Online Coded Caching

Ramtin Pedarsani
UC Berkeley
Berkeley, USA
ramtin@eecs.berkeley.edu

Mohammad Ali Maddah-Ali
Bell Labs, Alcatel-Lucent
New Jersey, USA
mohammadali.maddah-ali@alcatel-lucent.com

Urs Niesen
Bell Labs, Alcatel-Lucent
New Jersey, USA
urs.niesen@alcatel-lucent.com

***Abstract*—We consider a basic content distribution scenario consisting of a single origin server connected through a shared bottleneck link to a number of users each equipped with a cache of finite memory. The users issue a sequence of content requests from a set of popular files, and the goal is to operate the caches as well as the server such that these requests are satisfied with the minimum number of bits sent over the shared link. Assuming a basic Markov model for renewing the set of popular files, we characterize approximately the optimal long-term average rate of the shared link. We further prove that the optimal online scheme has approximately the same performance as the optimal offline scheme, in which the cache contents can be updated based on the entire set of popular files before each new request. To support these theoretical results, we propose an online coded caching scheme termed *coded least-recently sent* (LRS) and simulate it for a demand time series derived from the dataset made available by Netflix for the Netflix Prize. For this time series, we show that the proposed coded LRS algorithm significantly outperforms the popular least-recently used (LRU) caching algorithm.**

I. Introduction

The demand for video streaming services such as those offered by Netflix, YouTube, Amazon, and others, is growing rapidly. This places a significant burden on networks. One way to mitigate this burden is to place memories into the network that can be used to cache files that users may request. In this paper, we investigate how to optimally use these caches. In particular, we are interested in *online* algorithms for this problem, in which the operations of the caches have to be performed on the fly and without knowledge of future requests.

The online caching problem (also known as the paging problem in the context of virtual memory systems) has a long history, dating back to the work by Belady in 1966 [1]. This problem has been investigated both for systems with a single cache [1]–[11] as well as for systems with multiple distributed caches [12]–[14]. One solution to the caching problem that is popular in practice and for which strong optimality guarantees can be proved [2], [5], [8]–[11] is the *least-recently used* (LRU) eviction policy. In LRU, each cache is continuously updated to hold the most recently requested files, allowing it to exploit the temporal locality of content requests.

The figure of merit adopted by the papers mentioned so far is the cache-miss rate (or page-fault rate in the context of the paging problem), sometimes weighted by the file size. This cache-miss rate is used as a proxy for the network load. For systems with a *single* cache, the weighted cache-miss rate and the network load are indeed proportional to each other, and hence minimizing the former also minimizes the latter. However, this proportionality no longer holds for systems with *multiple* caches. For such systems with multiple caches, a fundamentally different so-called *coded caching* approach is required. This coded caching approach has been recently introduced in [15]–[17] for the *offline* caching problem.

In this paper, we investigate *online* coded caching, focusing on a basic content distribution scenario consisting of a single origin server connected through a shared (bottleneck) link to a number of users each equipped with a cache of finite memory (see Fig. 1). The users issue a sequence of content requests from a set of popular files, and the goal is to operate the caches as well as the server such as to satisfy these requests with the minimum number of bits sent over the shared link. The set of popular files evolve according to a Markov model and users select their demand uniformly from this set.

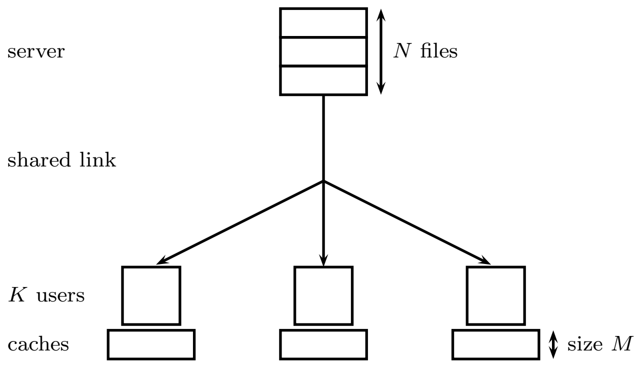

Fig. 1. Caching system considered in this paper. A server containing N files of size F bits each is connected through a shared link to K users each with a cache of size MF bits. In the figure, $N = K = 3$ and $M = 1$.

