Stochastic Analysis of Coded Multicasting for Shared Caches Networks

Adeel Malik, Berksan Serbetci, Emanuele Parrinello, Petros Elia
Dept. of Communication Systems, EURECOM, Sophia Antipolis, 06410, France
{malik, serbetci, parrinel, elia}@eurecom.fr

***Abstract*—The work establishes the exact fundamental limits of stochastic coded caching when users share a bounded number of cache states, and when the association between users and caches, is random. This association can greatly affect performance, which improves when the association is more balanced across the caches, and which deteriorates when this association becomes less uniform. Our work provides a statistical analysis of the average performance of such networks, quantifying the effect of randomness by identifying in closed-form, the exact optimal average delivery time. To insightfully capture this delay, we derive the exact scaling laws of the optimal average delivery time. In the scenario where delivery involves K users, we conclude that the multiplicative performance deterioration due to randomness — as compared to the well-known deterministic uniform case — can be unbounded and can scale as $\Theta\left(\frac{\log\Lambda}{\log\log\Lambda}\right)$ at $K=\Theta(\Lambda)$, and that as K increases, this deterioration gradually reduces, and ceases to scale when $K=\Omega(\Lambda\log\Lambda)$. The above analysis is validated numerically.**

***Index Terms*—Coded caching, shared caches, heterogeneous networks, femtocaching.**

I. Introduction

Data traffic in mobile networks is rapidly growing, and is expected to increase in the upcoming years. Existing network infrastructures will not be able to support the demand and due to this enormous growth, there emerged a need for new solutions that can serve continuously increasing number of users with a limited amount of network bandwidth resources. A promising means to increase efficiency is to proactively cache data in the base stations and transform the storage capability of the nodes into a new and powerful network resource.

The potential of such cache-enabled wireless networks has been strikingly elevated following the seminal publication in [1] which introduced the concept of *coded caching*, and which revealed that — in theory — an unbounded number of users can be served even with a bounded amount of network resources. This boost was a consequence of a novel cache placement algorithm that enabled the delivery of independent content to many users simultaneously. Since then, several extensions of the basic coded caching setting have been studied. For a thorough review of the existing coded caching works, the reader is strongly encouraged to refer to the longer version of this work [2].

The work is supported by the European Research Council under the EU Horizon 2020 research and innovation program / ERC grant agreement no. 725929 (project DUALITY).

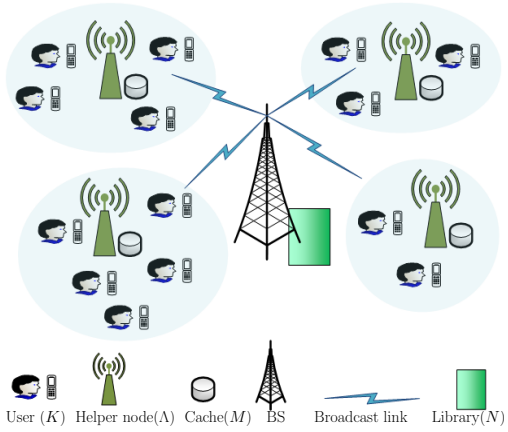

Fig. 1: An instance of a cache-aided heterogeneous network.

A. Coded caching networks with shared caches

Essential to the development of larger, realistic coded caching networks is the so-called *shared-caches setting*, where different users are coerced to benefit from the same cache content. This setting is of great importance because it reflects promising practical scenarios as well as unavoidable constraints.

Such a promising scenario can be found in the context of cache-enabled heterogeneous networks (HetNets), where a central transmitter (a base station) delivers content to a set of interfering users, with the assistance of cache-enabled helper nodes that serve as caches to the users. An instance of a cache-aided HetNet is illustrated in Figure 1. Such networks capture modern trends that envision a central base-station covering a larger area, in tandem with a multitude of smaller helper nodes each covering smaller cells. In this scenario, any user that appears in a particular small cell, can benefit from the cache-contents of the single helper node covering that cell.

In the context of coded caching, an early work on this scenario can be found in [3], which considered the uniform user-to-cache association case where each helper node is associated to an equal number of users. This assumption was eliminated in [4], which — under the assumption that content cache placement is uncoded as well as agnostic to the user-to-cache association — identified the exact optimal worst-case delivery time, as a function of the user-to-cache association profile that describes the number of users served by each

cache. A similar setting was studied in [5] for the case of non-distinct requests, as well as in [6]–[8] for the topology-aware (non-agnostic) scenario, where the user-to-cache association is known during cache placement. It is interesting to note that this same shared-caches setting also applies (to a certain extent) to the scenario where each user requests multiple files (see for example [9], [10]).

At this point, it is essential to note that this same shared-caches setting is directly related to the inevitable subpacketization bottleneck because this bottleneck can force the use of a reduced number of distinct cache states that must be indispensably shared among the many users. This number of distinct cache states, henceforth denoted as Λ, will be forced under most realistic assumptions, to be substantially less than the total number of users, simply because most known coded caching techniques require file sizes that scale exponentially with Λ (see [2] for a detailed list of works).

