive a tight minimax regret lower bound for the Bipartite Caching problem. The proof of the new lower bound critically uses a fundamental result in graph coloring, known as Brook's theorem (Diestel [2005]). Our lower bound improves upon a recent result by Bhattacharjee et al. [2020] and establishes a $\tilde{O}(n^{3/8})$ approximation guarantee for the regret of the `LeadCache` policy.

4 The `LeadCache` Policy

In this Section, we describe `LeadCache`- an efficient online network caching policy guaranteeing near-optimal regret. Since the reward function (1) is non-linear in the action variable $\boldsymbol{y}$, to use the FTPL policy, we *linearize* the problem by switching to a *virtual* action domain $\mathcal{Z}$ as detailed below.

The Virtual Caching Problem: Consider an associated Online Linear Optimization (OLO) problem, called Virtual Caching. In this problem, at each slot t, a virtual action $\boldsymbol{z}_t \equiv \big(\boldsymbol{z}_t^i, i \in \mathcal{I}\big)$ is taken in response to the file requests $\{\boldsymbol{x}_\tau\}_1^{t-1}$ received so far. The i^{th} component of the virtual action, denoted by $\boldsymbol{z}_t^i \in \{0,1\}^N$, roughly indicates the availability of the files in the caches connected to the i^{th} user. The set of all admissible virtual actions, denoted by $\mathcal{Z} \subseteq \{0,1\}^{N\times n}$, is given below in Eqn. (4). The slot-wise reward $r(\boldsymbol{x}_t, \boldsymbol{z}_t)$ accrued by the virtual action $\boldsymbol{z}_t$ for the file request vector $\boldsymbol{x}_t$ at the t^{th} slot is given by their inner product, *i.e.,*

$$r(\boldsymbol{x}_t, \boldsymbol{z}_t) := \langle \boldsymbol{x}_t, \boldsymbol{z}_t \rangle = \sum_{i\in\mathcal{I}} \boldsymbol{x}_t^i \cdot \boldsymbol{z}_t^i. \tag{3}$$

Virtual Actions: The set $\mathcal{Z}$ of all admissible virtual actions is defined as the set of all binary vectors $\boldsymbol{z} \in \{0,1\}^{N\times n}$ such that the following condition holds componentwise for some admissible physical cache configuration vector $\boldsymbol{y} \in \mathcal{Y}$:

$$\boldsymbol{z}^i \leq \min\Big\{\mathbf{1}_{N\times 1}, \Big(\sum_{j\in\partial^+(i)} \boldsymbol{y}^j\Big)\Big\}, \quad 1 \leq i \leq n. \tag{4}$$

In other words, the set $\mathcal{Z}$ be can be explicitly characterized as the set of all binary vectors $\boldsymbol{z} \in \{0,1\}^{N\times n}$ satisfying the following constraints for some feasible $\boldsymbol{y}$:

$$z_f^i \leq \sum_{j\in\partial^+(i)} y_f^j, \quad \forall i \in \mathcal{I}, f \in [N] \tag{5}$$

$$\sum_{f=1}^{N} y_f^j \leq C, \quad \forall j \in \mathcal{J}, \tag{6}$$

$$y_f^j, z_f^i \in \{0,1\}, \quad \forall i \in \mathcal{I}, \forall j \in \mathcal{J}, f \in [N]. \tag{7}$$

Let $\psi : \mathcal{Z} \to \mathcal{Y}$ be a mapping that maps any admissible virtual action $\boldsymbol{z} \in \mathcal{Z}$ to a corresponding physical caching action $\boldsymbol{y}$ satisfying the condition (4). Hence, the binary variable $z_{tf}^i = 1$ *only if* the file f is hosted in one of the caches connected to the i^{th} user at time t in the physical configuration $\boldsymbol{y}_t = \psi(\boldsymbol{z}_t)$. The mapping ψ may be used to map any virtual caching policy $\pi^{\text{virtual}} = \{\boldsymbol{z}_t\}_{t\geq 1}$, to a physical caching policy $\pi^{\text{phy}} \equiv \psi(\pi^{\text{virtual}}) = \{\boldsymbol{y}_t\}_{t\geq 1}$ through the correspondence $\boldsymbol{y}_t = \psi(\boldsymbol{z}_t), \forall t \geq 1$. The following lemma relates the regret of these two online policies:

Lemma 1. *For any virtual caching policy $\pi^{virtual}$, define a physical caching policy $\pi^{phy} = \psi(\pi^{virtual})$. Then the regret of π^{phy} is bounded above by that of the policy $\pi^{virtual}$,* i.e., $R_T^{\pi^{phy}} \leq R_T^{\pi^{virtual}}, \forall T.$

See Section 9.1 in the supplementary material for the proof of the above lemma. Lemma 1 implies that a low-regret virtual caching policy may be used to obtain a low-regret physical caching policy using the mapping $\psi(\cdot)$. The key advantage of the virtual caching problem is that it is a standard OLO problem. In our proposed `LeadCache` policy, we use an anytime version of the FTPL policy for solving virtual caching problem. The overall `LeadCache` policy is described below:

Algorithm 1 The `LeadCache` Policy

1: $\boldsymbol{X}(0) \leftarrow \mathbf{0}$
2: Sample $\boldsymbol{\gamma} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \mathbb{1}_{Nn\times 1})$
3: **for** $t = 1$ to T **do**
4: $\quad \boldsymbol{X}(t) \leftarrow \boldsymbol{X}(t-1) + \boldsymbol{x}_t$
5: $\quad \eta_t \leftarrow \frac{n^{3/4}}{(2d(\log\frac{N}{C}+1))^{1/4}}\sqrt{\frac{t}{Cm}}$
6: $\quad \boldsymbol{\Theta}(t) \leftarrow \boldsymbol{X}(t) + \eta_t \boldsymbol{\gamma}$
7: $\quad \boldsymbol{z}_t \leftarrow \max_{\boldsymbol{z}\in\mathcal{Z}} \langle \boldsymbol{\Theta}(t), \boldsymbol{z} \rangle$.
8: $\quad \boldsymbol{y}_t \leftarrow \psi(\boldsymbol{z}_t)$.
9: **end for**

In Algorithm 1, the flattened $Nn \times 1$ dimensional vector $\boldsymbol{X}(t)$ denotes the cumulative count of the file requests (for each (user, file) tuple), and the vector $\boldsymbol{\Theta}(t) = \big(\boldsymbol{\theta}^i(t), 1 \leq i \leq n\big)$, denotes the perturbed cumulative file request counts obtained upon adding a scaled i.i.d. Gaussian noise vector to $\boldsymbol{X}(t)$. Note that, in step 6, we can also sample a fresh Gaussian vector $\boldsymbol{\gamma}_t$ at every time, leading to a high probability regret bound [Devroye et al., 2015]. The following result gives an upper bound on regret:

Theorem 1. *The expected regret for the* `LeadCache` *policy is upper bounded by:*

$$\mathbb{E}\big(R_T^{\texttt{LeadCache}}\big) \leq \kappa n^{3/4} d^{1/4} \sqrt{mCT},$$

where $\kappa = O\big(\textsf{poly-log}(N/C)\big)$*, and the expectation is taken over the randomness of the policy.*

Note that, in sharp contrast with the generic bound of Suggala and Netrapalli [2020], our regret-bound is almost dimension-free, as it depends only logarithmically with the ambient dimension N.

Proof outline: Our analysis of the `LeadCache` policy exploits the elegant stochastic smoothing framework developed in Abernethy et al. [2016, 2014], Cohen and Hazan [2015], Lee [2018]. However, the generic regret bound given by Theorem 1.7 of Abernethy et al. [2016] is quite loose as it depends polynomially on the dimension N of the problem. Furthermore, as noted in Bhattacharjee et al. [2020], the regret bounds in Theorem 1 and Lemma 3 of Cohen and Hazan [2015] are also loose by a factor of $\Theta(\sqrt{C})$. In this paper, we derive a tight regret bound by carefully analyzing the Hessian of the stochastically smoothed potential function and exploiting the structure of the problem. See Section 9.2 of the supplementary material for the complete proof of Theorem 1.

4.1 Fast approximate implementation

The only computationally intensive steps in Algorithm 1 are (1) solving the optimization problem in step 7 to determine the virtual caching action, and (2) translating the virtual action back to the physical caching action in step (8). Since the perturbed vector $\boldsymbol{\Theta}(t)$ is obtained by adding white Gaussian noise (with unbounded support in both positive and negative real axes) to the cumulative request vector $\boldsymbol{X}(t)$, some of its components could be negative. In order to maximize the objective (7), it is clear that if the coefficient $\theta_f^i(t)$ is negative for some (i, f) tuple, it is feasible and optimal to set the corresponding virtual action variable z_f^i to zero. Hence, steps (7) and (8) may be combined as:

$$\boldsymbol{y}(t) \leftarrow \arg\max_{\boldsymbol{y}\in\mathcal{Y}} \underbrace{\sum_{i\in\mathcal{I}, f\in[N]} (\theta_f^i(t))^+ \Bigg(\min\Big(1, \sum_{j\in\partial^+(i)} y_f^j\Big)\Bigg)}_{L(\boldsymbol{y})}, \tag{8}$$

where $x^+ \equiv \max(0, x)$. Incidentally, we find that the problem (8) is mathematically identical to the uncoded *Femtocaching* problem with known file request probabilities studied in Section III of Shanmugam et al. [2013]. In Lemma 1 of Shanmugam et al. [2013], the authors showed that the problem (8) is NP-hard. They also described a complex rounding method, which involves repeatedly computing matchings in a graph. In the following, we propose an efficient *linear time* rounding technique that rounds a relaxation of (7) and enjoys the same performance guarantee.

LP Relaxation: First, we introduce a new set of variables $z_f^i := \min(1, \sum_{j\in\partial^+(i)} y_f^j), \forall i, f$, and relax the integrality constraints to arrive at the following LP:

$$\max \sum_{i,f} (\theta_f^i(t))^+ z_f^i \tag{9}$$

Subject to,

$$z_f^i \leq \sum_{j\in\partial^+(i)} y_f^j, \;\; \forall i \in \mathcal{I}, f \in [N] \tag{10}$$

$$\sum_{f=1}^N y_f^j \leq C, \;\; \forall j \in \mathcal{J}, \tag{11}$$

$$0 \leq y_f^j \leq 1, \;\; \forall j \in \mathcal{J}, f \in [N] \tag{12}$$

$$0 \leq z_f^i \leq 1, \;\; \forall i \in \mathcal{I}, f \in [N]. \tag{13}$$

Denote the objective function of the problem (8) by $L(\boldsymbol{y})$ and its optimal value (over integers) by $\textsf{OPT}$. Let $\boldsymbol{y}^*$ be an optimal solution to the relaxed LP (9). Since the LP is a relaxation to (8), it automatically holds that $L(\boldsymbol{y}^*) \geq \textsf{OPT}$. To round the cache allocation vector $\boldsymbol{y}^*$ to an integral one, we use the general framework of *Pipage rounding* introduced by Ageev and Sviridenko [2004].

Pipage Rounding: Our rounding technique is markedly simpler compared to Algorithm 1 of Shanmugam et al. [2013]. While we round one fractional allocation of a single cache at a time, the rounding procedure of Shanmugam et al. [2013] jointly rounds several allocations in multiple caches at the same time by computing matchings in a bipartite graph. Our simplification stems from the fact that Shanmugam et al. [2013] directly use the Pipage rounding procedure designed for a different template problem, which includes additional spurious constraints on the maximum number of copies a file may have on different caches at any point in time. Shanmugam et al. [2013] relax these additional constraints by setting constraint violation budgets sufficiently high. However, due to the spurious constraints, the resulting rounding procedure becomes unnecessarily complicated. Moreover, despite its high complexity, their rounding procedure is not general, as it requires the coefficients $\boldsymbol{\theta}$ of the objective function (8) to have a product form structure, which does not necessarily hold. In Algorithm 2, we design a linear-time rounding technique without making any assumption on the coefficients.

Algorithm 2 Cache-wise Pipage rounding

1: $\boldsymbol{y} \leftarrow$ Solution of the LP (9).
2: **while** $\boldsymbol{y}$ is not integral **do**
3: Select a cache j with two fractional variables $y^j_{f_1}$ and $y^j_{f_2}$.
4: Set $\epsilon_1 \leftarrow \min(y^j_{f_1}, 1-y^j_{f_2}), \epsilon_2 \leftarrow \min(1-y^j_{f_1}, y^j_{f_2})$.
5: Define two new feasible cache-allocation vectors $\boldsymbol{\alpha}, \boldsymbol{\beta}$ as follows:

$$\alpha^j_{f_1} \leftarrow y^j_{f_1} - \epsilon_1, \alpha^j_{f_2} \leftarrow y^j_{f_2} + \epsilon_1, \quad \text{and} \quad \alpha^k_f \leftarrow y^k_f, \text{ otherwise}, \tag{14}$$

$$\beta^j_{f_1} \leftarrow y^j_{f_1} + \epsilon_2, \beta^j_{f_2} \leftarrow y^j_{f_2} - \epsilon_2, \quad \text{and} \quad \beta^k_f \leftarrow y^k_f, \text{ otherwise}. \tag{15}$$

6: Set $\boldsymbol{y} \leftarrow \arg\max_{\boldsymbol{x} \in \{\boldsymbol{\alpha}, \boldsymbol{\beta}\}} \phi(\boldsymbol{x})$.
7: **end while**
8: **return** $\boldsymbol{y}$.

