Coded Caching in Networks With Heterogeneous User Activity

Adeel Malik, Berksan Serbetci, Petros Elia

Coded Caching in Networks with Heterogeneous User Activity

Adeel Malik, Berksan Serbetci, Petros Elia

***Abstract*—This work elevates coded caching networks from their purely information-theoretic framework to a stochastic setting, by exploring the effect of random user activity and by exploiting correlations in the activity patterns of different users. In particular, the work studies the K-user cache-aided broadcast channel with a limited number of cache states (i.e., the content stored at the cache of a certain user), and explores the effect of cache state association strategies in the presence of arbitrary user activity levels; a combination that strikes at the very core of the coded caching problem and its crippling subpacketization bottleneck. We first present a statistical analysis of the average worst-case delay performance of such subpacketization-constrained (state-constrained) coded caching networks, and provide computationally efficient performance bounds as well as scaling laws for any arbitrary probability distribution of the user-activity levels. The achieved performance is a result of a novel user-to-cache state association algorithm that leverages the knowledge of probabilistic user-activity levels.**

We then follow a data-driven approach that exploits the prior history on user-activity levels and correlations, in order to predict interference patterns, and thus better design the caching algorithm. This optimized strategy is based on the principle that users that overlap more, interfere more, and thus have higher priority to secure complementary cache states. This strategy is proven here to be within a small constant factor from the optimal. Finally, the above analysis is validated numerically using synthetic data following the Pareto principle. To the best of our understanding, this is the first work that seeks to exploit user-activity levels and correlations, in order to map future interference and design optimized coded caching algorithms that better handle this interference.

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

I. Introduction

THE volume of mobile data traffic is rapidly growing, and soon existing networks will not have enough bandwidth resources to support this dramatically increasing demand [1]. In this context, caching offers a promising means of increasing efficiency by proactively storing part of the data at the network edge [2], including at wireless communication stations as well as on end-user devices [3], [4].

While generally caching is based on the idea that storing data can allow a receiving node to have easy access to *its own* desired file, recent work has shown the powerful effects of exploiting the existence of the aforementioned desired file *at the caches of other receiving users* [5]–[16]. In interference-limited scenarios — such as in downlink settings exemplified by the broadcast channel where each user has access to their own cache and requires their own distinct file — the findings in [5] suggest that a proper use of caching can allow for single multicast transmissions to simultaneously serve many users each having their own distinct demands. This breakthrough in the way caching is perceived, is based on the ideas of index coding which tells us that when the stored content in one user's cache overlaps with other users' requests, one can design multicast transmissions (in the form of XORs or other linear combinations of desired data), that allow for rapid delivery of any possible set of demands. Index coding — which is generally a computationally hard problem [17] — has received significant attention in a literature that has explored its performance limits [18], [19], as well as its strong connections to the network coding problem [20], [21]. One main difference between index coding and network coding is that index coding specializes on cache-related cases in the sense that it considers receivers that benefit from side-information, which in our case can be found, for example, in the caches.

Motivated by index coding and its ability to exploit receivers' side information to create coded multicasting opportunities for users requesting different files, the seminal work in [5] has introduced the concept of *coded caching*. This work revealed that — under some theoretical assumptions, and in the presence of a deterministic information-theoretic broadcast framework — the use of caching at the receivers can allow the simultaneous delivery of an unlimited number of user-requests, with a limited delay. This astounding conclusion was achieved by carefully designing a combinatorial clique-based cache placement algorithm, and a synergistic delivery scheme that enables transmitting independent content to multiple users at a time. In essence, coded caching associates each receiving user to its own cache state in a manner that allows for the custom design of a long sequence of high-capacity index coding problems that are served one after the other. Each cache state defines the content stored at the cache of a certain user. As we will see soon though, this requirement that each user has their own cache state, is an assumption that — in essence — cannot hold.

To see this, let us quickly recall that in its original setting, coded caching considers a unit-capacity single-stream broadcast channel (BC), where a transmitting base station (BS) has access to a library (catalog) of N unit-sized files, and serves K receiving users each equipped with a cache of size equal to the size of M files, or equivalently equal to a fraction $\gamma = \frac{M}{N}$ of the library. In this context, the work in [5] provides a novel placement and delivery scheme that can serve any set

Adeel Malik and Petros Elia are with the Communication Systems Department, EURECOM, 06410 Sophia Antipolis, France (e-mail: malik@eurecom.fr, elia@eurecom.fr).

B. Serbetci was with the EURECOM, 06410 Sophia Antipolis, France. He resides in Ankara, Türkiye (e-mail: fberks@gmail.com).

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).

of K simultaneous requests with a worst-case delivery time of $T = \frac{K(1-\gamma)}{1+K\gamma} \approx \frac{1-\gamma}{\gamma}$. This ability to serve a theoretically ever-increasing number of users with a bounded delay, is a direct result of exploiting the cache-enabled multicasting opportunities that allow for delivery to $K\gamma + 1$ users at a time.

As suggested above though, coded caching has a serious Achilles' heel. In particular, for the above performance to be guaranteed, coded caching requires that each user be allocated their own specifically-designed cache state (cache content), which — without delving into the esoteric details of coded caching — effectively requires the partitioning of each library-file into $\binom{K}{K\gamma}$ subpackets. This number scales exponentially in K, and thus requires files to be of truly astronomical sizes. Thus given any reasonable constraint on the file sizes, the number of cache states is effectively forced to be reduced, and the aforementioned coding gains are indeed diminished to gains that are considerably less than $K\gamma + 1$. What this file-size constraint (also known as the subpacketization bottleneck) effectively forces is the reduction of the number of cache states[1] to some $\Lambda \ll K$, which — under the basic principles of the clique-based cache-placement in [5] — allows for a smaller subpacketization level $\binom{\Lambda}{\Lambda\gamma} \ll \binom{K}{K\gamma}$ at the expense though of a much reduced coding gain $\Lambda\gamma + 1 \ll K\gamma + 1$ and a much larger delay $T = \frac{K(1-\gamma)}{1+\Lambda\gamma}$ which is now unbounded. As a consequence, this reflects the case of having the normalized cache size γ at the end users being typically very low, which is consistent with the common assumption considered in standard wireless cellular settings [7], [22]. Even though there are several alternative solutions in the literature that deal with the subpacketization bottleneck [23], [24], we consider the aforementioned setting i) for its simplicity, and ii) for its versatility, e.g., for its applicability in practical and promising coded caching enabled settings, c.f., [22].

Another important —perhaps rather oversimplified— assumption that detracts coded caching from practical settings is that an overwhelming majority of aforementioned coded-caching studies solely focus on the deterministic information-theoretic broadcast framework, which are mostly based on the assumption of deterministic topologies. We believe that it is of utmost importance to transcend this boundary, and study coded caching in more realistic wireless communication networks, which are random in nature. In this context, this work elevates coded caching from their purely information-theoretic framework to a stochastic setting where the stochasticity of the networks originates from the heterogeneity in users' request behaviors. Analyzing the coded caching in the presence of such heterogeneity can help us understand the actual gains of the coded caching in stochastic network settings. In addition, these studies are poised to play a vital role in resolving the bottlenecks of coded caching. Let us see an example of how incorporating the users' request behavior in coded caching can help us resolve the subpacketization bottleneck.

[1]This simply means that even though there are K different users, each with their own physical cache, in essence, there can only exist Λ distinct caches, that must be shared among the users. This effectively means that groups of users are forced to have identical, rather than complementary, cache contents.

Example 1. *Let us assume that we want to design a coded caching system for $K = 200$ cache-enabled users, each with a normalized storage capacity of $\gamma = 0.1$. Now consider that we study the users' request behavior, and find a pattern that users are divided into four groups, where each group consists of 50 distinct users, and at any given instance, only 50 users belonging to the same group request a file, while the other 150 users are inactive. With this knowledge, for each group, we can design a separate coded caching system. As at any instance only one of the groups is active, from [5], the worst-case delivery time takes the form $= \frac{50(1-0.1)}{1+5} = \frac{45}{6} = 7.5$, and the required subpacketization rate is $\binom{50}{5} \approx 2.12 \times 10^6$. However, if we do not exploit this knowledge, and design a single coded caching system for all $K = 200$ users, then at any instance, when only 50 users are requesting, from [11], the corresponding worst-case delivery time takes the form $\frac{\binom{200}{21}-\binom{150}{21}}{\binom{200}{20}} = 8.558$, and the required subpacketization would be $\binom{200}{20} \approx 1.61 \times 10^{27}$. We can see that exploiting the users' request behavior allows us to not only reduce the delivery time, but also reduce the required subpacketization rate by a factor of 7.6×10^{20}.*

In practice, users' request behavior will be stochastic, and we may not see such clear patterns as we observed in the simplistic scenario illustrated above. However, this example serves as a baseline to underline the need of exploring the user activity patterns in the design of coded caching in realistic wireless communication networks.

A. Exploiting the connection between coded caching, complementary cache states, user-activity levels and user-activity correlations

The performance of coded caching in the presence of an inevitably reduced number of cache states, has been explored in various works that include the work in [25] which introduced a new scheme for this setting, and the work in [26] which established the fundamental limits of the state-limited coded caching setting, by deriving the exact optimal worst-case delivery time as a function of the user-to-cache state association profile that represents the number of users served by each cache.

As we witness in the above works, in order to maintain the ability to jointly exploit multicasting opportunities, users must be associated to complementary cache states that are carefully designed and which cannot be identical. The above findings reveal that a basic problem with the state-limited scenario (where $\Lambda \ll K$) in coded caching is simply the fact that if two or more users are forced to share the same cache state (i.e., the same content in their caches), then these users generally do not have the ability to jointly receive a multicasting message that can be useful to all. Such state-limited scenario results in a large deterioration in performance, irrespective of the user-to-cache association policy. What we additionally learn from the work in [27] is that if the users are assigned to cache states at random, then this randomness imposes an additional *unbounded* performance deterioration that is a result of 'unfortunate' associations where too many

users share the same cache state. That is why the task of user-to-cache state association is important.

At the same time though, coded caching experiences a certain synchronization aspect, which is a direct outcome of the fact that users are expected to be partially asynchronous in their timing of requesting files. Hence, the notion of time is of essence. This asynchronicity has a negative aspect, but also a positive one; both of which we explore here. On the one hand, having only a fraction of the users appear simultaneously, implies a smaller number of users that can simultaneously participate in coded caching and thus implies potentially fewer multicasting opportunities and thus a smaller coding gain. On the other hand, such asynchronicity implies less instantaneous interference. This is where user activity levels come into the picture, and this is where user activity correlations can be exploited. In essence — as it will become clearer later on — any users that are correlated in terms of their activity in time, should be associated to different cache states, as this is essential in using caches for handling their mutual interference. Therefore, having the information of some users rarely request data at the same time, allows us to allocate them the same cache state resource. In essence, users that overlap more, interfere more, and thus have higher priority to secure complementary cache states. This optimization effort is particularly important because, as we recall, these resources are indeed scarce. By exploring user activities and learning from their history, we are able to predict interference patterns, and then we are able to assign cache states accordingly.

Recently, [28] presented a work to find the optimal cache placement and delivery strategies in the presence of user inactivity. This work considers identical inactivity levels for all users, i.e., at any transmission instance, each user is inactive with the same constant probability, and the placement strategy follows standard centralized and decentralized policies. The work in [28] provides an optimization-oriented delivery scheme, and compares the proposed scheme with the standard and genie-aided (i.e., assuming full knowledge of user activity in the placement phase) scheme presented in [5]. Differently from [28], our work focuses on both identical and arbitrary activity levels, and also considers a data-driven approach for user activities. We consider the subpacketization-constrained uncoded cache placement scheme, and we provide the fundamental bounds, and characterize the scaling laws of the average delay. In a nutshell, our work, to the best of our knowledge, is the first work that seeks to exploit arbitrary user-activity levels and correlations, in order to map future interference and provide optimized caching algorithms that better handle this interference by presenting the fundamental bounds and the scaling laws of the problem.

B. Notations

Throughout this paper, for n a positive integer, we use the notation $[n] \triangleq [1, 2, \dots, n], \forall n \in \mathbb{Z}^+$. We use $\mathbf{A}/\mathbf{B}$ to denote the difference set that consists of all the elements of set $\mathbf{A}$ not in set $\mathbf{B}$. Unless otherwise stated, logarithms are assumed to have base 2. We also use the following asymptotic notation: i) $f(x) = O(g(x))$ will mean that there exist constants a and c such that $f(x) \leq ag(x), \forall x > c$, ii) $f(x) = o(g(x))$ will mean that $\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$, iii) $f(x) = \Omega(g(x))$ will be used if $g(x) = O(f(x))$, iv) $f(x) = \omega(g(x))$ will mean that $\lim_{x\to\infty} \frac{g(x)}{f(x)} = 0$, and finally v) $f(x) = \Theta(g(x))$ will be used 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 $\log x$.

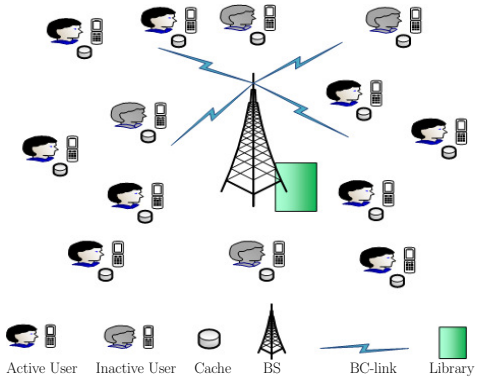

Fig. 1: An instance of a cache-aided wireless network.

II. System Model & Main Contribution

In this section, we present the system model and the communication process that consists of placement and delivery phases in detail. We then propose our metrics of interest, and finally wrap up the section by listing our contributions.

