Coded Caching with Optimized Shared-Cache Sizes

Emanuele Parrinello and Petros Elia
Communication Systems Department, EURECOM, Sophia Antipolis, France
Email: {parrinel,elia}@eurecom.fr

***Abstract*— This work studies the K-user broadcast channel where each user is assisted by one of Λ caches with a cumulative memory constraint that is equal to t times the size of the library, and where each cache serves an arbitrary number of users. In this setting, under the assumption of uncoded cache placement, no prior scheme is known to achieve a *sum degrees of freedom* (DoF) of $t+1$, other than in the uniform case where all caches serve an equal number of users. We here show for the first time that allowing an optimized memory allocation across the caches as a function of the number of users served per cache, provides for the aforementioned DoF. A subsequent index-coding based converse proves that this performance can be close to optimal for bounded values of t.**

I. Introduction

The well known work by Maddah-Ali and Niesen in [1] revealed that in the shared-link broadcast channel (BC) with cache-aided users, a carefully designed cache-placement phase can allow content delivery that employs multicasting transmissions that can be simultaneously useful to many users having different content requests. This delivery speedup, commonly referred to as the *coding gain* or equivalently as the Degrees-of-Freedom (DoF), was interestingly shown to scale with the cumulative cache capacity of the network, and it was shown in [2], [3] to be essentially optimal for the specific network.

Subsequently a variety of works explored coded caching in different scenarios such as D2D networks [4], multi-transmitters wireless settings [5]–[7], subpacketization constrained settings [8], [9], non-uniform file popularities [10], multiple access networks [11], and in other settings as well.

A. Coded Caching with Shared Caches

A variety of realistic scenarios — such as when cache-aided base stations serve users in cellular networks — calls for exploring coded caching in the presence of *shared caches*, where instead of having each user with its own dedicated cache, now caches (cache-aided base stations) can serve multiple users/receivers at the same time. Crucial to such scenarios is naturally the ability to account for the load density of each cache (i.e., the number of users served by each cache); for example, as noted in [15], a base station serving an office building is statistically likely to serve many more users/demands than a base station serving a more sparse residential area. Such considerations motivated the recent work in [16] which characterized the optimal (under uncoded cache placement) worst-case delivery time when each cache assists an arbitrary number of users and where the corresponding user-to-cache association is not known during the cache-placement phase. Related work can be found in [17] which extends the setting in [16] to account for error-prone links, as well as the work in [18] which quantifies — for the same setting as in [16] — the extra gains from knowing the user-to-cache association during the cache placement phase.

This same heterogeneous context also naturally supports the scenario that the more popular caches (those known to serve many users) will be allocated more memory. Motivated by this, we will consider a shared-cache setting — where each cache serves an arbitrary number of users/requests — and we will seek to optimally allocate a given storage memory across the caches and then to design a caching and delivery strategy that minimizes the delivery time. For this problem, we propose a novel caching scheme and an information theoretic converse, together with an upper bound on the gap to optimal.

B. Notation

For $\Lambda \in \mathbb{N}$, we use $[\Lambda] \triangleq \{1, 2, \dots, \Lambda\}$ and we use $\mathcal{C}_k^{\Lambda} \triangleq \{\tau : \tau \subseteq [\Lambda], |\tau| = k\}$ to denote the set of all k-combinations of $[\Lambda]$. For any $n \in \mathbb{N}$, we use $\mathcal{S}_n$ to denote the symmetric group of all permutations of $[n]$. Finally for an ordered set τ, we will refer to the j-th element of τ as $\tau(j)$.

II. System Model and Problem Definition

We consider a network where a server with access to a library of $N (N \geq K)$ unit-sized files $W^{(1)}, W^{(2)}, \dots, W^{(N)}$ is connected via a shared-link broadcast channel[1] to K users, each enjoying direct access (at zero cost) to *one* of Λ different caches, where the size of each cache $\lambda \in \{1, 2, \dots, \Lambda\}$ is a design parameter $M_\lambda \in (0, N]$ (in units of file), adhering to a cumulative sum cache-size constraint $\sum_{\lambda=1}^{\Lambda} M_\lambda = M_\Sigma$. For $\gamma_\lambda \triangleq \frac{M_\lambda}{N}$, this constraint takes the form

$$\sum_{\lambda=1}^{\Lambda} \gamma_\lambda = t \triangleq \frac{M_\Sigma}{N}. \tag{1}$$

Each cache λ is connected to a set of L_λ users $\mathcal{U}_\lambda \subset [K]$, thus defining a so-called *user profile* vector

$$\mathbf{L} = (L_1, \dots, L_\Lambda) \tag{2}$$

where naturally $\sum_{\lambda=1}^{\Lambda} |\mathcal{U}_\lambda| = K$ and where without loss of generality we assume that $L_i \geq L_j$ for $i \leq j$.