Design: The key to our deterministic rounding procedure is to consider the following surrogate objective function $\phi(\boldsymbol{y})$ instead of the original objective $L(\boldsymbol{y})$ in Eqn. (8):

$$\phi(\boldsymbol{y}) \equiv \sum_{i,f} (\theta^i_f(t))^+ \big(1 - \prod_{j \in \partial^+(i)} (1-y^j_f)\big). \tag{16}$$

Following a standard algebraic argument (see eqn. (16) of Ageev and Sviridenko [2004]), we have:

$$L(\boldsymbol{y}) \overset{(a)}{\geq} \phi(\boldsymbol{y}) \geq \bigg(1 - (1-\frac{1}{\Delta})^{\Delta}\bigg) L(\boldsymbol{y}), \quad \forall \boldsymbol{y} \in [0,1]^N, \tag{17}$$

where $\Delta \equiv \max_{i \in \mathcal{I}} |\partial^+(i)|$. Note that, the inequality (a) holds with equality for all binary vectors $\boldsymbol{y} \in \{0,1\}^{mN}$. Our Pipage rounding procedure, described in Algorithm 2, starts with an optimal solution of the LP (9). Then it iteratively perturbs two fractional variables (if any) in a single cache in such a way that the value of the surrogate objective function $\phi(\boldsymbol{y})$ never decreases (Steps (15)-(6)) while at least one of the two fractional variables is rounded to an integer. Step (4) ensures that the feasibility is maintained at every step of the perturbation. Upon termination (which happens within $O(mN)$ steps), the rounding procedure yields a feasible integral allocation vector $\hat{\boldsymbol{y}}$ with an objective value $L(\hat{\boldsymbol{y}})$, which is within a factor of $1-(1-\frac{1}{\Delta})^{\Delta}$ of the optimum objective, where $\Delta \equiv \max_{i \in \mathcal{I}} |\partial^+(i)|$. The following theorem formalizes this claim.

Theorem 2. *Algorithm 2 is an $1-(1-\frac{1}{\Delta})^{\Delta}$ approximation algorithm for solving the problem 8.*

See Section 10 of the supplementary material for the complete proof of Theorem 2.

5 Bounding the Number of Fetches

Repeatedly fetching files from the remote server to the local caches consumes bandwidth and increases the network congestion. Hence, it is imperative that the average number of new file fetches to the caches remains small over a long time interval. In the classical caching policies, such as LRU, FIFO, and LFU, a file is fetched if there is a cache-miss. Since the number of fetching is equal to the number of cache-misses for these policies, it is enough for them to minimize the cache-miss rates only. However, since the `LeadCache` policy decouples the fetching process from the cache-misses, in addition to low regret, we need to ensure that the number of file fetches remains small as well. In this paper, we prove the surprising result that if the file request process satisfies a mild stochastic regularity property, the file fetches *stop almost surely after a finite time*. Note that the above result is of a different flavor from the long line of work that minimizes the switching regret Devroye et al. [2015], Kalai and Vempala [2005], Geulen et al. [2010]. These papers do not make any stochastic assumption but give a much weaker guarantee on the number of fetches.

A. Stochastic Regularity Assumption: Let $\{\boldsymbol{X}(t)\}_{t\geq 1}$ be the cumulative request-arrival process. We assume that, there exists a set of non-negative numbers $\{p^i_f\}_{i\in\mathcal{I}, f\in[N]}$ such that for any $\epsilon > 0$:

$$\sum_{t=1}^{\infty} \sum_{i\in\mathcal{I}, f\in[N]} \mathbb{P}\Big(\Big|\frac{\boldsymbol{X}^i_f(t)}{t} - p^i_f\Big| \geq \epsilon\Big) < \infty. \tag{18}$$

Using the first Borel-Cantelli Lemma, the regularity assumption **A** implies that the process $\{\boldsymbol{X}(t)\}_{t\geq 1}$ satisfies the strong-law: $\boldsymbol{X}^i_f(t)/t \to p^i_f$, a.s., $\forall i \in \mathcal{I}, f \in [N]$. However, the converse may not be true. Nevertheless, the assumption **A** is quite mild and holds, *e.g.*, when the file request sequence is generated by a renewal process having an inter-arrival distribution with a finite fourth moment (See Section 12 of the supplementary material for the proof). Following is our main result for this section:

Theorem 3. *Under the stochastic regularity assumption* **A***, the file fetches to the caches stop after a finite time with probability* 1 *under the* `LeadCache` *policy.*

Please refer to Section 11 of the supplementary material for the proof of Theorem 3. Policies, such as LRU, LFU, FIFO, which immediately fetch every missed file, *do not* enjoy the above property.

6 A Minimax Lower Bound

We now establish a minimax lower bound on the regret achievable by any online policy for the Bipartite Caching problem. Before we state and prove our result, we first explain what makes this lower bound fundamentally new. Recall that our reward function (1) is non-linear. As such, the standard lower bounds for the OLO problems, such as Theorem 5.1 and 5.12 of Orabona [2019], Theorem 5 of Abernethy et al. [2008] are insufficient for our purpose. The first non-trivial regret lower bound for the Bipartite Caching problem was given by Bhattacharjee et al. [2020]. In the following, we strengthen this bound by utilizing results from graph coloring.

Theorem 4 (Regret Lower Bound)**.** *For a catalog of size* $N \geq \max(2\frac{d^2Cm}{n}, 2mC)$ *the regret of any online caching policy* π *is lower bounded as:* $R^\pi_T \geq \max(\sqrt{\frac{mnCT}{2\pi}}, d\sqrt{\frac{mCT}{2\pi}}) - \Theta(\frac{1}{\sqrt{T}})$.

Proof outline: We construct an i.i.d. file request sequence where all users request an identical file sampled uniformly at random. Then we tightly lower-bound the total reward accrued by the offline optimal policy. While Bhattacharjee et al. [2020] constructs an offline cache configuration with distinct files in each cache (*global exclusivity*), we construct an offline static caching configuration, where each user is guaranteed to find distinct files on each of its connected caches (*local exclusivity*). This construction effectively linearizes the reward function that can be analyzed by the probabilistic *Balls-into-Bins* framework. Using results from graph coloring theory, we find a small catalog size that guarantees the existence of such a construction. Our result improves the lower bound of Bhattacharjee et al. [2020] in the regime $d > \sqrt{n}$. See Section 13 of the supplementary material for the proof.

Approximation guarantee for `LeadCache`: Theorem 1 and Theorem 4, taken together, imply that the proposed `LeadCache` policy achieves the optimal regret up to a factor of $\tilde{O}(\min((nd)^{1/4}, (n/d)^{3/4}))$. Clearly, the worst-case factor is obtained when $d = \sqrt{n}$. Hence, irrespective of the value of d, the `LeadCache` policy is regret-optimal up to a factor of $\tilde{O}(n^{3/8})$. Furthermore, when d is a constant, the approximation ratio can be further improved to $\tilde{O}(n^{1/4})$.

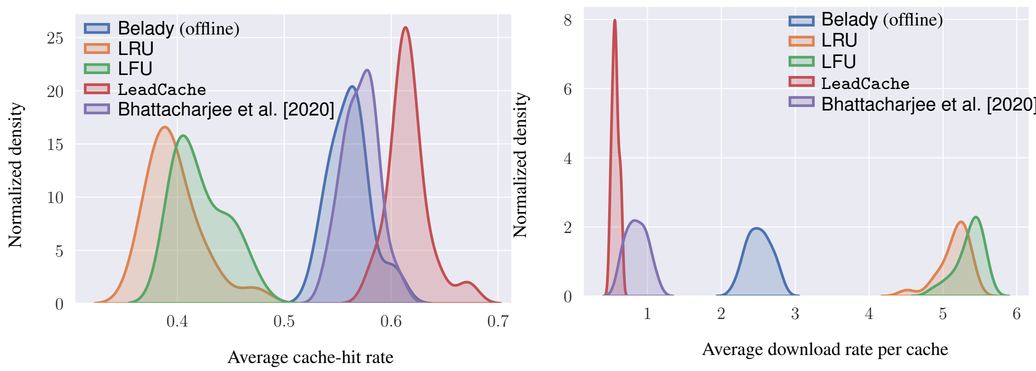


Figure 3: Empirical distributions of (a) Cache-Hit rates and (b) Fetch-rates of different caching policies.

7 Experiments

Baseline policies: In this section, we compare the performance of the `LeadCache` policy with other baselines on a standard dataset. In the LRU policy, each cache independently considers the set of all requested files from its connected users. In the case of a *cache-miss*, the LRU policy fetches the requested file into the cache while evicting a file from the cache that was requested *least recently*. The LFU policy works similarly to the LRU policy with the exception that, in the case of a cache-miss, the policy evicts a file that was requested the *least number of times* among all files currently on the cache. Finally, for the purpose of benchmarking, we also experiment with Belady's offline MIN algorithm Aho et al. [1971], which is optimal for each *individual* cache when the entire file request sequence is known a priori. Note that Belady's algorithm is *not* optimal in the network caching setting as it does not consider the adjacency relations between the caches and the users. As a result, similar to LRU and LFU, multiple users may find the same files cached in more than one connected caches, which does not help in improving the hit-rate. To the best of our knowledge, the characterization of the optimal offline policy in the network caching setting remains an open problem. The heuristic multi-cache policy by Bhattacharjee et al. [2020] uses an FTPL strategy assuming the reward function to be linear.

Experimental Setup: In our experiments, we use a publicly available anonymized production trace from a large CDN provider available under a BSD 2-Clause License [Berger et al., 2018, Berger, 2018]. The trace data consists of three fields, namely, request number, file-id, and file size. In our experiments, we construct a Bipartite caching network with $n = 30$ users and $m = 10$ caches. Each cache is connected to $d = 8$ randomly chosen users. Thus, every user is connected to ≈ 2.67 caches on average. The storage capacity C of each cache is taken to be 10% of the catalog size. We divide the trace consisting of the first $\sim 375K$ requests into $k = 20$ consecutive sub-intervals. File requests are uniformly assigned to the users before running the experiments on each of the sub-intervals.

Results and Discussion: Figure 3 shows the performance distribution of different caching policies in terms of the average cache-hit rate per file and the average number of file-fetches per cache. The average values of the above metrics for different file request sequences are listed in Table 1 in the supplementary section. From the plots, it is clear that the `LeadCache` policy outperforms the baseline policies in terms of *both* cache-hit rate and fetch-rate. Recall that Belady's policy is an *offline* policy that knows the future file requests. Figure 7 in the supplementary section shows a joint plot of the cache-hit rates and download rates of different policies. From the plots, it is clear that the `LeadCache` policy excels by coordinating the caching decisions among different caches and quickly adapting with the file request sequence. Section 14 of the supplementary material gives additional plots for the popularity distribution and the temporal dynamics of the policies for a given file request sequence, where we observe a similar pattern.

8 Conclusion and Future Work

We proposed an efficient network caching policy and showed that it is competitive with the existing caching policies, both theoretically and empirically. We proved a tight lower bound for the achievable regret and proved that the number of file-fetches incurred by our policy is finite under reasonable assumptions. Although any existing CDN load-balancer can be used along with `LeadCache`, a major limitation of our work is that it does not jointly optimize the load balancing and cache allocations. Furthermore, it will be interesting to design caching policies with strong guarantees for both regret and the competitive ratio [Daniely and Mansour, 2019].

References

Wouter M Koolen, Manfred K Warmuth, Jyrki Kivinen, et al. Hedging structured concepts. In *COLT*, pages 93–105. Citeseer, 2010.

Alon Cohen and Tamir Hazan. Following the perturbed leader for online structured learning. In *International Conference on Machine Learning*, pages 1034–1042, 2015.

Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. The akamai network: A platform for high-performance internet applications. *SIGOPS Oper. Syst. Rev.*, 44(3):2–19, August 2010. ISSN 0163-5980. doi: 10.1145/1842733.1842736. URL `https://doi.org/10.1145/1842733.1842736`.