A. Network setting

We consider a cache-aided wireless network, which consists of a base station and K cache-enabled receiving users. The base station (BS) has access to a library of N equisized files $\mathcal{F} = [F_1, F_2, \dots, F_N]$ and delivers content via a broadcast link to K receiving users. Each user $k \in [K]$ is equipped with a cache of normalized storage capacity of $\gamma \triangleq \frac{M}{N} \in [0, 1]$, and requests a file from the content library with probability p_k. We use $\mathbf{p} = [p_1, p_2, \dots, p_K]$ to denote the users *activity level vector*. At any instance, if a user k is requesting a file, then we say that the user $k \in [K]$ is an *active user*. Naturally $K_{\mathbf{p}} = \sum_{k=1}^{K} p_k$ is the expected number of active users. Figure 1 depicts an instance of our cache-aided wireless network.

The communication process consists of two phases; the *placement phase* and the *delivery phase*. During the placement phase, each user's cache is filled with the content from the library, and this phase is oblivious to the upcoming number of users in the delivery phase, as well as is oblivious to the upcoming file demands. The delivery phase begins with *the active users* simultaneously requesting one file each, and continues with the BS delivering this content to the users. This phase is naturally aware of the demands of the active users, as well as is aware of the content cached at each user.

***Placement phase*:** We consider the subpacketization-constrained uncoded cache placement scheme based on [5]. The placement phase consists of two parts. In the first part, we generate the cache states based on the maximum allowable subpacketization B_{max} of a file. Each cache state defines the

content stored in the cache of a certain user. For a maximum allowable subpacketization B_{max} of a file, we define the maximum number of cache states as follows

$$\Lambda = \arg\max_{k\leq K}\left\{\binom{k}{k\gamma} \leq B_{max}\right\}.$$

Two users assigned to the same cache state must store the exact same content in their caches. We will see later that having fewer cache states generally implies a smaller DoF. Given the number of cache states Λ, each file $F_i \in \mathcal{F}$ is partitioned into $\binom{\Lambda}{t}$ distinct equisized subpackets, where $t \triangleq \Lambda\gamma$, and where $t \in [\Lambda]$ holds[2]. Then we index each subpacket of a file by a distinct subset $\tau \subseteq [\Lambda]$ of size t. The set of indexed subpackets corresponding to file $F_i \in \mathcal{F}$ is given by $\{F_{i,\tau} : \tau \subseteq [\Lambda], |\tau| = t\}$. The content corresponding to each cache state $\lambda \in [\Lambda]$ is then given by

$$C_\lambda = \{F_{i,\tau} : i \in [N], \lambda \in \tau, \tau \subseteq [\Lambda], |\tau| = t\},$$

where each cache state consists of $|C_\lambda| = N\binom{\Lambda-1}{t-1}$ subpackets, which abides by the cache-size constraint since $N\frac{\binom{\Lambda-1}{t-1}}{\binom{\Lambda}{t}} = M$.

In the second part, each user's cache is filled with the content of one of the cache states $\lambda \in [\Lambda]$. The employed user-to-cache state association is defined by a matrix $\mathbf{G} = [0,1]^{\Lambda\times K}$, of which the (λ, k) element $g_{\lambda,k}$ takes the value 1 if user k is storing the content of cache state $\lambda \in [\Lambda]$, else $g_{\lambda,k} = 0$. We denote by $\mathbf{G}_\lambda$ the set of users caching the content of cache state $\lambda \in [\Lambda]$.

***Delivery phase*:** The delivery phase commences with each active user requesting a single file from the content library. In line with the common assumptions in coded caching, we assume that requests are generated simultaneously by active users, and that each active user requests a different file. During this phase, the BS is aware of the user-to-cache state association matrix $\mathbf{G}$. Once the BS receives the users' requests, it commences delivery of the coded subpackets over a unit-capacity[3] error-free broadcast link. Here, together with the aforementioned optimal placement, we also consider the optimal[4] multi-round delivery scheme of [25], [26]. At any instance of the problem, the *cache load vector* given the user-to-cache state association $\mathbf{G}$ is denoted by $\mathbf{V} = [v_1, \ldots, v_\Lambda]$, where v_λ represents the number of active users that are associated with cache state $\lambda \in [\Lambda]$. Additionally, we use $\mathbf{L} = [l_1, \ldots, l_\Lambda] = sort(\mathbf{V})$ to be the *profile vector*, which is the sorted (in descending order) version of the cache load vector $\mathbf{V}$.

For any cache load vector $\mathbf{V}$ such that $sort(\mathbf{V}) = \mathbf{L}$, multi-round delivery scheme in [25], [26] proposes to complete the delivery in l_1 rounds, where the content is delivered to at most one user from each cache state $\lambda \in [\Lambda]$ in each round. The delivery time corresponding to each round $j \in [l_1]$ is given as $\frac{\binom{\Lambda}{t+1}-\binom{\Lambda-A_j}{t+1}}{\binom{\Lambda}{t}}$, where A_j denotes the number of users being served in delivery round j. Then, for any cache load vector $\mathbf{V}$ such that $sort(\mathbf{V}) = \mathbf{L}$, the delivery time takes the form $\sum_{j=1}^{l_1}\frac{\binom{\Lambda}{t+1}-\binom{\Lambda-A_j}{t+1}}{\binom{\Lambda}{t}}$.

Example 2. *For $K = 7$, $\Lambda = 4$, and $\gamma = 0.5$, each file is divided into $\binom{\Lambda}{\Lambda\gamma} = \binom{4}{2} = 6$ subpackets. The set of indexed subpackets corresponding to each file $F_i \in \mathcal{F}$ is given by*

$$\left\{F_{i,(1,2)}, F_{i,(1,3)}, F_{i,(1,4)}, F_{i,(2,3)}, F_{i,(2,4)}, F_{i,(3,4)}\right\}.$$

The content corresponding to each cache state is given by

$$C_1 = \{F_{i,\tau} : \tau \in [(1,2),(1,3),(1,4)], i \in [N]\},$$
$$C_2 = \{F_{i,\tau} : \tau \in [(1,2),(2,3),(2,4)], i \in [N]\},$$
$$C_3 = \{F_{i,\tau} : \tau \in [(1,3),(2,3),(3,4)], i \in [N]\},$$
$$C_4 = \{F_{i,\tau} : \tau \in [(1,4),(2,4),(3,4)], i \in [N]\}.$$

Now, let us move to the content delivery phase. Let us consider the case of $\mathbf{V} = [3,2,1,1]$, i.e., the case where users 1, 2 and 3 are associated to cache state 1, and request the content F_1, F_2, and F_3, respectively; users 4 and 5 are associated to cache state 2, and request F_4 and F_5, respectively; user 6 is associated to cache state 3, and requests F_6; and user 7 is associated to cache state 4, and requests F_7. Following the multi-round delivery scheme of [25], [26], the BS delivers the content in three rounds.

In the first round of delivery, the BS serves user 1 associated to cache state 1, user 4 associated to cache state 2, user 6 associated to cache state 3, and user 7 associated to cache state 4. For this round, the BS transmits the following four subpackets.

$$F_{1,(2,3)} \oplus F_{4,(1,3)} \oplus F_{6,(1,2)},$$
$$F_{1,(2,4)} \oplus F_{4,(1,4)} \oplus F_{7,(1,2)},$$
$$F_{1,(3,4)} \oplus F_{6,(1,4)} \oplus F_{7,(1,3)},$$
$$F_{4,(3,4)} \oplus F_{6,(2,4)} \oplus F_{7,(2,3)}.$$

After this round user 1, 4, 6, and 7 can successfully decode their requested files using the content received from the BS and the content stored in their own caches. The delivery time corresponding to this round is $\frac{\binom{4}{3}-\binom{(4-4)}{3}}{\binom{4}{2}} = \frac{4}{6}$.

Then, in the second round of delivery, the BS serves user 2 associated to cache state 1, user 5 associated to cache state 2. As there is no active user associated to cache states 3 and 4, for this round, the BS transmits the following four subpackets

$$F_{2,(2,3)} \oplus F_{5,(1,3)}, \quad F_{2,(3,4)},$$
$$F_{2,(2,4)} \oplus F_{5,(1,4)}, \quad F_{5,(3,4)}.$$

After this round user 2 and 5 can successfully decode their required file using the content received from the BS and the content stored in their own caches. The delivery time corresponding to this round is $\frac{\binom{4}{3}-\binom{(4-2)}{3}}{\binom{4}{2}} = \frac{4}{6}$

Then, in the final round of delivery, the BS only serves user 3 associated to cache state 1. As there is no active user associated to cache states 2, 3, and 4, for this round, the BS transmits the following three subpackets

$$F_{3,(2,3)}, \quad F_{3,(2,4)}, \quad F_{3,(3,4)}.$$

After this round user 1 can successfully decode the required file using the content received from the BS and the content

[2]In this work $t = \Lambda\gamma$ is a function of maximum allowable subpacketization B_{max} of a file and it represents the reduced achievable coded caching gain of $t+1 = \Lambda\gamma+1$ as compared to $K\gamma+1$ of conventional coded caching setting with no limit on the subpacketization of a file.

[3]Here the capacity is measured in units of file.

[4]Optimality here refers to the performance of the scheme over the traditional (deterministic) coded caching problem with *constant* user activity.

stored in their own cache. The delivery time corresponding to this round is $\frac{\binom{4}{3}-\binom{(4-1)}{3}}{\binom{4}{2}} = \frac{3}{6}$. *This completes the content delivery phase, which results in a delivery time of* $\frac{11}{6}$.

Remark 1. *For the case where multiple users request common files, a reduced delivery time can be obtained by using the delivery strategy proposed in [11] at each delivery round.*

B. Metrics of interest

To capture the randomness in user activity, we consider — for any given user-to-cache state association matrix $\mathbf{G}$ — the averaging metric

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

where $P(\mathbf{V})$ is the probability of $\mathbf{V}$ *given the user-to-cache state association* $\mathbf{G}$, and where $T(\mathbf{V})$ is the delivery time[5] needed to complete the delivery of requested files given a certain cache load vector $\mathbf{V}$ associated to matrix $\mathbf{G}$. For any cache load vector $\mathbf{V}$ such that $sort(\mathbf{V}) = \mathbf{L}$, the information-theoretically optimal delivery time — achieved with the multi-round delivery scheme [25], [26] — takes the form

$$T(\mathbf{L}) = \sum_{\lambda=1}^{\Lambda-t} l_\lambda \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}. \tag{2}$$

Thus, the average delay takes the form

$$\begin{aligned}\overline{T}(\mathbf{G}) = \sum_{\mathbf{L}\in\mathcal{L}} P(\mathbf{L})T(\mathbf{L}) &= \sum_{\lambda=1}^{\Lambda-t}\sum_{\mathbf{L}\in\mathcal{L}} P(\mathbf{L}) l_\lambda \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \\ &= \sum_{\lambda=1}^{\Lambda-t} E[l_\lambda]\frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}},\end{aligned} \tag{3}$$

where $\mathcal{L}$ describes the set of all possible profile vectors $\mathbf{L}$, where $P(\mathbf{L})$ is the probability of a profile vector $\mathbf{L}$ given the user-to-cache state association $\mathbf{G}$, and where $E[l_\lambda]$ is the expected number of active users in the λ-th most loaded cache, again given $\mathbf{G}$.

Our interest is in finding the optimal user-to-cache association that minimizes the average delay. This corresponds to the following optimization problem.

Problem II.1:

$$\min_{\mathbf{G}} \quad \overline{T}(\mathbf{G}) \tag{4}$$

subject to

$$\sum_{i=1}^{\Lambda} g_{i,k} = 1 \quad \forall k \in [K]. \tag{5}$$

C. Our contribution

In this work, we analyze a state-constrained coded caching network of K cache-aided users with Λ cache states, when users have different activity levels, and the association between users and cache states is subject to an arbitrary grouping strategy $\mathbf{G}$. Our aim is to provide analytical bounds on the performance. We will do so either in a manner that is numerically tractable, or in the form of asymptotic approximations that offer direct insight. The following are our contributions, step by step.

- In Section III-A, for any arbitrary user activity level vector $\mathbf{p}$ and any arbitrary user-to-cache state association $\mathbf{G}$,
  - We derive upper and lower bounds on the average delay $\overline{T}(\mathbf{G})$. These bounds can be evaluated in a computationally efficient manner.
  - We characterize the scaling laws of $\overline{T}(\mathbf{G})$ which take clear and insightful forms.
  - Based on the insights from the derived bounds, we propose a new user-to-cache association algorithm that seeks to minimize the average delay.
- In Section III-B, we analyze the special case of uniform user activity statistics, and uniform user-to-cache state association $\mathbf{G}$. For this setting, we provide analytical upper and lower bounds on the performance, and show that the bounds have a bounded gap between them and thus a bounded gap to the optimal. Then, we proceed to characterize the exact scaling laws of $\overline{T}(\mathbf{G})$.
- In Section IV, we extend our analysis to the data-driven setting, where — in designing the caching policy — we are able to learn from the past S different demand vectors. Using this bounded-depth user-request history, we propose a heuristic user-to-cache state association algorithm which is simple to implement and which we prove here to be at most at a factor of $\frac{\log S}{\log\log S}$ from the optimal. This factor, as we argue later below, remains less than 3-4 for any reasonable scenario[6], which is validated numerically using synthetic data following the Pareto principle.
- In Section V, we perform extensive numerical evaluations that validate our analysis.

III. Main Results: Statistical Approach

In this section, we present our main results on the performance of a coded caching network of K cache-aided users and Λ cache states, where the user-activity levels follow an arbitrary probability distribution $\mathbf{p}$ and where the association between users and cache states is subject to an arbitrary association strategy $\mathbf{G}$.