We approximately characterize the optimal long-term average rate of the shared link for this setting. We show further that the optimal online scheme performs approximately the same as the optimal offline scheme. This is perhaps surprising, since in the offline scheme caches are allowed to be updated in an offline fashion each time the set of popular files changes, whereas in the online scheme caches are updated in an online fashion based solely on the limited observations they have

This work was done when R. Pedarsani was a summer intern at Bell Labs, Alcatel-Lucent.
The work of Urs Niesen was supported in part by AFOSR under grant FA9550-09-1-0317.

through the sequence of requests.[1]

To evaluate the gain of coded caching in practical scenarios, we propose an online coded caching scheme termed *coded least-recently sent* (LRS) and simulate it on a demand time series derived from the dataset made available by Netflix for the Netflix Prize. For this time series, we show that the proposed coded LRS algorithm significantly outperforms the baseline LRU algorithm.

The remainder of this paper is organized as follows. Section II provides background information on coded caching. Section III formally introduces the problem setting. Section IV contains the main results. Due to space constraints, only proof sketches are proved; detailed proofs can be found in the full version of the paper [18].

II. Background on Coded Caching

Coded caching, recently introduced in [15]–[17], is a novel approach to the distributed caching problem. It can achieve a significant reduction in network load by creating and exploiting coded multicasting opportunities between users with different demands. We make essential use of the offline coded caching scheme from [16] in the present paper. Therefore, we now briefly overview that algorithm.

The setting in [16] is the offline version of the one depicted in Fig. 1 in Section I. In particular, a single origin server is connected to K users through a shared link. There is a fixed set of $N \geq K$ files of length F bits, and each user has a memory of size MF bits with $M \leq N$. The cache memories are prefetched in an *offline* fashion during a placement phase (during a period of low network load, say the early morning) so as to minimize the peak load $R(M, N, K)F$ over the shared link during a later delivery phase (say in the evening) during which each user requests a single file. We refer to the normalized peak load $R(M, N, K)$ as the peak rate.

The offline coded caching scheme proposed in [16] achieves, for F large enough, a peak rate of

$$R(M,N,K) \triangleq K\cdot(1-M/N)\cdot\frac{N}{KM}\big(1-(1-M/N)^K\big), \quad (1)$$

which is shown to be within a constant factor of the optimum.

In the placement phase of the algorithm in [16], each user caches a random subset of MF/N bits of each of the N files. In the delivery phase, the server sends an appropriate linear combination of those bits over the shared link to enable all users to recover the requested files. This is illustrated with the following toy example.

Example 1 (*Decentralized Caching Scheme [16]*). Consider the caching problem with $N = 2$ files say A, B and $K = 2$ users, each with a cache of size MF. In the placement phase, each user caches $MF/2$ bits of each file independently at random, satisfying the memory constraint. We partition

$$A = (A_\emptyset, A_1, A_2, A_{1,2}),$$

where $A_\mathcal{S}$ denotes the bits of file A that are stored at the users in the set $\mathcal{S}$, and similarly for B. For large F, the size of the subfile $A_\mathcal{S}$ tends to $(M/2)^{|\mathcal{S}|}(1-M/2)^{2-|\mathcal{S}|}F$ bits by the law of large numbers.

In the delivery phase, suppose that users one and two request files A and B, respectively. User one already has access to file parts A_1 and $A_{1,2}$ of its requested file, and needs $A_\emptyset$ and A_2, which are not cached its memory. Similarly user two already has access to parts B_2 and $B_{1,2}$ of its requested file and needs $B_\emptyset$ and B_1. The server can then satisfy these user requests by sending $A_\emptyset$, $B_\emptyset$, and $A_2 \oplus B_1$ over the shared link, where $\oplus$ denotes the XOR operation applied element-wise to to A_2 and B_1 treated as vectors of bits.

Observe that user one has B_1 stored in its cache. From this and the output $A_2 \oplus B_1$ of the shared link, user one can recover the desired file part A_2. Similarly, user two has A_2 stored in its cache and can use this to recover the desired file part B_1 from the output $A_2 \oplus B_1$ of the shared link. Thus, using the contents of their caches and the outputs of the shared link, both users can recover all the required file parts.