Georgios Paschos, George Iosifidis, and Giuseppe Caire. Cache optimization models and algorithms. *Foundations and Trends® in Communications and Information Theory*, 16(3–4):156–345, 2020. ISSN 1567-2190. doi: 10.1561/0100000104. URL `http://dx.doi.org/10.1561/0100000104`.

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`.

Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge university press, 2006.

Arun Sai Suggala and Praneeth Netrapalli. Online non-convex learning: Following the perturbed leader is optimal. In *Algorithmic Learning Theory*, pages 845–861, 2020.

James Hannan. Approximation to bayes risk in repeated play. *Contributions to the Theory of Games*, 3:97–139, 1957.

Adam Kalai and Santosh Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3):291–307, 2005.

Luc Devroye, Gábor Lugosi, and Gergely Neu. Random-walk perturbations for online combinatorial optimization. *IEEE Transactions on Information Theory*, 61(7):4099–4106, 2015.

Jacob Abernethy, Chansoo Lee, and Ambuj Tewari. Perturbation techniques in online learning and optimization. *Perturbations, Optimization, and Statistics*, page 233, 2016.

Amit Daniely, Alon Gonen, and Shai Shalev-Shwartz. Strongly adaptive online learning. In *International Conference on Machine Learning*, pages 1405–1411, 2015.

Kwang-Sung Jun, Francesco Orabona, Stephen Wright, and Rebecca Willett. Improved strongly adaptive online learning using coin betting. In *Artificial Intelligence and Statistics*, pages 943–951. PMLR, 2017.

Stefano Traverso, Mohamed Ahmed, Michele Garetto, Paolo Giaccone, Emilio Leonardi, and Saverio Niccolini. Temporal locality in today's content caching: why it matters and how to model it. *ACM SIGCOMM Computer Communication Review*, 43(5):5–12, 2013.

Benjamin Van Roy. A short proof of optimality for the min cache replacement algorithm. *Information processing letters*, 102(2-3):72–73, 2007.

Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H Noh, Sang Lyul Min, Yookun Cho, and Chong-Sang Kim. On the existence of a spectrum of policies that subsumes the least recently used (lru) and least frequently used (lfu) policies. In *SIGMETRICS*, volume 99, pages 1–4. Citeseer, 1999.

Asit Dan and Don Towsley. *An approximate analysis of the LRU and FIFO buffer replacement schemes*, volume 18. ACM, 1990.

Amit Daniely and Yishay Mansour. Competitive ratio vs regret minimization: achieving the best of both worlds. In *Algorithmic Learning Theory*, pages 333–368, 2019.

Nick Littlestone and Manfred K Warmuth. The weighted majority algorithm. *Information and computation*, 108(2):212–261, 1994.

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

Karthikeyan Shanmugam, Negin Golrezaei, Alexandros G Dimakis, Andreas F Molisch, and Giuseppe Caire. Femtocaching: Wireless content delivery through distributed caching helpers. *IEEE Transactions on Information Theory*, 59(12):8402–8413, 2013.

Georgios S Paschos, Apostolos Destounis, Luigi Vigneri, and George Iosifidis. Learning to cache with no regrets. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 235–243. IEEE, 2019.

Francesco Orabona. A modern introduction to online learning. *arXiv preprint arXiv:1912.13213*, 2019.

Alexander A Ageev and Maxim I Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8(3): 307–328, 2004.

Georgios S Paschos, George Iosifidis, Meixia Tao, Don Towsley, and Giuseppe Caire. The role of caching in future communication systems and networks. *IEEE Journal on Selected Areas in Communications*, 36(6):1111–1125, 2018.

Reinhard Diestel. Graph theory 3rd ed. *Graduate texts in mathematics*, 173, 2005.

Jacob Abernethy, Chansoo Lee, Abhinav Sinha, and Ambuj Tewari. Online linear optimization via smoothing. In *Conference on Learning Theory*, pages 807–823, 2014.

Chansoo Lee. *Analysis of Perturbation Techniques in Online Learning*. PhD thesis, 2018.

Sascha Geulen, Berthold Vöcking, and Melanie Winkler. Regret minimization for online buffering problems using the weighted majority algorithm. In *COLT*, pages 132–143. Citeseer, 2010.

Jacob Abernethy, Peter L Bartlett, Alexander Rakhlin, and Ambuj Tewari. Optimal strategies and minimax lower bounds for online convex games. 2008.

Alfred V Aho, Peter J Denning, and Jeffrey D Ullman. Principles of optimal page replacement. *Journal of the ACM (JACM)*, 18(1):80–93, 1971.

Daniel S Berger, Nathan Beckmann, and Mor Harchol-Balter. Practical bounds on optimal caching with variable object sizes. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(2):1–38, 2018.

Daniel S Berger. CDN Traces. `https://github.com/dasebe/optimalwebcaching`, 2018.

Martin J Wainwright. *High-dimensional statistics: A non-asymptotic viewpoint*, volume 48. Cambridge University Press, 2019.

Pascal Massart. *Concentration inequalities and model selection*, volume 6. Springer, 2007.

Jacob Abernethy and Alexander Rakhlin. Optimal strategies and minimax lower bounds for online convex games. In *In Proceedings of the Nineteenth Annual Conference on Computational Learning Theory*, 2008.

Noga Alon and Joel H Spencer. *The probabilistic method*. John Wiley & Sons, 2004.

Godfrey Harold Hardy, John Edensor Littlewood, George Pólya, György Pólya, DE Littlewood, et al. *Inequalities*. Cambridge university press, 1952.

F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

9 Supplementary Material for the paper ***LeadCache*: Regret-Optimal Caching in Networks**

9.1 Proof of Lemma 1

From the equation (4), we have that for any file request vector $\boldsymbol{x}$ and virtual caching configuration $\boldsymbol{z}$:

$$r(\boldsymbol{x}, \boldsymbol{z}) = \langle \boldsymbol{x}, \boldsymbol{z} \rangle \leq q(\boldsymbol{x}, \psi(\boldsymbol{z})).$$

Thus, for any file request sequence $\{\boldsymbol{x}_t\}_{t\geq 1}$, we have:

$$\sum_{t=1}^{T} r(\boldsymbol{x}_t, \boldsymbol{z}_t) \leq \sum_{t=1}^{T} q(\boldsymbol{x}_t, \psi(\boldsymbol{z}_t)). \tag{19}$$

On the other hand, let $\boldsymbol{y}_* \in \arg\max_{\boldsymbol{y}\in\mathcal{Y}} \sum_{t=1}^{T} q(\boldsymbol{x}_t, \boldsymbol{y})$ be an optimal offline static cache configuration vector for the file request sequence $\{\boldsymbol{x}_t\}_{t\geq 1}$. Consider a candidate static virtual cache configuration vector $\boldsymbol{z}_*$ defined as

$$\boldsymbol{z}^i = \min\left\{\mathbf{1}_{N\times 1}, \Big(\sum_{j\in\partial^+(i)} \boldsymbol{y}_*^j\Big)\right\}, \quad 1 \leq i \leq n.$$

We have

$$\max_{\boldsymbol{z}\in\mathcal{Z}} \sum_{t=1}^{T} r(\boldsymbol{x}_t, \boldsymbol{z}) \geq \langle \sum_{t=1}^{T} \boldsymbol{x}_t, \boldsymbol{z}_* \rangle = \max_{\boldsymbol{y}\in\mathcal{Y}} \sum_{t=1}^{T} q(\boldsymbol{x}_t, \boldsymbol{y}). \tag{20}$$

Combining Eqns. (19) and (20), we have

$$\max_{\boldsymbol{y}\in\mathcal{Y}} \sum_{t=1}^{T} q(\boldsymbol{x}_t, \boldsymbol{y}) - \sum_{t=1}^{T} q(\boldsymbol{x}_t, \psi(\boldsymbol{z}_t)) \leq \max_{\boldsymbol{z}\in\mathcal{Z}} \sum_{t=1}^{T} r(\boldsymbol{x}_t, \boldsymbol{z}) - \sum_{t=1}^{T} r(\boldsymbol{x}_t, \boldsymbol{z}).$$

Taking supremum of both sides of the above w.r.t. the file request sequence $\{\boldsymbol{x}_t\}_{t\geq 1}$ yields the result. ■

9.2 Proof of Theorem 1

Keeping Lemma 1 in view, it is enough to upper-bound the regret for the virtual policy π^{virtual} only. Following Cohen and Hazan [2015] we derive a general expression for the regret upper-bound applicable to any linear reward function under the anytime FTPL policy. For this, we extend the argument of Cohen and Hazan [2015] to the anytime setting. Finally, we specialize this bound to the specific setting of our problem.
Recall that, the file-request sequence from the users is denoted by by $\{\boldsymbol{x}_t\}_{t\geq 1}$ and the virtual cache configuration sequence is denoted by $\{\boldsymbol{z}_t\}_{t\geq 1}$. Define the aggregate arrival up to time t by

$$\boldsymbol{X}_t = \sum_{\tau=1}^{t-1} \boldsymbol{x}_\tau.$$

Note that the LeadCache policy chooses the virtual cache configuration at time slot t as the solution of the following problem

$$\boldsymbol{z}_t = \arg\max_{\boldsymbol{z}\in\mathcal{Z}} \langle \boldsymbol{z}, \boldsymbol{X}_t + \eta_t \boldsymbol{\gamma} \rangle, \tag{21}$$

where each of the Nn components of $\boldsymbol{\gamma}$ are sampled i.i.d. from a standard Gaussian distribution, and $\mathcal{Z}$ denotes the set of feasible virtual cache configurations. Next, we define the following potential function:

$$\Phi_{\eta_t}(\boldsymbol{x}) \equiv \mathbb{E}_{\boldsymbol{\gamma}}\Big[\max_{\boldsymbol{z}\in\boldsymbol{\mathcal{Z}}} \langle \boldsymbol{z}, \boldsymbol{x} + \eta_t \boldsymbol{\gamma} \rangle\Big]. \tag{22}$$

From the above definition and Cohen and Hazan [2015], it follows that $\nabla \Phi_{\eta_t}(\boldsymbol{X}_t) = \mathbb{E}(\boldsymbol{z}_t)$. Hence, we have:

$$\langle \nabla \Phi_{\eta_t}(\boldsymbol{X}_t), \boldsymbol{x}_t \rangle = \mathbb{E}\langle \boldsymbol{z}_t, \boldsymbol{x}_t \rangle. \tag{23}$$

To bound the regret of the LeadCache policy from the above, we expand $\Phi_{\eta_t}(\boldsymbol{X}_{t+1})$ in second-order Taylor's series as follows:

$$
\begin{aligned}
& \Phi_{\eta_t}(\boldsymbol{X}_{t+1}) \\
= \;& \Phi_{\eta_t}(\boldsymbol{X}_t + \boldsymbol{x}_t) \\
= \;& \Phi_{\eta_t}(\boldsymbol{X}_t) + \langle \nabla \Phi_{\eta_t}(\boldsymbol{X}_t), \boldsymbol{x}_t \rangle + \frac{1}{2}\boldsymbol{x}_t^T \nabla^2 \Phi_{\eta_t}(\tilde{\boldsymbol{X}}_t)\boldsymbol{x}_t,
\end{aligned}
$$

where $\tilde{\boldsymbol{X}}_t$ lies on the line segment joining $\boldsymbol{X}_t$ and $\boldsymbol{X}_{t+1}$. Plugging in Eqn. (23) in the above expression, we obtain

$$
\mathbb{E}\langle \boldsymbol{z}_t, \boldsymbol{x}_t \rangle = \Phi_{\eta_t}(\boldsymbol{X}_{t+1}) - \Phi_{\eta_t}(\boldsymbol{X}_t) - \frac{1}{2}\boldsymbol{x}_t^T \nabla^2 \Phi_{\eta_t}(\tilde{\boldsymbol{X}}_t)\boldsymbol{x}_t. \tag{24}
$$

Summing up the equation (24) from $t = 1$ to T, we obtain the total expected reward obtained by the LeadCache policy as:

$$
\begin{aligned}
& \mathbb{E}\big(Q^{\text{LeadCache}}(T)\big) \\
= \;& \sum_{t=1}^{T} \mathbb{E}\langle \boldsymbol{z}_t, \boldsymbol{x}_t \rangle \\
= \;& \sum_{t=1}^{T} \bigg(\Phi_{\eta_t}(\boldsymbol{X}_{t+1}) - \Phi_{\eta_t}(\boldsymbol{X}_t)\bigg) - \frac{1}{2}\sum_{t=1}^{T} \boldsymbol{x}_t^T \nabla^2 \Phi_{\eta_t}(\tilde{\boldsymbol{X}}_t)\boldsymbol{x}_t \\
= \;& \sum_{t=1}^{T} \bigg(\Phi_{\eta_t}(\boldsymbol{X}_{t+1}) - \Phi_{\eta_{t-1}}(\boldsymbol{X}_t) + \Phi_{\eta_{t-1}}(\boldsymbol{X}_t) - \Phi_{\eta_t}(\boldsymbol{X}_t)\bigg) - \frac{1}{2}\sum_{t=1}^{T} \boldsymbol{x}_t^T \nabla^2 \Phi_{\eta_t}(\tilde{\boldsymbol{X}}_t)\boldsymbol{x}_t \\
= \;& \sum_{t=1}^{T} \bigg(\Phi_{\eta_t}(\boldsymbol{X}_{t+1}) - \Phi_{\eta_{t+1}}(\boldsymbol{X}_{t+1})\bigg) + \bigg(\Phi_{\eta_{T+1}}(\boldsymbol{X}_{T+1}) - \Phi_{\eta_1}(\boldsymbol{X}_1)\bigg) - \frac{1}{2}\sum_{t=1}^{T} \boldsymbol{x}_t^T \nabla^2 \Phi_{\eta_t}(\tilde{\boldsymbol{X}}_t)\boldsymbol{x}_t
\end{aligned}
$$