We can see from (3) that for any given user-to-cache state association $\mathbf{G}$ and user-activity statistics $\mathbf{p}$, the exact evaluation of (3) is computationally expensive especially for large system parameters, as the creation of $\mathcal{L}$ is an integer partition problem, and the cardinality of $\mathcal{L}$ is known to be growing exponentially with system parameters K and Λ [29].

[5]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 BS to the user, had there been no caching and no interference.

[6]If we consider a scenario where we assign caches to users once a day, and assuming that independent demand vectors appear once every 30 minutes, then the number S is at most $2 \times 24 = 48$ which implies a gap of approximately 2.3. If instead we assign caches once a week, S becomes $7 \times 2 \times 24 = 336$ and the gap is approximately 2.7. If this depth changes to a much larger $S = 12 \times 336$ corresponding to a history window of 4 weeks, and a demand vector — for those same K co-located users — every 10 minutes, then the gap is bounded at 3.3.

Motivated by this complexity, we here proceed to provide computationally efficient bounds on the performance. After doing so, we resort to asymptotic analysis of the impact of $\mathbf{G}$ and $\mathbf{p}$ on the performance, and provide an insightful characterization of the scaling laws of this performance. Finally, based on the insights from these scaling laws, we propose a heuristic user-to-cache state association algorithm that aims to minimize the worst-case delivery time.

A. Performance analysis with arbitrary activity levels

In this subsection, we present the statistical analysis of our problem for the general setting of an arbitrary user-to-cache state association strategy $\mathbf{G}$ and an arbitrary activity level vector $\mathbf{p}$. Crucial to our analysis for this setting will be the mean $\mu_\lambda = \sum_{k\in\mathbf{G}_\lambda} p_k$ and the variance $\sigma_\lambda^2 = \sum_{k\in\mathbf{G}_\lambda} p_k(1-p_k)$ of the number of active users that are caching the content of cache state $\lambda \in [\Lambda]$. Now we proceed to present our first result which is the characterization of faster-to-compute analytical bounds on the performance.

Theorem 1. *In a state-constrained coded caching network of Λ cache states, K cache-aided users with normalized cache capacity γ and activity level vector $\mathbf{p}$, the average delay $\overline{T}(\mathbf{G})$ for a given user-to-cache state association strategy $\mathbf{G}$ is bounded as follows*

$$\overline{T}(\mathbf{G}) \leq \frac{\Lambda-t}{1+t}\left(A - \sum_{x=0}^{A-1}\max\left(0, 1-\Lambda+\sum_{\lambda=1}^{\Lambda}F_1(\lambda,x)\right)\right), \quad (6)$$

$$\overline{T}(\mathbf{G}) \geq \frac{\Lambda-t}{1+t}\frac{t}{\Lambda-1}\left(A - \sum_{x=0}^{A-1}\frac{\sum_{\lambda=1}^{\Lambda}F_2(\lambda,x)}{\Lambda}\right) + \frac{\Lambda-t}{1+t}\frac{K_{\mathbf{p}}}{\Lambda}\frac{\Lambda-t-1}{\Lambda-1}, \quad (7)$$

where $t = \Lambda\gamma$, $A = \max\left(\{|\mathbf{G}_\lambda|\}_{\lambda=1}^{\Lambda}\right)$, where $\mathbf{G}_\lambda$ is the set of users caching the content of cache state λ,

$$F_1(\lambda,x) = \begin{cases} 0 & \text{if } 0 \leq x \leq \mu_\lambda - 1 \\ F_{bin}\left(|\mathbf{G}_\lambda|, \frac{\mu_\lambda}{|\mathbf{G}_\lambda|}, x\right) & \text{if } \mu_\lambda \leq x \leq |\mathbf{G}_\lambda| \\ 1 & \text{if } x > |\mathbf{G}_\lambda|, \end{cases} \quad (8)$$

$$F_2(\lambda,x) = \begin{cases} F_{bin}\left(|\mathbf{G}_\lambda|, \frac{\mu_\lambda}{|\mathbf{G}_\lambda|}, x\right) & \text{if } 0 \leq x \leq \mu_\lambda - 1 \\ 1 & \text{if } x > \mu_\lambda - 1, \end{cases} \quad (9)$$

where $F_{bin}(n,q,x) = \sum_{i=0}^{x}\binom{n}{i}q^i(1-q)^{n-i}$ and where $\mu_\lambda = \sum_{k\in\mathbf{G}_\lambda} p_k$.

Proof. The proof is deferred to Appendix A. □

Remark 2. *The bounds in Theorem 1 can be computed in a computationally-efficient manner, as for each $\lambda \in [\Lambda]$, their evaluation only requires to compute the binomial cumulative distribution function $F_{bin}\left(|\mathbf{G}_\lambda|, \frac{\mu_\lambda}{|\mathbf{G}_\lambda|}, x\right)$ for all $x \in [0,1,2,\cdots,|\mathbf{G}_\lambda|]$ of a random variable with $|\mathbf{G}_\lambda|$ independent trials and $\frac{\mu_\lambda}{|\mathbf{G}_\lambda|}$ success probability*[7].

Next, we proceed to our next result, which provides the asymptotic analysis of the average delay $\overline{T}(\mathbf{G})$, in the limit of large Λ and K. Let us quickly recall that $\mu_\lambda = \sum_{k\in G_\lambda} p_k$ and $\sigma_\lambda^2 = \sum_{k\in G_\lambda} p_k(1-p_k)$ are respectively the mean and variance of the number of active users that are associated with cache state λ.

Theorem 2. *In a state-constrained coded caching network of Λ cache states, K cache-aided users with normalized cache capacity γ and activity level vector $\mathbf{p}$, the average delay $\overline{T}(\mathbf{G})$ for a given association strategy $\mathbf{G}$ scales as*

$$\overline{T}(\mathbf{G}) = O\left(\left(\frac{K_{\mathbf{p}}}{\Lambda} + \sqrt{\sum_{i=1}^{\Lambda}(\sigma_i^2 + (\mu_i-\mu)^2)}\right)\frac{\Lambda-t}{1+t}\right), \quad (10)$$

and

$$\overline{T}(\mathbf{G}) = \Omega\left(\frac{K_{\mathbf{p}}}{\Lambda}\frac{\Lambda-t}{1+t}\right), \quad (11)$$

where $\mu = \frac{1}{\Lambda}\sum_{\lambda=1}^{\Lambda}\mu_\lambda = \frac{K_{\mathbf{p}}}{\Lambda}$.

Proof. This proof is deferred to Appendix B. □

Furthermore we have the following.

Corollary 1. *Any association strategy $\mathbf{G}$ that satisfies $\sqrt{\sum_{i=1}^{\Lambda}(\sigma_i^2 + (\mu_i-\mu)^2)} = O\left(\frac{K_{\mathbf{p}}}{\Lambda}\right)$ is order-optimal.*

Proof. Since the lower bound in (11) is independent of the association strategy $\mathbf{G}$, this implies that the optimal average delay $\overline{T}^*$ (corresponding to an optimal association $\hat{\mathbf{G}}$) is lower bounded by

$$\overline{T}^* = \Omega\left(\frac{K_{\mathbf{p}}(1-\gamma)}{1+t}\right). \quad (12)$$

Therefore any association $\mathbf{G}$ for which the gap factor $\sqrt{\sum_{i=1}^{\Lambda}(\sigma_i^2 + (\mu_i-\mu)^2)}$ scales as $O\left(\frac{K_{\mathbf{p}}}{\Lambda}\right)$ would be order optimal as the scaling order of (10) yields $O\left(\frac{K_{\mathbf{p}}(1-\gamma)}{1+t}\right)$, thus, giving the exact scaling law of $\overline{T}(\mathbf{G}) = \Theta\left(\frac{K_{\mathbf{p}}(1-\gamma)}{1+t}\right)$. □

Following the insights from Corollary 1, we now propose an algorithm that solves Problem II.1.

1) Algorithm 1:

The algorithm aims to heuristically minimize $\sum_{i=1}^{\Lambda}(\sigma_i^2 + (\mu_i-\mu)^2)$, and it works in K iterations, where for each iteration the algorithm finds a user and cache state pair $(\hat{k},\hat{\lambda})$ in accordance to step 02 of this algorithm. Consequently user $\hat{k}$ is assigned cache state $\hat{\lambda}$.

[7] In theory, $F_{bin}\left(|\mathbf{G}_\lambda|, \frac{\mu_\lambda}{|\mathbf{G}_\lambda|}, x\right)$ needs to be calculated for all values of $x \in [0, |\mathbf{G}_\lambda|]$. However, it is known that there exists a $\tilde{x} \in [0, |\mathbf{G}_\lambda|]$, where $F_{bin}\left(|\mathbf{G}_\lambda|, \frac{\mu_\lambda}{|\mathbf{G}_\lambda|}, \tilde{x}\right) \approx 1$. By De Moivre-Laplace Theorem, it is known that binomial distribution can be approximated by the normal distribution in the limit of large $|\mathbf{G}_\lambda|$, and the well-known 68–95–99.7 rule states that $\tilde{x} << |\mathbf{G}_\lambda|$. Since $F_{bin}\left(|\mathbf{G}_\lambda|, \frac{\mu_\lambda}{|\mathbf{G}_\lambda|}, x\right) \approx 1$ for any $x \geq \tilde{x}$, both (8) and (9), and consequently (6) and (7) can be quickly evaluated with high accuracy.

Algorithm 1

Input: $\mathbf{p}$, K, and Λ
Output: $\mathbf{G}$
Initialization: $\mathbf{G} \leftarrow 0$; $\mathcal{K} \leftarrow [K]$
Step 01: **while** $\mathcal{K} \neq \varnothing$ **do**
Step 02: $[\hat{\lambda}, \hat{k}] \leftarrow \arg\min_{\lambda \in [\Lambda], k \in \mathcal{K}} \sum_{i=1}^{\Lambda} (\sigma_i^2 + (\mu_i - \mu)^2)$
Step 03: $g_{\hat{\lambda},\hat{k}} \leftarrow 1$
Step 04: $\mathcal{K} \leftarrow \mathcal{K} \backslash \hat{k}$
Step 05: **end while**

In Section V, we will verify that the bounds presented in Theorem 1 are valid for any user-to-cache state association strategy. We will also show that Algorithm 1 provides an efficient user-to-cache state association that yields a performance very close to the performance of optimal user-to-cache state association.

B. Performance analysis with uniform activity level

In this subsection, we analyze a special setting where users have a uniform activity level p, corresponding to the equiprobable case of $p_1 = p_2 \cdots = p_K = p$. As is common, we will also assume that $I \triangleq \frac{K}{\Lambda}$ is an integer[8].

Lemma 1. *In the presence of uniform activity level probabilities p, the optimal user-to-cache state association policy is the uniform one where each cache state is allocated to K/Λ users.*

Proof. We first note that (3) implies that the average delay is minimized when $[E[l_1], E[l_2], \ldots, E[l_\Lambda]]$ is uniform. In the case where we have uniform activity levels, setting $|G_\lambda| = I$ $\forall \lambda \in [\Lambda]$ provides uniformity. □

We now proceed to provide computationally efficient analytical bounds on the average delay $\overline{T}(\mathbf{G})$ achieved by the uniform association policy, and subsequently to provide the exact scaling laws of this policy.

Theorem 3. *In a state-constrained coded caching network of Λ cache states, K cache-aided users with normalized cache capacity γ and activity level of p, the average delay $\overline{T}(\mathbf{G})$ corresponding to the uniform user-to-cache state association strategy $\mathbf{G}$ is bounded by*

$$\overline{T}(\mathbf{G}) \leq \frac{\Lambda - t}{1+t} E[l_1] \tag{13}$$

and

$$\overline{T}(\mathbf{G}) \geq \frac{\Lambda - t}{1+t} \left(\frac{E[l_1]t}{\Lambda - 1} + \frac{Kp}{\Lambda} \frac{\Lambda - t - 1}{\Lambda - 1} \right) \tag{14}$$

where

$$E[l_1] = I - \sum_{j=0}^{I-1} \left(\sum_{i=0}^{j} \binom{I}{i} p^i (1-p)^{I-i} \right)^{\Lambda}. \tag{15}$$

Proof. The proof is deferred to Appendix C. □

[8]In the case where $\frac{K}{\Lambda}$ is not an integer, it was shown in [26] that the optimal strategy is to set $I = \lfloor \frac{K}{\Lambda} \rfloor$, and associate the remaining $K - I\Lambda$ user cache states randomly (without replacement), with the expense of an additional delivery round.

Furthermore, the following shows that the bounds remain relatively close to the exact $\overline{T}(\mathbf{G})$.

Corollary 2. *For any fixed $\gamma \leq 1 - \frac{1}{\Lambda}$, the multiplicative gap between the analytical upper bound (AUB) in* (13) *and the analytical lower bound (ALB) in* (14)*, is at most $\frac{\Lambda - 1}{t} < 1/\gamma$. This allows us to identify the exact $\overline{T}(\mathbf{G})$ within a factor that is independent of both Λ as well as K.*

Proof. The proof follows directly from the fact that $\frac{\Lambda - t}{1+t} \frac{E[l_1]t}{\Lambda - 1} \leq \overline{T}(\mathbf{G}) \leq \frac{\Lambda - t}{1+t} E[l_1]$. □

Remark 3. *We note that the range of $\gamma \leq 1 - \frac{1}{\Lambda}$ covers in essence the entire range of γ and most certainly covers the range of pertinent γ values.*

Our proposed bounds in Theorem 3 identify that $E[l_1]$ plays the main role in defining the performance of the system. Therefore, it is sufficient to characterize the exact scaling laws of $E[l_1]$ in order to obtain the exact scaling laws of the average delay. We now proceed to exploit the bounds in Theorem 3, in order to provide in a simple and insightful form, the exact scaling laws of performance. The following theorem provides the asymptotic analysis of the average delay $\overline{T}(\mathbf{G})$, in the limit of large Λ and K.