This work was supported by the European Research Council under the EU Horizon 2020 research and innovation program / ERC grant agreement no. 725929.

[1]The channel capacity is normalized to one file per unit of time.

The scenario calls for three different phases.

1) A *memory allocation phase* during which the total memory M_Σ is allocated to the caches, yielding the allocated set $\{\gamma_\lambda\}_{\lambda=1}^{\Lambda}$.
2) A *cache placement phase* during which each cache λ — of allocated size γ_λ — is filled with content $\mathcal{Z}_\lambda$ from the library, according to a certain strategy $\boldsymbol{\mathcal{Z}} = (\mathcal{Z}_1, \ldots, \mathcal{Z}_\Lambda)$.
3) A *delivery phase* which starts when each user $k \in [K]$ requests a file, $W^{(d_k)}$. With the demand vector $\boldsymbol{d} = (d_1, d_2, \ldots, d_K)$ known, the server will aim to deliver each requested file to its corresponding user.

A. Problem definition

For any given *uncoded cache placement* strategy $\boldsymbol{\mathcal{Z}}$ and any given demand $\boldsymbol{d}$, we denote by $T^*(\boldsymbol{\mathcal{Z}}, \boldsymbol{d})$ the minimum (over all delivery schemes) time that guarantees delivery to all users $k \in [K]$ of their desired files $W^{(d_k)}$. Our goal is to characterize the minimum (over all memory-allocation strategies and all placement-and-delivery schemes, under the assumption of uncoded cache placement) worst-case delivery time

$$T^* = \min_{\boldsymbol{\mathcal{Z}}} \max_{\boldsymbol{d}} T^*(\boldsymbol{\mathcal{Z}}, \boldsymbol{d}) \tag{3}$$

as a function of K, M_Σ, N and $\mathbf{L}$.

III. Main Results

In this section we state our main contributions.

Theorem 1. *For the K-user BC with Λ shared caches, a sum-cache constraint M_Σ and a user profile $\mathbf{L}$, the worst-case delivery time*

$$T = \frac{\sum_{\lambda=1}^{\Lambda} L_\lambda (1 - \gamma_\lambda)}{t+1} \tag{4}$$

is achievable for any integer[2] *$t \in [\Lambda]$, where*

$$\gamma_\lambda = \frac{\sum_{\tau \in \mathcal{C}_t^\Lambda:\ \tau \ni \lambda} \prod_{j=1}^{t} L_{\tau(j)}}{\sum_{\tau \in \mathcal{C}_t^\Lambda} \prod_{j=1}^{t} L_{\tau(j)}}. \tag{5}$$

Proof: The placement and delivery schemes are presented in Section IV.

A. DoF Performance

In the context of uneven $\mathbf{L}$ and uneven $\{\gamma_\lambda\}$, the concept of the sum DoF naturally generalizes to $DoF \triangleq \frac{K - \sum_{\lambda=1}^{\Lambda} \gamma_\lambda L_\lambda}{T}$ reflecting the rate of delivery of the non-cached desired information. By recalling that $\sum_{\lambda=1}^{\Lambda} L_\lambda = K$, the delay in (4) directly implies that $DoF = t + 1$, which is an improvement over the case where $\mathbf{L}$ is unknown to the placement phase. In fact, we know from [16] that without knowledge of $\mathbf{L}$ during the (uncoded) cache placement, the $DoF = t + 1$ can be achieved only in the uniform case of having $\frac{K}{\Lambda}$ users per cache, and that any non-uniformity in $\mathbf{L}$ forces a DoF penalty. The above theorem shows that knowledge of the profile $\mathbf{L}$ allows for a redesigned and skewed memory allocation[3] which — in conjunction with the coded caching scheme — achieves this $DoF = t + 1$ regardless of $\mathbf{L}$.

[2]For non-integer t, the lower convex envelope is achievable.

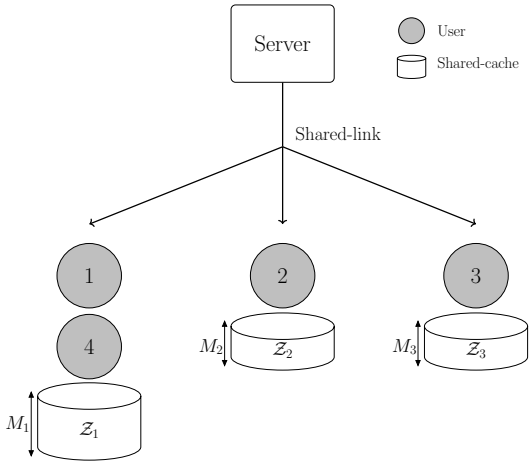

Fig. 1. Caching setting considered in this work for $\mathbf{L} = (2, 1, 1)$.

We proceed with the converse.