Both of the above isomorphic settings entail that during the content delivery that follows the allocation of cache states to each user, different broadcast sessions would experience user populations that differently span the spectrum of cache states. In the best-case scenario, a transmitter would have to deliver to a set of K users that uniformly span the Λ states (such that each cache state is found in exactly K/Λ users), while in the most unfortunate of scenarios, a transmitter would encounter K users that happen to have an identical cache state. Both cases are rare instances of a stochastic process, which we explore here in order to identify the exact optimal performance of such systems.

Most of our results apply both to the HetNet scenario as well as the aforementioned related subpacketization-constrained setting which was nicely studied in [11]. For ease of exposition, we will focus the wording of our description to the first scenario corresponding to a HetNet where Λ plays the role of the number of helper nodes. All the results though of Section II certainly apply to the latter setting as well.

B. Shared-caches setting & problem statement

We consider the shared-caches coded-caching setting where a transmitter having access to a library of N equisized files, delivers content via a broadcast link to K receiving users, with the assistance of Λ cache-enabled helper nodes. Each helper node $\lambda \in [1, 2, \dots, \Lambda]$ is equipped with a cache of storage capacity equal to the size of M files, thus being able to store a fraction $\gamma \triangleq \frac{M}{N} \in \left[\frac{1}{\Lambda}, \frac{2}{\Lambda}, \dots, 1\right]$ of the library. Each such helper node, which will be henceforth referred to as a 'cache', can assist in the delivery of content to any number of receiving users.

The communication process consists of three phases; the *content placement phase*, the *user-to-cache association phase*, and the *delivery phase*. The first phase involves the placement of library-content in the caches, and it is oblivious to the outcome of the next two phases. The second phase is when each user is assigned — independently of the placement phase — to exactly one cache from which it can download content at zero cost. This second phase is also oblivious of the other two phases[1]. The final phase begins with users simultaneously requesting one file each, and continues with the transmitter delivering this content to the receivers. Naturally this phase is aware of the content of the caches, as well as aware of which cache assists each user.

User-to-cache association: For any cache $\lambda \in [1, 2, \dots, \Lambda]$, we denote with v_λ the number of users that are assisted by it, and we consider the *cache population vector* $\mathbf{V} = [v_1, v_2, \dots, v_\Lambda]$. Additionally we consider the sorted version $\mathbf{L} = [l_1, l_2, \dots, l_\Lambda] = sort(\mathbf{V})$, where $sort(\mathbf{V})$ denotes the sorting of vector $\mathbf{V}$ in descending order. We refer to $\mathbf{L}$ as a *profile vector*, and we note that each entry l_λ is simply the number of users assisted by the λ-th most populous (most heavily loaded) cache. Figure 1 depicts an instance of our shared-caches setting where $\mathbf{L} = [5, 4, 3, 2]$.

Delivery phase: The delivery phase commences with each user $k \in [1, 2, \dots, K]$ requesting a single library file that is indexed by $d_k \in [1, 2, \dots, N]$. As is common in coded caching works, we assume that each user requests a different file [1], [4], [11]. Once the transmitter is notified of the *request vector* $\mathbf{d} = [d_1, d_2, \dots, d_K]$, it commences delivery over an error-free broadcast link of bounded capacity of one file per time slot.

C. Metric of interest

As one can imagine, any given instance of the problem, experiences a different user-to-cache association and thus[2] a different $\mathbf{V}$. Our measure of interest is thus the average delay

$$\overline{T}(\gamma) \triangleq E_{\mathbf{V}}[T(\mathbf{V})] = \sum_{\mathbf{V}} P(\mathbf{V}) T(\mathbf{V}), \tag{1}$$

where $T(\mathbf{V})$ is the worst-case delivery time[3] corresponding to any specific cache population vector $\mathbf{V}$, and where $P(\mathbf{V})$ is the probability that the user-to-cache association corresponds to vector $\mathbf{V}$.

More precisely, we use $T(\mathbf{V}, \mathbf{d}, \mathcal{X})$ to define the delivery time required by some generic caching-and-delivery scheme $\mathcal{X}$ to satisfy request vector $\mathbf{d}$ when the user-to-cache association is described by the vector $\mathbf{V}$. Our aim here is to characterize the optimal average delay

$$\begin{aligned} \overline{T}^*(\gamma) &= \min_{\mathcal{X}} E_{\mathbf{V}} \left[\max_{\mathbf{d}} T(\mathbf{V}, \mathbf{d}, \mathcal{X}) \right] \\ &= \min_{\mathcal{X}} E_{\mathbf{L}} \left[E_{\mathbf{V}_\mathbf{L}} \left[\max_{\mathbf{d}} T(\mathbf{V}, \mathbf{d}, \mathcal{X}) \right] \right] \end{aligned} \tag{2}$$

[1]This assumption is directly motivated by the time-scales of the problem, as well as by the fact that in the heterogeneous setting, the user-to-cache association is a function of the geographical location of the user. Note that users can only be associated to caches when users are within the coverage of caches, and a dynamic user-to-cache association that requires continuous communication between the users and the server may not be desirable as one seeks to minimize the network load overhead and avoid the handover.