Theorem 4. *In a coded caching setting with Λ cache states and K cache-aided users with equal cache size γ and activity level p, the average delay $\overline{T}(\mathbf{G})$ corresponding to the uniform association strategy $\mathbf{G}$ scales as*

$$\overline{T}(\mathbf{G}) = \begin{cases} \Theta\left(\frac{Kp(1-\gamma)}{1+\Lambda\gamma}\right) & \text{if } Ip = \Omega(\log\Lambda) \\ \Theta\left(\frac{Kp(1-\gamma)\log\Lambda}{(1+\Lambda\gamma)Ip\log\frac{\log\Lambda}{Ip}}\right) & \text{if } Ip \in \left[\Omega\left(\frac{1}{\text{polylog}\Lambda}\right), o(\log\Lambda)\right]. \end{cases} \tag{16}$$

Proof. The proof is deferred to Appendix D. □

In identifying the exact scaling laws of the problem, Theorem 4 nicely captures the following points.

- It highlights that randomness in users activity can bring an additional reduction in the coding caching gain.
- It reveals that coding caching gain of $1 + \Lambda\gamma$, which is already less than $1 + K\gamma$ due to subpacketization bottleneck, is only achievable when $\frac{Kp}{\Lambda} = \Omega(\log\Lambda)$.
- It shows that when $\frac{Kp}{\Lambda} = o(\log\Lambda)$, we expect an additional reduction in the coding caching gain. The extent of this reduction can now be readily computed by using (16). For example, it scales as $\Theta\left(\frac{(1+\Lambda\gamma)\log\log\Lambda}{\log\Lambda}\right)$ at $Kp = \Theta(\Lambda)$, and as Kp increases, the coding caching gradually increases, and ceases to scale when $Kp = \Omega(\Lambda\log\Lambda)$.

IV. Main Results: Data-Driven Approach

In this section, we will extend our analysis to the data-driven setting. Unlike in the previous section where we used a predetermined set of statistics $\mathbf{p}$, we will now exploit the users' content request histories to define the user activity levels as well as correlations. To proceed with our analysis we need to define the time scales involved. In our setting, the entire time horizon is equal to the time it takes between two user-to-cache associations. This time horizon will be here subdivided into

S independent time slots, where one time slot corresponds to the amount of time that elapses from the appearance of one demand vector to the next demand vector. This dynamic time refinement captures the amount of memory of the system, and will capture how far back in history we can learn from regarding user activities.

Example 3. *In a scenario where users are assigned cache states once a week, then the time frame is equal to one week which is equal to 10080 minutes. In this same example, if we assume that independent demand vectors appear once every* 10 *minutes, then the number of independent time slots S is simply $S = \frac{10080}{10} = 1008$.*

The main assumption in our data-driven approach is that each user's activity is predictable using their content request history as user's activity is highly correlated in time. For example let us suppose that we record the content request history of an office worker, and find out that the user is mostly active from 1pm to 2pm (i.e., lunch break) in the week days. Then, we conclude that we expect a similar activity pattern in the upcoming weeks from this user. Consequently, our aim will be to associate users to one of the cache states based on this prior user activity data.

In our setting, users' requests are served simultaneously, starting at the very beginning of each time slot. Any content request received during a time slot is put on hold, to be served in the beginning of the next time slot. This justifies the use of the term *dynamic duration* of each time slot $s \in [S]$, where this duration will be equal to the time needed to transmit all files that were requested during the previous time slot from the BS to the users. We can now proceed with the details of our data-driven approach.

Let $\mathbf{D} \in [0,1]^{S\times K}$ denote the *user activity matrix*, of which the (s,k) element $d_{s,k}$ is equal to 1 if user k requests content at time slot s, else $d_{s,k} = 0$. Then, for a given user-to-cache state association $\mathbf{G}$, the cache load vector for time slot $s \in [S]$ is denoted as $\mathbf{V}_s = [v_{s,1}, \ldots, v_{s,\Lambda}]$, where $v_{s,\lambda} = \sum_{k=1}^{K} g_{\lambda,k} d_{s,k}$ is the number of active users at time slot s that are storing the content of cache state $\lambda \in [\Lambda]$. The profile vector at time slot s is denoted as $\mathbf{L}_s = [l_{s,1}, \ldots, l_{s,\Lambda}]$, which is the sorted version of the cache load vector $\mathbf{V}_s$ in descending order. The average delay for a given user-to-cache state association $\mathbf{G}$ and a given user activity matrix $\mathbf{D}$, is given by

$$\overline{T}(\mathbf{G}) \triangleq \frac{1}{S}\sum_{s=1}^{S}\sum_{\lambda=1}^{\Lambda-t} l_{s,\lambda}\frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}. \tag{17}$$

Unlike in the statistical approach of Section III, where an enormous number of possible profile vectors rendered the exact calculation of $\overline{T}(\mathbf{G})$ computationally intractable, in this current data-driven setting, the calculation of $\overline{T}(\mathbf{G})$ is direct even for large system parameters. This will allow us to design an algorithm that will find a user-to-cache association policy that is provably order-optimal.

An additional difference of the proposed data-driven problem formulation is that now this formulation inherits a crucial property of exploiting users' activity correlation in time. As previously discussed, users with similar request patterns will be associated with different cache states as this would guarantee more multicasting opportunities during the delivery phase. On the other hand, users that rarely request files at the same time, can be allocated the same cache state without any performance deterioration.

We now proceed to find an order-optimal user-to-cache state association $\hat{\mathbf{G}}$ corresponding to Problem II.1. At this point we note that it is computationally intractable to brute-force solve Problem II.1 for large system parameters K, Λ and S, since there are Λ^K possible user-to-cache state associations, corresponding to an exhaustive-search computational complexity of $O\left(S\Lambda^{K+1}\right)$. Under these circumstances, the most common approach is to use computationally efficient algorithms to obtain an approximate solution that is away from the optimal solution within provable gaps. In the following subsection, we will present two such computationally efficient algorithms.

A. Computationally efficient algorithms & bounds on the performance

We start with the following lemma which lower bounds the optimal average delay $\overline{T}^*$, optimized over all policies $\mathbf{G}$.

Lemma 2. *The optimal average delay, optimized over all association policies, is lower bounded by*

$$\overline{T}^* \geq \frac{1}{S}\sum_{s\in[S]}\left(\left\lfloor\frac{d_s}{\Lambda}\right\rfloor+1\right)\frac{\Lambda-t}{1+t} - \frac{1}{S}\sum_{s\in\mathbf{S}_2}\frac{\binom{\Lambda-A_s}{t+1}}{\binom{\Lambda}{t}}, \tag{18}$$

where $d_s = \sum_{k\in[K]} d_{s,k}$, $A_s = d_s - \Lambda\left\lfloor\frac{d_s}{\Lambda}\right\rfloor$, and where $\mathbf{S}_2 \subseteq [S]$ is the set of time slots for which $A_s < \Lambda - t$.

Proof. The proof is deferred to Appendix E. □

The bound provided in Lemma 2 will serve as a benchmark for numerical performance evaluation of various user-to-cache state association algorithms.

We now proceed to present our computationally efficient algorithms. In the following, $\mathcal{G}$ will denote the set form of the user-to-cache state association matrix $\mathbf{G}$, where $(\lambda,k) \in \mathcal{G}$ if $g_{\lambda,k} = 1$. Similarly, $\mathcal{G}^{(\lambda)} = \{k : (\lambda,k) \in \mathcal{G}\}$ will denote the set of users that are storing the content of cache state $\lambda \in [\Lambda]$. Note that there is a direct correspondence between $\mathbf{G}$ and $\mathcal{G}$, and the two terms can be used interchangeably.

1) Algorithm 2:

Problem II.1 belongs to the family of well-known vector scheduling problems [30], [31], whose aim is to optimally assign each of the S-dimensional K jobs (i.e., the S-dimensional K vectors that are drawn from the columns of the user activity matrix $\mathbf{D}$) to one of the machines $\lambda \in [\Lambda]$ (i.e., cache states) with the objective of minimizing the maximum machine load (i.e., $\max_{s\in[S]} l_{s,1}$), or with the objective of minimizing the norm of the machine loads. One can see that the vector scheduling problem is the generalization of a classical load balancing problem, where each job has a vector load instead of a scalar load.

We adopt the vector scheduling algorithm of [31, Section II-B3] to find the optimal user-to-cache state association within provable gaps. **Algorithm 2** consists of three parts. The first part is the data transformation, where the user activity matrix $\mathbf{D}$ is scaled according to step 00. The second part (steps 01 to

08) is the deterministic user-to-cache state association, where for each user $k \in [K]$, we find the cache state $\hat{\lambda} \in [\Lambda]$ according to step 02. If the scaled load (cf. step 03) of cache $\hat{\lambda}$ after the assignment of user k is less than $\frac{30 \log S}{\log \log S} + 1$ for all time slots $s \in [S]$, then user k is assigned to cache state $\hat{\lambda}$. Otherwise user k is not assigned to any of the cache states, and is instead added to a set of residual users denoted by $\mathcal{K}_r$, and will be associated to a cache state later in the third part of **Algorithm 2**. The outcome of the second part is the user-to-cache state association $\mathcal{G}_1$ for users in $[K]/\mathcal{K}_r$. Next, the third part (steps 09 to 13) completes the association of the residual users in $\mathcal{K}_r$. Each user $k \in \mathcal{K}_r$ is assigned to cache $\hat{\lambda} \in [\Lambda]$ according to step 11. The outcome of this part is the user-to-cache state association $\mathcal{G}_2$ for users in $\mathcal{K}_r$. The final user-to-cache state association strategy for all users is then given by $\mathcal{G} = \mathcal{G}_1 \cup \mathcal{G}_2$.

Algorithm 2

Input: $\mathbf{D}$, K, Λ, and S
Output: $\mathcal{G}$
Initialization: $\mathcal{G}_1 \leftarrow \varnothing$; $\mathcal{G}_2 \leftarrow \varnothing$; $\mathcal{K}_r \leftarrow \varnothing$; $\alpha = \frac{10 \log S}{\log \log S}$
Step 00: $\bar{d}_{s,k} \leftarrow \min\left(\frac{\Lambda\ d_{s,k}}{\sum_{i\in[K]} d_{s,i}}, 1\right) \quad \forall\ s \in [S], k \in [K]$
Step 01: **for** k **from** 1 **to** K **do**
Step 02: $\hat{\lambda} \leftarrow \underset{\lambda\in[\Lambda]}{\operatorname{argmin}} \sum_{s=1}^{S} \sum_{\lambda=1}^{\Lambda} \left(\frac{1}{\alpha}\right)^{\frac{\alpha}{\Lambda} \sum_{i\in(\mathcal{G}_1\cup(\lambda,k))} \bar{d}_{s,i} - \sum_{j\in(\mathcal{G}_1\cup(\lambda,k))^{\lambda}} \bar{d}_{s,j}}$
Step 03: **if** $\sum_{k\in(\mathcal{G}_1\cup(\hat{\lambda},k))^{\hat{\lambda}}} \bar{d}_{s,k} < 3\alpha + 1\ \forall\ s \in [S]$
Step 04: $\mathcal{G}_1 \leftarrow \mathcal{G}_1 \cup (\hat{\lambda}, k)$
Step 05: **else**
Step 06: $\mathcal{K}_r \leftarrow \mathcal{K}_r \cup k$
Step 07: **end if**
Step 08: **end for**
Step 09: **for** c **from** 1 **to** $|\mathcal{K}_r|$ **do**
Step 10: $k = \mathcal{K}_r(c)$
Step 11: $\hat{\lambda} \leftarrow \underset{\lambda\in[\Lambda]}{\operatorname{argmin}} \left(\max_{s\in[S]} \sum_{j\in(\mathcal{G}_2\cup(\lambda,k))^{\lambda}} \bar{d}_{s,j}\right)$
Step 12: $\mathcal{G}_2 \leftarrow \mathcal{G}_2 \cup (\hat{\lambda}, k)$
Step 13: **end for**
Step 14: $\mathcal{G} \leftarrow \mathcal{G}_1 \cup \mathcal{G}_2$

Theorem 5. *When there are at least Λ requests at each time slot $s \in [S]$, the average delay $\overline{T}(\mathbf{G})$ corresponding to the user-to-cache state association $\mathbf{G}$ obtained from **Algorithm 2** is bounded by*

$$\overline{T}(\mathbf{G}) = O\left(\frac{\log S}{\log \log S} \overline{T}^{*}\right), \tag{19}$$

*which proves that **Algorithm 2** is at most a factor $O\left(\frac{\log S}{\log \log S}\right)$ from the optimal.*

Proof. The proof is deferred to Appendix F. □

Proposition 1. *The time complexity of **Algorithm 2** is $O(\Lambda^2 K S)$.*

Proof. The first part of **Algorithm 2** runs for K iterations and in each iteration, the evaluation at step 02 takes at most $\Lambda^2 S$ basic operations. Then, the second part of **Algorithm 2** runs for at most K iterations and in each iteration, the evaluation at step 11 takes at most ΛS basic operations. Thus the time complexity of **Algorithm 2** is $O(\Lambda^2 K S)$. □

Directly from above, we can see that **Algorithm 2** is significantly faster than the exhaustive search algorithm for which as we recall the time complexity was $O\left(S\Lambda^{K+1}\right)$.