Theorem 2. *For the K-user, Λ-shared caches problem with cache redundancy $t \in [\Lambda]$ and profile $\mathbf{L}$, under the assumption of uncoded cache placement, any delivery time must satisfy*

$$T \geq \frac{K - \sum_{\lambda=1}^{t} L_\lambda}{t+1}. \tag{6}$$

Proof: The proof can be found in Section V.

The following bounds the gap to optimal under the assumption of uncoded cache placement.

Proposition 1. *For any set $\{K, N, M_\Sigma, \mathbf{L}\}$, the achieved delivery time in* (4) *is at most a multiplicative factor of $t + 1$ from the optimal. When $\mathbf{L} = (2, 1, 1)$ the delivery time is exactly optimal under the assumption of uncoded cache placement.*

Proof: The proof[4] can be found in Section VII.

IV. Placement-and-Delivery Scheme

In this section we present our caching and delivery scheme that achieves the performance in (4). We start with a simple example to help the reader understand the idea behind the scheme.

A. Example

Consider the problem with $N = 4$ files, $\Lambda = 3$ caches and $K = 4$ users (see Fig. 1) distributed according to the profile

[3]It is this skewness that allows for multicasting messages that serve $t + 1$ users at a time. It is interesting to observe that the skewness of the allocated $\{\gamma_\lambda\}_{\lambda=1}^{\Lambda}$ reduces as t increases. For $t = 1$, the skewness is maximal and it corresponds to $\gamma_\lambda = \frac{L_\lambda}{K}$, while as t increases, the allocation approaches the uniform allocation.

[4]The exact optimality when $\mathbf{L} = (2, 1, 1)$ is proven by means of a more sophisticated variant of the index coding technique in Section V. The proof will appear in the extended version of this work [19].

$\mathbf{L} = (2,1,1)$. We assume that a sum memory of $M_\Sigma = 4$ units of file is available to be allocated across the caches.

In the caching phase, we split each file W^n ($n \in \{1,2,3,4\}$) into 4 equally-sized subfiles indexed by the pairs $\{(1,1),(1,2),(2,1),(3,1)\}$. Based on the first index, we fill the caches as follows;

$$\mathcal{Z}_1 = \{W^{(1)}_{1,1}, W^{(1)}_{1,2}, W^{(2)}_{1,1}, W^{(2)}_{1,2}, W^{(3)}_{1,1}, W^{(3)}_{1,2}, W^{(4)}_{1,1}, W^{(4)}_{1,2}\},$$
$$\mathcal{Z}_2 = \{W^{(1)}_{2,1}, W^{(2)}_{2,1}, W^{(3)}_{2,1}, W^{(4)}_{2,1}\},$$
$$\mathcal{Z}_3 = \{W^{(1)}_{3,1}, W^{(2)}_{3,1}, W^{(3)}_{3,1}, W^{(4)}_{3,1}\}$$

thus adhering to $\gamma_1 = \frac{1}{2}, \gamma_2 = \frac{1}{4}, \gamma_3 = \frac{1}{4}$, as stated in (5).

In the delivery phase, we consider the demand vector $\boldsymbol{d} = (1,2,3,4)$ where $W^{(1)}$ and $W^{(2)}$ are each requested by the two users associated to cache 1, $W^{(3)}$ by the user with cache 2, and $W^{(4)}$ by the user with cache 3. For $\mathcal{R}_\lambda$ denoting the set of uncached subfiles wanted by the users of cache λ, we have

$$\mathcal{R}_1 = \{W^{(1)}_{2,1}, W^{(1)}_{3,1}, W^{(2)}_{2,1}, W^{(2)}_{3,1}\},$$
$$\mathcal{R}_2 = \{W^{(3)}_{1,1}, W^{(3)}_{3,1}, W^{(3)}_{1,2}\},$$
$$\mathcal{R}_3 = \{W^{(4)}_{1,1}, W^{(4)}_{2,1}, W^{(4)}_{1,2}\}.$$

These files are transmitted — two at a time — in the form of the following 5 XORs

$$X_1 = W^{(1)}_{2,1} \oplus W^{(3)}_{1,1} \qquad X_2 = W^{(1)}_{3,1} \oplus W^{(4)}_{1,1}$$
$$X_3 = W^{(3)}_{3,1} \oplus W^{(4)}_{2,1} \qquad X_4 = W^{(2)}_{2,1} \oplus W^{(3)}_{1,2}$$
$$X_5 = W^{(2)}_{3,1} \oplus W^{(4)}_{1,2}$$

and can be decoded in the standard clique-based manner of [1], thus yielding a delay $T = \frac{5}{4}$ which can be shown to be optimal under the assumption of uncoded cache placement.

We proceed with the description of the general scheme.