[2]We briefly note that focusing on $\mathbf{V}$ rather than the sets of users connected to each cache, maintains all the pertinent information, as what matters for the analysis is the number of users connected to each cache and not the index (identity) of the users connected to that cache.

[3]This delay corresponds to the time needed to complete the delivery of any file-request vector $\mathbf{d}$, where the time scale is normalized such that a unit of time corresponds to the optimal amount of time needed to send a single file from the transmitter to the receiver, had there been no caching and no interference.

where the minimization is over all possible caching and delivery schemes $\mathcal{X}$, and where $E_{\mathbf{V_L}}$ denotes the expectation over all vectors $\mathbf{V}$ whose sorted version is equal to some fixed $sort(\mathbf{V}) = \mathbf{L}$. Consequently the metric of interest takes the form

$$\overline{T}(\gamma) = E_{\mathbf{L}}[T(\mathbf{L})] = \sum_{\mathbf{L}} P(\mathbf{L})T(\mathbf{L}) \tag{3}$$

where $T(\mathbf{L}) \triangleq E_{\mathbf{V_L}}\left[\max_{\mathbf{d}} T(\mathbf{V},\mathbf{d})\right]$ and where

$$P(\mathbf{L}) \triangleq \sum_{\mathbf{V}:sort(\mathbf{V})=\mathbf{L}} P(\mathbf{V})$$

is simply the cumulative probability over all $\mathbf{V}$ for which $sort(\mathbf{V}) = \mathbf{L}$.

We will consider here the uncoded cache placement scheme in [1] and the coded multicasting delivery scheme in [4], [11], which will prove to be optimal for our setting under the common assumption of uncoded cache placement. This multi-round delivery scheme introduces — for any $\mathbf{V}$ such that $sort(\mathbf{V}) = \mathbf{L}$ — a worst-case delivery time of

$$T(\mathbf{L}) = \sum_{\lambda=1}^{\Lambda-t} l_\lambda \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}, \tag{4}$$

where $t = \Lambda\gamma$ and l_λ is the number of users in the λ-th most populous cache.

From equation (4) we can see that the minimum delay corresponds to the case when $\mathbf{L}$ is uniform. When Λ divides K, this minimum (uniform) delay takes the well known form

$$T_{min} = \frac{K(1-\gamma)}{1+\Lambda\gamma} \tag{5}$$

while for general K, Λ, it takes the form[4]

$$T_{min} = \frac{\Lambda - t}{1+t}\left(\left\lfloor \frac{K}{\Lambda} \right\rfloor + 1 - f(\hat{K})\right), \tag{6}$$

where $\hat{K} = K - \left\lfloor \frac{K}{\Lambda} \right\rfloor \Lambda$, $f(\hat{K}) = 1$ when $\hat{K} = 0$, $f(\hat{K}) = 0$ when $\hat{K} \geq \Lambda - t$, and $f(\hat{K}) = \frac{\prod_{i=t+1}^{\hat{K}+t}(\Lambda-i)}{\prod_{j=0}^{\hat{K}-1}(\Lambda-j)}$ when $\hat{K} < \Lambda - t$. The proof of this is straightforward, but a complete proof can be found in the longer version of this work [2]. The above T_{min} is optimal under the assumption of uncoded placement (cf. [4]).

On the other hand, for any other (now non-uniform) $\mathbf{L}$, the associated delay $T(\mathbf{L})$ will exceed T_{min} (see [4] for the proof, and see Figure 2 for a few characteristic examples), and thus so will the average delay

$$E_{\mathbf{L}}[T(\mathbf{L})] = \sum_{\lambda=1}^{\Lambda-t} \sum_{\mathbf{L}\in\mathcal{L}} P(\mathbf{L}) l_\lambda \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}, \tag{7}$$

where $\mathcal{L}$ describes the set of all possible profile vectors $\mathbf{L}$ (where naturally $\sum_{\lambda=1}^{\Lambda} l_\lambda = K$).

D. Our contribution

In this work we assume that each user can be associated to any particular cache (i.e., can appear in any particular cell) with equal probability. We will identify the optimal average delay $\overline{T}^*(\gamma)$ and the corresponding (multiplicative) performance deterioration

$$G(\gamma) = \frac{\overline{T}^*(\gamma)}{T_{min}} \tag{8}$$

experienced in this random setting. Our aim is to obtain an exact characterization of the performance measure, and provide asymptotic analysis in order to yield clear insight of the scaling laws. The following are our main contributions.