2) Algorithm 3:

The main intuition behind **Algorithm 3** is to exploit the fact that both $\binom{\Lambda-\lambda}{t}$ and $l_{s,\lambda}$ are non-increasing with λ; a fact that directly follows from (17). Thus, the optimal user-to-cache state association strategy is the one that minimizes the variances of the cache load vectors $\mathbf{V}_s$ over all time slots. **Algorithm 3** aims to heuristically minimize the sum of squares of cache populations over all time slots, which is equivalent to minimizing the sum of variances of the cache load vectors over all time slots. **Algorithm 3** works in K iterations. At each iteration, it finds a pair of a user $\hat{k}$ and a cache state $\hat{\lambda}$ according to step 02 of **Algorithm 3** and assigns user $\hat{k}$ to cache state $\hat{\lambda}$.

Algorithm 3

Input: $\mathbf{D}$, K, and Λ
Output: $\mathcal{G}$
Initialization: $\mathcal{G} \leftarrow \varnothing$; $\mathcal{K} \leftarrow [K]$
Step 01: **while** $\mathcal{K} \neq \varnothing$ **do**
Step 02: $[\hat{\lambda}, \hat{k}] \leftarrow \underset{\lambda\in[\Lambda], k\in\mathcal{K}}{\arg\min} \sum_{s\in[S]} \sum_{i\in[\Lambda]} \left(\sum_{j\in(\mathcal{G}\cup(\lambda,k))^{(i)}} d_{s,j}\right)^2$
Step 03: $\mathcal{G} \leftarrow \mathcal{G} \cup (\hat{\lambda}, \hat{k})$
Step 04: $\mathcal{K} \leftarrow \mathcal{K} \backslash \hat{k}$
Step 05: **end while**

Proposition 2. *The time complexity of **Algorithm 3** is $O(\Lambda^2 K^2 S)$.*

Proof. **Algorithm 3** runs for K iterations and in each iteration, the evaluation at step 02 takes at most $K\Lambda^2 S$ basic operations. Thus the time complexity of **Algorithm 3** is $O(\Lambda^2 K^2 S)$. □

We can see that the time complexity of **Algorithm 3** is K times higher than the time complexity of **Algorithm 2**. However, in Section V we numerically show that **Algorithm 3** performs better than **Algorithm 2**.

V. Numerical Validation

In this section, we numerically validate our analytical bounds, and evaluate the performance of the different proposed user-to-cache state association algorithms.

A. Statistical approach

We first evaluate our proposed analytical bounds in Theorem 1 and Theorem 3 for the statistical setting using the *sampling-based numerical* (SBN) approximation method, where for any given $\mathbf{G}$, we generate a sufficiently large set $\mathcal{L}_1$ of randomly generated profile vectors $\mathbf{L}$ based on user activity vector $\mathbf{p}$ and where we subsequently approximate $\overline{T}(\mathbf{G})$ as

$$\overline{T}(\mathbf{G}) \approx \frac{1}{|\mathcal{L}_1|} \sum_{\mathbf{L}\in\mathcal{L}_1} T(\mathbf{L}), \tag{20}$$

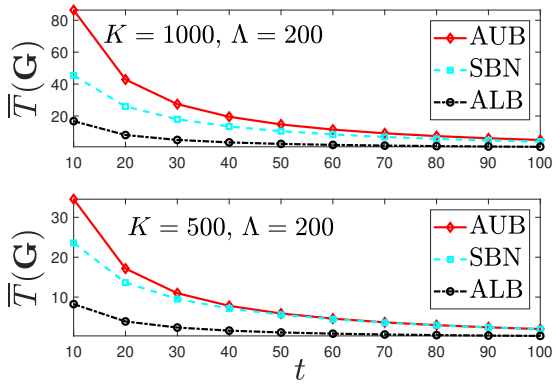

Fig. 2: Analytical upper bound (AUB) from (6) vs. analytical lower bound (ALB) from (7) vs. sampling-based numerical (SBN) approximation in (20) (for $|\mathcal{L}_1| = 20000$, $\mathbf{p}$ in (21), and random user-to-cache state association).

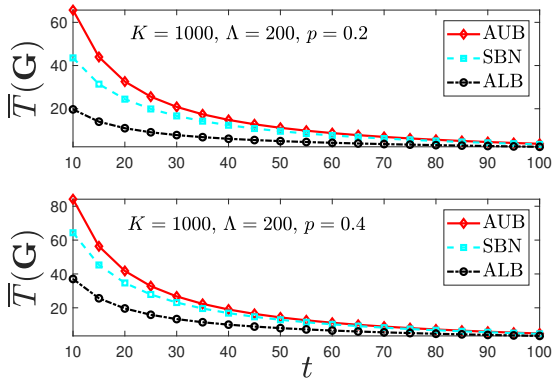

Fig. 3: Analytical upper bound (AUB) from (13) vs. analytical lower bound (ALB) from (14) vs. sampling-based numerical (SBN) approximation in (20) (for $|\mathcal{L}_1| = 20000$ and uniform user-to-cache state association).

where $T(\mathbf{L})$ is defined in (2). For our evaluations involving an arbitrary user activity level vector $\mathbf{p}$, we adopt the Pareto principle to generate the synthetic user activity level vector $\mathbf{p}$. According to the Pareto principle, 80% of consequences (content requests) come from 20% of causes (users). To be exact, each user $k \in [K]$ has a request with probability

$$p_k = \begin{cases} \frac{1}{\sum_{i=1}^{5} i^{-2.7}} & \text{if} \quad k = [1, 2, \cdots, 0.2K] \\ \frac{2^{-2.7}}{\sum_{i=1}^{5} i^{-2.7}} & \text{if} \quad k = [0.2K+1, 0.2K+2, \cdots, 0.4K] \\ \frac{3^{-2.7}}{\sum_{i=1}^{5} i^{-2.7}} & \text{if} \quad k = [0.4K+1, 0.4K+2, \cdots, 0.6K] \\ \frac{4^{-2.7}}{\sum_{i=1}^{5} i^{-2.7}} & \text{if} \quad k = [0.6K+1, 0.6K+2, \cdots, 0.8K] \\ \frac{5^{-2.7}}{\sum_{i=1}^{5} i^{-2.7}} & \text{if} \quad k = [0.8K+1, 0.8K+2, \cdots, K]. \end{cases} \tag{21}$$

The intuition behind (21) is that users are divided into 5 equipopulated groups, and the users that belong to the same group have the same activity levels.

The activity levels corresponding to these 5 groups then follow the Power law with parameter $\alpha = 2.7$, and with these carefully selected parameters, the user activity pattern satisfies the Pareto principle (80/20 rule) [32].

In Figure 2, we compare the analytical bounds in (6) and (7) for an arbitrary activity level vector $\mathbf{p}$, where this comparison uses the sampling-based numerical (SBN) approximation which is done for $|\mathcal{L}_1| = 20000$ and random user-to-cache state association. Subsequently, Figure 3 compares the analytical bounds in (13) and (14) for uniform user activity level, where again the comparison is with sampling-based numerical (SBN) approximation which is done for $|\mathcal{L}_1| = 20000$ and uniform user-to-cache state association. Both figures reveal the proposed analytical bounds to be very tight, where in particular, analytical upper bounds are indeed very close to the exact performance.

Next, we evaluate the performance of our first proposed user-to-cache state association algorithm (**Algorithm 1**) by comparing it with the numerical lower bound (NLB) on the delay $\overline{T}^*$ corresponding to the optimal user-to-cache state association $\hat{\mathbf{G}}$ of Lemma 2. Figure 4 compares SBN approximation (once again done for $|\mathcal{L}_1| = 20000$) for the user-to-cache state association obtained from **Algorithm 1** with the numerical lower bound (NLB) on $\overline{T}^*$ in (18). Again we observe that the performance corresponding to the user-to-cache state association $\mathbf{G}$ obtained from **Algorithm 1** is very close to NLB for $\overline{T}^*$.

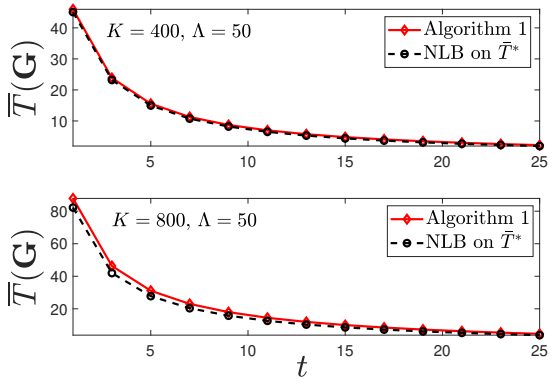

Fig. 4: SBN from (20) of **Algorithm 1** vs. Numerical lower bound (NLB) on $\overline{T}^*$ from (18) (for $|\mathcal{L}_1| = 20000$ and $\mathbf{p}$ in (21)).

Figure 5 emphasizes on how the knowing and exploiting the user activity can play a positive role in dealing with the subpacketization bottleneck of the coded caching. Figure 5 compares sampling-based numerical (SBN) approximation in (20) (for $|\mathcal{L}_1| = 20000$ and uniform user-to-cache state association) for several user activity levels. As expected, reducing the number of cache states leads to the increase in the average delay. An important observation that we can draw from this result is that the actual deterioration due to the limit on subpacketization is significant when the user activity level is high. For example when $p = 1$ (i.e., MAN setting), reducing the number of cache states from $\Lambda = 1500$ (required subpacketization of $\approx 10^{210}$) to $\Lambda = 50$ (required subpacketization of $\approx 10^{6}$) increases the delay by 25.16 times[9]. However, when $p = 0.2$, a similar reduction in the number of cache states increases the delay by only 7.4 times.

[9]Note that this is equal to the ratio between the DoFs, i.e., $151/6$.

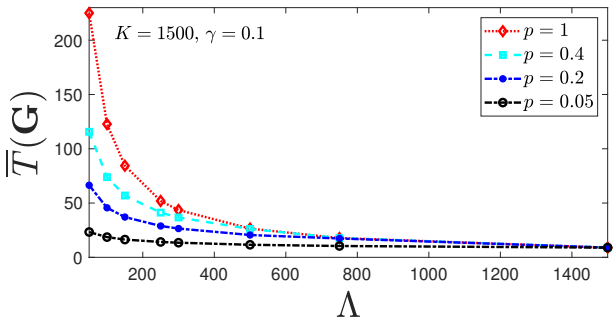

Fig. 5: Sampling-based numerical (SBN) approximation in (20) (for $|\mathcal{L}_1| = 20000$ and uniform user-to-cache state association).

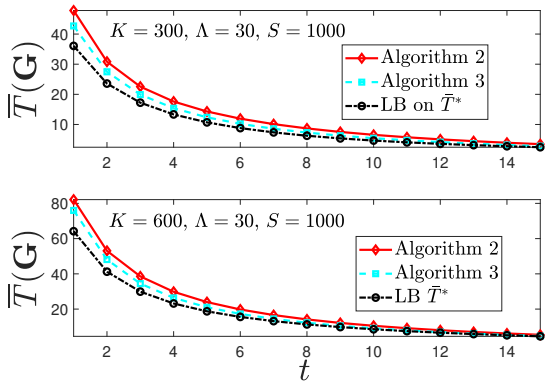

Fig. 6: $\overline{T}(\mathbf{G})$ of **Algorithm 2** and **Algorithm 3** from (17) vs. lower bound (LB) on $\overline{T}^*$ from (18).

This highlights the importance of analyzing the coded caching in the presence of such heterogeneity in order to understand the actual gains of the coded caching.

B. Data-driven approach

For the data-driven approach, we synthetically generate a user activity matrix $\mathbf{D}$ following the Pareto principle. To be exact, we assume that user $k \in [K]$ develops a request (i.e., is active) with probability p_k as in (21) at each time slot $s \in [S]$. Then, for each time slot $s \in [S]$, we pick a random number r_k between 0 and 1 for each user $k \in [K]$, and set $d_{s,k} = 1$ if $r_k \leq p_k$, and $d_{s,k} = 0$ if $r_k > p_k$, which yields a user activity matrix $\mathbf{D}$ satisfying the Pareto principle [32].

In Figure 6, we compare the average delay $\overline{T}(\mathbf{G})$ in (17) corresponding to the user-to-cache state association obtained from **Algorithm 2** and **Algorithm 3** with the lower bound (LB) on $\overline{T}^*$ in (18). It turns out that both algorithms yield performances that are over close to the optimal LB on $\overline{T}^*$, with **Algorithm 3** having a slight advantage over **Algorithm 2**.

In Figure 7, we highlight the importance of exploiting the user-activity patterns and finding an efficient user-to-cache state association. Figure 7 compares $\overline{T}(\mathbf{G})$ values for random user-to-cache state association, and the user-to-cache state associations obtained from **Algorithm 2** and **Algorithm 3**, where the lower bound (LB) of (18) serves as a benchmark.

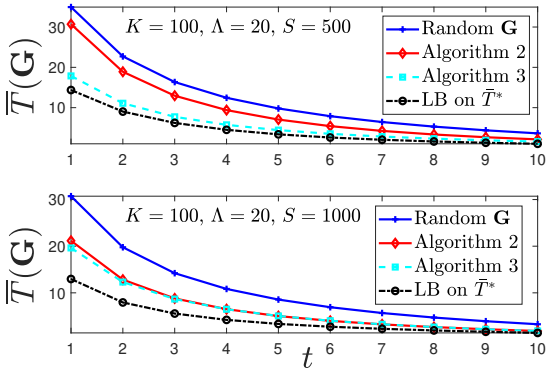

Fig. 7: $\overline{T}(\mathbf{G})$ of random user-to-cache state association, the user-to-cache state associations obtained from **Algorithm 2**, and **Algorithm 3** from (17), and the lower bound (LB) of (18).

