Coded Caching with Nonuniform Demands

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

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

***Abstract*—We consider a network consisting of a file server connected through a shared link to a number of users, each equipped with a cache. Knowing the popularity distribution of the files in the database, the goal is to optimally populate the caches such as to minimize the expected load of the shared link. For a single user, it is well known that caching the most popular files is optimal in this setting. However we show here that this is no longer the case for multiple users. Indeed, caching only the most popular files can be highly suboptimal. Instead, a fundamentally different approach is needed, in which the cache contents are used as side information for coded communication over the shared link. We propose such a coded caching scheme and prove that it is close to optimal.**

I. Introduction

Caching or prefetching is a technique to reduce network loads by storing part of the content to be distributed at or near end users. In this paper, we design near-optimal caching strategies for a basic network scenario consisting of one server connected through a shared, error-free link to K users as illustrated in Fig. 1. The server has access to a database of N files each of size F bits. Each user has an isolated cache memory of size MF bits.

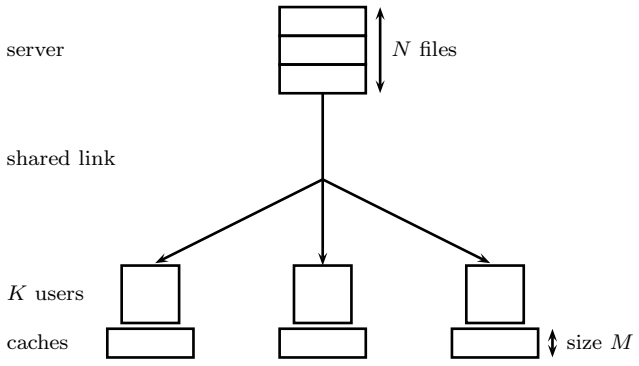

Fig. 1. The caching problem with $N = 3$ files and $K = 3$ users.

During times of low traffic demand (say early in the morning) or when connected to a network with large available bandwidth (say a mobile handset connected to WiFi), users can save some of the content in the database to their local caches. During a later time, when a user requests one of the files, the local cache can be used to reduce network load. More formally, the system operates in two distinct phases: a *placement phase* and a *delivery phase*. In the placement phase, each user can save part of the N files in its cache memory. In the delivery phase, each user randomly requests one of the files in the database independently of the other users and with identical distribution. We denote by p_n the probability[1] that user k requests file n. The server is informed of these requests and proceeds by transmitting a message over the shared link. Using the content of its cache and the message received over the shared link in the delivery phase, each user aims to reconstruct its requested file. The placement and delivery phases of the system should be designed to minimize the load of the shared link subject to the memory constraint in the placement phase and the reconstruction constraint in the delivery phase.

Designing and analyzing caching systems for such (and more complicated) networks has a long history, see for example [1]–[6]. The impact of specific file popularities $p_1, p_2, \cdots, p_N$ (such as Zipf or other heavy-tail distributions) on the performance of caching has been analyzed in [7]–[9], among others. These papers consider *uncoded* caching. For the basic network scenario considered here with only a *single* user ($K = 1$), it turns out that such uncoded caching strategies are optimal. Indeed, it is well known that the optimal strategy in this case is the least-frequently used (LFU) caching scheme, in which each user caches the M most popular files in its cache.[2]

In this paper, we show that this intuition for a single user does not carry over to *multiple* users ($K > 1$). In fact, LFU can be arbitrarily suboptimal in the multi-user setting. Instead, a fundamentally different approach to the caching problem is required. We propose the use of a *coded* caching scheme, recently introduced by the present authors in [10], [11]. These coded caching schemes work by carefully designing the placement phase so as to enable a simultaneous coded multicasting gain among the users, even if they request different files. These schemes were shown in [10], [11] to approximately minimize the *peak* load (i.e., for the worst-case user requests) of the shared link.

However, in many situations the file popularities $p_1, p_2, \ldots, p_N$ differ over many orders of magnitudes, and hence the *expected* load is more relevant than the

The work of Urs Niesen was supported in part by AFOSR under grant FA9550-09-1-0317.

[1]These probabilities can, for example, be estimated from past user requests.

[2]The name least-frequently used refers to the cache eviction policy, in which, when a new item is requested, the item that is least-frequently used is evicted from the cache. If the actual file popularities are known, as is assumed here, then this reduces to caching the M most popular files.