B. Memory Allocation and Cache Placement

We first split each file $W^{(n)}, n \in [N]$, into

$$S = \sum_{\tau \in \mathcal{C}^\Lambda_t} \prod_{j=1}^{t} L_{\tau(j)} \tag{7}$$

subfiles of equal size, and we label each subfile as

$$W^{(n)} = \left\{ W^{(n)}_{\tau,1}, W^{(n)}_{\tau,2}, \ldots, W^{(n)}_{\tau,|A_\tau|} \mid \tau \in \mathcal{C}^\Lambda_t \right\}$$

where $A_\tau \triangleq \left\{1,2,\ldots,\prod_{j=1}^{t} L_{\tau(j)}\right\}$. Cache $\lambda \in [\Lambda]$ caches all subfiles $W^{(n)}_{\tau,m}, m \in A_\tau$ whose first subscript τ includes λ, which in turn yields caches

$$\mathcal{Z}_\lambda = \left\{ W^{(n)}_{\tau,m} : W^{(n)}_{\tau,m} \in W^{(n)}, \tau \ni \lambda, n \in [N] \right\}.$$

This automatically yields the memory allocation $\{\gamma_\lambda\}_{\lambda=1}^{\Lambda}$ from (5) because the aforementioned placement condition that a subfile $W^{(n)}_{\tau,m}$ is placed in $\mathcal{Z}_\lambda$ if only if $\lambda \in \tau$, manages to automatically guarantee the numerator of (5), while the subpacketization in (7) guarantees the denominator of (5). This same placement also assures that each subfile is cached in exactly t caches (because each τ satisfies $|\tau| = t$), which in turn guarantees the sum memory constraint in (1).

This placement yields an interesting property, which is described in the following lemma.

Lemma 1. *For any $t+1$-tuple $\mathcal{Q} \subset [\Lambda]$, the total number of subfiles with first subscript $\tau = \mathcal{Q} \setminus \{\lambda\}$ that are missing from all the users associated to any specific cache $\lambda \in \mathcal{Q}$, is independent of λ and it equals*

$$P_\mathcal{Q} \triangleq \prod_{j=1}^{t+1} L_{\mathcal{Q}(j)}. \tag{8}$$

Proof. For any $t+1$-tuple $\mathcal{Q} \subset [\Lambda]$, consider cache $\lambda \in \mathcal{Q}$ and let $\tau = \mathcal{Q}\setminus\{\lambda\}$. There are L_λ requested files from the users $\mathcal{U}_\lambda$ of cache λ, each having $\prod_{j=1}^{t} L_{\tau(j)}$ subfiles with first index τ. This in turn means that the total number of subfiles that need to be sent to satisfy users in $\mathcal{U}_\lambda$ is $L_\lambda \prod_{j=1}^{t} L_{\tau(j)} = \prod_{j=1}^{t+1} L_{\mathcal{Q}(j)}$, which does not depend on λ. □

C. Delivery phase

For ease of presentation, we will use $\boldsymbol{d^\lambda}$ to denote the vector of indices of the files requested by the users in $\mathcal{U}_\lambda$.

For a fixed $t+1$-tuple $\mathcal{Q}$ and any $\lambda \in \mathcal{Q}$, consider the set

$$\{W^{(\boldsymbol{d^\lambda}(j))}_{\tau,m} : j \in [L_\lambda], m \in A_\tau\}$$

of subfiles, with first subscript $\tau = \mathcal{Q}\setminus\{\lambda\}$, that are requested from users in $\mathcal{U}_\lambda$, and relabel these as

$$\{F^{(\lambda)}_{\tau,j} : j \in [P_\mathcal{Q}]\}.$$

The two sets are the same.

Because of the cache placement phase, we notice that for any $t+1$-tuple $\mathcal{Q}$ and any $j \in [P_\mathcal{Q}]$, the set of subfiles

$$F^{(\lambda)}_{\mathcal{Q}\setminus\{\lambda\},j}, \ \forall \lambda \in \mathcal{Q} \tag{9}$$

forms a clique of $t+1$ nodes. By Lemma 1, for any $t+1$-tuple $\mathcal{Q} \in [\Lambda]$, we have $P_\mathcal{Q}$ cliques as in (9), all of $t+1$ nodes. Consequently we transmit, for each $t+1$-tuple $\mathcal{Q} \subseteq [\Lambda]$, the following $P_\mathcal{Q}$ XORs

$$X_\mathcal{Q}(j) = \bigoplus_{\lambda \in \mathcal{Q}} F^{(\lambda)}_{\mathcal{Q}\setminus\{\lambda\},j}, \quad \forall j \in [P_\mathcal{Q}] \tag{10}$$

whose structure allows for clique-based decoding as in [1].