- In Section II-A, we characterize in closed form the exact optimal average-case delay $\overline{T}^*(\gamma)$, optimized over all placement and delivery schemes under the assumption of uncoded cache placement and under the assumption that each user can be associated to any particular cache with equal probability.
- In Section II-B, we characterize the exact scaling laws of performance. It is interesting to see that the aforementioned multiplicative deterioration $G(\gamma) = \frac{\overline{T}^*(\gamma)}{T_{min}}$ can in fact be unbounded, as Λ increases. For example, when $K = \Theta(\Lambda)$ (i.e., when K matches the order of Λ), the performance deterioration scales exactly as $\Theta\left(\frac{\log\Lambda}{\log\log\Lambda}\right)$, whereas when K increases, this deterioration gradually reduces, and ceases to scale when $K = \Omega(\Lambda\log\Lambda)$.
- In Section III, we perform numerical evaluations that validate our analysis.

E. Notations

Throughout the paper, we use the following asymptotic notation: i) $f(x) = O(g(x))$ means that there exist constants a and c such that $f(x) \leq ag(x), \forall x > c$, ii) $f(x) = o(g(x))$ means that $\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$, iii) $f(x) = \Omega(g(x))$ if $g(x) = O(f(x))$, iv) $f(x) = \omega(g(x))$ means that $\lim_{x\to\infty} \frac{g(x)}{f(x)} = 0$, v) $f(x) = \Theta(g(x))$ if $f(x) = O(g(x))$ and $f(x) = \Omega(g(x))$. We use the term $\text{polylog}(x)$ to denote the class of functions $\bigcup_{k\geq 1} O((\log x)^k)$ that are polynomial in $\log x$. Unless otherwise stated, logarithms are assumed to have base 2.

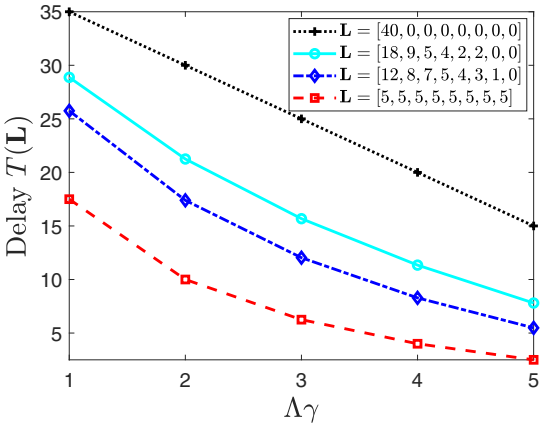

Fig. 2: Delay $T(\mathbf{L})$ for different profile vectors $\mathbf{L}$, for K=40 and Λ=8.

[4]When $K/\Lambda \notin \mathbb{Z}^+$, the best-case delay corresponds to having $l_\lambda = \lfloor K/\Lambda \rfloor + 1$ for $\lambda \in \left[1, 2, \cdots, \hat{K}\right]$ and $l_\lambda = \lfloor K/\Lambda \rfloor$ for $\lambda \in \left[\hat{K}+1, \hat{K}+2, \cdots, \Lambda\right]$, where $\hat{K} = K - \lfloor K/\Lambda \rfloor \Lambda$.

II. Main Results

In this section we present our main results on the performance of the K-user broadcast channel with Λ caches, each of normalized size γ, and a uniformly random user-to-cache association process. As noted, the analysis applies both to the Λ-cell HetNet, as well as to the isomorphic subpacketization-constrained setting.

A. Exact characterization of the optimal average delay

We proceed to characterize the exact optimal average delay $\overline{T}^*(\gamma)$. Crucial in this characterization will be the vector $\mathbf{B_L} = \left[b_1, b_2, \ldots, b_{|B_\mathbf{L}|}\right]$, where each element $b_j \in \mathbf{B_L}$ indicates the number of caches in a distinct group of caches in which each cache has the same load[5]. Under the assumption that each user can be associated to any particular cache with equal probability, the optimal average delay $\overline{T}^*(\gamma)$ — optimized over all coded caching strategies with uncoded placement — is given by the following theorem.

Theorem 1. *In the K-user, Λ-caches setting with normalized cache size γ and a random user-to-cache association, the average delay*

$$\overline{T}^*(\gamma) = \sum_{\lambda=1}^{\Lambda-t} \sum_{\mathbf{L}\in\mathcal{L}} \frac{K!\ t!\ (\Lambda-t)!\ l_\lambda \binom{\Lambda-\lambda}{t}}{\Lambda^K \prod_{i=1}^{\Lambda} l_i!\ \prod_{j=1}^{|\mathbf{B_L}|} b_j!} \tag{9}$$

is exactly optimal under the assumption of uncoded placement.