978-1-4799-3088-3/14/$31.00 ©2014 IEEE

peak load. In this paper, we describe how to deal with such situations. In particular, we propose a coded caching algorithm that approximately minimizes the expected load of the shared link and that is able to handle widely differing file popularities such as those arising from heavy-tail distributions. The proof of approximate optimality of the proposed scheme links the optimal expected rate to the optimal peak rate and is quite intricate. We apply the proposed algorithm to the file popularities of the Netflix video catalog and show that it can significantly outperform LFU. Due to space constraints we only present partial proofs here; detailed proofs can be found in the full version of the paper [12].

II. Background on Coded Caching

In [10], [11], we have shown that in a system with N files, each of size F bits, and K users, each with an isolated cache memory of size MF bits, a peak load of $R(M,N,K)F$ bits over the shared link is achievable with high probability for F large enough, where

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

We refer to the normalized (by F) peak load $R(M,N,K)$ as the *peak rate*. We now briefly describe how the peak rate (1) can be achieved; the discussion here follows [11].

In the placement phase, each user saves a random subset of MF/N bits of each file into its cache memory. These random subsets are chosen uniformly and independently for each user and file. Since there are a total of N files, this satisfies the memory constraint of MF bits. In the delivery phase, after the user's requests are revealed, the server delivers the requested files while maximally exploiting the side information available in each user's cache. This is done by coding several requested files together.

The placement and delivery procedures are formally stated in Algorithm 1.[3] In the statement of Algorithm 1, $[K]$ denotes the set $\{1,2,\dots,K\}$ and similarly for $[N]$. Furthermore, $V_{k,\mathcal{S}}$ denotes the vector of file bits that are requested by user k and that are available in the cache of every user in $\mathcal{S}$ and missing in the cache of every user outside $\mathcal{S}$. The following example from [11] illustrates the algorithm.

Example 1 (*Illustration of Algorithm 1*)**.** We consider the caching problem with $N=2$ files A and B and with $K=2$ users. In the placement phase of Algorithm 1, each user caches a random subset of $MF/2$ bits of each file. Any fixed bit of a file is thus cached by a fixed user with probability $M/2$.

Focusing on file A, we see that the placement procedure partitions this file into four subfiles

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

[3]The reader may have noticed that Algorithm 1 contains two possible delivery procedures; the server chooses whichever one is better.

Algorithm 1 Coded Caching Scheme from [11] achieving peak rate (1)

procedure PLACEMENT
  for $k\in[K], n\in[N]$ **do**
    User k caches a random $\frac{MF}{N}$-bit subset of file n
  end for
end procedure
procedure DELIVERY$(d_1,\dots,d_K)$
  for $s=K,K-1,\dots,1$ **do**
    for $\mathcal{S}\subset[K]:|\mathcal{S}|=s$ **do**
      Server sends $\oplus_{k\in\mathcal{S}}V_{k,\mathcal{S}\setminus\{k\}}$
    end for
  end for
end procedure
procedure DELIVERY'$(d_1,\dots,d_K)$
  for $n\in[N]$ **do**
    Server sends enough random linear combinations of bits in file n for all users requesting it to decode
  end for
end procedure

where $A_\mathcal{S}$ denotes the bits of file A that are stored in the cache memories of users in $\mathcal{S}$. E.g., $A_{1,2}$ are the bits of file A stored in the cache of users one and two, and A_2 are the bits of A stored exclusively in the cache of user two. For large enough file size F, the law of large numbers guarantees that

$$|A_\mathcal{S}|/F\approx(M/2)^{|\mathcal{S}|}(1-M/2)^{2-|\mathcal{S}|}$$

with high probability and similarly for file B.

Consider next the delivery procedure.[4] Assume users one and two request files A and B, respectively. In this case, $V_{1,2}=A_2$, $V_{2,1}=B_1$, $V_{1,\emptyset}=A_\emptyset$, and $V_{2,\emptyset}=B_\emptyset$. Hence, the server sends $A_2\oplus B_1$, $A_\emptyset$, and $B_\emptyset$ over the shared link.