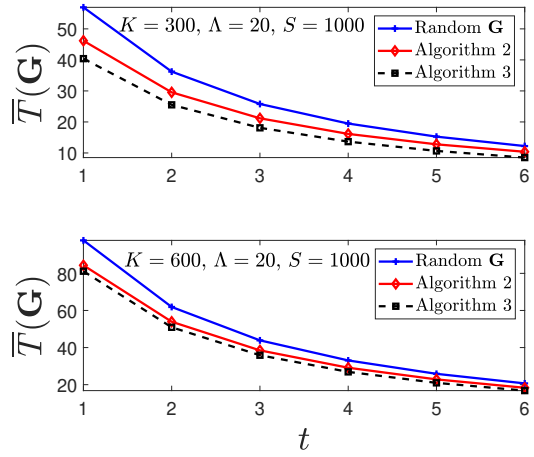

Fig. 8: $\overline{T}(\mathbf{G})$ of random user-to-cache state association, the user-to-cache state associations obtained from **Algorithm 2**, and **Algorithm 3** from (17).

It turns out that both algorithms outperform random user-to-cache state association, and the corresponding delay performances perform very close to the optimal LB on $\overline{T}^*$. If the time horizon S is subdivided into a small number of independent time slots, Algorithm 3 further outperforms Algorithm 2. As the time complexity of Algorithm 2 is $O(\Lambda^2 KS)$, and the time complexity of Algorithm 3 is $O(\Lambda^2 K^2 S)$, one can perceive a trade-off between the number of users and the reduction in the delay by adopting Algorithm 3 instead of Algorithm 2. Let us also note that increasing S to reflect having more frequent information on the dynamics of the user activity patterns, diminishes the performance gap between the two algorithms. In a nutshell, in the expense of an increase by a factor of K in the time complexity, it is worth adopting Algorithm 3 over Algorithm 2 when the time horizon is divided into a smaller number of independent time slots.

In our final evaluation, we validate our assumption that designing the user-to-cache state associations using past data can achieve similar performance superiority over random user-to-cache state association. For the following results, we generate

two user activity matrices D_{train} and D_{test} following the same Pareto principle approach defined above. We use D_{train} to obtain the user-to-cache state associations from **Algorithm 2** and **Algorithm 3**. Then, we use D_{test} to calculate the performance. Figure 8 compares $\overline{T}(\mathbf{G})$ values for random user-to-cache state association, and the user-to-cache state associations obtained from **Algorithm 2** and **Algorithm 3**. We can see that both algorithms outperform random user-to-cache state association, with once again **Algorithm 3** having a slight advantage over **Algorithm 2**.

VI. Conclusions

In this work we analyzed coded caching networks with finite number of cache states and a user-to-cache state association subject to a grouping strategy in the presence of heterogeneous user activity. Even though coded caching techniques rely on the assumption of having enough number of users to provide its theoretically promised gains, all the earlier works ignored the fact of heterogeneity in user activities, which in our opinion has direct practical ramifications, as it captures practical wireless networks more accurately.

We first presented a statistical analysis of the average worst-case delivery performance of state-constrained coded caching networks, and provided bounds and scaling laws under the assumption of probabilistic user-activity levels. We also proposed a heuristic user-to-cache state association algorithm with the ultimate goal of minimizing the average delay.

Next, we extended our analysis to the data-driven setting, where we were able to learn from the past S different demand vectors in designing the caching policy. By exploiting this bounded-depth user request history, the emphasis then was placed on finding the optimal user-to-cache state association, as computing the average delay for any given data is trivial. We proposed two algorithms for finding the optimal user-to-cache state association strategy, with the first algorithm providing the optimal within a constant gap, and with the second algorithm numerically outperforming the first one.

For both aforementioned settings, the results highlighted the essence of exploiting the user activity level, and the importance of carefully associating users to cache states based on their activity patterns.

Appendix

A. Proof of Theorem 1

Exploiting the fact that in (3), both $\binom{\Lambda-\lambda}{t}$ and $E[l_\lambda]$ are non-increasing with λ, the average delay $\overline{T}(\mathbf{G})$ is bounded by

$$\overline{T}(\mathbf{G}) \leq \sum_{\lambda=1}^{\Lambda-t} E[l_1] \frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \overset{(a)}{=} E[l_1] \frac{\binom{\Lambda}{t+1}}{\binom{\Lambda}{t}} = E[l_1] \frac{\Lambda-t}{1+t}, \quad (22)$$

and

$$\begin{aligned}\overline{T}(\mathbf{G}) &\overset{(b)}{\geq} \frac{E[l_1]\binom{\Lambda-1}{t} + \sum_{\lambda=2}^{\Lambda-t} \frac{K_{\mathbf{p}}-E[l_1]}{\Lambda-1}\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \\ &\overset{(c)}{=} E[l_1]\frac{\binom{\Lambda-1}{t}}{\binom{\Lambda}{t}} + \frac{K_{\mathbf{p}}-E[l_1]}{\Lambda-1}\frac{\binom{\Lambda-1}{t+1}}{\binom{\Lambda}{t}} \\ &= \frac{\Lambda-t}{1+t}\left(\frac{E[l_1]t}{\Lambda-1} + \frac{K_{\mathbf{p}}}{\Lambda}\frac{\Lambda-t-1}{\Lambda-1}\right), \quad (23)\end{aligned}$$

where in steps (a) and (c), we inherit the the column-sum property of Pascal's triangle yielding $\sum_{k=0}^{n}\binom{k}{t} = \binom{n+1}{t+1}$, while in step (b), we have[10] $K_{\mathbf{p}} = \sum_{\lambda=1}^{\Lambda} E[l_\lambda]$, and the fact that uniformity in $\mathbf{L}$ leads to the minimum $\overline{T}(\mathbf{G})$.

Next, to complete the proof, we proceed to derive the expected number of active users that are storing the content of the most loaded cache state (i.e., $E[l_1]$), which is given by

$$E[l_1] = \sum_{x=0}^{A-1} P[l_1 > x] = \sum_{x=0}^{A-1} \left(1 - P[l_1 \leq x]\right), \quad (24)$$

where $A = \max\left(\{|\mathbf{G}_\lambda|\}_{\lambda=1}^{\Lambda}\right)$, $\mathbf{G}_\lambda$ is the set of users caching the content of cache state λ, and $P[l_1 \leq x]$ is the probability that number of active users storing the content of the most loaded cache state are less than or equal to x. From [33, Proposition 3], we have

$$P[l_1 \leq x] \geq \max\left(0, 1 - \Lambda + \sum_{\lambda=1}^{\Lambda} F_{v_\lambda}(x)\right) \quad (25)$$

and

$$P[l_1 \leq x] \leq \frac{\sum_{\lambda=1}^{\Lambda} F_{v_\lambda}(x)}{\Lambda}, \quad (26)$$

where $F_{v_\lambda}(x)$ is the probability that no more than x users that are caching the content of cache state $\lambda \in [\Lambda]$ are active (i.e., $P[v_\lambda \leq x]$). Then $E[l_1]$ is bounded by

$$E[l_1] \leq A - \sum_{x=0}^{A-1} \max\left(0, 1 - \Lambda + \sum_{\lambda=1}^{\Lambda} F_{v_\lambda}(x)\right) \quad (27)$$

and

$$E[l_1] \geq A - \sum_{x=0}^{A-1} \frac{\sum_{\lambda=1}^{\Lambda} F_{v_\lambda}(x)}{\Lambda}. \quad (28)$$

For each cache state $\lambda \in [\Lambda]$, the corresponding random variable v_λ follows the Poisson binomial distribution. Using Hoeffding's inequalities [34, Theorem 2.1], $F_{v_i}(x)$ is bounded by

$$F_{v_\lambda}(x) \geq \begin{cases} 0 & \text{for } 0 \leq x \leq \mu_\lambda - 1 \\ F_{bin}\left(|\mathbf{G}_\lambda|, \frac{\mu_\lambda}{|\mathbf{G}_\lambda|}, x\right) & \text{for } \mu_\lambda \leq x \leq |\mathbf{G}_\lambda| \end{cases} \quad (29)$$

and

$$F_{v_\lambda}(x) \leq \begin{cases} F_{bin}\left(|\mathbf{G}_\lambda|, \frac{\mu_\lambda}{|\mathbf{G}_\lambda|}, x\right) & \text{for } 0 \leq x \leq \mu_\lambda - 1 \\ 1. & \text{for } x > \mu_\lambda - 1, \end{cases} \quad (30)$$

where $\mu_\lambda = \sum_{k \in \mathbf{G}_\lambda} p_k$ is the expected number active users that are storing the content of cache state $\lambda \in [\Lambda]$ and $F_{bin}(n, q, x) = \sum_{i=0}^{x}\binom{n}{i} q^i (1-q)^{n-i}$ is the Binomial cumulative distribution function.

Finally, the upper bound in (6) can be obtained from (22), (27), and (29); and the lower bound in (7) can be obtained from (23), (28), and (30).

[10] It straightforward to see that $\sum_{\lambda \in [\Lambda]} E[l_\lambda] = \sum_{i=0}^{K} \sum_{\mathbf{L} \in \mathcal{L}: i = \sum_{j \in [\Lambda]} l_j} \sum_{\lambda \in [\Lambda]} l_\lambda P(\mathbf{L}) = \sum_{i=0}^{K} \sum_{\mathbf{L} \in \mathcal{L}: i = \sum_{j \in [\Lambda]} l_j} i P(\mathbf{L}) = K_{\mathbf{p}}$.

B. Proof of Theorem 2

From (22) and (23), we have,

$$\overline{T}(\mathbf{G}) = O\left(E[l_1]\frac{\Lambda - t}{1+t}\right), \tag{31}$$

and

$$\overline{T}(\mathbf{G}) = \Omega\left(\frac{\Lambda - t}{1+t}\frac{E[l_1]t}{\Lambda - 1}\right). \tag{32}$$

As $\frac{t}{\Lambda-1} \approx \gamma$ is a constant, we get the exact scaling law of $\overline{T}(\mathbf{G})$, which is given by

$$\overline{T}(\mathbf{G}) = \Theta\left(E[l_1]\frac{\Lambda - t}{1+t}\right). \tag{33}$$

We know from [35, Proposition 1] that $E[l_1]$ is bounded by

$$\frac{1}{\Lambda}\sum_{\lambda=1}^{\Lambda}\mu_\lambda \leq E[l_1] \leq \frac{1}{\Lambda}\sum_{\lambda=1}^{\Lambda}\mu_\lambda + \sqrt{\frac{\Lambda-1}{\Lambda}\sum_{\lambda=1}^{\Lambda}\left(\sigma_\lambda^2 + \left(\mu_\lambda - \frac{1}{\Lambda}\sum_{\lambda=1}^{\Lambda}\mu_\lambda\right)^2\right)}, \tag{34}$$

where $\mu_\lambda = \sum_{k\in G_\lambda} p_k$ and $\sigma_\lambda^2 = \sum_{k\in G_\lambda} p_k(1-p_k)$ are the mean and the variance of the number of active users that are caching the content of cache state $\lambda \in [\Lambda]$ respectively. After defining a new parameter $\mu = \frac{1}{\Lambda}\sum_{\lambda=1}^{\Lambda}\mu_\lambda$, we have

$$\overline{T}(\mathbf{G}) = O\left(\left(\mu + \sqrt{\sum_{i=1}^{\Lambda}[\sigma_i^2 + (\mu_i - \mu)^2]}\right)\frac{\Lambda - t}{1+t}\right), \tag{35}$$

and

$$\overline{T}(\mathbf{G}) = \Omega\left(\mu\frac{\Lambda - t}{1+t}\right). \tag{36}$$

This concludes the proof of Theorem 2.

C. Proof of Theorem 3

We start our proof by deriving the expected number of active users that are storing the content of the most loaded cache state (i.e., $E[l_1]$). Assuming that each user independently requests a content with probability p, the probability that no more than x out of I users are active and storing the content of cache state $\lambda \in [\Lambda]$ is given by

$$F_{v_\lambda}(x) = \sum_{i=0}^{x}\binom{I}{i}p^i(1-p)^{I-i}. \tag{37}$$

Then, the probability that l_1 (i.e., $\max(\mathbf{V})$, the maximum number of active users among all caches) is less than or equal to j, is equal to the probability of the event $v_\lambda \leq j$, $\forall\, \lambda \in [\Lambda]$, and given by

$$P[l_1 \leq x] = \prod_{\lambda=1}^{\Lambda}F_{v_\lambda}(x) = \left(\sum_{i=0}^{x}\binom{I}{i}p^i(1-p)^{I-i}\right)^{\Lambda}. \tag{38}$$

Now, we can characterize $E[l_1]$ as follows

$$E[l_1] = \sum_{x=0}^{I-1}(1 - P[l_1 \leq x]) = I - \sum_{x=0}^{I-1}\left(\sum_{i=0}^{x}\binom{I}{i}p^i(1-p)^{I-i}\right)^{\Lambda}. \tag{39}$$

Finally, we obtain the upper and lower bounds in Theorem 3 by combining (22) and (23) with (39), respectively.

D. Proof of Theorem 4

To prove Theorem 4, we will follow a similar approach as in [36]. For each cache state $\lambda \in [\Lambda]$, we denote Y_λ to be an indicator random variable, which is equal to 1 if $v_\lambda \geq k_\alpha$, and it is equal to 0 otherwise. It immediately follows that $E[Y_\lambda] = P[v_\lambda \geq k_\alpha]$, $\forall \lambda \in [\Lambda]$. Let $Y = \sum_{\lambda=1}^{\Lambda} Y_\lambda$ be the sum of the indicators over all cache states. Then, we have

$$E[Y] = E\left[\sum_{\lambda=1}^{\Lambda}Y_\lambda\right] = \sum_{\lambda=1}^{\Lambda}E[Y_\lambda] = \Lambda P[v_\lambda \geq k_\alpha]. \tag{40}$$

From [36, Section 2], we inherit the following properties that are drawn from the outcomes of Markov's inequality and Chebyshev's inequality

$$P[Y=0] = \begin{cases} 1-o(1) & \text{if } \log(E[Y]) \to -\infty \\ o(1) & \text{if } \log(E[Y]) \to \infty. \end{cases} \tag{41}$$

Consequently, the probability that there exists at least one cache state λ for which the number of active users v_λ is at least k_α is given by

$$P[Y \geq 1] = \begin{cases} o(1) & \text{if } \log(E[Y]) \to -\infty \\ 1-o(1) & \text{if } \log(E[Y]) \to \infty. \end{cases} \tag{42}$$

We now proceed with the following results which are crucial for the derivation of the asymptotics of $E[l_1]$.