The rate over the shared link is

$$(M/2)(1-M/2) + 2(1-M/2)^2 = R(M, 2, 2),$$

with $R(M, N, K)$ is defined in (1). The delivery phase is explained here for a specific set of user requests, however this rate is achievable for all other possible requests as well. $\Diamond$

The main gain from using this scheme derives from the coded multicasting opportunities between users with different demands. These coded multicasting opportunities are created in the placement phase and are exploited in the delivery phase. As the size M of the cache memories increases, this coded multicasting gain increases as well. This gain, called the *global* gain in [15], [16], is captured by the factor

$$\frac{N}{KM}\big(1-(1-M/N)^K\big)$$

in $R(M, N, K)$. There is a second, *local*, gain deriving from having part of the requested file available in a user's local cache. This local gain is captured by the factor

$$(1 - M/N)$$

in $R(M, N, K)$. As is shown in [15]–[17], this local gain is usually less significant than the global coded multicasting gain.

III. Problem Setting

We consider a content distribution system with a server connected through a shared, error-free link to K users as illustrated in Fig. 1 in Section I. At time $t \in \mathbb{N}$, each user k requests a file $d_t(k)$ from a time-varying set $\mathcal{N}_t$ of popular files with cardinality $N \geq K$. The K user's requests, collectively denoted by the vector d_t, are chosen uniformly at random without replacement from $\mathcal{N}_t$. Each file has size F bits, and each user is equipped with a cache memory of size MF bits.

The content distribution system operates as follows. At the beginning of each time slot t, the users reveal their requests d_t to the server. The server, having access to the database of all

[1]This definition of an offline caching scheme differs from the definition adopted by other papers in the caching literature such as [2]. In those papers, an offline scheme has noncausal knowledge of the entire sequence of demands.

the files in the system, responds by transmitting a message of size R_tF bits over the shared link. Using their cache contents and the message received over the shared link, each user k aims to reconstruct its requested file $d_t(k)$.

The goal is to design the actions of the users and the server to minimize the long-term average rate

$$\bar{R} \triangleq \limsup_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}(R_t),$$

while satisfying the memory and reconstruction constraints. Observe that the rate $\bar{R}$ is the long-term average load $\bar{R}F$ over the shared link normalized by the file size F. In order to obtain a rate $\bar{R}$ independent of the file size and to simplify the analysis, we allow the file size F to be as large as needed.

In this paper, we are interested in *online* caching schemes, which place additional restrictions on the actions of the server and the caches. In such online schemes, the cache content at user k at the beginning of time slot t is a function of the cache content at the same user at the previous time $t-1$, the output of the shared link at time $t-1$, and the requests $d_1, d_2, \dots, d_{t-1}$ up until time $t-1$. In particular, the cache content may *not* be a function of the outputs of the shared link at times prior to $t-1$. Furthermore, for an online scheme, the message sent by the server over the shared link at time t is a function of only the demands d_t and the cache contents of the users at that time. We denote by $\bar{R}^\star$ the long-term average rate over the shared link of the optimal online caching scheme.

Example 2 (*LRU*)**.** A popular online caching scheme is *least-recently used* (LRU). In this scheme, each user caches M entire files. When a user requests a file that is already contained in its cache, that request can be served out of the cache without any communication from the server. When a user requests a file that is not contained in the cache, the server sends the entire file over the link. The user then evicts the least-recently requested (or used) file from its cache and replaces it with the newly requested one.

Observe that this is a valid online caching strategy. Indeed, the content of a user's cache at the beginning of time slot t is a function of the cache content at time $t-1$, the output of the shared link at time $t-1$, and the past requests (in order to determine which file was least-recently used). Moreover, the message sent by the server at time t is only a function of the demands d_t and the cache contents of the users at that time (in order to decide if a file needs to be transmitted at all). We will adopt LRU as the baseline scheme throughout this paper. $\Diamond$

We next provide a formal description of the dynamics of the set of popular files $\mathcal{N}_t$. The initial set $\mathcal{N}_1$ consists of N distinct files. The set $\mathcal{N}_t$ evolves from the set $\mathcal{N}_{t-1}$ using an arrival/departure process. With probability $1-p$, there is no arrival and $\mathcal{N}_t = \mathcal{N}_{t-1}$. With probability p, there is an arrival, and the set $\mathcal{N}_t$ is constructed by choosing a file uniformly at random from $\mathcal{N}_{t-1}$ and replacing it with a new, so far unseen, file. Note that this guarantees that $|\mathcal{N}_t| = N$ for all t.