$A_\emptyset$ and $B_\emptyset$ are the file parts that are not cached at any of the users, and hence they obviously have to be sent from the server for successful recovery of the requested files. The more interesting transmission is $A_2\oplus B_1$. Observe that user one has B_1 stored in its cache memory. Hence, user one can solve for the desired file part A_2 from the received message $A_2\oplus B_1$. Similarly, user two has A_2 stored in its cache memory. Hence, it can solve for the desired file part B_1 from the received message $A_2\oplus B_1$. In other words, the transmission $A_2\oplus B_1$ is simultaneously useful for both users. Thus, even though the two users request different files, the server can successfully multicast useful information to both of them. The rate of the messages sent by the server is

$$(M/2)(1-M/2)+2(1-M/2)^2 = 2\cdot(1-M/2)\cdot\frac{1}{M}\big(1-(1-M/2)^2\big).$$

While the analysis here was for file requests (A,B), the same arguments hold for all other possible file requests as

[4]It can be checked that in this setting the first delivery procedure is better and will hence be used by the server.

well. In each case, the side information in the caches is used to create coded multicasting opportunities for users with (possibly) different demands. In other words, the content placement is performed such as to enable coded multicasting opportunities *simultaneously* for all possible demands. The above rate holds therefore for every possible user demands, i.e., it is an achievable peak rate for the caching problem. $\Diamond$

III. Theoretical Results

In [11], we prove that the rate $R(M,N,K)$ (defined in (1)) of the caching scheme reviewed in Section II is within a constant factor of the optimal *peak* rate. In this paper, we are instead interested in the *expected* rate. As a corollary to the results presented later in this section, we show that the same rate $R(M,N,K)$ is also within a constant factor of the optimal expected rate for files with uniform popularities. The question is how to achieve approximately optimal expected rate in the more realistic case of files with popularities varying over several orders of magnitude.

Before explaining the proposed algorithm for such cases, we first highlight two important features of Algorithm 1. The first feature is that for every possible user demands the delivery algorithm exploits coded multicasting opportunities among every subset of users. The second feature is the symmetry in the placement phase. It is this symmetry that permits to easily identify and quantify the coded multicasting opportunities (see also the discussion in Section V). These two features together are at the core of the approximate optimality of Algorithm 1 for uniform file popularities.

Consider now some of the options for nonuniform file popularities. One option is to choose a number, say $\hat{N}$, of the most popular files and apply the placement procedure of Algorithm 1 only to those files. In the delivery phase, the user requests selected from these $\hat{N}$ files are handled using the delivery procedure of Algorithm 1. For the remaining requests, the server simply transmits the entire files uncoded over the shared link. The parameter $\hat{N}$ can be optimized to minimize the expected rate of the scheme. The advantage of this scheme is that the symmetry of the content placement is preserved, and therefore, in the delivery phase, we can again identify and quantify the coded multicasting opportunities among the $\hat{N}$ files. The disadvantage is that the difference in the popularities among the $\hat{N}$ cached files is ignored. Since these files can have widely different popularities, it is wasteful to dedicate the same fraction of memory to each one of them. As a result, this approach does not perform well in general.

Another option is to dedicate a different amount of memory to each file in the placement phase. For example the amount of allocated memory could be proportional to the popularity of a file. While this option takes the different file popularities into account, it breaks the symmetry of the content placement. As a result, the delivery phase becomes difficult to analyze.

Here we propose an alternative solution having the advantages of both of theses approaches and can be proved to be approximately optimal. In the proposed scheme, we partition the files into groups with approximately uniform popularities. In the placement phase, each file within the same group is allocated the same amount of cache memory. However, files in different groups may have different memory allocations. In the delivery phase, the demands within each group are delivered using Algorithm 1. Since the symmetry within each group has been preserved, the delivery phase is analytically tractable. Moreover, since different groups have different memory allocations, we can use more memory for files with higher popularity.

We now describe the proposed scheme in detail. By relabeling the files, we can assume without loss of generality that $p_1 \geq p_2 \geq \cdots \geq p_N > 0$. We partition the N files

$$\mathcal{N} \triangleq \{1,2,\ldots,N\}$$

into L groups $\mathcal{N}_1,\mathcal{N}_2,\ldots,\mathcal{N}_L$. Denote by N_ℓ the size of group $\mathcal{N}_\ell$ so that $\sum_{\ell=1}^{L} N_\ell = N$.