Next, note that

$$
\begin{aligned}
\Phi_{\eta_{T+1}}(\boldsymbol{X}_{T+1}) \;&=\; \mathbb{E}_{\boldsymbol{\gamma}}\big[\max_{\boldsymbol{z}\in\mathcal{Z}} \langle \boldsymbol{z}, \boldsymbol{X}_{T+1} + \eta_{T+1}\boldsymbol{\gamma}\rangle\big] \\
&\overset{\text{(Jensen's ineq.)}}{\geq}\; \max_{\boldsymbol{z}\in\mathcal{Z}} \big[\mathbb{E}_{\boldsymbol{\gamma}}\langle \boldsymbol{z}, \boldsymbol{X}_{T+1} + \eta_{T+1}\boldsymbol{\gamma}\rangle\big] \\
&=\; \max_{\boldsymbol{z}\in\mathcal{Z}} \langle \boldsymbol{z}, \boldsymbol{X}_{T+1}\rangle \\
&=\; Q^*(T),
\end{aligned}
$$

where, we recall that $Q^*(T)$ denotes the optimal cumulative reward up to time T obtained by the best static policy in the hindsight. Hence, from the above two inequalities, we can upper bound the expected regret (2) of the LeadCache policy as:

$$
\begin{aligned}
& \mathbb{E}(R_T^{\text{LeadCache}}) \\
= \;& Q^*(T) - \mathbb{E}\big(Q^{\text{LeadCache}}(T)\big) \\
\leq \;& \Phi_{\eta_1}(\boldsymbol{X}_1) + \underbrace{\sum_{t=1}^{T} \bigg(\Phi_{\eta_{t+1}}(\boldsymbol{X}_{t+1}) - \Phi_{\eta_t}(\boldsymbol{X}_{t+1})\bigg)}_{(a)} + \frac{1}{2}\sum_{t=1}^{T} \boldsymbol{x}_t^T \nabla^2 \Phi_{\eta_t}(\tilde{\boldsymbol{X}}_t)\boldsymbol{x}_t.
\end{aligned} \tag{25}
$$

Bounding the Term (a): Next, we analyze the term (a) in the inequality above. From Eqns. (21) and (22), we can write:

$$
\Phi_{\eta_{t+1}}(\boldsymbol{X}_{t+1}) = \mathbb{E}\big[\langle \boldsymbol{z}_{t+1}, \boldsymbol{X}_{t+1} + \eta_{t+1}\boldsymbol{\gamma}\rangle\big],
$$

and

$$
\Phi_{\eta_t}(\boldsymbol{X}_{t+1}) \geq \mathbb{E}\big[\langle \boldsymbol{z}_{t+1}, \boldsymbol{X}_{t+1} + \eta_t\boldsymbol{\gamma}\rangle\big].
$$

Thus, each individual term in the summation (a) may be upper bounded as follows:

$$\begin{aligned}\Phi_{\eta_{t+1}}(\boldsymbol{X}_{t+1})-\Phi_{\eta_t}(\boldsymbol{X}_{t+1}) &\le \mathbb{E}\big[\langle \boldsymbol{z}_{t+1},\boldsymbol{X}_{t+1}+\eta_{t+1}\boldsymbol{\gamma}\rangle\big]-\mathbb{E}\big[\langle \boldsymbol{z}_{t+1},\boldsymbol{X}_{t+1}+\eta_t\boldsymbol{\gamma}\rangle\big]\\ &= \mathbb{E}\big[\langle \boldsymbol{z}_{t+1},(\eta_{t+1}-\eta_t)\boldsymbol{\gamma})\rangle\big]\\ &= (\eta_{t+1}-\eta_t)\mathbb{E}\big[\langle \boldsymbol{z}_{t+1},\boldsymbol{\gamma}\rangle\big]\\ &\le (\eta_{t+1}-\eta_t)\mathbb{E}\big[\max_{z\in\mathcal{Z}}\langle \boldsymbol{z},\boldsymbol{\gamma}\rangle\big]\\ &= (\eta_{t+1}-\eta_t)\mathcal{G}(\mathcal{Z}),\end{aligned}$$

where the quantity $\mathcal{G}(\mathcal{Z})$ is known as the *Gaussian complexity* of the set $\mathcal{Z}$ of virtual configurations Wainwright [2019]. Substituting the above upper bound in Eqn. (25), we notice that the summation in part (a) telescopes, yielding the following bound for the expected regret:

$$\mathbb{E}(R_T^{\mathsf{LeadCache}}) \le \eta_{T+1}\underbrace{\mathcal{G}(\mathcal{Z})}_{(b)}+\frac{1}{2}\sum_{t=1}^{T}\underbrace{\boldsymbol{x}_t^T\nabla^2\Phi_{\eta_t}(\tilde{\boldsymbol{X}}_t)\boldsymbol{x}_t}_{(c)}. \tag{26}$$

Bounding the Term (b): Recall that

$$\mathcal{G}(\mathcal{Z}) \equiv \mathbb{E}_{\boldsymbol{\gamma}}\big[\max_{\boldsymbol{z}\in\mathcal{Z}}\langle \boldsymbol{z},\boldsymbol{\gamma}\rangle\big],$$

In the following, we bound this term from the above. From equation (4), for any feasible $\boldsymbol{z}\in\mathcal{Z}$, we have:

$$\begin{aligned}\sum_{i\in\mathcal{I},f\in[N]}\boldsymbol{z}_f^i &\le \sum_{i\in\mathcal{I},f\in[N]}\sum_{j\in\partial^+(i)}\boldsymbol{y}_f^j\\ &= \sum_{j\in\mathcal{J}}\sum_{f\in[N]}\sum_{i\in\partial^-(j)}\boldsymbol{y}_f^j\\ &\overset{(a)}{\le} d\sum_{j\in\mathcal{J}}\sum_{f\in[N]}\boldsymbol{y}_f^j\\ &\overset{(b)}{\le} dmC.\end{aligned} \tag{27}$$

where in step (a), we have used the fact that the right-degree of the bipartite graph $\mathcal{G}$ is upper-bounded by d, and in (b), we have used the fact that the capacity of each cache is bounded by C.

Next, recall that each component of the virtual cache-configuration vector $\boldsymbol{z}$ is binary. Moreover, for any fixed $\boldsymbol{z}\in\mathcal{Z}$, the random variable $\langle \boldsymbol{z},\boldsymbol{\gamma}\rangle$ follows a normal distribution with mean zero and variance σ^2 where

$$\sigma^2 \equiv \mathbb{E}\langle \boldsymbol{z},\boldsymbol{\gamma}\rangle^2 \overset{(a)}{=} \sum_{i\in\mathcal{I},f\in[N]}(\boldsymbol{z}_f^i)^2 \overset{(b)}{=} \sum_{i\in\mathcal{I},f\in[N]}\boldsymbol{z}_f^i \overset{(c)}{\le} dmC.$$

In the above, the equality (a) follows from the fact the the each component of the random vector $\boldsymbol{\gamma}$ are independent, the equality (b) follows from the fact that z_f^i's are binary-valued (hence, $(z_f^i)^2 = z_f^i$), and the equality (c) follows from Eqn. (27).

Now we make the simple observation that if $\boldsymbol{z}_*\in\arg\max_{\boldsymbol{z}\in\mathcal{Z}}\langle \boldsymbol{z},\boldsymbol{\gamma}\rangle$, then $\gamma_f^i<0$ implies $z_{*f}^i=0$. Hence, we can simplify the expression for the Gaussian complexity of the set $\mathcal{Z}$ as

$$\mathcal{G}(\mathcal{Z}) \equiv \mathbb{E}_{\boldsymbol{\gamma}}\big[\max_{\boldsymbol{z}\in\mathcal{Z}}\langle \boldsymbol{z},\boldsymbol{\gamma}\rangle\big] = \mathbb{E}_{\boldsymbol{\gamma}}\big[\max_{\boldsymbol{z}\in\mathcal{Z}}\sum_{(i,f):\gamma_f^i>0} z_f^i\gamma_f^i\big].$$

Since all the coefficients γ_f^i in the above summation is positive, from the definition of the constraint set $\mathcal{Z}$, we conclude that there exists an optimal vector $\boldsymbol{z}_*$ such that the inequality in Eqn. (4) is met with an equality, *i.e.,*

$$\boldsymbol{z}_{*f}^i = \min(1, \sum_{j\in\partial^+(i)} y_{*f}^j),\ \ \forall (i,f):\gamma_f^i>0,$$

for some $\boldsymbol{y}_*\in\mathcal{Y}$. Let $\mathcal{Z}_*$ be the set of all virtual caching vectors obtained this way. This immediately implies that $|\mathcal{Z}_*|\le|\mathcal{Y}|$. Since any of the m caches can be loaded with any arbitrary C files, we can write

$$|\mathcal{Z}_*|\le|\mathcal{Y}|\le\binom{N}{C}^m\le\Big(\frac{Ne}{C}\Big)^{mC},$$

where the last inequality is a standard upper bound for binomial coefficients. Finally, using Massart [2007]'s lemma for Gaussian variables, we have

$$\begin{aligned}\mathcal{G}(\mathcal{Z}) \equiv \mathbb{E}_{\boldsymbol{\gamma}}\big[\max_{\boldsymbol{z}\in\mathcal{Z}}\langle \boldsymbol{z}, \boldsymbol{\gamma}\rangle\big] = \mathbb{E}_{\boldsymbol{\gamma}}\big[\max_{\boldsymbol{z}\in\mathcal{Z}_*}\sum_{(i,f):\gamma_f^i>0} z_f^i\gamma_f^i\big] \quad &\leq \quad \sqrt{dmC}\sqrt{2\log|\mathcal{Z}_*|} \\ &\leq \quad mC\sqrt{2d\Big(\log\frac{N}{C}+1\Big)}. \qquad (28)\end{aligned}$$

Bounding the Term (c): Let us denote the file requested by the i^{th} user at time t by f_i. For bounding the last term in (25), we use Lemma 7 from Abernethy and Rakhlin [2008], which yields:

$$\big(\nabla^2\Phi_{\eta_t}(\tilde{\boldsymbol{X}}_{\boldsymbol{t}})\big)_{\boldsymbol{pq}} = \frac{1}{\eta_t}\mathbb{E}\big[\hat{\boldsymbol{z}}_{\boldsymbol{p}}\gamma_{\boldsymbol{q}}\big], \qquad (29)$$

where $\hat{\boldsymbol{z}} \in \arg\max_{\boldsymbol{z}\in\mathcal{Z}}\langle \boldsymbol{z}, \tilde{\boldsymbol{X}}_t + \eta_t\boldsymbol{\gamma}\rangle$, and each of the indices $\boldsymbol{p}, \boldsymbol{q}$ is a (user, file) tuple. Hence, using Eqn. (29), and noting that each user requests only one file at a time, we can expand the quadratic as:

$$\begin{aligned}\boldsymbol{x}_t^T\nabla^2\Phi_{\eta_t}(\tilde{\boldsymbol{X}}_{\boldsymbol{t}})\boldsymbol{x}_t \quad &= \quad \frac{1}{\eta_t}\sum_{i,j\in\mathcal{I}}\mathbb{E}[\hat{z}_{f_i}^i\gamma_{f_j}^j] \\ &= \quad \frac{1}{\eta_t}\mathbb{E}(\sum_{i\in\mathcal{I}}\hat{z}_{f_i}^i)(\sum_{j\in\mathcal{I}}\gamma_{f_j}^j) \\ &\overset{(a)}{\leq} \quad \frac{1}{\eta_t}\sqrt{\mathbb{E}(\sum_{i\in\mathcal{I}}\hat{z}_{f_i}^i)^2\mathbb{E}(\sum_{j\in\mathcal{I}}\gamma_{f_j}^j)^2} \\ &\overset{(b)}{\leq} \quad \frac{1}{\eta_t}\sqrt{n^2\times n} \\ &= \quad \frac{1}{\eta_t}n^{3/2}, \qquad (30)\end{aligned}$$