Example 3 (*Popular-File Dynamics*)**.** Consider a toy system with $N = 2$ popular files. A possible evolution of the set $\mathcal{N}_t$ of popular files is as follows.

$t = 1$: The initial set of popular files is $\mathcal{N}_1 = \{B, C\}$.
$t = 2$: There is an arrival. The file C is randomly chosen and replaced with file D, so that $\mathcal{N}_2 = \{B, D\}$.
$t = 3$: There is no arrival, and $\mathcal{N}_3 = \mathcal{N}_2 = \{B, D\}$.

$\Diamond$

IV. Main Results

We start by introducing a new, online, coded caching algorithm in Section IV-A. A simplified variant of this algorithm is shown in Section IV-B to have performance close to the optimal online caching scheme. Section IV-C provides simulation results comparing the proposed coded caching algorithm to the baseline LRU scheme for an empirical demand time series derived from the Netflix Prize database.

A. An Online, Coded Caching Algorithm

We now introduce an online version of the caching algorithm in [16], which we term *coded least-recently sent* (LRS). The coded LRS algorithm is presented formally in the listing Algorithm 1. The statement of the algorithm uses the shorthand $[K]$ for $\{1, 2, \dots, K\}$.

Algorithm 1 The coded LRS caching algorithm for time t.

```
1: procedure DELIVERY
2:     for $s = K, K-1, \dots, 1$ do
3:         for $\mathcal{S} \subset [K] : |\mathcal{S}| = s$ do
4:             Server sends $\oplus_{k\in\mathcal{S}} V_{\mathcal{S}\setminus\{k\}}(k)$
5:         end for
6:     end for
7: end procedure
8: procedure CACHE UPDATE
9:     for $k, k' \in [K]$ do
10:        if $d_t(k')$ is not partially cached at user $k$ then
11:            User $k$ replaces the least recently sent file in its cache with a random subset of $\frac{MF}{N'}$ bits of file $d_t(k')$
12:        end if
13:    end for
14: end procedure
```

Algorithm 1 consists of a delivery procedure and a cache update procedure. The delivery procedure is formally similar to the delivery procedure of the decentralized caching algorithm in [16]. $V_{\mathcal{S}}(k)$ denotes the bits of the file $d_t(k)$ requested by user k cached exclusively at users in $\mathcal{S}$. In other words, a bit of file $d_t(k)$ is in $V_{\mathcal{S}}(k)$ if it is present in the cache of every user in $\mathcal{S}$ and if it is absent from the cache of every user outside $\mathcal{S}$. The XOR operation $\oplus$ in Line 4 is to be understood as being applied element-wise to $V_{\mathcal{S}}(k)$ treated as a vector of bits. If those vectors do not have the same length, they are assumed to be zero padded for the purposes of the XOR. We thus see that the delivery procedure of the coded LRS algorithm consists of sending one linear file combination for each subset $\mathcal{S}$ of users.

It is worth pointing out that, whenever a requested file is not cached at any user, then $V_\emptyset(k)$ is equal to the entire requested file, and hence when $\mathcal{S}=\{k\}$ in Line 3 the delivery procedure sends in this case the entire requested file uncoded over the shared link.

Consider next the cache update procedure. In each time slot t, the users maintain a list of $N' \triangleq \alpha N$ partially cached files for some $\alpha \geq 1$. The parameter α can be chosen to optimize the caching performance; a simple and reasonable choice is $\alpha \in (1, 2]$. At time t, after the delivery procedure is executed, the caches are updated as follows. If a requested file $d_t(k')$ of *any* user k' is not currently partially cached, *all* users evict the least-recently used file and replace it with MF/N' randomly chosen bits from file $d_t(k')$. This is feasible since the uncached file $d_t(k')$ was sent uncoded over the shared link during the delivery procedure. This update procedure guarantees that the number of partially cached files remains N', and that the same files (but not necessarily the same bits) are partially cached at each of the K users.

We illustrate the proposed coded LRS algorithm with a small toy example (applications to more realistic scenarios are analyzed in Section IV-C). This example also illustrates that the rate of the proposed scheme can be related to the rate $R(M, N, K)$ defined in (1) of the decentralized caching algorithm from [16].