In general, the choice of groups can be optimized to minimize the average rate. Here we introduce one option that is analytically tractable. Let $\mathcal{N}_1$ be the files $1,2,\ldots,N_1$ with N_1 such that $p_{N_1} \geq p_1/2$ and $p_{N_1+1} < p_1/2$. Thus, $\mathcal{N}_1$ are the most popular files and all files in this group have popularity differing by at most a factor two. Similarly, define $\mathcal{N}_2$ as the group of next most popular files, and so on. In general, for any two files n,n' in the same group $\mathcal{N}_\ell$ the file popularities p_n and $p_{n'}$ differ by at most a factor two. In other words, let n be the smallest number in $\mathcal{N}_\ell$. Then,

$$p_n \geq p_{n+N_\ell-1} \geq p_n/2 \qquad \text{and} \qquad p_{n+N_\ell} < p_n/2.$$

We say that the files $\mathcal{N}$ are maximally partitioned to within popularity factor of two into $\mathcal{N}_1,\mathcal{N}_2,\ldots,\mathcal{N}_L$.

For the placement phase, we allocate a fraction of the memory to each of the groups $\mathcal{N}_\ell$ of files. Denote by $M_\ell F$ the number of bits allocated to cache files in $\mathcal{N}_\ell$. M_ℓ must be chosen such that the total memory constraint is satisfied, i.e.,

$$\sum_{\ell=1}^{L} M_\ell = M.$$

Once the memory allocation is done, we proceed with the actual placement phase. In the placement phase, each user randomly selects $M_\ell F/N_\ell$ bits from each file in group $\mathcal{N}_\ell$ and stores them in its cache memory. With this, the total number of bits cached at each user is

$$\sum_{\ell=1}^{L} N_\ell \cdot \frac{M_\ell F}{N_\ell} = MF,$$

satisfying the memory constraint.

In the delivery phase, each user requests a file. Denote by $\mathcal{K}_\ell$ those users that request a file in the group $\mathcal{N}_\ell$ of files. Note that $\mathcal{K}_1,\mathcal{K}_2,\ldots,\mathcal{K}_L$ partitions the users into L groups. Denote by K_ℓ the cardinality of user group $\mathcal{K}_\ell$. Note that, since the groups $\mathcal{K}_\ell$ depend on the random choice of the user requests, the cardinalities K_ℓ are random variables, and we highlight this fact by writing them in sans-serif font.

The server uses the same delivery procedure as in Algorithm 1 L times, once for each group of users $\mathcal{K}_\ell$. The next

theorem analyzes the rate of this scheme for large file size F. This yields an upper bound on the optimal expected rate $R^\star(M,\mathcal{N},K)$ for the caching problem.

Theorem 1. *For N files $\mathcal{N}$ partitioned maximally to within popularity factor of two into L groups $\mathcal{N}_1,\mathcal{N}_2,\ldots,\mathcal{N}_L$ and K users each with normalized cache size M, we have*

$$R^\star(M,\mathcal{N},K) \le \min_{\{M_\ell\}:\sum_\ell M_\ell = M} \sum_{\ell=1}^{L} \mathbb{E}\big(R(M_\ell, N_\ell, \mathsf{K}_\ell)\big) \le \sum_{\ell=1}^{L} \mathbb{E}\big(R(M/L, N_\ell, \mathsf{K}_\ell)\big),$$

where $R(M,N,K)$ is defined in (1)*. We recall that N_ℓ denotes the size of file group $\mathcal{N}_\ell$ and that the random variable K_ℓ represents the number of users $\mathcal{K}_\ell$ requesting files from group $\mathcal{N}_\ell$. All expectations are with respect to K_ℓ.*

The first inequality in Theorem 1 upper bounds the optimal expected rate $R^\star(M,\mathcal{N},K)$ by the rate of the proposed scheme. Each term in the sum corresponds to the rate of serving the users in one of the subgroups $\mathcal{K}_\ell$, and the sum rate is minimized over the choice of memory allocation M_ℓ. The second inequality follows from the simple memory allocation $M_\ell = M/L$ for all ℓ. Note that, even if each group is allocated the same amount of memory M/L, the memory allocated to an individual file in group $\mathcal{N}_\ell$ is $M/(N_\ell L)$, which varies as a function of ℓ. In particular, if the files follow a heavy-tail distribution, then N_ℓ increases rapidly in ℓ, and the amount of memory allocated to each file decreases quickly in ℓ.