where the inequality (a) follows from the Cauchy-Schwartz inequality, and the inequality (b) follows from the fact that $\boldsymbol{z}$ are binary variables and the components of the random vector $\boldsymbol{\gamma}$ are i.i.d. Finally, from Eqn. (26) and the upper bounds for the constituent terms, we may bound the expected regret from the above as:

$$\begin{aligned}\mathbb{E}\big(R_T^{\textsf{LeadCache}}\big) \quad &\leq \quad \eta_{T+1}\mathcal{G}(\mathcal{Z}) + \frac{n^{3/2}}{2}\sum_{t=1}^{T}\frac{1}{\eta_t} \\ &\leq \quad \eta_{T+1}mC\sqrt{2d\Big(\log\frac{N}{C}+1\Big)} + \frac{n^{3/2}}{2}\sum_{t=1}^{T}\frac{1}{\eta_t}.\end{aligned}$$

where the bound in the last inequality follows from Eqn. (28). Choosing $\eta_t = \beta\sqrt{t}$ with an appropriate constant $\beta > 0$ yields the following bound:

$$\mathbb{E}\big(R_T^{\textsf{LeadCache}}\big) \leq kn^{3/4}d^{1/4}\sqrt{mCT},$$

for some $k = O(\textsf{poly-log}(N/C))$. ∎

10 Proof of Theorem 2

Denote the objective function of the problem (8) by $L(\boldsymbol{y}) \equiv \sum_{i,f}\theta_f^i\min(1, \sum_{j\in\partial^+(i)}y_f^j)$, where, to simplify the notations, we have not explicitly shown the dependence of the $\boldsymbol{\theta}$ coefficients on the time index t. Recall the definition of surrogate objective function $\phi(\boldsymbol{y})$ given in Eqn. (16):

$$\phi(\boldsymbol{y}) = \sum_{i,f}(\theta_f^i)^+\big(1 - \prod_{j\in\partial^+(i)}(1-y_f^j)\big), \qquad (31)$$

From Eqn. (16) of Ageev and Sviridenko [2004], we have the following algebraic inequality:

$$L(\boldsymbol{y}) \overset{(a)}{\geq} \phi(\boldsymbol{y}) \geq \Bigg(1 - (1-\frac{1}{\Delta})^{\Delta}\Bigg)L(\boldsymbol{y}), \qquad (32)$$

where $\Delta \equiv \max_{i\in\mathcal{I}} |\partial^+(i)|$. Note that, the inequality (a) holds with equality for binary vectors $\boldsymbol{y} \in \{0,1\}^{mN}$.
Let $\boldsymbol{y}^*$ be a solution of the relaxed LP (9), and OPT be the optimal value of the problem (8). Obviously, $L(\boldsymbol{y}^*) \geq \mathsf{OPT}$, which, combined with (32), yields:

$$\phi(\boldsymbol{y}^*) \geq \bigg(1 - (1 - \frac{1}{\Delta})^{\Delta}\bigg)\mathsf{OPT}. \tag{33}$$

Since $\boldsymbol{y}^*$ is a solution to the relaxed LP, it may possibly contain fractional coordinates. In the following, we show that the Pipage rounding procedure, described in Algorithm 2, rounds at least one fractional variable of a cache at a time *without* decreasing the value of the surrogate objective function $\phi(\cdot)$ (Steps 3-6).
For a given fractional allocation vector $\boldsymbol{y}$, and another vector $\boldsymbol{v_y}$ of our choice depending on $\boldsymbol{y}$, define a one-dimensional function $g_{\boldsymbol{y}}(\cdot)$ as:

$$g_{\boldsymbol{y}}(s) = \phi(\boldsymbol{y} + s\boldsymbol{v_y}). \tag{34}$$

The vector $\boldsymbol{v_y}$ denotes the direction along which the fractional allocation vector $\boldsymbol{y}$ is rounded in the current step. The Pipage rounding procedure, Algorithm 2, chooses the vector $\boldsymbol{v_y}$ as follows: consider any cache j that has at least two fractional coordinates $y^j_{f_1}$ and $y^j_{f_2}$ in the current allocation $\boldsymbol{y}$ (Step 3 of Algorithm 2). Take $\boldsymbol{v_y} = e_{j,f_1} - e_{j,f_2}$, where $e_{j,l}$ denotes the standard unit vector with 1 in the coordinate corresponding to the l^{th} file of the j^{th} cache, $l = f_1, f_2$. We now claim that the function $g_{\boldsymbol{y}}(s) = \phi(\boldsymbol{y} + s\boldsymbol{v_y})$ is linear in s. To see this, consider any one of the constituent terms of $g_{\boldsymbol{y}}(s)$ as given in Eqn. (31). Examining each term, we arrive at the following two cases:

1. If both $f \neq f_1$ and $f \neq f_2$ then that term is independent of s.
2. If either $f = f_1$, or $f = f_2$, the variables $y^j_{f_1}$ or $y^j_{f_2}$ may appear in each product term in (31) at most once. Since, the products involve variable corresponding to a specific file f, the variables $y^j_{f_1}$ and $y^j_{f_2}$ can not appear in the same product term together.

The above two cases imply that the function $g_{\boldsymbol{y}}(s)$ is linear in s. By increasing and decreasing the variable s to the maximum extent possible, so that the candidate allocation $\boldsymbol{y} + s\boldsymbol{v_y}$ does not violate the constraint (12), we construct two new candidate allocation vectors $\boldsymbol{\alpha} = \boldsymbol{y} - \epsilon_1\boldsymbol{v_y}$ and $\boldsymbol{\beta} = \boldsymbol{y} + \epsilon_2\boldsymbol{v_y}$, where the constants ϵ_1 and ϵ_2 are chosen in such a way that at least one of the fractional variables of $\boldsymbol{y}$ becomes integral (Steps 4-5). It is easy to see that, all the capacity cache constraints in Eqn. (11) continue to hold in both of these two candidate allocations. In step 6, we choose the best of the candidate allocations $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, with respect to the surrogate function $\phi(\cdot)$. Let $\boldsymbol{y}^{\text{new}}$ denote the new candidate allocation vector. Since the maximum of a linear function over an interval is achieved on one of its two boundaries, we conclude that $\phi(\boldsymbol{y}^{\text{new}}) \geq \phi(\boldsymbol{y})$. As argued above, the rounded solution has at least one less fractional coordinates and feasible. Hence, by repeated application of the above procedure, we finally arrive at a feasible integral allocation $\hat{\boldsymbol{y}}$ such that:

$$L(\hat{\boldsymbol{y}}) = \phi(\hat{\boldsymbol{y}}) \geq \phi(\boldsymbol{y}^*) \geq \bigg(1 - (1 - \frac{1}{\Delta})^{\Delta}\bigg)\mathsf{OPT},$$

where the first equality follows from that fact that the functions $\phi(\boldsymbol{y}) = L(\boldsymbol{y})$ on integral points. ■

11 Proof of Theorem 3

Discussion: To understand why the number of fetches is expected to be small under the LeadCache policy, consider the simplest case of a single cache with a single user. At every slot, the LeadCache policy populates the cache with a set of C files having the highest perturbed cumulative requests $\boldsymbol{\Theta}(t)$. To simplify the argument, assume that the learning-rate η_t is time-invariant. Since at most one file is requested per slot, only one component of $\boldsymbol{\Theta}(t)$ changes, and hence, the LeadCache policy fetches at most *one* new file at a slot. In fact, as the following proof shows, the average number of fetches over a large time interval could be far less than one.

Recall that, under the LeadCache policy, the optimal virtual caching configuration $\boldsymbol{z}_t$ for the t^{th} slot is obtained by solving the problem P:

$$\max_{\boldsymbol{z}\in\mathcal{Z}} \sum_{i\in\mathcal{I}} \langle \boldsymbol{\theta}^i(t), \boldsymbol{z}^i \rangle, \tag{35}$$

where we assume that the ties (if any) are broken according to some fixed tie-breaking rule. As we have seen before, the corresponding physical cache configuration $\boldsymbol{y}_t$ may be obtained as a by-product of the above optimization problem. Now consider a static virtual cache-configuration $\tilde{\boldsymbol{z}}$ obtained by replacing the perturbed-count vectors $\boldsymbol{\theta}^i(t)$ in the objective function (35) by the vectors $\boldsymbol{p}^i, \forall i \in \mathcal{I}$, where $\boldsymbol{p} = \big(\boldsymbol{p}^i, i \in \mathcal{I}\big)$ is the vector of long-term file-request probabilities, given by Eqn. (18). In other words,

$$\tilde{\boldsymbol{z}} \in \arg\max_{\boldsymbol{z} \in \mathcal{Z}} \sum_{i \in \mathcal{I}} \langle \boldsymbol{p}^i, \boldsymbol{z}^i \rangle. \tag{36}$$

Since the set of all possible virtual caching configurations $\mathcal{Z}$ is finite, the objective value corresponding to any other non-optimal caching configuration must be a non-zero gap $\Delta > 0$ away from that of an optimal configuration. Let us denote the set of all *sub-optimal* virtual cache configuration vectors by $\mathcal{B}$. Hence, for all any $\boldsymbol{z} \in \mathcal{B}$, we must have:

$$\sum_{i \in \mathcal{I}} \langle \boldsymbol{p}^i, \tilde{\boldsymbol{z}}^i \rangle \geq \sum_{i \in \mathcal{I}} \langle \boldsymbol{p}^i, \boldsymbol{z}^i \rangle + \Delta. \tag{37}$$

Let us define an "error" event $\mathcal{E}(t)$ to be event such that the LeadCache policy yields a sub-optimal virtual cache configuration (and possibly, a sub-optimal physical cache configuration (4)) at time t. Let G be a zero-mean Gaussian random variable with variance $2Nn$. We now bound the probability of this error event from the above as shown below:

$$\begin{aligned}
&\mathbb{P}(\mathcal{E}(t)) \\
\overset{(a)}{\leq}\; & \mathbb{P}\bigg(\sum_{i \in \mathcal{I}} \langle \boldsymbol{\theta}^i(t), \boldsymbol{z}^i(t) \rangle > \sum_{i \in \mathcal{I}} \langle \boldsymbol{\theta}^i(t), \tilde{\boldsymbol{z}}^i \rangle, \boldsymbol{z}(t) \in \mathcal{B} \bigg) \\
\overset{(b)}{\leq}\; & \mathbb{P}\bigg(\eta_t G \geq \sum_{i \in \mathcal{I}} \langle \boldsymbol{X}^i(t), \tilde{\boldsymbol{z}}^i \rangle - \sum_{i \in \mathcal{I}} \langle \boldsymbol{X}^i(t), \boldsymbol{z}^i(t) \rangle, \boldsymbol{z}(t) \in \mathcal{B} \bigg) \\
\overset{(c)}{=}\; & \mathbb{P}\bigg(\eta_t G \geq \sum_{i \in \mathcal{I}} \langle \boldsymbol{X}^i(t), \tilde{\boldsymbol{z}}^i \rangle - \sum_{i \in \mathcal{I}} \langle \boldsymbol{X}^i(t), \boldsymbol{z}^i(t) \rangle, \sum_{i \in \mathcal{I}} \langle \boldsymbol{X}^i(t), \tilde{\boldsymbol{z}}^i \rangle - \sum_{i \in \mathcal{I}} \langle \boldsymbol{X}^i(t), \boldsymbol{z}^i(t) \rangle > \frac{\Delta t}{2}, \boldsymbol{z}(t) \in \mathcal{B} \bigg) \\
& + \mathbb{P}\bigg(\eta_t G \geq \sum_{i \in \mathcal{I}} \langle \boldsymbol{X}^i(t), \tilde{\boldsymbol{z}}^i \rangle - \sum_{i \in \mathcal{I}} \langle \boldsymbol{X}^i(t), \boldsymbol{z}^i(t) \rangle, \sum_{i \in \mathcal{I}} \langle \boldsymbol{X}^i(t), \tilde{\boldsymbol{z}}^i \rangle - \sum_{i \in \mathcal{I}} \langle \boldsymbol{X}^i(t), \boldsymbol{z}^i(t) \rangle \leq \frac{\Delta t}{2}, \boldsymbol{z}(t) \in \mathcal{B} \bigg) \\
\overset{(d)}{\leq}\; & \mathbb{P}\bigg(\eta_t G \geq \frac{\Delta t}{2} \bigg) + \mathbb{P}\bigg(\frac{1}{t} \sum_{i \in \mathcal{I}} \langle \boldsymbol{X}^i(t), \tilde{\boldsymbol{z}}^i - \boldsymbol{z}^i(t) \rangle \leq \frac{\Delta}{2}, \boldsymbol{z}(t) \in \mathcal{B} \bigg) \\
=\; & \mathbb{P}\bigg(\eta_t G \geq \frac{\Delta t}{2} \bigg) + \mathbb{P}\bigg(\sum_{i \in \mathcal{I}} \langle \frac{\boldsymbol{X}^i(t)}{t} - \boldsymbol{p}^i, \tilde{\boldsymbol{z}}^i - \boldsymbol{z}^i(t) \rangle + \sum_{i \in \mathcal{I}} \langle \boldsymbol{p}^i, \tilde{\boldsymbol{z}}^i - \boldsymbol{z}^i(t) \rangle \leq \frac{\Delta}{2}, \boldsymbol{z}(t) \in \mathcal{B} \bigg) \\
\overset{(e)}{\leq}\; & \mathbb{P}\bigg(\eta_t G \geq \frac{\Delta t}{2} \bigg) + \mathbb{P}\bigg(\sum_{i \in \mathcal{I}} \langle \frac{\boldsymbol{X}^i(t)}{t} - \boldsymbol{p}^i, \tilde{\boldsymbol{z}}^i - \boldsymbol{z}^i(t) \rangle \leq -\frac{\Delta}{2} \bigg) \\
\overset{(f)}{\leq}\; & \mathbb{P}\bigg(\eta_t G \geq \frac{\Delta t}{2} \bigg) + \mathbb{P}\bigg(\sum_{i \in \mathcal{I}} \sum_{f \in [N]} \bigg| \frac{\boldsymbol{X}^i_f(t)}{t} - p^i_f \bigg| \geq \frac{\Delta}{2} \bigg) \\
\overset{(g)}{\leq}\; & \mathbb{P}\bigg(\eta_t G \geq \frac{\Delta t}{2} \bigg) + \mathbb{P}\bigg(\bigcup_{i,f} \bigg| \frac{\boldsymbol{X}^i_f(t)}{t} - p^i_f \bigg| \geq \frac{\Delta}{2Nn} \bigg) \\
\overset{(h)}{\leq}\; & \mathbb{P}\bigg(\eta_t G \geq \frac{\Delta t}{2} \bigg) + \sum_{i,f} \mathbb{P}\bigg(\bigg| \frac{\boldsymbol{X}^i_f(t)}{t} - p^i_f \bigg| \geq \frac{\Delta}{2Nn} \bigg) \\
\overset{(i)}{\leq}\; & \exp(-ct) + Nn\alpha_\epsilon(t).
\end{aligned}$$