Example 4 (*Coded LRS*)**.** We consider again a system with $N = 2$ popular files and assume the same popular-file dynamics as in Example 3 in Section III. Assume there are $K = 2$ users with a cache memory of $M = 1$. Let $\alpha = 3/2$ so that each user caches $1/3$ of the bits of $N' = \alpha N = 3$ files. We assume that initially each user partially caches the files $\{A, B, C\}$.

$t = 1$: The set of popular files is $\mathcal{N}_1 = \{B, C\}$. Assume the users request $d_1 = (B, C)$. Both of the requested files are partially cached at the users. In the delivery procedure, the server sends $B_\emptyset$, $C_\emptyset$, and $B_2 \oplus C_1$. For F large enough, so that each of these file parts has close to expected size (as discussed in Example 1 in Section II), this results in a rate of

$$(M/N')(1 - M/N') + 2(1 - M/N')^2 = R(M, N', 2) = 10/9.$$

with $R(M, N, K)$ as defined in (1). Since all the requested files are already partially cached, the set of cached files stays the same in the cache update procedure, and each user still partially caches $\{A, B, C\}$.

$t = 2$: The set of popular files changes to $\mathcal{N}_2 = \{B, D\}$. Assume the users request $d_2 = (B, D)$. Here, file B is partially cached at the users but file D is not. The server sends $B_\emptyset$, $D_\emptyset$, and $B_2 \oplus D_1$. Since D is not cached at any of the users, we have in this case that $D_\emptyset = D$ and $D_1 = \emptyset$. Hence, the transmission of the server is equivalently $B_\emptyset$, D and B_2. This results in a rate of

$$(M/N')(1 - M/N') + (1 - M/N')^2 + 1 = R(M, N', 1) + 1 = 15/9.$$

The least-recently sent file A is evicted from each cache and replaced by a random third of the file D. The new set of partially cached files is $\{B, C, D\}$.

$t = 3$: The set of popular files stays $\mathcal{N}_3 = \{B, D\}$. Assume the users request $d_3 = (D, B)$, both of which are now partially cached at the users. The server now sends $B_\emptyset$, $D_\emptyset$, and $B_2 \oplus D_1$. Unlike before, D_1 is now no longer empty, and the resulting rate is

$$R(M, N', 2) = 10/9$$

as calculated before. The set of partially cached files stays the same, namely $\{B, C, D\}$.

$\Diamond$

It is worth comparing the proposed coded LRS algorithm with the well-known LRU algorithm described in Example 2 in Section III. Both of them are online algorithms. However, there are three key differences. First, coded LRS uses a coded delivery procedure whereas the transmissions in LRU are uncoded. Second, coded LRS caches many (N') partial files whereas LRU caches fewer (M) whole files. Third, coded LRS uses a least-recently sent eviction rule, taking into account the files requested by all users jointly, compared to the least-recently used eviction rule, taking into account only the files requested by every user individually. The impact of these differences will be explored in more detail later.

B. Theoretical Results

The main result of this paper is the following theorem for the setting described in Section III with N files and K users each with a cache of size M.

Theorem 1. *The long-term average rate $\bar{R}^\star$ of the optimal online caching scheme satisfies*

$$\frac{1}{12} R(M, N, K) \leq \bar{R}^\star \leq 2R(M, N, K) + 6$$

with $R(M, N, K)$ as defined in (1).

A proof sketch for Theorem 1 is presented in Section V. The upper bound in Theorem 1 results from the analysis of a simplified version of the proposed coded LRS caching scheme, showing that this algorithm is approximately optimal. For the lower bound, we use the rate of the optimal offline scheme, whose rate is approximately $R(M, N, K)$.

The theorem thus implies that the rate of the optimal online caching scheme is approximately the same as the rate of the optimal offline scheme. Recall that in an offline scheme, the cache memories are given access to the entire set of popular files each time it changes. Moreover, these cache updates are performed offline, meaning that the data transfer needed to update and maintain the caches is not counted towards the load of the shared link. In contrast, in the online scenario, caches are updated based on the limited observations they have through the sequence of demands. Moreover, the cache updates are performed through the same shared link, and therefore affect the average rate. Theorem 1 thus indicates that these significant restrictions for the online problem have only a small effect on the rate compared to the offline scheme.