D. Performance of the scheme

The fact that there are $P_\mathcal{Q}$ XORs for each $t+1$-tuple Q, implies a total of

$$\sum_{\mathcal{Q} \in \mathcal{C}^\Lambda_{t+1}} P_\mathcal{Q} = \sum_{\mathcal{Q} \in \mathcal{C}^\Lambda_{t+1}} \prod_{j=1}^{t+1} L_{\mathcal{Q}(j)}$$

transmissions and a corresponding delay of

$$T = \frac{\sum_{\mathcal{Q} \in \mathcal{C}^\Lambda_{t+1}} \prod_{j=1}^{t+1} L_{\mathcal{Q}(j)}}{\sum_{\tau \in \mathcal{C}^\Lambda_t} \prod_{j=1}^{t} L_{\tau(j)}} \overset{(a)}{=} \frac{\sum_{\lambda=1}^{\Lambda} L_\lambda (1-\gamma_\lambda)}{t+1} \tag{11}$$

where the denominator $\sum_{\tau\in\mathcal{C}_t^\Lambda}\prod_{j=1}^{t}L_{\tau(j)}$ is due to (7), and where (a) follows from basic mathematical manipulations which are omitted here due to lack of space. The above expression reflects the fact that each user from cache λ enjoys a local caching gain γ_λ, the fact that each cache λ "requests" L_λ files, and the fact that all transmitted XORs served $t+1$ users at a time.

V. Proof of the Converse

In this section we present the proof of Theorem 2.

The technique used to develop the bound draws from [2] and some aspects are drawn from [16]. Key to the bound is the conversion of the coded caching problem into an equivalent index coding problem, along with the use of the outer bound on the index coding capacity presented in [20, Corollary 1].

a) Translation to index coding: Let $\mathcal{D}_{dif}$ be the set of vectors $\boldsymbol{d}$ comprised of K different file indices. For every demand vector $\boldsymbol{d}\in\mathcal{D}_{dif}$, we assume that each requested file $W^{(\boldsymbol{d^\lambda}(l))}, l\in[L_\lambda]$ is split in a generic manner into 2^Λ disjoint subfiles $W_\tau^{(\boldsymbol{d^\lambda}(l))}, \tau\in 2^{[\Lambda]}$ according to the power set $2^{[\Lambda]}$ of $[\Lambda]$, where $\tau\in[\Lambda]$ denotes the set of caches in which $W_\tau^{(\boldsymbol{d^\lambda}(l))}$ is cached, and where the size of each subfile $|W_\tau^{(\boldsymbol{d^\lambda}(l))}|$ can take any value. The equivalent index coding problem (equivalent to the caching problem defined by $\boldsymbol{d}$) is then fully defined by the side information graph $\mathcal{G}_{\boldsymbol{d}}=(\mathcal{V}_\mathcal{G},\mathcal{E}_\mathcal{G})$, where $\mathcal{V}_\mathcal{G}$ is the set of vertices corresponding to the requested subfiles $W_\tau^{(\boldsymbol{d^\lambda}(l))}, \lambda\notin\tau$, where $\mathcal{E}_\mathcal{G}$ is the set of directed edges of the graph. A directed edge from vertex $v\in\mathcal{V}_\mathcal{G}$ to $v'\in\mathcal{V}_\mathcal{G}$ exists if and only if the index coding user requesting vertex/subfile v' knows the subfile corresponding to vertex v.

Helpful to our proof will be the adaptation of two lemmas from [20] and [16], which we state below.

Lemma 2. *(Cut-set-type converse [20]) For a given side information graph $\mathcal{G}_{\boldsymbol{d}}=(\mathcal{V}_\mathcal{G},\mathcal{E}_\mathcal{G})$, the following inequality holds*

$$T\geq\sum_{v\in\mathcal{V}_\mathcal{J}}|v| \tag{12}$$

for every acyclic induced subgraph $\mathcal{J}=(\mathcal{V}_\mathcal{J},\mathcal{E}_\mathcal{J})$ of $\mathcal{G}_{\boldsymbol{d}}$, where $|v|$ is the size of the subfile/node v.

The following lemma is an adaptation of Lemma 1 in [2] taken here directly from [16].

Lemma 3. *An acyclic subgraph $\mathcal{J}$ of $\mathcal{G}_{\boldsymbol{d}}$, is designed here to consist of all subfiles $W_{\tau_\lambda}^{\boldsymbol{d^{\sigma\lambda}}(j)}$, $\forall j\in[L_\lambda]$, $\forall\lambda\in[\Lambda]$ for all $\tau_\lambda\subseteq[\Lambda]\setminus\{\sigma_1,\dots,\sigma_\lambda\}$ where $\boldsymbol{\sigma}\stackrel{\triangle}{=}(\sigma_1,\dots,\sigma_\Lambda)$ is a permutation on vector $(1,\dots,\Lambda)$.*