Proof. The proof can be found in Appendix A. □

One can now easily see that when $\frac{K}{\Lambda} \in \mathbb{Z}^+$, the optimal multiplicative deterioration $G(\gamma) = \frac{\overline{T}^*(\gamma)}{T_{min}}$ takes the form

$$G(\gamma) = \sum_{\lambda=1}^{\Lambda-t} \sum_{\mathbf{L}\in\mathcal{L}} \frac{(K-1)!\ (\Lambda-t-1)!\ (t+1)!\ l_\lambda \binom{\Lambda-\lambda}{t}}{\Lambda^{K-1} \prod_{i=1}^{\Lambda} l_i! \prod_{j=1}^{|\mathbf{B_L}|} b_j!}. \tag{10}$$

Remark 1. *Theorem 1 provides the exact optimal performance in the random association setting, as well as a more efficient way to evaluate this performance compared to the state of the art (SoA) (cf. [11, Theorem 1]). This speedup is due to the averaging being over the much smaller set $\mathcal{L}$ of all $\mathbf{L}$, rather than over the set $\mathcal{V}$ of all $\mathbf{V}$ (see Table I for a brief comparison). We note that the creation of $\mathcal{V}$ is a so-called weak composition problem, whereas the creation of $\mathcal{L}$ is an integer partition problem. It is easy to verify that the complexities of the algorithms for the integer partition problem are significantly lower than the ones for the weak composition problem [12], [13].*

B. Scaling laws of coded caching with random association

The following provides the asymptotic analysis of the optimal $\overline{T}^*(\gamma)$, in the limit of large Λ.

[5]For example, for a profile vector $\mathbf{L} = [5, 5, 3, 3, 3, 2, 1, 0, 0]$, there are five distinct groups of caches in terms of having the same load, then the corresponding vector $\mathbf{B_L} = [2, 3, 1, 1, 2]$, because two caches have a similar load of five users, three caches have a similar load of three users, two caches have a similar load of zero and all other caches have distinct number of users.

	$\lvert\mathcal{L}\rvert$	$\lvert\mathcal{V}\rvert$
$K = 10$	42	92378
$K = 20$	530	10015005
$K = 30$	3590	211915132
$K = 40$	16928	2.054455634×10^9
$K = 50$	62740	$1.2565671261 \times 10^{10}$

TABLE I: Size of $\mathcal{L}$ and $\mathcal{V}$ ($\Lambda = 10$)

Theorem 2. *In the K-user, Λ-caches setting with normalized cache size γ and random user-to-cache association, the optimal delay scales as*

$$\overline{T}^*(\gamma) = \begin{cases} \Theta\left(\frac{T_{min}\Lambda\log\Lambda}{K\log\frac{\Lambda\log\Lambda}{K}}\right) & \text{if } K \in \left[\frac{\Lambda}{\text{polylog}(\Lambda)}, o\left(\Lambda\log\Lambda\right)\right] \\ \Theta\left(T_{min}\right) & \text{if } K = \Omega\left(\Lambda\log\Lambda\right). \end{cases} \tag{11}$$

Proof. Due to lack of space, the proof is relegated to the longer version of this work [2]. □

Directly from the above, we now know that the performance deterioration due to user-to-cache association randomness, scales as

$$G(\gamma) = \begin{cases} \Theta\left(\frac{\Lambda\log\Lambda}{K\log\frac{\Lambda\log\Lambda}{K}}\right) & \text{if } K \in \left[\frac{\Lambda}{\text{polylog}(\Lambda)}, o\left(\Lambda\log\Lambda\right)\right] \\ \Theta\left(1\right) & \text{if } K = \Omega\left(\Lambda\log\Lambda\right) \end{cases} \tag{12}$$

which in turn leads to the following corollary.

Corollary 1. *The performance deterioration $G(\gamma)$ due to association randomness, scales as $\Theta\left(\frac{\log\Lambda}{\log\log\Lambda}\right)$ at $K = \Theta\left(\Lambda\right)$, and as K increases, this deterioration gradually reduces, and ceases to scale when $K = \Omega\left(\Lambda\log\Lambda\right)$.*

Proof. The proof is straightforward from Theorem 2. □

In identifying the exact scaling laws of the problem, Theorem 2 nicely captures the following points.

- It describes the extent to which the performance deterioration increases with Λ and decreases with $\frac{K}{\Lambda}$.
- It reveals that the performance deterioration can in fact be unbounded.
- It shows how in certain cases, increasing Λ may yield diminishing returns due to the associated exacerbation of the random association problem. For example, to avoid a scaling $G(\gamma)$, one must approximately keep Λ below $e^{W(K)}$ ($W(.)$ is the Lambert W-function) such that $\Lambda\log\Lambda \leq K$.

C. Furthering the state of the art on the subpacketization-constrained shared-caches setting

As mentioned before, our setting is isomorphic to the subpacketization-constrained setting recently studied in [11]. We briefly mention below the utility of our results in this latter context.

- Theorem 1 now identifies the exact optimal performance, as well as provides a more efficient way to evaluate this performance, compared to the state of the art (cf. [11, Theorem 1]). As explained before, this speedup is due

to the focus on averaging over the much smaller set $\mathcal{L}$ rather than $\mathcal{V}$ (see Remark 1).

- Theorem 2 completes our understanding of the scaling laws of the random association setting. For example, for the case where $K = \Theta(\Lambda)$, prior to our work, $G(\gamma)$ was known to scale at most as $\Theta\left(\sqrt{\Lambda}\right)$, whereas now we know that this deterioration scales exactly as $\Theta\left(\frac{\log \Lambda}{\log \log \Lambda}\right)$. Refer to Table II for a detailed comparison of the known upper bounds and our exact scaling results.