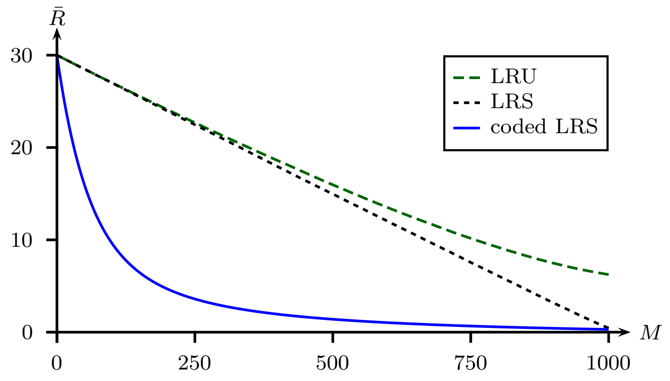

Fig. 2. Performance of various caching approaches for a system with $N = 1000$ popular files, $K = 30$ users, and arrival probability $p = 0.1$. The figure plots the long-term average rate $\bar{R}$ over the shared link as a function of cache memory size M for LRU (dashed green), uncoded LRS (dashed black), and the proposed coded LRS (solid blue).

We now compare the proposed coded LRS scheme to the baseline LRU scheme. The performances of these two schemes are shown in Fig. 2 for a system with $N = 1000$ popular files, $K = 30$ users, and arrival probability $p = 0.1$. As is visible from the figure, coded LRS provides significant gains over LRU both for small and large memory sizes. For example, for $M = 250$ (meaning that the cache is large enough to hold $1/4$ of the popular files), LRU results in a rate of 22.7 (meaning that we need to send the equivalent of 22.7 files over the shared link on average), whereas coded LRS results in a rate of 3.6. Similarly, for $M = 1000$, LRU results in a rate of 6.2, whereas coded LRS results in a rate of 0.3.

As mentioned in Section IV-A, the three main differences between coded LRS and LRU are coded delivery, partial caching, and LRS eviction. To get a sense of the impact of these three differences, Fig. 2 also depicts the performance of the uncoded LRS scheme. In this scheme, M whole files are cached and uncoded delivery is used (as in LRU), however the LRS eviction rule is used (unlike in LRU).

Comparing (uncoded) LRS to LRU, we see that the two schemes perform quite similarly for small and moderate values of M, say $0 \leq M \leq 750$. For large values of M, say $750 < M \leq 1000$, LRS provides a significant improvement over LRU.

Comparing uncoded LRS to coded LRS, we see that only when M is very close to the number of popular files N are the performances of the schemes similar. For all other values, coded LRS significantly outperforms uncoded LRS. This implies that, except for large values of M, the main gain of the coded LRS scheme derives from the partial caching of many files and from the coded delivery.

C. Empirical Results

To validate the theoretical analysis in Section IV-B as well as our model for the evolution of popular files, we now evaluate the performance of the proposed coded LRS and the baseline LRU schemes for a real-life time series of demands. This time series is derived from the dataset made available by Netflix for the Netflix Prize as follows. Each entry in the Netflix dataset consists of a user ID, a movie ID, a time stamp, and a rating that the user gave to the movie at the specified time. We would like to use the time of rating a movie as a proxy for the time of viewing that movie.

This approach is problematic for old movies, which users may rate long after they have seen them. However, it is reasonable to expect that the rating time is close to the viewing time for recently released movies. To ensure that this is the case, we selected all user ratings in the database from the year 2005 (the last full year for which ratings are available), and kept only those that are for movies released in either 2004 or 2005. The resulting filtered time series contains about 10^7 user ratings for 1948 unique movies.

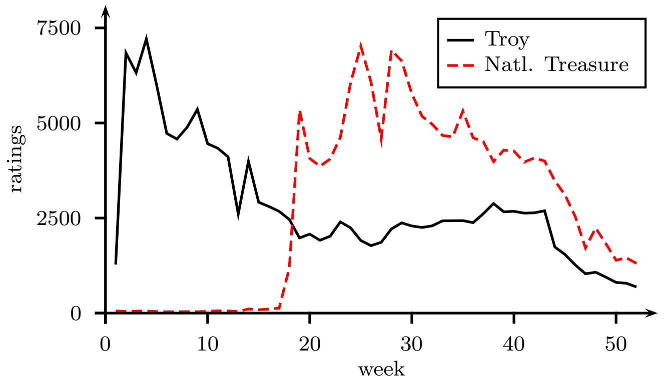

Fig. 3. Number of ratings in the Netflix database for two movies (“Troy” and “National Treasure”) as a function of week in 2005.