For any acyclic subgraph of $\mathcal{G}_{\boldsymbol{d}}$ induced by a permutation $\boldsymbol{\sigma}$ according to Lemma 3, we use Lemma 2 to get

$$T^*(\boldsymbol{\mathcal{Z}},\boldsymbol{d})\geq T^{lb}_{\boldsymbol{\sigma}}(\boldsymbol{\mathcal{Z}},\boldsymbol{d})$$

where

$$T^{lb}_{\boldsymbol{\sigma}}(\boldsymbol{\mathcal{Z}},\boldsymbol{d})\stackrel{\triangle}{=}\sum_{l=1}^{L_1}\sum_{\tau_1\subseteq[\Lambda]\setminus\{\sigma_1\}}|W_{\tau_1}^{(\boldsymbol{d^{\sigma_1}}(l))}|+\cdots+$$
$$\sum_{l=1}^{L_\lambda}\sum_{\tau_\lambda\subseteq[\Lambda]\setminus\{\sigma_1,\dots,\sigma_\lambda\}}|W_{\tau_\lambda}^{(\boldsymbol{d^{\sigma\lambda}}(l))}|+\cdots+\sum_{l=1}^{L_\Lambda}\sum_{\tau_\Lambda\subseteq[\Lambda]\setminus\{\sigma_1,\dots,\sigma_\Lambda\}}|W_{\tau_\Lambda}^{(\boldsymbol{d^{\sigma\Lambda}}(l))}|. \tag{13}$$

To symmetrize, we average (13) over all $|\mathcal{S}_\Lambda|=\Lambda!$ permutations $\boldsymbol{\sigma}\in\mathcal{S}_\Lambda$ to get

$$T^*(\boldsymbol{\mathcal{Z}},\boldsymbol{d})\geq\frac{1}{|\mathcal{S}_\Lambda|}\sum_{\boldsymbol{\sigma}\in\mathcal{S}_\Lambda}T^{lb}_{\boldsymbol{\sigma}}(\boldsymbol{\mathcal{Z}},\boldsymbol{d}) \tag{14}$$

and a subsequent lower bounding of T^* gives

$$\begin{aligned}T^*&=\min_{\boldsymbol{\mathcal{Z}}}\max_{\boldsymbol{d}\in[N]^K}T^*(\boldsymbol{\mathcal{Z}},\boldsymbol{d})\\&\geq\min_{\boldsymbol{\mathcal{Z}}}\frac{1}{|\mathcal{D}_{dif}|}\sum_{\boldsymbol{d}\in\mathcal{D}_{dif}}T^*(\boldsymbol{\mathcal{Z}},\boldsymbol{d})\\&\geq\min_{\boldsymbol{\mathcal{Z}}}\frac{1}{|\mathcal{D}_{dif}||\mathcal{S}_\Lambda|}\sum_{\boldsymbol{d}\in\mathcal{D}_{dif}}\sum_{\boldsymbol{\sigma}\in\mathcal{S}_\Lambda}T^{lb}_{\boldsymbol{\sigma}}(\boldsymbol{\mathcal{Z}},\boldsymbol{d}).\end{aligned} \tag{15}$$

At this point, the inherent symmetry with respect to the file indices, allows us to rewrite the double summation in (15) as

$$\sum_{\boldsymbol{d}\in\mathcal{D}_{dif}}\sum_{\boldsymbol{\sigma}\in\mathcal{S}_\Lambda}T^{lb}_{\boldsymbol{\sigma}}(\boldsymbol{\mathcal{Z}},\boldsymbol{d})=\sum_{i=0}^{\Lambda}\sum_{\tau\in2^{[\Lambda]}:|\tau|=i}g_\tau|W_\tau^\Sigma| \tag{16}$$

where g_τ denotes the number of times that $|W_\tau^\Sigma|$ appears in (15). We can prove that

$$\frac{g_\tau}{|\mathcal{D}_{dif}||\mathcal{S}_\Lambda|}=\frac{K-\sum_{j\in\tau}L_j}{N(|\tau|+1)} \tag{17}$$

the proof of which can be found in the extended version [19] of this work. Next, we combine (17) with (15)-(16) to obtain

$$T^*\geq\min_{\boldsymbol{\mathcal{Z}}}\sum_{i=0}^{\Lambda}\sum_{\tau\in2^{[\Lambda]}:|\tau|=i}\frac{K-\sum_{j\in\tau}L_j}{N(|\tau|+1)}|W_\tau^\Sigma| \tag{18}$$

which, together with the sum file size constraint and the cache size constraint

$$\sum_{\tau\in2^{[\Lambda]}}|W_\tau^\Sigma|=N \tag{19}$$

$$\sum_{\tau\in2^{[\Lambda]}}|\tau||W_\tau^\Sigma|=tN \tag{20}$$

yields the linear program

$$\begin{aligned}&\underset{|W_\tau^\Sigma|}{\text{minimize}}\ \sum_{i=0}^{\Lambda}\sum_{\tau\in2^{[\Lambda]}:|\tau|=i}\frac{K-\sum_{j\in\tau}L_j}{N(|\tau|+1)}|W_\tau^\Sigma|\\&\text{subject to}\ \ (19),(20)\ \ \text{and}\ \ \ |W_\tau^\Sigma|\geq0,\ \forall\tau\in2^{[\Lambda]}\end{aligned}$$

whose solution is exactly

$$\frac{K-\sum_{j=1}^{t}L_j}{N(t+1)}. \tag{21}$$

The LP solution in (21) can be verified by solving the KKT conditions, and this verification can be found in the extended version [19] of this work. This concludes the proof. □