	$\overline{T}^*(\gamma)$ in [11]	$\overline{T}^*(\gamma)$ in our work
$K = \Theta(\Lambda)$	$O\left(\sqrt{\Lambda}\right)$	$\Theta\left(\frac{\log \Lambda}{\log \log \Lambda}\right)$
$K = \Theta(\Lambda^a)$ for $1 < a < 2$ and $K = \Omega(\Lambda \log \Lambda)$	$O(\Lambda^{a/2})$	$\Theta(T_{min}) = \Theta\left(\frac{K}{\Lambda}\right)$ $= \Theta(\Lambda^{a-1})$
$K = \Omega(\Lambda^2)$	$O\left(\frac{K}{\Lambda}\right)$	$\Theta(T_{min}) = \Theta\left(\frac{K}{\Lambda}\right)$

TABLE II: State of the art comparison of scaling laws.

III. Numerical Validation

We proceed to numerically evaluate $E_{\mathbf{L}}[T(\mathbf{L})]$, and subsequently numerically validate our analytical results from Section II. We use two basic evaluation approaches. The first is the basic *sampling-based numerical* (SBN) approximation method, where we generate a sufficiently large set $\mathcal{L}_1$ of randomly generated profile vectors $\mathbf{L}$, to then approximate $E_{\mathbf{L}}[T(\mathbf{L})]$ as

$$E_{\mathbf{L}}[T(\mathbf{L})] \approx \frac{1}{|\mathcal{L}_1|} \sum_{\mathbf{L} \in \mathcal{L}_1} T(\mathbf{L}), \tag{13}$$

where we recall that $T(\mathbf{L})$ is defined in (4). The corresponding approximate performance deterioration is then evaluated by dividing the above by T_{min}.

The second approach is a *threshold-based numerical* method. We first generate a set $\mathcal{L}_2 \subseteq \mathcal{L}$ of profile vectors $\mathbf{L}$ such that $\sum_{\mathbf{L} \in \mathcal{L}_2} P(\mathbf{L}) \approx \rho$, for some chosen threshold value $\rho \in [0, 1]$. Note that the closed form expression for $P(\mathbf{L})$ is given in (16) in Appendix A. Then, with this subset $\mathcal{L}_2$ at hand, we simply have the numerical lower bound (NLB)

$$E_{\mathbf{L}}[T(\mathbf{L})] \geq \sum_{\mathbf{L} \in \mathcal{L}_2} P(\mathbf{L}) T(\mathbf{L}) + (1-\rho)\, T_{min}, \tag{14}$$

by considering the best-case delay for every $\mathbf{L} \notin \mathcal{L}_2$, and similarly have the numerical upper bound (NUB)

$$E_{\mathbf{L}}[T(\mathbf{L})] \leq \sum_{\mathbf{L} \in \mathcal{L}_2} P(\mathbf{L}) T(\mathbf{L}) + (1-\rho)\, K(1-\gamma), \tag{15}$$

by considering the worst possible delay $K(1-\gamma)$ for every $\mathbf{L} \notin \mathcal{L}_2$. The bounding of $G(\gamma)$ is direct by dividing the above with T_{min}. Naturally the larger the threshold ρ, the tighter the bounds, the higher the computational cost. The additive gap between the bounds on $G(\gamma)$, takes the form $(1-\rho)\left(\frac{K(1-\gamma)}{T_{min}} - 1\right) \approx (1-\rho)\, t$, revealing the benefit of increasing ρ.

Figure 3 compares the exact $G(\gamma)$ with the sampling-based numerical (SBN) approximation in (13) for $\Lambda = 20$ and $|\mathcal{L}_1| = 10000$, where it is evident that the SBN approximation is consistent with the exact performance. Figure 4 compares the exact $G(\gamma)$ (for $\Lambda = 30$) with the threshold-based numerical bounds that are based on (14) and (15), using $\rho = 0.95$. Interestingly, the threshold-based NLB turns out to be very tight in the entire range of γ, whereas the NUB tends to move away from the exact performance as γ increases.

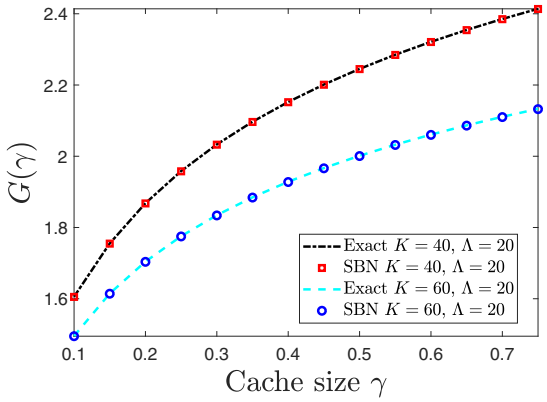

Fig. 3: Exact $G(\gamma)$ from (10) vs. sampling-based numerical (SBN) approximation from (13) ($|\mathcal{L}_1| = 10000$).