The next theorem establishes a lower bound on the optimal expected rate $R^\star(M,\mathcal{N},K)$ for the caching problem.

Theorem 2. *For N files $\mathcal{N}$ partitioned maximally to within popularity factor of two into L groups $\mathcal{N}_1,\mathcal{N}_2,\ldots,\mathcal{N}_L$ and K users each with normalized cache size M, we have*

$$R^\star(M,\mathcal{N},K) \ge \frac{1}{cL} \sum_{\ell=1}^{L} \mathbb{E}\big(R(M, N_\ell, \mathsf{K}_\ell)\big),$$

where c is a strictly positive universal[5] *constant and where $R(M,N,K)$ is defined in* (1)*.*

This theorem states that if each user has (normalized) cache size ML, rather than M, and we apply the proposed caching scheme, then the resulting expected rate is at most cL times larger than the expected rate of the optimal scheme for the original problem, in which each user has cache size M.

The proofs of Theorems 1 and 2 are presented in Appendices A and B, respectively. The proof of Theorem 1 analyzes the rate of the proposed scheme using the results from [11] for each subgroup of files and is straightforward. The proof of Theorem 2 links the optimal expected rate to the optimal peak rate and is quite intricate, involving a genie-based uniformization argument as well as a symmetrization argument.

[5]I.e., not depending on any of the problem parameters such as N, K, M, or the popularity distribution $p_1,\ldots,p_N$.

Theorems 1 and 2 together approximate the optimal memory-rate tradeoff $R^\star(M,\mathcal{N},K)$. They show that the proposed caching scheme achieves the optimal tradeoff to within a factor cL in the rate direction and to within a factor L in the memory direction. By the exponential scaling construction of $\mathcal{N}_\ell$, the parameter L is usually small, i.e., the factor L in the approximation gap is usually modest (see also the discussion in Section IV).

For the special case of uniform file popularities, we have $L = 1$. Hence Theorems 1 and 2 imply that the optimal expected rate $R^\star(M,\mathcal{N},K)$ satisfies

$$\frac{1}{c} R(M,N,K) \le R^\star(M,\mathcal{N},K) \le R(M,N,K),$$

showing that the peak and expected rates are approximately the same in this case. From the results in [11] this also implies that the expected rate of the scheme proposed here can be up to a factor $\Theta(K)$ smaller than LFU. Thus, we see that, while LFU minimizes the expected rate for a single user ($K = 1$), it can be significantly suboptimal for multiple users ($K > 1$).

Consider next the important special case of heavy-tail popularity distributions. In this case, there are several groups $\mathcal{N}_\ell$, and their sum popularities (i.e., the sum of the popularities of the files in $\mathcal{N}_\ell$) decay only slowly or not at all as a function of ℓ. The proposed coded caching scheme deals with this heavy tail by careful allocation of the cache memory among the different file groups. As a result, the proposed scheme is able to exploit both the fact that *individually* the file popularities differ over several orders of magnitude, but at the same time *collectively* each group of files may have high probability. We discuss the behavior of the proposed caching scheme for such heavy-tail distribution in detail in the next section.

IV. Empirical Results

We now compare the performance of the proposed coded caching scheme to the well-known LFU caching scheme. Recall that, for known file popularity, LFU caches the M most popular files in each user's cache (see the discussion in the Section I). We choose the file popularities p_n to be those of the $N = 10\,000$ most popular movies from the Netflix catalog. Following the approach in [13], we estimate p_n from the dataset made available by Netflix for the Netflix Prize. The file popularities p_n are shown in Fig. 2.

As can be seen from the figure, the popularities p_n exhibit a flat "head", consisting of the first 600 or so most popular files. This is followed by a power-law "tail" with exponent of approximately -2. This is in line with the behavior of other multimedia content [9], [14].