for some positive constants c and ϵ, which depend on the parameters of the problem. In the above chain of inequalities

(a) follows from the fact that on the error event $\mathcal{E}(t)$, the virtual cache configuration vector $\boldsymbol{z}(t)$ must be in the sub-optimal set $\mathcal{B}$ and it must yield more objective value in (35) than the vector $\tilde{\boldsymbol{z}}$,

(b) follows by writing $\boldsymbol{\Theta}(t) = \boldsymbol{X}(t) + \eta_t \boldsymbol{\gamma}$, and observing that the virtual configurations $\boldsymbol{z}(t) \in \mathcal{B}$ and $\tilde{\boldsymbol{z}}(t)$ may differ in at most Nn coordinates, and that the normal random variables are increasing

(in the convex ordering sense) with their variances,
(c) follows from the law of total probability,
(d) follows from the monotonicity of the probability measures,
(e) follows from Eqn. (37),
(f) follows from the fact that for any two equal-length vectors $\boldsymbol{a}, \boldsymbol{b}$, triangle inequality yields:

$$\langle \boldsymbol{a}, \boldsymbol{b} \rangle \geq -\sum_k |\boldsymbol{a}_k||\boldsymbol{b}_k|,$$

and that $|\tilde{\boldsymbol{z}}_f^i - \boldsymbol{z}_f^i(t)| \leq 1, \forall i, f$,
(g) follows from the simple observation that at least one number in a set of some numbers must be at least as large as the average,
(h) follows from the union bound,
and finally, the inequality (i) follows from the concentration inequality for Gaussian variables and the concentration inequality for the request process $\{\boldsymbol{X}(t)\}_{t\geq 1}$, as given by Eqn. (18). Using the above bound and the assumptions on the request sequence, we have

$$\sum_{t\geq 1} \mathbb{P}(\mathcal{E}(t)) \leq \sum_{t\geq 1} \exp(-ct) + Nn \sum_{t\geq 1} \alpha_\epsilon(t) < \infty.$$

Hence, the first Borel-Cantelli Lemma implies that

$$\mathbb{P}(\mathcal{E}(t) \text{ i.o}) = 0.$$

Hence, almost surely, the error events stop after a finite time. Thus, with a fixed tie-breaking rule, the new file fetches stop after a finite time w.p. 1. ■

12 Renewal Processes satisfies the Regularity Condition (A)

Proof. Suppose that the random process $\{\boldsymbol{X}_f^i(t)\}_{t\geq 1}$ constitutes a renewal process of with expected renewal interval length $1/p_f^i$ and a finite fourth moment. Then we have

$$\begin{aligned}
\mathbb{P}\bigg(\frac{\boldsymbol{X}_f^i(t)}{t} - p_f^i \leq -\epsilon\bigg) &= \mathbb{P}\bigg(\boldsymbol{X}_f^i(t) \leq t(p_f^i - \epsilon)\bigg) \\
&\leq \mathbb{P}\bigg(S_{\lfloor t(p_f^i - \epsilon)\rfloor} \geq t\bigg) \\
&\leq \mathbb{P}\bigg((S_{\lfloor t(p_f^i-\epsilon)\rfloor} - \lfloor t(p_f^i - \epsilon)\rfloor)^4 \geq (t(1-p_f^i-\epsilon))^4\bigg) \\
&\overset{(a)}{\leq} O(\frac{1}{t^2}),
\end{aligned}$$

where in the Markov inequality (a), we have used the fact that the fourth moment of the centred random walk is $O(t^2)$. Using a similar line of arguments, we can show that

$$\mathbb{P}\bigg(\frac{\boldsymbol{X}_f^i(t)}{t} - p_f^i \geq \epsilon\bigg) \leq O(\frac{1}{t^2}).$$

Combining the above two bounds, we conclude that

$$\sum_{t\geq 1} \mathbb{P}\bigg(|\frac{\boldsymbol{X}_f^i(t)}{t} - p_f^i| \geq \epsilon\bigg) < \infty.$$

This verifies the regularity condition A. □

13 Proof of Theorem 4

We establish a slightly stronger result by proving the lower bound for a regular bipartite network with uniform left-degree d_L and uniform right degree d. Counting the total number of edges in two different ways, we have $nd_L = md$. Hence, $d_L = \frac{md}{n}$. For pedagogical reasons, we divide the proof into several parts.

(A) Some Observations and Preliminary Lemmas: To facilitate the analysis, we introduce the following surrogate linear reward function:

$$q_{\text{linear}}(\boldsymbol{x}, \boldsymbol{y}) \equiv \sum_{i \in \mathcal{I}} \boldsymbol{x}_t^i \cdot \Big(\sum_{j \in \partial^+(i)} \boldsymbol{y}_t^j \Big). \tag{38}$$

We begin our analysis with the following two observations:

1. **Upper Bound:** From the definition (1) of the rewards, we clearly have:

$$q(\boldsymbol{x}, \boldsymbol{y}) \leq q_{\text{linear}}(\boldsymbol{x}, \boldsymbol{y}), \quad \forall \boldsymbol{x}, \boldsymbol{y}. \tag{39}$$

2. **Local Exclusivity implies Linearity:** In the case when all caches connected to each user host *distinct* files, *i.e.,* the cached files are *locally exclusive* in the sense that they not duplicated from each user's local point-of-view, *i.e.,*

$$\boldsymbol{y}^{j_1} \cdot \boldsymbol{y}^{j_2} = 0, \quad \forall j_1 \neq j_2 : j_1, j_2 \in \partial^+(i), \forall i \in \mathcal{I}, \tag{40}$$

the reward function reduces to a linear one:

$$q(\boldsymbol{x}, \boldsymbol{y}) = q_{\text{linear}}(\boldsymbol{x}, \boldsymbol{y}), \quad \forall \boldsymbol{x}, \boldsymbol{y}. \tag{41}$$

The equation (41) follows from the fact that with the local exclusivity constraint, we have

$$\sum_{j \in \partial^+(i)} \boldsymbol{y}_t^j \leq \mathbf{1}, \quad \forall i \in \mathcal{I},$$

where the inequality holds componentwise. Hence, the '$\min$' operator in the definition of the reward function (Eqn. (1)) is vacuous in this case. To make use of the linearity of the rewards as in Eqn. (41), the caches need to store items in such a way that the local exclusivity condition in Eqn. (40) holds. Towards this, we now define a special coloring of the nodes in the set $\mathcal{J}$ for a given bipartite graph $\mathcal{G}(\mathcal{I} \cup \mathcal{J}, E)$.

Definition 1 (Valid χ-coloring of the caches)**.** *Let χ be a positive integer. A valid χ-coloring of the caches of a bipartite network $\mathcal{G}(\mathcal{I} \cup \mathcal{J}, E)$ is an assignment of colors from the set $\{1, 2, \ldots, \chi\}$ to the vertices in $\mathcal{J}$ (i.e., the caches) in such a way that all neighboring caches $\partial^+(i)$ to every node $i \in \mathcal{I}$ (i.e., the users) are assigned distinct colors.*

Obviously, for a given bipartite graph $\mathcal{G}$, a valid χ-coloring of the caches exists only if the number of possible colors χ is large enough. The following lemma gives an upper bound to the value of χ so that a valid χ-coloring of the caches exists.

Lemma 2. *Suppose we are given a bipartite network $\mathcal{G}(\mathcal{I} \cup \mathcal{J}, E)$, where each user $i \in \mathcal{I}$ is connected to at most d_L caches, and each cache $j \in \mathcal{J}$ is connected to at most d users. Then there exists a valid χ-coloring of the caches where $\chi \leq d_L d$.*

Proof. From the given bipartite network $\mathcal{G}(V, E)$, construct another graph $H(V', E')$, where the caches form the vertices of H, *i.e.,* $V' \equiv \mathcal{J}$. For any two vertices in $u, v \in V'$, there is an edge $(u, v) \in E'$ if and only if a user $i \in \mathcal{I}$ is connected to both the caches u and v in the bipartite network $\mathcal{G}$. Next, consider any cache $j \in \mathcal{J}$. By our assumption, it is connected to at most d users. On the other hand, each of the users is connected to at most $d_L - 1$ caches other than j. Hence, the degree of any node j in the graph H is upper bounded as:

$$\Delta \leq d(d_L - 1) \leq d_L d - 1.$$

Finally, using Brook's theorem Diestel [2005], we conclude that the vertices of the graph H may be colored using at most $1 + \Delta \leq d_L d$ distinct colors. □

(B) Probabilistic Method for Regret Lower Bounds: With the above results at our disposal, we now employ the well-known probabilistic method of Alon and Spencer [2004] for proving the regret lower bound. The basic principle of the probabilistic method is quite simple. We compute a lower bound L_T to the expected regret under any online network caching policy π for a selected joint probability distribution $\boldsymbol{p}(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_T)$ of the incoming file request sequence. Since the maximum of a set of numbers is at least as large as the expectation, the quantity L_T also gives a lower bound to the regret under an adversarial file request sequence. Hence, the power of the probabilistic method largely comes from the ability to identify a distribution $\boldsymbol{p}$ which is amenable to analysis and, at the same time, yields a tight bound. In the following, we flesh out the above principle.