VI. Conclusions and Future Works

The work explored the coded caching problem with shared caches where each cache serves an arbitrary number of users, and proposes a scheme which — under a cumulative cache-size constraint — yields a sum DoF of $t+1$ users served at a time. Compared to settings where all caches have the same size, the current solution also allows for increased local caching gains. The work also presented an information theoretic converse that allows us to identify the optimal performance (under the assumption of uncoded cache placement) within a gap of no more than $t+1$, irrespective of K and $\mathbf{L}$.

An interesting extension of this work could be the study of the average delivery time. As always, another direction would be to account for subpacketization constraints, which are known to severely limit theoretical gains. Also motivated by the approach in [18], an additional extension would be to explore potential gains from allowing coded cache placement. Finally one could explore — under a non-uniform file popularity — the tradeoffs between increased local caching gain (cf. [15]) and the current approach that focuses on coding gain.

VII. Appendix

A. Gap from Optimality

To bound the gap to the optimal T^*, we employ the achievable T from (4) and the lower bound from (6), and proceed as follows

$$\frac{T}{T^*} \leq \frac{K-\sum_{\lambda=1}^{\Lambda} L_\lambda \gamma_\lambda}{K-\sum_{\lambda=1}^{t} L_\lambda} \tag{22}$$

$$= 1+\frac{\sum_{\lambda=1}^{t} L_\lambda - \sum_{\lambda=1}^{\Lambda} L_\lambda \gamma_\lambda}{\sum_{\lambda=t+1}^{\Lambda} L_\lambda} \tag{23}$$

$$\overset{(a)}{=} 1+\frac{\sum\limits_{\lambda=1}^{t} L_\lambda \sum\limits_{\tau\in\mathcal{C}_t^\Lambda} \prod\limits_{j=1}^{t} L_{\tau(j)} - \sum\limits_{\lambda=1}^{\Lambda} L_\lambda \sum\limits_{\tau\in\mathcal{C}_t^\Lambda:\ \tau\ni\lambda} \prod\limits_{j=1}^{t} L_{\tau(j)}}{\sum_{\lambda=t+1}^{\Lambda} L_\lambda \sum\limits_{\tau\in\mathcal{C}_t^\Lambda} \prod\limits_{j=1}^{t} L_{\tau(j)}} \tag{24}$$

$$= 1+\frac{\sum\limits_{\lambda=1}^{t} L_\lambda \sum\limits_{\tau\in\mathcal{C}_t^\Lambda:\lambda\notin\tau} \prod\limits_{j=1}^{t} L_{\tau(j)} - \sum\limits_{\lambda=t+1}^{\Lambda} L_\lambda \sum\limits_{\tau\in\mathcal{C}_t^\Lambda:\ \tau\ni\lambda} \prod\limits_{j=1}^{t} L_{\tau(j)}}{\sum_{\lambda=t+1}^{\Lambda} L_\lambda \sum\limits_{\tau\in\mathcal{C}_t^\Lambda} \prod\limits_{j=1}^{t} L_{\tau(j)}} \tag{25}$$

$$= 1+\frac{\sum\limits_{\lambda=1}^{t} L_\lambda \sum\limits_{\tau\in\mathcal{C}_t^\Lambda:\lambda\notin\tau} \prod\limits_{j=1}^{t} L_{\tau(j)} - \overbrace{\sum\limits_{\lambda=t+1}^{\Lambda} L_\lambda \sum\limits_{\tau\in\mathcal{C}_t^\Lambda:\ \tau\ni\lambda} \prod\limits_{j=1}^{t} L_{\tau(j)}}^{B}}{\sum\limits_{\lambda=t+1}^{\Lambda} L_\lambda \left(\sum\limits_{\tau\in\mathcal{C}_t^\Lambda:\lambda\in\tau} \prod\limits_{j=1}^{t} L_{\tau(j)} + \sum\limits_{\tau\in\mathcal{C}_t^\Lambda:\lambda\notin\tau} \prod\limits_{j=1}^{t} L_{\tau(j)} \right)} \tag{26}$$

$$\overset{(b)}{\leq} 1+\frac{\sum_{\lambda=1}^{t} L_\lambda \left(\sum\limits_{\tau\in\mathcal{C}_t^\Lambda:\lambda\notin\tau} \prod\limits_{j=1}^{t} L_{\tau(j)} \right)}{\sum_{\lambda=t+1}^{\Lambda} L_\lambda \left(\sum\limits_{\tau\in\mathcal{C}_t^\Lambda:\lambda\notin\tau} \prod\limits_{j=1}^{t} L_{\tau(j)} \right)} \tag{27}$$