We start with the analysis of LFU. For LFU, the expected rate is equal to the expected number of users with a request outside the first M most popular files, i.e., $K\sum_{n=M+1}^{N} p_n$. This is depicted in Fig. 3 for $K = 300$ users and various values of cache size M. Increasing the cache size from M to $M+1$ decreases the rate of LFU over the shared link by Kp_{M+1}. From Fig. 2, we expect the rate to initially decay rather quickly with M until the end of the "head" of the file

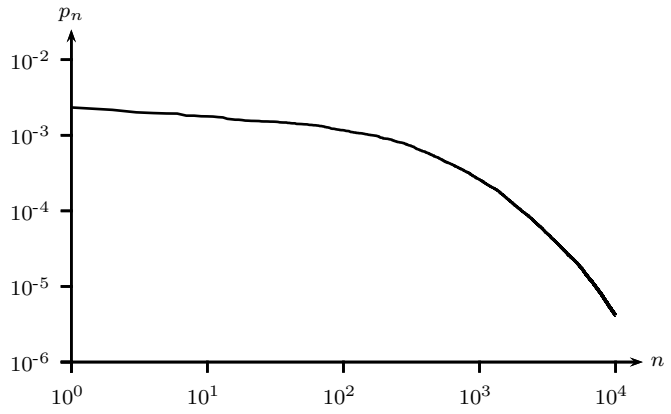

Fig. 2. File popularities p_n for the Netflix movie catalog. Observe the flattened "head" of the distribution for the first roughly 600 files followed by a power-law "tail" with exponent roughly -2.

popularity curve. Once M is big enough for the entire "head" to be cached, we expect further decreases in M to lead to diminishing returns. This behavior is indeed clearly visible in Fig. 3. We conclude that a reasonable choice of M for LFU is thus the size of the "head" of the popularity distribution, which in this case corresponds to about $M = 600$.

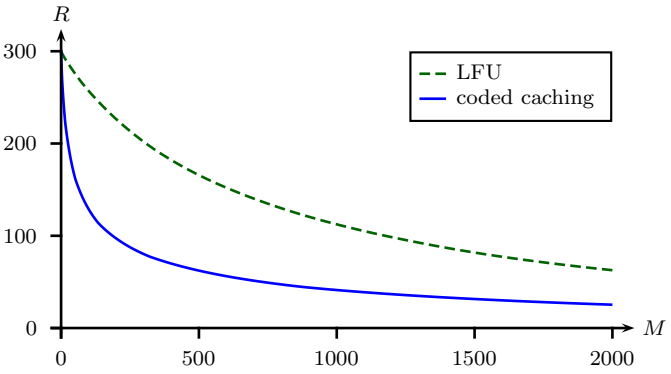

Fig. 3. Memory-rate tradeoff under Netflix file popularities for the baseline LFU scheme (dashed green line) and the proposed coded caching scheme (solid blue line). The number of users is $K = 300$, and the number of files is $N = 10\,000$.

We continue with the evaluation of the proposed coded caching scheme. From Theorem 1, its rate is

$$\min_{\{M_\ell\}:\sum_\ell M_\ell = M} \sum_{\ell=1}^{L} \mathbb{E}\big(R(M_\ell, N_\ell, \mathsf{K}_\ell)\big).$$

Note that $R(M, N, K)$ is a concave function of K. We can thus apply Jensen's inequality to upper bound the rate of the coded caching scheme by

$$\min_{\{M_\ell\}:\sum_\ell M_\ell = M} \sum_{\ell=1}^{L} R\big(M_\ell, N_\ell, \mathbb{E}(\mathsf{K}_\ell)\big),$$

where $\mathbb{E}(\mathsf{K}_\ell) = K \sum_{n\in\mathcal{N}_\ell} p_n$. We use this upper bound, depicted in Fig. 3, in the following.

Comparing the curves in Fig. 3, it is clear that the proposed coded caching scheme significantly improves upon the baseline LFU scheme. In particular, for a cache size of $M = 600F$ bits (where F is the file size), LFU results in $152F$ bits being sent over the shared link. In contrast, for the same value of M, the proposed scheme results in $56F$ bits being sent over the shared link—an improvement by more than a factor 2.7. Similarly, assume we want to operate at the same load of $152F$ bits of the shared link as achieved by LFU with $M = 600F$ bits. The proposed coded caching scheme can achieve the same load with only $M = 63F$ bits in cache memory—an improvement by a factor 9.5.