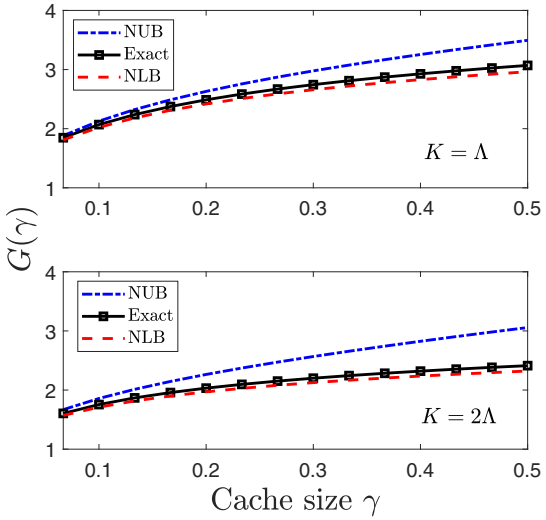

Fig. 4: Threshold-based numerical upper bound (NUB) from (15) vs. threshold-based numerical lower bound (NLB) from (14) vs. exact $G(\gamma)$ from (10) ($\Lambda = 30$ and $\rho = 0.95$).

IV. Conclusions

In this work we identified the exact optimal performance of coded caching with random user-to-cache association. In our opinion, the random association problem has direct practical ramifications, as it captures promising scenarios (such as the HetNet scenario) as well as operational realities (namely, the subpacketization constraint). The problem becomes even more

pertinent as we now know that its effect can in fact scale indefinitely.

Key to our effort to identify the effect of association randomness, has been the need to provide expressions that can either be evaluated numerically, or that can be rigorously approximated in order to yield clear insight. The first part was achieved by deriving exact expressions that can be evaluated directly or by using the proposed numerical approaches, while the second part was achieved by studying the asymptotics of the problem which yielded simple performance expressions and direct operational guidelines.

APPENDIX

A. Proof of Theorem 1

We first note that the probability $P(\mathbf{L})$ of observing a specific profile vector $\mathbf{L} \in \mathcal{L}$ is simply the cumulative probability over all $\mathbf{V}$ for which $sort(\mathbf{V}) = \mathbf{L}$. This probability takes the form

$$P(\mathbf{L}) = \overbrace{\frac{1}{\Lambda^K} \times \frac{K!}{\prod_{i=1}^{\Lambda} l_i!}}^{\text{term 1}} \times \overbrace{\frac{\Lambda!}{\prod_{j=1}^{|B_L|} b_j!}}^{\text{term 2}}. \tag{16}$$

To see this, we break down the above equation. The first term in (16) accounts for the fact that there are Λ^K different user-to-cache associations, i.e., there are Λ^K different ways that K users can be allocated to the Λ different caches. It also accounts for the fact that each user can be associated to any particular cache, with equal probability $\frac{1}{\Lambda}$. The second term in (16) indicates the total number of user-to-cache associations that leads[6] to a specific $\mathbf{V}$ for which $sort(\mathbf{V}) = \mathbf{L}$, for some fixed $\mathbf{L}$. Consequently term 1 in (16) is simply $P(\mathbf{V})$, which naturally remains fixed for any $\mathbf{V}$ for which $sort(\mathbf{V}) = \mathbf{L}$, and which originates from the well known probability mass function of the multinomial distribution. Consequently this implies that $P(\mathbf{L}) = |\{\mathbf{V} : sort(\mathbf{V}) = \mathbf{L}\}| \times P(\mathbf{V})$. Finally, term 2 describes the number of all possible cache population vectors $\mathbf{V}$ for which $sort(\mathbf{V})$ is equal to some fixed $\mathbf{L}$.

We now proceed to insert (16) into (7), which yields the average delay

$$\begin{aligned} E_{\mathbf{L}}[T(\mathbf{L})] &= \sum_{\lambda=1}^{\Lambda-t} \sum_{\mathbf{L}\in\mathcal{L}} P(\mathbf{L}) l_\lambda \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \\ &= \sum_{\lambda=1}^{\Lambda-t} \sum_{\mathbf{L}\in\mathcal{L}} \frac{l_\lambda K!\Lambda!}{\Lambda^K \prod_{i=1}^{\Lambda} l_i! \prod_{j=1}^{|\mathbf{B_L}|} b_j!} \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \\ &= \sum_{\lambda=1}^{\Lambda-t} \sum_{\mathbf{L}\in\mathcal{L}} \frac{K!\ t!\ (\Lambda-t)!\ l_\lambda \binom{\Lambda-\lambda}{t}}{\Lambda^K \prod_{i=1}^{\Lambda} l_i! \prod_{j=1}^{|\mathbf{B_L}|} b_j!}, \end{aligned} \tag{17}$$

which concludes the achievability part of the proof for the expression in Theorem 1.

[6]Recall that different user-to-cache associations can lead to the same cache population vector $\mathbf{V}$. For example, when $K = \Lambda = 3$, the following 6 user-to-cache associations, $[1, 2, 3]$, $[1, 3, 2]$, $[2, 1, 3]$, $[2, 3, 1]$, $[3, 2, 1]$, and $[3, 1, 2]$ — each describing which user is associated to which cache — in fact all correspond to the same $\mathbf{V} = [1, 1, 1]$, because always each cache is associated to one user.