To validate this approach, Fig. 3 plots the number of ratings for the two most-rated movies (“Troy” and “National Treasure”) as a function of time measured in weeks. The movie “Troy” was released on DVD on January 4, 2005 (at which time it was likely also available on Netflix), corresponding to week 1. The movie “National Treasure” was released on DVD on May 3, 2005, corresponding to week 18. As can be seen from Fig. 3, the number of ratings increases strongly on the DVD release week, stays relatively high for a number of weeks, and then drops. This suggests that the rating time is indeed a valid proxy for the viewing time when applied to recently released movies. It also suggests that the model of time-varying popular files described in Section III is a reasonable model for the viewing behavior of users.

Fig. 4 compares the performance of the proposed coded LRS scheme to the baseline LRU scheme for the Netflix demand time series for a system with $K = 30$ caches (each here corresponding to many users that are attached to it). The figure shows that coded LRS again significantly outperforms

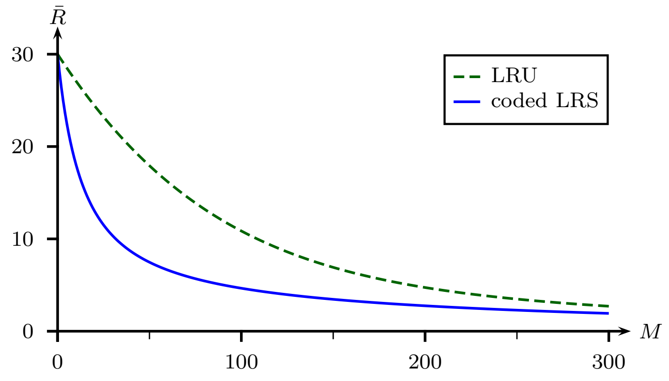

Fig. 4. Performance of various caching approaches for the Netflix demand time series and a system with $K = 30$ caches. The figure plots the long-term average rate $\bar{R}$ over the shared link as a function of cache memory size M for LRU (dashed green) and the proposed coded LRS (solid blue).

LRU. In particular, for a cache size of $M = 100$, LRU achieves a rate of 10.9 compared to a rate of 4.7 for coded LRS.

V. Proof Sketch for Theorem 1

For the upper bound on $\bar{R}^\star$, we analyze a simplified version of the proposed coded LRS scheme referred to as *coded random eviction*. In coded random eviction, if Y_t files are requested by users at time t that are not currently partially cached, then Y_t of the cached files are randomly chosen, evicted from *all* cache memories, and replaced with MF/N' randomly chosen bits from each of the Y_t newly requested files. This guarantees that the same collection of files is partially cached at each user. The remainder of the algorithm is the same as coded LRS as listed in Algorithm 1. In particular the coded random-eviction algorithm partially caches $N' = \alpha N$ files with $\alpha = 1.4$ at each user, where N is the cardinality of the set of popular files $\mathcal{N}_t$.

Recall that, according to the system model described in Section III, at each time slot t, the users request K randomly chosen files from the set of popular files $\mathcal{N}_t$ without replacement. At any time t, not all files in $\mathcal{N}_t$ may be partially cached at the users. As in the previous paragraph, we denote by Y_t the (random) number of uncached files requested by the users at time t. Note that Y_t takes value in set $\{0, 1, \dots, K\}$.