From Theorems 1 and 2, we also know that the proposed coded caching scheme achieves the optimal memory-rate tradeoff to within a factor cL in the rate direction and to within a factor L in the memory direction. In this example, $L = 10$.

V. Discussion and Conclusion

In this paper, we have analyzed the benefits of coding for caching systems with nonuniform file popularities. While (uncoded) LFU is optimal for such systems with a single user, we show here that coding is required for the optimal operation of caching systems with multiple users.

The caching problem analyzed in in this paper is related to the index coding problem [15], [16] (which was recently shown in [17] to be equivalent to the network coding problem [18]). In the index coding problem, a single server is connected via a broadcast channel to K users. The server has access to N files, and each of the users has access to a given subset of those files. For a given user demand, the aim is to minimize the number of transmissions from the server over the broadcast channel in order to satisfy all the demands.

From the above description, we see that for *fixed* cache content and for *fixed* user demands, the delivery phase of the caching problem as considered in this paper is in fact an index coding problem. Since there are N^K possible user demands, this means that the delivery phase actually consists of exponentially many parallel such index coding problems. To complicate matters, the index coding problem itself is known to be computationally intractable [19]. One contribution of this paper is hence to design the placement phase (i.e., choose the cache contents) such that each of these exponentially many index coding problems simultaneously have an efficient and analytical solution.

Appendix A
Proof of Theorem 1

We analyze the performance of the scheme described in Section III. The rate $R(M, N, K)$ defined in (1) is the peak rate achieved by Algorithm 1 for N files and K users each with a cache memory of MF bits. Thus, the rate $R(M, N, K)$ of this scheme is achievable for every possible user request. As a result, the expected value (over all requests) of the rate of Algorithm 1 is the same as the rate for any specific request.

Consider now a specific random request $(\mathsf{d}_1, \ldots, \mathsf{d}_K)$. This request results in the users being partitioned into subsets $\mathcal{K}_1, \ldots, \mathcal{K}_L$ with cardinalities $\mathsf{K}_1, \ldots, \mathsf{K}_L$.

Since the delivery algorithm treats each of the groups $\mathcal{K}_\ell$ independently, the rate for request $(\mathsf{d}_1, \ldots, \mathsf{d}_K)$ is

$$\sum_{\ell=1}^{L} R(M_\ell, N_\ell, \mathsf{K}_\ell).$$

Note that the only randomness in this expression is due to the random size K_ℓ of the group $\mathcal{K}_\ell$. Taking the expectation over all K_ℓ then yields the following upper bound on the expected rate $R^\star(M, N, K)$ of the optimal caching scheme:

$$R^\star(M, N, K) \le \sum_{\ell=1}^{L} \mathbb{E}\big(R(M_\ell, N_\ell, \mathsf{K}_\ell)\big).$$

We can minimize this upper bound by optimizing over the choice of memory allocation. This yields

$$R^\star(M, N, K) \le \min_{\{M_\ell\}: \sum_\ell M_\ell = M} \sum_{\ell=1}^{L} \mathbb{E}\big(R(M_\ell, N_\ell, \mathsf{K}_\ell)\big).$$

One particular choice of M_ℓ is M/L for each ℓ, which yields

$$R^\star(M, N, K) \le \sum_{\ell=1}^{L} \mathbb{E}\big(R(M/L, N_\ell, \mathsf{K}_\ell)\big).$$

Together, these two equations prove Theorem 1. ∎

Appendix B
Proof of Theorem 2

We will prove the equivalent statement that

$$\sum_{\ell=1}^{L} \mathbb{E}\big(R(M, N_\ell, \mathsf{K}_\ell)\big) \le cLR^\star(M, \mathcal{N}, K). \tag{2}$$

The proof of (2) is based on the following three claims.

Claim 1*:* We have

$$R(M, N_\ell, K_\ell) \le c_1 \bar{R}(M, \mathcal{N}_\ell, K_\ell),$$

where $\bar{R}(M, \mathcal{N}_\ell, K_\ell)$ denotes the expected rate of the optimal scheme for a system with K_ℓ users and files $\mathcal{N}_\ell$ with *uniform* popularity.

This claim upper bounds the peak rate of Algorithm 1 by the optimal expected rate for the caching problem with uniform file popularities. The proof of this claim is based on a symmetrization and a cut-set argument.