Lemma 3 ([36, Lemma 2] - adaptation)**.** *For a positive constant c, if $Ip + 1 \leq x \leq (\log I)^c$, then*

$$P[v_\lambda \geq x] = e^{x(\log Ip - \log x + 1) - Ip + O\left(log^{(2)} I\right)} \tag{43}$$

and if $x = Ip + o\left((p(1-p)I)^{\frac{2}{3}}\right)$ and $z = \frac{x - Ip}{\sqrt{p(1-p)I}}$ tends to infinity, then

$$P[v_\lambda \geq x] = e^{-\frac{z^2}{2} - \log z - \log\sqrt{2\pi} + o(1)} \tag{44}$$

Proof. The result comes directly from [36, Lemma 2]. □

Lemma 4. *In a K-user Λ-cache state setting where each user requests a content with probability p, the probability that the maximum number of active users among all caches is less than or equal to k_α, takes the form*

$$P[l_1 \geq k_\alpha] = \begin{cases} o(1) & \text{if } \alpha > 1 \\ 1 - o(1) & \text{if } 0 < \alpha < 1, \end{cases} \tag{45}$$

for

$$k_\alpha = \begin{cases} Ip + \sqrt{2\alpha Ip(1-p)\log(\Lambda)}, \text{ if } Ip = \omega\left((\log\Lambda)^3\right) \\ \left(1 + \alpha\sqrt{\frac{2\log\Lambda}{Ip}}\right)Ip, \text{ if } Ip \in [\omega(\log\Lambda), O(\text{polylog}\Lambda)] \\ (\alpha + e - 1)\,Ip, \text{ if } Ip = \Theta(\log\Lambda) \\ \frac{\log\Lambda}{\log\frac{\log\Lambda}{Ip}}\left(1 + \alpha\frac{\log^{(2)}\frac{\log\Lambda}{Ip}}{\log\frac{\log\Lambda}{Ip}}\right), \text{ if } Ip \in \left[\Omega\left(\frac{1}{\text{polylog}\Lambda}\right), o(\log\Lambda)\right]. \end{cases} \tag{46}$$

Proof. We begin the proof for the case of $Ip = \omega\left((\log\Lambda)^3\right)$. Let $k_\alpha = Ip + \sqrt{2\alpha Ip(1-p)\log\Lambda}$, then from (40) and (44), we have

$$\log(E[Y]) = \log\Lambda - \frac{z^2}{2} - \log z - \log\sqrt{2\pi} + o(1) = \log\Lambda\left(1 - \alpha - \frac{\log 2\alpha + \log^{(2)}\Lambda}{2\log\Lambda}\right) - \log\sqrt{2\pi} + o(1). \tag{47}$$

Using (42), we conclude the proof for this case as for $\Lambda \to \infty$, we have

$$\log(E[Y]) \longrightarrow \begin{cases} -\infty & \text{if } \alpha > 1 \\ \infty & \text{if } 0 < \alpha < 1. \end{cases} \tag{48}$$

Next, we proceed with the case of $Ip \in [\omega(\log\Lambda), O(\text{polylog}(\Lambda))]$. We first define $g \triangleq O(\text{polylog}(\Lambda))$. Then, assuming that $k_\alpha = \left(1+\alpha\sqrt{\frac{2}{g}}\right) Ip$ and $Ip = g\log(\Lambda)$, from (40) and (43), we have

$$\begin{aligned}
&\log(E[Y]) = \log\Lambda + k_\alpha(\log Ip - \log k_\alpha + 1) - Ip + O\left(\log^{(2)} I\right) \\
&= \log\Lambda - k_\alpha \log\left(1+\alpha\sqrt{\frac{2}{g}}\right) + k_\alpha - Ip + O\left(\log^{(2)} I\right) \\
&\overset{(a)}{=} \log\Lambda - k_\alpha\alpha\sqrt{\frac{2}{g}}\left(1-\alpha\sqrt{\frac{1}{2g}} + o\left(\alpha\sqrt{\frac{2}{g}}\right)\right) + \alpha Ip\sqrt{\frac{2}{g}} \\
&+ O\left(\log^{(2)} I\right) \\
&= \log\Lambda\left(1 - \alpha^2(1+o(1)) + (1-o(1))\,\alpha^3\sqrt{\frac{2}{g}} + O\left(\frac{\log^{(2)} I}{\log\Lambda}\right)\right),
\end{aligned} \tag{49}$$

where in step (a), we used the Maclaurin series expansion of the logarithm function, i.e., $\log(1+x) = x - 0.5x^2 + o(x^2)$. Using (42), we conclude the proof for this case as for $\Lambda \to \infty$, $\log(E[Y])$ converges to $(1-\alpha^2)\log\Lambda$, and we obtain

$$\log(E[Y]) \longrightarrow \begin{cases} -\infty & \text{if } \alpha > 1 \\ \infty & \text{if } 0 < \alpha < 1. \end{cases} \tag{50}$$

Now, we proceed with the case of $Ip = \Theta(\log\Lambda)$. Assuming that $k_\alpha = (\alpha + e - 1)\, Ip$ and $Ip = \log\Lambda$, from (40) and (43), we obtain

$$\begin{aligned}
&\log(E[Y]) = \log\Lambda + k_\alpha(\log(Ip) - \log k_\alpha + 1) - Ip + O\left(\log^{(2)} I\right) \\
&= k_\alpha(1 - \log(\alpha + e - 1)) + O\left(log^{(2)} I\right) \\
&= \log\Lambda\left((\alpha+e-1)(1-\log(\alpha+e-1)) + O\left(\frac{log^{(3)}\Lambda}{p\log\Lambda}\right)\right).
\end{aligned} \tag{51}$$

Using (42), we conclude the proof for this case as for $\Lambda \to \infty$, we have

$$\log(E[Y]) \longrightarrow \begin{cases} -\infty & \text{if } \alpha > 1 \\ \infty & \text{if } 0 < \alpha < 1 \end{cases} \tag{52}$$

Finally, we consider the case of $Ip \in \left[\Omega\left(\frac{1}{\text{polylog}\,\Lambda}\right), o(\log(\Lambda))\right]$. We first define $g \triangleq O(\text{polylog}(\Lambda))$. Then, assuming that $k_\alpha = \frac{\log\Lambda}{\log g}\left(1+\alpha\frac{\log^{(2)} g}{\log g}\right)$ and $Ip = \frac{\log(\Lambda)}{g}$, from (40) and (43), we obtain

$$\begin{aligned}
&\log(E[Y]) = \log\Lambda + k_\alpha(\log(Ip) - \log k_\alpha + 1) - Ip + O\left(\log^{(2)} I\right) \\
&= \log\Lambda + k_\alpha\left(\log^{(2)}\Lambda - \log g - \log^{(2)}\Lambda + \log^{(2)} g\right. \\
&\left. - \log\left(1+\alpha\frac{\log^{(2)} g}{\log g}\right) + 1\right) - \frac{\log(\Lambda)}{g} + O\left(\log^{(2)} I\right) \\
&\overset{(a)}{=} \log\Lambda + k_\alpha\left(1 - \log g + \log^{(2)} g - \left[\alpha\frac{\log^{(2)} g}{\log g} - \frac{1}{2}\left(\alpha\frac{\log^{(2)} g}{\log g}\right)^2\right.\right. \\
&\left.\left. + o\left(\left(\alpha\frac{\log^{(2)} g}{\log g}\right)^2\right)\right]\right) - \frac{\log(\Lambda)}{g} + O\left(\log^{(2)} I\right) \\
&= \frac{\log\Lambda\log^{(2)} g}{\log g}\left(1 - \alpha + \alpha\frac{\log^{(2)} g}{\log g} + \frac{1}{\log^{(2)} g} - \frac{\log g}{g\log^{(2)} g}\right. \\
&+ O\left(\frac{\log^{(2)} I\log g}{\log\Lambda\log^{(2)} g}\right) + \alpha^2\frac{\log^{(2)} g}{(\log g)^2}\left[-0.5 + 0.5\left(\alpha\frac{\log^{(2)} g}{\log g}\right)\right. \\
&\left.\left. - o\left(\alpha\frac{\log^{(2)} g}{\log g}\right) - o(1)\right]\right),
\end{aligned} \tag{53}$$

where in step (a), we used the Maclaurin series expansion of the logarithm function, i.e., $\log(1+x) = x - 0.5x^2 + o(x^2)$. Using (42), we conclude the proof for this case as for $\Lambda \to \infty$, $\log(E[Y])$ converges to $\frac{\log\Lambda\log^{(2)} g}{\log g}(1-\alpha)$, and we obtain

$$\log(E[Y]) \longrightarrow \begin{cases} -\infty & \text{if } \alpha > 1 \\ \infty & \text{if } 0 < \alpha < 1 \end{cases} \tag{54}$$

This concludes the proof of Lemma 4. □

With Lemma 4 at hand, we proceed to characterize $E[l_1]$. Let us first consider the case of $\alpha > 1$, for which we have

$$\begin{aligned}
E[l_1] &= \sum_{j=1}^{k_\alpha - 1} P[l_1 \geq j] + P[l_1 \geq k_\alpha] + \sum_{j=k_\alpha+1}^{I} P[l_1 \geq j] \\
&\overset{(a)}{\leq} k_\alpha - 1 + o(1) + (I - k_\alpha)o(1) = O(k_\alpha),
\end{aligned} \tag{55}$$

where in step (a), we use the fact that $P[l_1 \geq j]$ is at most 1 for $j = [1, \cdots k_\alpha - 1]$, and if $P[l_1 \geq k_\alpha] = o(1)$ then $P[l_1 \geq j]$ is at most $o(1)$ for $j = [k_\alpha + 1, \cdots I]$.

Similarly, for $0 < \alpha < 1$, we have

$$\begin{aligned}
E[l_1] &= \sum_{j=1}^{k_\alpha - 1} P[l_1 \geq j] + P[l_1 > k_\alpha] + \sum_{j=k_\alpha+1}^{I} P[l_1 \geq j] \\
&\overset{(a)}{\geq} (k_\alpha - 1)(1 - o(1)) + 1 - o(1) = \Omega(k_\alpha),
\end{aligned} \tag{56}$$

where in step (a), we use the fact that $\sum_{j=k_\alpha+1}^{I} P[l_1 \geq j] \geq 0$, and if $P[l_1 \geq k_\alpha] = 1 - o(1)$ then $P[l_1 \geq j]$ is at least $1 - o(1)$ for $j = [1, \cdots, k_\alpha - 1]$. Combining (46), (55), and (56), we have

$$E[l_1] = \begin{cases} \Theta\left(Ip + \sqrt{Ip(1-p)\log(\Lambda)}\right), & \text{if } Ip = \omega\left((\log\Lambda)^3\right) \\ \Theta\left(Ip + \sqrt{Ip\log\Lambda}\right), & \text{if } Ip \in [\Omega(\log\Lambda), O(\text{polylog}\Lambda)] \\ \Theta\left(\frac{\log\Lambda}{\log\frac{\log\Lambda}{Ip}}\right), & \text{if } Ip \in \left[\Omega\left(\frac{1}{\text{polylog}\Lambda}\right), o(\log\Lambda)\right]. \end{cases} \tag{57}$$

From (13) and (14), we have,

$$\overline{T}(\mathbf{G}) = O\left(\frac{Kp(1-\gamma)}{1+t}\frac{E[l_1]}{Ip}\right) \tag{58}$$

and

$$\overline{T}(\mathbf{G}) = \Omega\left(\frac{Kp(1-\gamma)}{1+t}\left(\frac{E[l_1]t}{Ip(\Lambda-1)}\right)\right). \tag{59}$$

As $\frac{t}{\Lambda-1} \approx \gamma$ is a constant, we get the exact scaling law of $\overline{T}(\mathbf{G})$, which is given by

$$\overline{T}(\mathbf{G}) = \Theta\left(\frac{Kp(1-\gamma)}{1+t}\frac{E[l_1]}{Ip}\right) \tag{60}$$

Combining (60) with (57), we obtain

$$\overline{T}(\mathbf{G}) = \begin{cases} \Theta\left(\frac{Kp(1-\gamma)}{1+t}\left(1+\sqrt{\frac{(1-p)\log\Lambda}{Ip}}\right)\right), \text{ if } Ip=\omega\left((\log\Lambda)^3\right) \\ \Theta\left(\frac{Kp(1-\gamma)}{1+t}\left(1+\sqrt{\frac{\log\Lambda}{Ip}}\right)\right), \text{ if } Ip\in[\Omega(\log\Lambda),O(\text{polylog}\Lambda)] \\ \Theta\left(\frac{Kp(1-\gamma)}{1+t}\frac{\log\Lambda}{Ip\log\frac{\log\Lambda}{Ip}}\right), \text{ if } Ip\in\left[\Omega\left(\frac{1}{\text{polylog}\Lambda}\right),o(\log\Lambda)\right], \end{cases} \tag{61}$$

which can be further simplified as

$$\overline{T}(\mathbf{G}) = \begin{cases} \Theta\left(\frac{Kp(1-\gamma)}{1+t}\right), \text{ if } Ip=\Omega\left(\log\Lambda\right) \\ \Theta\left(\frac{Kp(1-\gamma)}{1+t}\frac{\log\Lambda}{Ip\log\frac{\log\Lambda}{Ip}}\right), \text{ if } Ip\in\left[\Omega\left(\frac{1}{\text{polylog}\Lambda}\right),o(\log\Lambda)\right]. \end{cases} \tag{62}$$

This concludes the proof of Theorem 4.