Fix a valid χ-coloring of the caches, and let $k = \chi C$. Upon setting $N = 2k$, we select a randomized and *identical* file request sequence $\{\boldsymbol{x}_t^i \equiv \boldsymbol{\alpha}_t\}_{t=1}^T$ for each user, where the common file request vector $\boldsymbol{\alpha}_t$ at the slot t is sampled independently across time and uniformly at random from the set of the *first* $2k$ unit vectors $\{\boldsymbol{e}_i \in \mathbb{R}^{2k}, 1 \leq i \leq 2k\}$. In other words, we choose:

$$p(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_T) \coloneqq \prod_{t=1}^{T} \left(\frac{1}{2k} 1\big(\boldsymbol{x}_t^{i_1} = \boldsymbol{x}_t^{i_2},\ \forall i_1, i_2 \in \mathcal{I}\big) \right).$$

(C) Upper-bounding the Total Reward accrued by any Online Policy: Making use of the observation (39), the expected total reward G_T^π accrued by any online network caching policy π may be upper bounded as follows:

$$\begin{aligned}
G_T^\pi \quad &\leq \quad \mathbb{E}\bigg(\sum_{t=1}^{T} \sum_{i \in \mathcal{I}} \boldsymbol{x}_t^i \cdot \sum_{j \in \partial^+(i)} \boldsymbol{y}_t^j \bigg) \\
&\overset{(a)}{=} \quad \sum_{t=1}^{T} \sum_{(i,j) \in E} \mathbb{E}\big(\boldsymbol{x}_t^i \cdot \boldsymbol{y}_t^j\big) \\
&\overset{(b)}{=} \quad \sum_{t=1}^{T} \sum_{(i,j) \in E} \mathbb{E}\big(\boldsymbol{x}_t^i\big) \cdot \mathbb{E}\big(\boldsymbol{y}_t^j\big) \\
&\overset{(c)}{=} \quad \frac{1}{2k} \sum_{t=1}^{T} \mathbb{E}\bigg(\sum_{(i,j) \in E} \sum_{f \in [N]} \boldsymbol{y}_{tf}^j \bigg) \\
&\overset{(d)}{\leq} \quad \frac{d}{2k} \sum_{t=1}^{T} \mathbb{E}\bigg(\sum_{j \in \mathcal{J}} \sum_{f \in [N]} \boldsymbol{y}_{tf}^j \bigg) \\
&\overset{(e)}{\leq} \quad \frac{dmCT}{2k},
\end{aligned} \tag{42}$$

where the eqn. (a) follows from the linearity of expectation, eqn. (b) follows from the fact that the cache configuration vector $\boldsymbol{y}_t$ is independent of the file request vector $\boldsymbol{x}_t$ at the same slot, as the policy is online, the eqn. (c) follows from the fact that each of the $N = 2k$ components of the vector $\mathbb{E}\big(\boldsymbol{x}_t^i\big)$ is equal to $\frac{1}{2k}$, the inequality (d) follows from the fact that each cache is connected to at most d users, and finally, the inequality (e) follows from the cache capacity constraints.

(D) Lower-bounding the Total Reward Accrued by the Static Oracle: We now lower bound the total expected reward accrued by the offline static oracle policy (*i.e.,* the first term in the definition (1)). Note that, due to the presence of the '$\min$' operator in (1), obtaining an exactly optimal static cache-configuration vector $\boldsymbol{y}^*$ is non-trivial, as it requires solving the NP-hard optimization problem (8) (with the vector $\boldsymbol{\theta}(t)$ in the objective replaced by the cumulative file request vector $\boldsymbol{X}(T) \equiv \sum_{t=1}^T \boldsymbol{x}_t$). However, since we only require a good lower bound to the total expected reward, a suitably constructed sub-optimal caching configuration will serve our purpose, provided that we can compute a lower bound to its expected reward in closed form. Towards this end, in the following, we construct a joint cache-configuration vector $\boldsymbol{y}_\perp$ that satisfies the local exclusivity constraint (40).

D.1 Construction of a "good" cache-configuration vector $\boldsymbol{y}_\perp$: Let $\mathcal{X}$ be a valid χ-coloring of the caches. Let the color c appear in f_c distinct caches in the coloring $\mathcal{X}$. To simplify the notations, we relabel the colors in non-increasing order of their frequency of appearance in the coloring $\mathcal{X}$, *i.e.,*

$$f_1 \geq f_2 \geq \dots \geq f_\chi. \tag{43}$$

Let the vector $\boldsymbol{v}$ be obtained by sorting the components of the vector $\sum_{t=1}^T \boldsymbol{\alpha}_t$ in non-increasing order. Partition the vector $\boldsymbol{v}$ into $\frac{2k}{C} = 2\chi$ segments by merging C contiguous coordinates of $\boldsymbol{v}$ at a time. Let c_j denote the color of the cache $j \in \mathcal{J}$ in the coloring $\mathcal{X}$. The cache configuration vector $\boldsymbol{y}_\perp$ is constructed by loading each cache $j \in \mathcal{J}$ with the set of all C files in the ${c_j}^{\text{th}}$ segment of the vector $\boldsymbol{v}$.

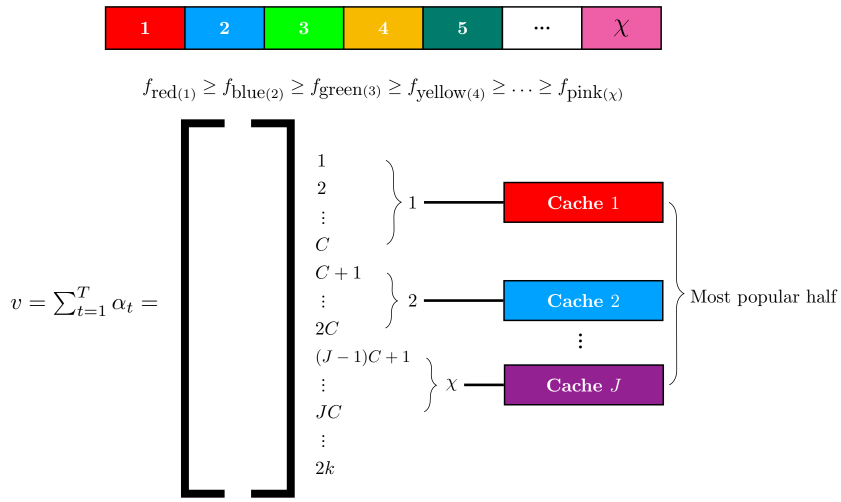


Figure 4: Construction of the caching configuration $\boldsymbol{y}_{\perp}$.

D.2 Observation: Since the vector $\boldsymbol{v}$ has 2χ segments (each containing C distinct files) and the number of possible colors in the coloring $\mathcal{X}$ is χ, it follows that only the most popular half of the files get loaded to some caches under $\boldsymbol{y}_{\perp}$. Moreover, it can be easily verified that the cache configuration vector $\boldsymbol{y}_{\perp}$ satisfies the local exclusivity property (40) by construction.

D.3 Analysis: Let $S_{\boldsymbol{v}}(m)$ denote the sum of the components in the m^{th} segment of the vector $\boldsymbol{v}$. In other words, $S_{\boldsymbol{v}}(m)$ denotes the aggregate frequency of requests for the files in the m^{th} segment of the vector $\boldsymbol{v}$. By construction, we have

$$S_{\boldsymbol{v}}(1) \geq S_{\boldsymbol{v}}(2) \geq \ldots \geq S_{\boldsymbol{v}}(\chi). \tag{44}$$

Since under the distribution $\boldsymbol{p}$, all users almost surely request the same file at each time slot (*i.e.,* $\boldsymbol{x}_t^i = \boldsymbol{\alpha}_t, \forall i$), and since the linearity holds due to the local exclusivity property of the cache configuration $\boldsymbol{y}_{\perp}$, the reward obtained by the files in the j^{th} cache under the caching configuration $\boldsymbol{y}_{\perp}$ is given by:

$$\begin{aligned}
\boldsymbol{y}_{\perp}^j \cdot \Big(\sum_{i \in \partial^-(j)} \sum_{t=1}^{T} \boldsymbol{x}_t^i \Big) &= \boldsymbol{y}_{\perp}^j \cdot \Big(\sum_{i \in \partial^-(j)} \sum_{t=1}^{T} \boldsymbol{\alpha}_t \Big) \\
&\overset{(a)}{=} d\boldsymbol{y}_{\perp}^j \cdot \Big(\sum_{t=1}^{T} \boldsymbol{\alpha}_t \Big) \\
&\overset{(b)}{=} dS_{\boldsymbol{v}}(c_j),
\end{aligned} \tag{45}$$

where the equation (a) follows from the fact that, in this converse proof, we are investigating a regular bipartite network where each cache is connected to exactly d users, and the equation (b) follows from the construction of the cache configuration vector $\boldsymbol{y}_{\perp}$. Hence, the expected aggregate reward accrued by the optimal stationary configuration $\boldsymbol{y}^*$ may be lower-bounded by that of the configuration $\boldsymbol{y}_{\perp}$ as follows:

$$\begin{aligned}
G_T^{\pi^*} &\overset{(a)}{\geq} \mathbb{E}\Big(\sum_{j \in \mathcal{J}} \boldsymbol{y}_{\perp}^j \cdot \big(\sum_{i \in \partial^-(j)} \sum_{t=1}^{T} \boldsymbol{x}_t^i \big) \Big) \\
&\overset{(b)}{=} d\mathbb{E}\Big(\sum_{j \in \mathcal{J}} S_{\boldsymbol{v}}(c_j) \Big) \\
&\overset{(c)}{=} d\mathbb{E}\Big(\sum_{c=1}^{\chi} f_c S_{\boldsymbol{v}}(c) \Big) \\
&\overset{(d)}{\geq} \frac{d}{\chi} \big(\sum_{c=1}^{\chi} f_c \big) \mathbb{E}\big(\sum_{c=1}^{\chi} S_{\boldsymbol{v}}(c) \big),
\end{aligned}$$

where
(a) follows from the local exclusivity property of the configuration $\boldsymbol{y}_\perp$,
(b) follows from Eqn. (45),
(c) follows after noting that the color c appears on f_c distinct caches in the coloring $\mathcal{X}$,
(d) follows from an algebraic inequality presented in Lemma 3 below, used in conjunction with the conditions (43) and (44).

Next, we lower bound the quantity $\mathbb{E}\big(\sum_{c=1}^{\chi} S_{\boldsymbol{v}}(c)\big)$ appearing on the right hand side of the equation (46). Conceptually, identify the catalog with $N = 2k$ "bins", and the random file requests $\{\boldsymbol{\alpha}_t\}_{t=1}^T$ as "balls" thrown uniformly into one of the "bins." With this correspondence in mind, a little thought reveals that the random variable $\sum_{c=1}^{\chi} S_{\boldsymbol{v}}(c)$ is distributed identically as the total load in the most popular $k = \chi C$ bins when T balls are thrown uniformly at random into $2k$ bins. Continuing with the above chain of inequalities, we have

$$\begin{aligned} G_T^{\pi^*} &\overset{(e)}{\geq} \frac{dmC}{k}\mathbb{E}\big(\text{load in the most popular half of } 2k \text{ bins with } T \text{ balls thrown u.a.r.}\big) \\ &\overset{(f)}{\geq} \frac{dmC}{k}\left(\frac{T}{2}+\sqrt{\frac{kT}{2\pi}}\right) - \Theta(\frac{1}{\sqrt{T}}) \\ &= \frac{dmCT}{2k} + dmC\sqrt{\frac{T}{2\pi k}} - \Theta(\frac{1}{\sqrt{T}}), \end{aligned} \tag{46}$$

where in the inequality (e), we have used the fact that $\sum_{c=1}^{\chi} f_c = m$, and the inequality (f) follows from Lemma 4 below proved in Bhattacharjee et al. [2020]. Hence, combining Eqns. (42) and (46), and noting that $k = \chi C \leq d_L dC = \frac{mCd^2}{n}$ from Lemma 2, we conclude that for any caching policy π :

$$R_T^{\pi} \geq G_T^{\pi^*} - G_T^{\pi} \geq \sqrt{\frac{mnCT}{2\pi}} - \Theta(\frac{1}{\sqrt{T}}). \tag{47}$$

Moreover, making use of a *Globally* exclusive configuration (where each caches store distinct files), in Theorem 7 of their paper, Bhattacharjee et al. [2020] proved the following regret lower bound for any online caching policy π:

$$R_T^{\pi} \geq d\sqrt{\frac{mCT}{2\pi}} - \Theta(\frac{1}{\sqrt{T}}). \tag{48}$$

Hence, combining the bounds from Eqns. (47) and (48), we conclude that

$$R_T^{\pi} \geq \max\left(\sqrt{\frac{mnCT}{2\pi}}, d\sqrt{\frac{mCT}{2\pi}}\right) - \Theta(\frac{1}{\sqrt{T}}).$$

■

Lemma 3. *Let $f_1 \geq f_2 \geq \ldots \geq f_n$ and $s_1 \geq s_2 \geq \ldots \geq s_n$ be two non-increasing sequences of n real numbers each. Then*

$$\sum_{i=1}^{n} f_i s_i \geq \frac{1}{n}\big(\sum_{i=1}^{n} f_i\big)\big(\sum_{i=1}^{n} s_i\big).$$

Proof. From the rearrangement inequality (Hardy et al. [1952]), we have for each $0 \leq j \leq n-1$:

$$\sum_{i=1}^{n} f_i s_i \geq \sum_{i=1}^{n} s_i f_{(i+j)(\text{mod } n)+1}, \tag{49}$$

where the modulo operator is used to cyclically shift the indices. Summing over the inequalities (49) for all $0 \leq j \leq n-1$, we have

$$n\sum_{i=1}^{n} f_i s_i \geq \big(\sum_{i=1}^{n} f_i\big)\big(\sum_{i=1}^{n} s_i\big),$$

which yields the result. □

Lemma 4 (Bhattacharjee et al. [2020])**.** *Suppose that T balls are thrown independently and uniformly at random into $2C$ bins. Let the random variable $M_C(T)$ denote the number of balls in the most populated C bins. Then*

$$\mathbb{E}(M_C(T)) \geq \frac{T}{2} + \sqrt{\frac{CT}{2\pi}} - \Theta\left(\frac{1}{\sqrt{T}}\right).$$

14 Additional Experimental Results

In this Section, we compare the performance of the `LeadCache` policy with four other standard benchmark policies on two datasets taken from different application domains. We observe that the relative performance of the algorithms remains qualitatively the same across the datasets with the `LeadCache` policy consistently maintaining the highest hit-rate. In our experiments, we instantiated a randomly generated bipartite network with $n = 30$ users and $m = 10$ caches. Each cache is connected to $d = 8$ randomly chosen users. The capacity of each cache is taken to be 10% of the library size. Our experiments are run on HPE Apollo XL170rGen10 Servers with Dual Intel Xeon Gold 6248 20-core and 192 GB RAM.