$$= 1+\sum_{\lambda=1}^{t} \frac{L_\lambda \left(\sum\limits_{\tau\in\mathcal{C}_t^\Lambda:\lambda\notin\tau} \prod\limits_{j=1}^{t} L_{\tau(j)} \right)}{\sum_{i=t+1}^{\Lambda} L_i \left(\sum\limits_{\tau\in\mathcal{C}_t^\Lambda:i\notin\tau} \prod\limits_{j=1}^{t} L_{\tau(j)} \right)} \tag{28}$$

$$\overset{(c)}{\leq} 1+t \tag{29}$$

where (a) is obtained by applying the value of γ_λ from (5), (b) is obtained after adding and subtracting B to the numerator and denominator of (26), and (c) is due to the fact that each term of the summation in (28) is smaller than 1. □

References

[1] M. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Trans. Inf. Theory*, May 2014.

[2] K. Wan, D. Tuninetti, and P. Piantanida, "On the optimality of uncoded cache placement," in *Information Theory Workshop (ITW), IEEE*, 2016.

[3] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Characterizing the rate-memory tradeoff in cache networks within a factor of 2," *IEEE Trans. Inf. Theory*, Jan 2019.

[4] M. Ji, G. Caire, and A. F. Molisch, "Fundamental limits of caching in wireless D2D networks," *IEEE Trans. Inf. Theory*, Feb 2016.

[5] S. P. Shariatpanahi, G. Caire, and B. H. Khalaj, "Physical-layer schemes for wireless coded caching," *IEEE Trans. Inf. Theory*, 2018.

[6] J. Zhang and P. Elia, "Fundamental limits of cache-aided wireless BC: Interplay of coded-caching and CSIT feedback," *IEEE Trans. Inf. Theory*, May 2017.

[7] E. Lampiris, J. Zhang, and P. Elia, "Cache-aided cooperation with no CSIT," in *Proc. IEEE Int. Symp. on Inform. Theory (ISIT)*, June 2017.

[8] Q. Yan, M. Cheng, X. Tang, and Q. Chen, "On the placement delivery array design for centralized coded caching scheme," *IEEE Trans. Inf. Theory*, Sept 2017.

[9] E. Lampiris and P. Elia, "Adding transmitters dramatically boosts coded-caching gains for finite file sizes," *IEEE JSAC (Special Issue on Caching)*, June 2018.

[10] J. Zhang, X. Lin, and X. Wang, "Coded caching under arbitrary popularity distributions," *IEEE Trans. Inf. Theory*, Jan 2018.

[11] J. Hachem, N. Karamchandani, and S. Diggavi, "Coded caching for multi-level popularity and access," *IEEE Trans. Inf. Theory*, May 2017.

[12] E. Lampiris and P. Elia, "Achieving full multiplexing and unbounded caching gains with bounded feedback resources," in *Proc. IEEE Int. Symp. on Inform. Theory (ISIT)*, June 2018.

[13] E. Lampiris and P. Elia, "Full coded caching gains for cache-less users," in *Information Theory Workshop (ITW), IEEE*, 2018.

[14] N. Zhang and M. Tao, "Fitness-aware coded multicasting for decentralized caching with finite file packetization," *IEEE Wireless Communications Letters*, Oct 2018.

[15] T. X. Vu, S. Chatzinotas, and B. Ottersten, "Coded caching and storage planning in heterogeneous networks," in *2017 IEEE Wireless Communications and Networking Conference (WCNC)*, March 2017.

[16] E. Parrinello, A. Ünsal, and P. Elia, "Fundamental limits of caching in heterogeneous networks with uncoded prefetching," *arXiv preprint https://arxiv.org/pdf/1811.06247.pdf*, 2018.

[17] N. S. Karat, S. Dey, A. Thomas, and B. S. Rajan, "An optimal linear error correcting delivery scheme for coded caching with shared caches," 2019. [Online]. Available: http://arxiv.org/abs/1901.03188

[18] K. Wan, D. Tuninetti, M. Ji, and G. Caire, "Novel inter-file coded placement and d2d delivery for a cache-aided fog-ran architecture," 2018. [Online]. Available: https://arxiv.org/pdf/1811.05498.pdf

[19] E. Parrinello and P. Elia, "Coded caching with optimized shared-cache sizes," *Manuscript in Preparation (arxiv)*, 2019.

[20] F. Arbabjolfaei, B. Bandemer, Y. H. Kim, E. Şaşoğlu, and L. Wang, "On the capacity region for index coding," in *Proc. IEEE Int. Symp. on Inform. Theory (ISIT)*, Jul 2013.