Claim 2*:* We have

$$\bar{R}(M, \mathcal{N}_\ell, K_\ell) \le c_2 R^\star(M, \mathcal{N}_\ell, K_\ell).$$

This claim upper bounds the optimal expected rate for a system with uniform file popularities by the optimal expected rate for a system with almost uniform file popularities (i.e., file popularities differing at most by a factor two). To prove this claim, we introduce a genie-based randomization argument to transform almost uniform to uniform file popularities.

Claim 3*:* We have

$$\mathbb{E}\big(R^\star(M, \mathcal{N}_\ell, \mathsf{K}_\ell)\big) \le R^\star(M, \mathcal{N}, K).$$

This claim simply states that if the server is only asked to handle the demands of users in $\mathcal{K}_\ell$, ignoring the demands of the remaining users, the rate of the optimal system decreases. As before, the expectation on the left-hand side is with respect to the random number of users K_ℓ.

Now, from Claims 1 and 2,

$$\sum_{\ell=1}^{L} \mathbb{E}\big(R(M, N_\ell, \mathsf{K}_\ell)\big) \le c_1 c_2 \sum_{\ell=1}^{L} \mathbb{E}\big(R^\star(M, \mathcal{N}_\ell, \mathsf{K}_\ell)\big).$$

Combined with Claim 3, this yields (2) with $c \triangleq c_1 c_2$. ∎

We omit the poofs of the three claims here and refer the reader to the full version of the paper [12] for the details.

References

[1] A. Leff, J. L. Wolf, and P. S. Yu, "Replication algorithms in a remote caching architecture," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, pp. 1185–1204, Nov. 1993.

[2] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, "Placement algorithms for hierarchical cooperative caching," in *Proc. ACM-SIAM SODA*, pp. 586–595, Jan. 1999.

[3] I. Baev and R. Rajaraman, "Approximation algorithms for data placement in arbitrary networks," in *Proc. ACM-SIAM SODA*, pp. 661–670, Jan. 2001.

[4] A. Meyerson, K. Munagala, and S. Plotkin, "Web caching using access statistics," in *Proc. ACM-SIAM SODA*, pp. 354–363, 2001.

[5] I. Baev, R. Rajaraman, and C. Swamy, "Approximation algorithms for data placement problems," *SIAM J. Comput.*, vol. 38, pp. 1411–1429, July 2008.

[6] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *Proc. IEEE INFOCOM*, pp. 1–9, Mar. 2010.

[7] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *Proc. IEEE INFOCOM*, pp. 126–134, Mar. 1999.

[8] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy, "On the scale and performance of cooperative web proxy caching," in *Proc. ACM SOSP*, pp. 16–31, Dec. 1999.

[9] M. Hefeeda and O. Saleh, "Traffic modeling and proportional partial caching for peer-to-peer systems," *IEEE/ACM Trans. Netw.*, vol. 16, pp. 1447–1460, Dec. 2008.

[10] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *arXiv:1209.5807 [cs.IT]*, Sept. 2012. Accepted for publication in IEEE Trans. Inf. Theory.

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

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

[13] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "I tube, you tube, everybody tubes: Analyzing the world's largest user generated content video system," in *Proc. ACM IMC*, Oct. 2007.

[14] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *Proc. ACM SOSP*, pp. 314–329, Oct. 2003.

[15] Y. Birk and T. Kol, "Coding on demand by an informed source (ISCOD) for efficient broadcast of different supplemental data to caching clients," *IEEE Trans. Inf. Theory*, vol. 52, pp. 2825–2830, June 2006.

[16] Z. Bar-Yossef, Y. Birk, T. S. Jayram, and T. Kol, "Index coding with side information," *IEEE Trans. Inf. Theory*, vol. 57, pp. 1479–1494, Mar. 2011.

[17] S. E. Rouayheb, A. Sprintson, and C. Georghiades, "On the index coding problem and its relation to network coding and matroid theory," *IEEE Trans. Inf. Theory*, vol. 56, pp. 3187–3195, July 2010.

[18] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, vol. 46, pp. 1204–1216, Apr. 2000.

[19] M. Langberg and A. Sprintson, "On the hardness of approximating the network coding capacity," *IEEE Trans. Inf. Theory*, vol. 57, pp. 1008–1014, Feb. 2011.