14.1 Experiments on the CMU dataset [Berger et al., 2018]

Dataset description: This dataset is obtained from the production trace of a global content-distribution networks. It consists of 500M requests for a total of 18M objects. The popularity distribution of the requests follow approximately a Zipf distribution with the parameter α between 0.85 and 1. Since we are interested only in the order in which the requests arrive, we ignore the size of the objects in our experiments. Due to the extremely large volume of the original dataset, we consider only the first ~ 375K requests in our experiments.

Table 1: Performance Evaluation on the CMU dataset [Berger et al., 2018]

Policies	Hit Rate	Fetch Rate
`LeadCache`	**0.616**	**0.568**
Heuristic [Bhattacharjee et al., 2020]	0.571	0.854
`LRU`	0.397	5.153
`LFU`	0.422	5.359
`Belady` (offline)	0.561	2.507

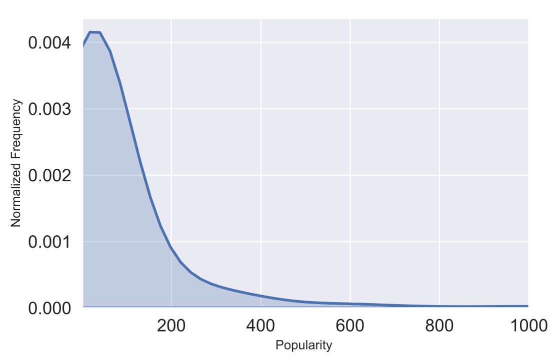


Figure 5: Plot showing the popularity distribution of the files in the CMU dataset

Figure 5 shows the sorted overall popularity distribution of the most popular files in the dataset. It is easy to see that the popularity distribution has a light tail - a small fraction of the files are extremely

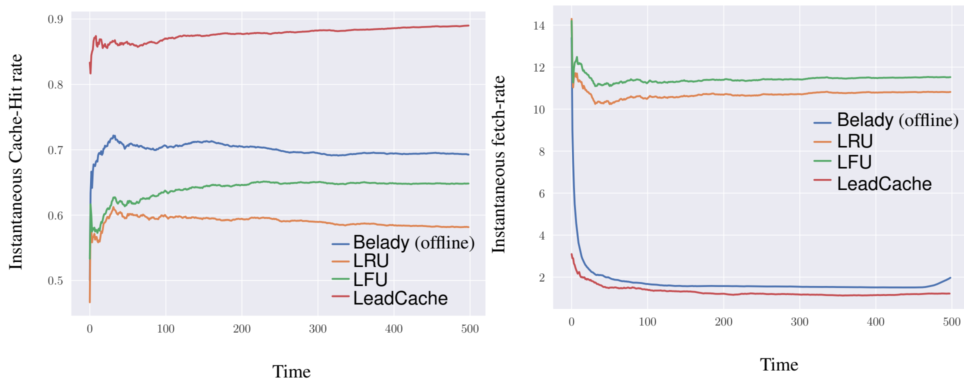


Figure 6: Temporal dynamics of instantaneous (a) Cache-Hit rates and (b) Fetch-rates of different caching policies for a particular file request sequence from the CMU dataset [Berger et al., 2018]

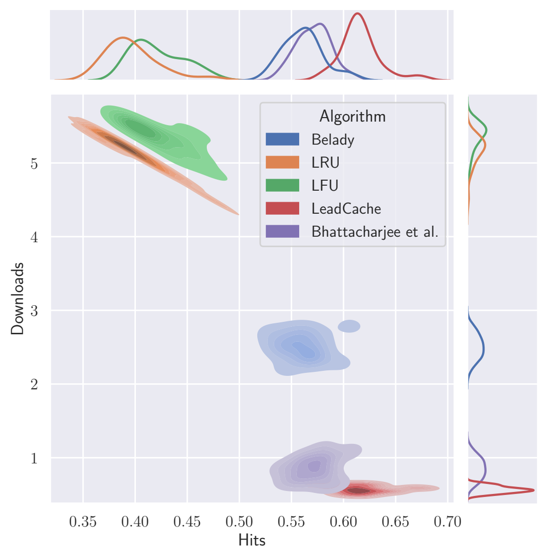


Figure 7: Bivariate plot of the Cache-Hit rates and the Fetch-rates of different caching policies for the CMU dataset [Berger et al., 2018]

popular, while others are rarely requested. The *Recall distance* measures the number of file requests between two successive requests of the same file. Figure 8 shows a plot of the empirical Recall distance distribution for this dataset.

Experimental Results: Figure 6 compares the dynamics of the caching policies for a particular file request pattern. It shows that the proposed LeadCache policy maintains a high cache-hit rate right from the beginning. In other words, the proposed policy quickly learns the file request pattern from all users and distributes the files near-optimally on different caches. This plot also shows that the fetch-rate of the LeadCache policy remains small compared to the other three caching policies. Figure 7 gives a bivariate plot of the cache-hits and cache-downloads by the LeadCache policy. From the plots, it is clear that the `LeadCache` policy outperforms the benchmarks on this dataset in terms of both Hit rate and Fetch rate. The average values of the performance indices are summarized in Table 1.

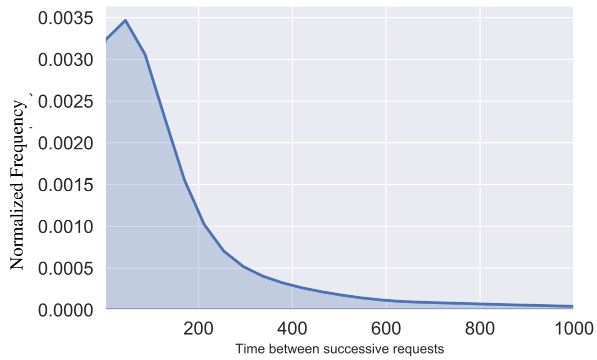


Figure 8: Distribution of time between two successive request of the same file on the CMU Dataset

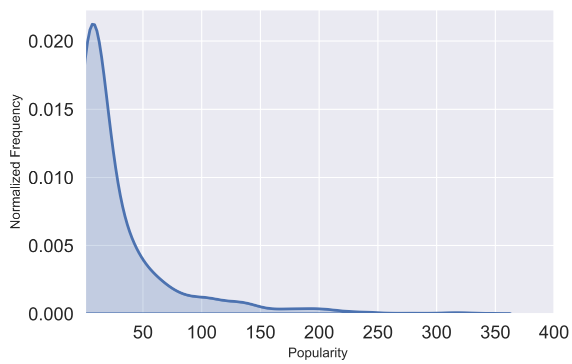


Figure 9: Empirical Popularity distribution of the number of ratings for the MovieLens Dataset [Harper and Konstan, 2015]

14.2 Experiments with the MovieLens Dataset [Harper and Konstan, 2015]

Dataset Description: MovieLens [1] is a popular dataset containing ~ 20M ratings for N ~ 27278 movies along with the timestamps of the ratings [Harper and Konstan, 2015]. The ratings were assigned by 138493 users over a period of approximately twenty years. Our working assumption is that a user rates a movie in the same sequence as she requests the movie file for download from the Content Distribution Network. Due to the sheer size of the dataset, in our experiments, we consider the first 1M ratings only. Figure 9 shows the empirical distribution of the number of times the movies have been rated (and hence, downloaded) by the users. Figure 10 shows the empirical distribution of time between two successive requests of the same file (*i.e.,* the *Recall distance*).

Table 2: Performance Evaluation with the MovieLens dataset [Harper and Konstan, 2015]

Policies	Hit Rate	Fetch Rate
`LeadCache`	**0.991**	1.509
Heuristic [Bhattacharjee et al., 2020]	0.694	**0.297**
`LRU`	0.312	3.234
`LFU`	0.595	2.028
`Belady` (offline)	0.560	1.589

[1] This dataset is freely available from `https://www.kaggle.com/grouplens/movielens-20m-dataset`

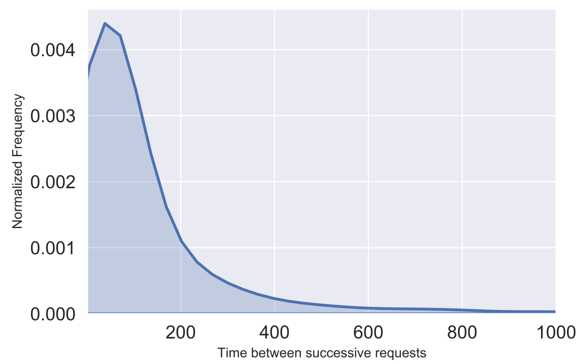


Figure 10: Distribution of time between two successive request of the same file on the MovieLens Dataset

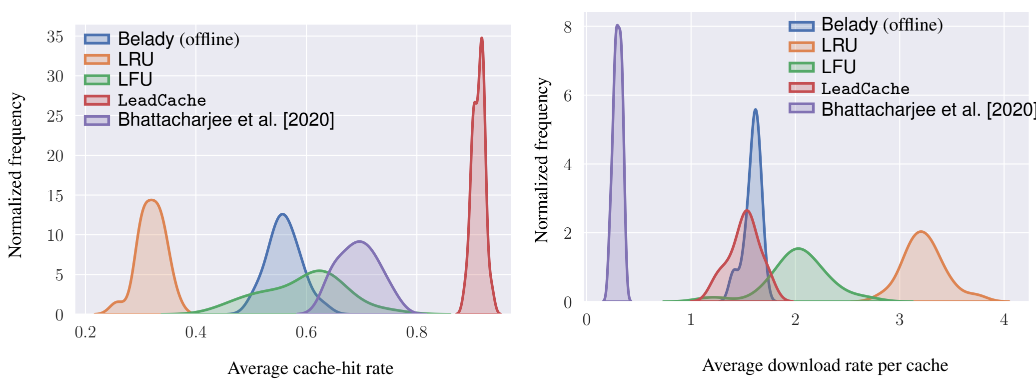


Figure 11: Empirical distributions of (a) Cache-Hit rates and (b) Fetch-rates of different caching policies on the MovieLens Dataset.

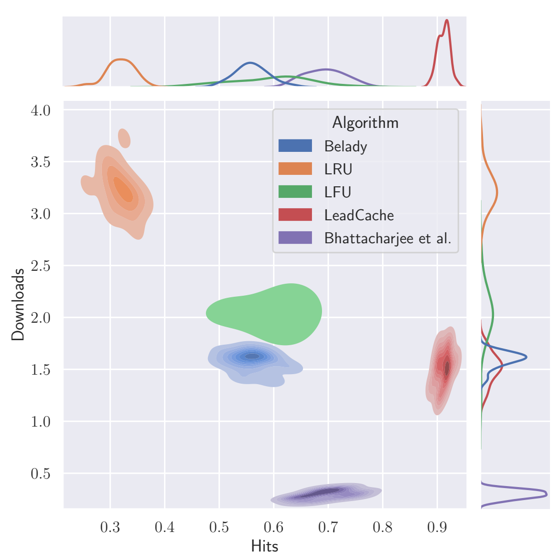


Figure 12: Bivariate plot of the Cache-Hit rates and the Fetch-rates of different caching policies for the MovieLens dataset [Harper and Konstan, 2015]

Experimental Results Figure 11 compares the performance of different policies in terms of the hit rates and fetch rates. The average values of the key performance indicators are shown in Table 2. From the plots and the table, we see that the `LeadCache` policy achieves the highest hit rate among all other policies, which is about 32% more than that of the Heuristic policy proposed by Bhattacharjee et al. [2020]. On the other hand, it incurs more file fetches compared to only the heuristic policy proposed by Bhattacharjee et al. [2020]. Figure 12 gives a joint plot of the hit rate and the fetch rate of different policies. It is clear from the plots that the `LeadCache` policy robustly learns the file request patterns and caches them on the caches near-optimally.