The delivery procedure in Algorithm 1 transmits these Y_t files uncoded over the shared link. To send the remaining $K - Y_t$ files that are partially cached at the users, the delivery procedure of Algorithm 1 uses coding. This coded part requires a rate of $R(M, \alpha N, K - Y_t)$ as described in Section II, and with $R(M, N, K)$ as defined in (1). Thus, the rate R_t over the shared link at time t is

$$R_t = R(M, \alpha N, K - Y_t) + Y_t \leq R(M, \alpha N, K) + Y_t.$$

The long-term average rate $\bar{R}$ of coded random eviction is therefore upper bounded by

$$\bar{R} \leq R(M, \alpha N, K) + \limsup_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}(Y_t). \qquad (2)$$

We show that the first term in (2) is approximately $R(M, N, K)$. Using a somewhat intricate analysis of the hidden Markov process Y_t, we further show that the second term is upper bounded by a constant. This second upper bound is perhaps surprising, since Y_t itself can take any value up to K and is hence not upper bounded by a constant independent of the problem parameters. The details of the remainder of the argument are omitted and can be found in [18].

For the lower bound, we consider the offline scenario in which all cache memories are fully aware of the set of popular files $\mathcal{N}_t$. In addition, at the beginning of each time slot t, before users decide on their requests, the caches are given full access to all the files in $\mathcal{N}_t$ to update their stored content at no cost. However, the cache memories are not aware of future requests. Clearly, the rate of the optimal scheme for this offline setting is a lower bound on the optimal rate for the online setting.

This offline problem is in fact equal to the prefetching problem investigated in [15]–[17], where it is shown that the instantaneous rate, and therefore also the long-term average rate, is lower bounded by $\frac{1}{12}R(M, N, K)$. Thus,

$$\bar{R}^\star \geq \frac{1}{12} R(M, N, K).$$

References

[1] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Syst. J.*, vol. 5, no. 2, pp. 78–101, 1966.

[2] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Comm. ACM*, vol. 28, pp. 202–208, Feb. 1985.

[3] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young, "Competitive paging algorithms," *J. Algorithms*, vol. 12, pp. 685–699, Dec. 1991.

[4] L. A. McGeoch and D. D. Sleator, "A strongly competitive randomized paging algorithm," *Algorithmica*, vol. 6, pp. 816–825, 1991.

[5] A. Borodin, S. Irani, P. Raghavan, and B. Schieber, "Competitive paging with locality of reference," in *Proc. ACM STOC*, pp. 249–259, 1991.

[6] A. R. Karlin, S. J. Phillips, and P. Raghavan, "Markov paging," in *Proc. IEEE FOCS*, pp. 208–217, Oct. 1992.

[7] P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms," pp. 193–206, Dec. 1997.

[8] M. Chrobak and J. Noga, "LRU is better than FIFO," in *Proc. ACM-SIAM SODA*, pp. 78–81, Jan. 1998.

[9] E. Torng, "A unified analysis of paging and caching," *Algorithmica*, vol. 20, pp. 175–200, Feb. 1998.

[10] E. Koutsoupias and C. H. Papadimitriou, "Beyond competitive analysis," *SIAM J. Comput.*, vol. 30, pp. 300–317, July 2000.

[11] S. Angelopoulos, R. Dorrigiv, and A. López-Ortiz, "On the separation and equivalence of paging strategies," in *Proc. ACM-SIAM SODA*, pp. 229–237, Jan. 2007.

[12] B. Awerbuch, Y. Bartal, and A. Fiat, "Distributed paging for general networks," in *Proc. ACM-SIAM SODA*, pp. 574–583, Jan. 1996.

[13] F. M. auf der Heide, B. Vöcking, and M. Westermann, "Caching in networks," in *Proc. ACM-SIAM SODA*, pp. 430–439, Jan. 2000.

[14] X. Li, C. G. Plaxton, M. Tiwari, and A. Venkataramani, "Online hierarchical cooperative caching," *Theory Comput. Syst.*, vol. 39, pp. 851–874, Nov. 2006.

[15] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *arXiv:1209.5807 [cs.IT]*, Sept. 2012. Submitted to IEEE Trans. Inf. Theory.

[16] M. A. Maddah-Ali and U. Niesen, "Decentralized coded caching attains order-optimal memory-rate tradeoff," *arXiv:1301.5848 [cs.IT]*, Jan. 2013.

[17] U. Niesen and M. A. Maddah-Ali, "Coded caching with nonuniform demands," *arXiv:1308.0178 [cs.IT]*, Aug. 2013.

[18] R. Pedarsani, M. A. Maddah-Ali, and U. Niesen, "Online coded caching," *arXiv:1311.3646 [cs.IT]*, Nov. 2013.