Optimality of the aforementioned expression can be proved by means of the lower bound developed in [4]. We notice that the optimal delay $\overline{T}^*(\gamma)$ can be lower bounded as

$$\begin{aligned} \overline{T}^*(\gamma) &= \min_{\mathcal{X}} E_{\mathbf{L}} \left[E_{\mathbf{V_L}} \left[\max_{\mathbf{d}} T(\mathbf{V}, \mathbf{d}, \mathcal{X}) \right] \right] \\ &\geq \min_{\mathcal{X}} E_{\mathbf{L}} \left[\max_{\mathbf{d}} E_{\mathbf{V_L}} \left[T(\mathbf{V}, \mathbf{d}, \mathcal{X}) \right] \right] \\ &\geq E_{\mathbf{L}} \left[\min_{\mathcal{X}} \max_{\mathbf{d}} E_{\mathbf{V_L}} \left[T(\mathbf{V}, \mathbf{d}, \mathcal{X}) \right] \right] \\ &\geq E_{\mathbf{L}} \Big[\underbrace{\min_{\mathcal{X}} E_{\mathbf{d}\in\mathcal{D}_{wc}} E_{\mathbf{V_L}} \left[T(\mathbf{V}, \mathbf{d}, \mathcal{X}) \right]}_{T^*(\mathbf{L})} \Big] \end{aligned} \tag{18}$$

where $\mathcal{D}_{wc}$ denoted the set of demand vectors with distinct users' file-requests. Next, exploiting the fact that $P(\mathbf{V})$ is the same for any $\mathbf{V}$ for which $sort(\mathbf{V}) = \mathbf{L}$, we notice that

$$T^*(\mathbf{L}) \triangleq \min_{\mathcal{X}} E_{\mathbf{d}\in\mathcal{D}_{wc}} E_{\mathbf{V_L}} \left[T(\mathbf{V}, \mathbf{d}, \mathcal{X}) \right]$$

is lower bounded by equation (53) in [4], which then proves that $T^*(\mathbf{L})$ is bounded as

$$T^*(\mathbf{L}) \geq \sum_{\lambda=1}^{\Lambda-t} l_\lambda \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}. \tag{19}$$

This concludes the proof for the optimality of the delivery time in Theorem 1.

REFERENCES

[1] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Trans. Inf. Theory*, vol. 60, no. 5, pp. 2856–2867, May 2014.

[2] A. Malik, B. Serbetci, E. Parrinello, and P. Elia, "Fundamental limits of stochastic caching networks," 2020. [Online]. Available: https://arxiv.org/pdf/2005.13847.pdf

[3] J. Hachem, N. Karamchandani, and S. Diggavi, "Coded caching for multi-level popularity and access," *IEEE Trans. Inf. Theory*, vol. 63, no. 5, pp. 3108–3141, May 2017.

[4] E. Parrinello, A. Unsal, and P. Elia, "Fundamental limits of coded caching with multiple antennas, shared caches and uncoded prefetching," *IEEE Trans. Inf. Theory*, vol. 66, no. 4, pp. 2252–2268, Apr. 2020.

[5] 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

[6] K. Wan, D. Tuninetti, M. Ji, and G. Caire, "On the fundamental limits of fog-ran cache-aided networks with downlink and sidelink communications," 2018. [Online]. Available: https://arxiv.org/pdf/1811.05498.pdf

[7] T. X. Vu, S. Chatzinotas, and B. Ottersten, "Coded caching and storage planning in heterogeneous networks," in *Proc. IEEE Wireless Commun. and Netw. Conf. (WCNC)*, San Francisco, CA, Mar. 2017, pp. 1–6.

[8] E. Parrinello and P. Elia, "Coded caching with optimized shared-cache sizes," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Visby, Sweden, Aug. 2019, pp. 1–5.

[9] A. Sengupta and R. Tandon, "Improved approximation of storage-rate tradeoff for caching with multiple demands," *IEEE Trans. Commun.*, vol. 65, no. 5, pp. 1940–1955, May 2017.

[10] Y. Wei and S. Ulukus, "Coded caching with multiple file requests," in *Proc. 55th Annual Allerton Conf. on Commun., Cont., and Comput. (Allerton)*, Monticello, IL, Oct. 2017, pp. 437–442.

[11] S. Jin, Y. Cui, H. Liu, and G. Caire, "A new order-optimal decentralized coded caching scheme with good performance in the finite file size regime," *IEEE Trans. Commun.*, vol. 67, no. 8, pp. 5297–5310, Aug. 2019.

[12] I. Stojmenović and A. Zoghbi, "Fast algorithms for generating integer partitions," *Int. J. Comput. Math.*, vol. 70, no. 2, pp. 319–332, 1998.

[13] M. Merca, "Fast algorithm for generating ascending compositions," *J. Math. Model. and Algorithms*, vol. 11, pp. 89–104, 2012.