E. Proof of Lemma 2

From (17), we know that $l_{s,\lambda}$ and $\binom{\Lambda-\lambda}{t}$ are non-increasing with λ, which implies that for each time slot $s \in [S]$, the profile vector $\mathbf{L}_s$, which minimizes the delay has components of the form

$$l_{s,\lambda} = \begin{cases} \left\lfloor \frac{d_s}{\Lambda} \right\rfloor + 1 & \text{for } \lambda \in [1,2,\ldots,A_s] \\ \left\lfloor \frac{d_s}{\Lambda} \right\rfloor & \text{for } \lambda \in [A_s+1, A_s+2, \ldots, \Lambda], \end{cases} \tag{63}$$

where $d_s = \sum_{k\in[K]} d_{s,k}$, and $A_s \triangleq d_s - \Lambda\left\lfloor \frac{d_s}{\Lambda} \right\rfloor$. Consequently, when $A_s \geq \Lambda - t$, the corresponding best-case delay T_s for time slot $s \in [S]$ is given by

$$T_s = \sum_{\lambda=1}^{\Lambda-t}\left(\left\lfloor \frac{d_s}{\Lambda} \right\rfloor + 1\right)\frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} = \left(\left\lfloor \frac{d_s}{\Lambda} \right\rfloor + 1\right)\frac{\Lambda-t}{1+t}, \tag{64}$$

while when $A_s < \Lambda - t$, this is given as

$$\begin{aligned} T_s &= \sum_{\lambda=1}^{A_s}\left(\left\lfloor \frac{d_s}{\Lambda} \right\rfloor + 1\right)\frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} + \sum_{\lambda=A_s+1}^{\Lambda-t}\left\lfloor \frac{d_s}{\Lambda} \right\rfloor\frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \\ &= \left\lfloor \frac{d_s}{\Lambda} \right\rfloor\sum_{\lambda=1}^{\Lambda-t}\frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} + \sum_{\lambda=1}^{A_s}\frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \\ &= \left(\left\lfloor \frac{d_s}{\Lambda} \right\rfloor + 1\right)\sum_{\lambda=1}^{\Lambda-t}\frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} - \sum_{\lambda=A+1}^{\Lambda-t}\frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}} \\ &= \left(\left\lfloor \frac{d_s}{\Lambda} \right\rfloor + 1\right)\frac{\binom{\Lambda}{t+1}}{\binom{\Lambda}{t}} - \frac{\binom{\Lambda-A_s}{t+1}}{\binom{\Lambda}{t}} \\ &= \left(\left\lfloor \frac{d_s}{\Lambda} \right\rfloor + 1\right)\frac{\Lambda-t}{1+t} - \frac{\binom{\Lambda-A_s}{t+1}}{\binom{\Lambda}{t}}. \end{aligned} \tag{65}$$

We denote $\mathbf{S}_2 \subseteq [S]$ to be the set of time slots for which $A_s < \Lambda - t$. Then, the average delay corresponding to the optimal user-to-cache state association $\hat{\mathbf{G}}$ is lower bounded by

$$\begin{aligned} \overline{T}^* &\geq \frac{1}{S}\sum_{s\in[S]/\mathbf{S}_2}\left(\left\lfloor \frac{d_s}{\Lambda} \right\rfloor + 1\right)\frac{\Lambda-t}{1+t} \\ &\quad + \frac{1}{S}\sum_{s\in[\mathbf{S}_2]}\left(\left(\left\lfloor \frac{d_s}{\Lambda} \right\rfloor + 1\right)\frac{\Lambda-t}{1+t} - \frac{\binom{\Lambda-A_s}{t+1}}{\binom{\Lambda}{t}}\right) \\ &= \frac{1}{S}\sum_{s\in[S]}\left(\left\lfloor \frac{d_s}{\Lambda} \right\rfloor + 1\right)\frac{\Lambda-t}{1+t} - \frac{1}{S}\sum_{s\in[\mathbf{S}_2]}\frac{\binom{\Lambda-A_s}{t+1}}{\binom{\Lambda}{t}}. \end{aligned} \tag{66}$$

This concludes the proof of Lemma 2.

F. Proof of Theorem 5

We denote $v_{s,\lambda}^{\mathcal{G}_1}$, $v_{s,\lambda}^{\mathcal{G}_2}$, and $v_{s,\lambda}^{\mathcal{G}}$ to be the scaled loads calculated using transformed user demand matrix $\mathbf{D}$ (Step 00 of **Algorithm 2**) of each cache state $\lambda \in [\Lambda]$ at time slot $s \in [S]$ following the user-to-cache state association given by $\mathcal{G}_1$, $\mathcal{G}_2$, and $\mathcal{G}$ respectively. It is straightforward to see from step 03 of **Algorithm 2** that $v_{s,\lambda}^{\mathcal{G}_1} = O\left(\frac{\log S}{\log\log S}\right)$ $\forall s \in [S],\ \lambda \in [\Lambda]$. By combining Lemma 15 and Lemma 18 of [31], we have $v_{s,\lambda}^{\mathcal{G}_2} = O(1)\ \forall s \in [S],\ \lambda \in [\Lambda]$. Thus, the combined scaled load of each cache $\lambda \in [\Lambda]$ at each time slot $s \in [S]$ is given by $v_{s,\lambda}^{\mathcal{G}} = O\left(\frac{\log S}{\log\log S}\right)$.

To complete the proof, we now proceed to convert the scaled load of each cache $\lambda \in [\Lambda]$ at time slot $s \in [S]$ to the actual load. Based on the assumption that for each time slot $s \in [S]$, $\sum_{k\in[K]} d_{s,k} \geq \Lambda$, we have $\bar{d}_{s,k} = \min\left(\frac{\Lambda\ d_{s,k}}{\sum_{i\in[K]} d_{s,i}}, 1\right) = \frac{\Lambda\ d_{s,k}}{\sum_{i\in[K]} d_{s,i}}\ \forall\ s \in [S], k \in [K]$. Then, the actual load corresponding to user-to-cache state association $\mathcal{G}$ is given by

$$v_{s,\lambda} = \frac{\sum_{k\in[K]} d_{s,k}}{\Lambda} v_{s,\lambda}^{\mathcal{G}} = O\left(\frac{\sum_{k\in[K]} d_{s,k}}{\Lambda}\frac{\log S}{\log\log S}\right)$$

$\forall\ s \in [S],\ \lambda \in [\Lambda]$. Consequently, from (17), we have

$$\overline{T}(\mathbf{G}) = O\left(\frac{1}{S}\sum_{s=1}^{S}\sum_{\lambda=1}^{\Lambda-t}\frac{\sum_{k\in[K]} d_{s,k}}{\Lambda}\frac{\log S}{\log\log S}\frac{\binom{\Lambda-\lambda}{t}}{\binom{\Lambda}{t}}\right) \tag{67}$$

$$= O\left(\frac{\log S}{\log\log S}\frac{1}{S}\sum_{s=1}^{S}\frac{\sum_{k\in[K]} d_{s,k}}{\Lambda}\frac{\Lambda-t}{1+t}\right). \tag{68}$$

We also have from (66) that

$$\overline{T}^* = \Omega\left(\frac{1}{S}\sum_{s\in[S]}\frac{\sum_{k\in[K]} d_{s,k}}{\Lambda}\frac{\Lambda-t}{1+t}\right). \tag{69}$$

This concludes the proof of Lemma 2.

REFERENCES

[1] "Cisco Annual Internet Report (2018-2023)," White paper, March 2020.

[2] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Trans. Inf. Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.

[3] N. Golrezaei, P. Mansourifard, A. F. Molisch, and A. G. Dimakis, "Base-station assisted device-to-device communications for high-throughput wireless video networks," *IEEE Trans. Wireless Commun.*, vol. 13, no. 7, pp. 3665–3676, 2014.

[4] M. Ji, G. Caire, and A. F. Molisch, "Wireless device-to-device caching networks: Basic principles and system performance," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 1, pp. 176–189, 2016.

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

[6] U. Niesen and M. A. Maddah-Ali, "Coded caching with nonuniform demands," *IEEE Trans. Inf. Theory*, vol. 63, no. 2, pp. 1146–1158, Feb 2017.

[7] K. Shanmugam, M. Ji, A. M. Tulino, J. Llorca, and A. G. Dimakis, "Finite-length analysis of caching-aided coded multicasting," *IEEE Trans. Inf. Theory*, vol. 62, no. 10, pp. 5524–5537, Oct. 2016.

[8] J. Zhang, X. Lin, and X. Wang, "Coded caching under arbitrary popularity distributions," *IEEE Trans. Inf. Theory*, vol. 64, no. 1, pp. 349–366, Jan 2018.

[9] 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*, vol. 65, no. 1, pp. 647–663, Jan 2019.
[10] K. Wan, D. Tuninetti, and P. Piantanida, "An index coding approach to caching with uncoded cache placement," *IEEE Trans. Inf. Theory*, vol. 66, no. 3, pp. 1318–1332, 2020.
[11] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "The exact rate-memory tradeoff for caching with uncoded prefetching," *IEEE Trans. Inf. Theory*, vol. 64, no. 2, pp. 1281–1296, 2018.
[12] E. Parrinello, P. Elia, and E. Lampiris, "Extending the optimality range of multi-antenna coded caching with shared caches," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, 2020, pp. 1675–1680.
[13] B. Serbetci, E. Lampiris, T. Spyropoulos, G. Caire, and P. Elia, "Multi-transmitter coded caching networks with transmitter-side knowledge of file popularity," *IEEE/ACM Trans. Netw.*, pp. 1–16, 2022.
[14] H. Zhao, A. Bazco-Nogueras, and P. Elia, "Wireless coded caching can overcome the worst-user bottleneck by exploiting finite file sizes," *IEEE Trans. Wireless Commun.*, vol. 21, no. 7, pp. 5450–5466, 2022.
[15] F. Brunero and P. Elia, "Unselfish coded caching can yield unbounded gains over selfish caching," *IEEE Trans. Inf. Theory*, vol. 68, no. 12, pp. 7871–7891, 2022.
[16] ——, "Fundamental limits of combinatorial multi-access caching," *IEEE Trans. Inf. Theory*, vol. 69, no. 2, pp. 1037–1056, 2023.
[17] M. Langberg and A. Sprintson, "On the hardness of approximating the network coding capacity," *IEEE Trans. Inf. Theory*, vol. 57, no. 2, pp. 1008–1014, 2011.
[18] Z. Bar-Yossef, Y. Birk, T. Jayram, and T. Kol, "Index coding with side information," *IEEE Trans. Inf. Theory*, vol. 57, no. 3, pp. 1479–1494, 2011.
[19] H. Maleki, V. R. Cadambe, and S. A. Jafar, "Index coding—an interference alignment perspective," *IEEE Trans. Inf. Theory*, vol. 60, no. 9, pp. 5402–5432, 2014.
[20] C. Fragouli and E. Soljanin, *Network coding fundamentals*. Now Publishers Inc, 2007.
[21] M. Effros, S. El Rouayheb, and M. Langberg, "An equivalence between network coding and index coding," *IEEE Trans. Inf. Theory*, vol. 61, no. 5, pp. 2478–2487, 2015.
[22] E. Lampiris and P. Elia, "Adding transmitters dramatically boosts coded-caching gains for finite file sizes," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 6, pp. 1176–1188, Jun. 2018.
[23] L. Tang and A. Ramamoorthy, "Coded caching schemes with reduced subpacketization from linear block codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 4, pp. 3099–3120, 2018.
[24] A. Malik, B. Serbetci, and P. Elia, "Stochastic coded caching with optimized shared-cache sizes and reduced subpacketization," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2022, pp. 2918–2923.
[25] 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.
[26] E. Parrinello, A. Ünsal, 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.
[27] A. Malik, B. Serbetci, E. Parrinello, and P. Elia, "Fundamental limits of stochastic shared-cache networks," *IEEE Trans. Commun.*, vol. 69, no. 7, pp. 4433–4447, 2021.
[28] J. Liao and O. Tirkkonen, "Fundamental rate-memory tradeoff for coded caching in presence of user inactivity," *arXiv preprint arXiv:2109.14680*, 2021.
[29] I. Stojmenović and A. Zoghbi, "Fast algorithms for generating integer partitions," *Int. J. Comput. Math.*, vol. 70, no. 2, pp. 319–332, 1998.
[30] C. Chekuri and S. Khanna, "On multidimensional packing problems," *SIAM J. Comput.*, vol. 33, no. 4, pp. 837–851, Apr. 2004.
[31] S. Im, N. Kell, J. Kulkarni, and D. Panigrahi, "Tight bounds for online vector scheduling," in *Proc. 56th Annu. Symp. Foundations Comput. Sci. (FOCS)*, Berkeley, CA, Oct. 2015, pp. 525–544.
[32] M. E. Newman, "Power laws, pareto distributions and zipf's law," *Contemporary physics*, vol. 46, no. 5, pp. 323–351, 2005.
[33] G. Caraux and O. Gascuel, "Bounds on distribution functions of order statistics for dependent variates," *Statist. Probab. Lett.*, vol. 14, no. 2, pp. 103–105, May 1992.
[34] W. Tang and F. Tang, "The poisson binomial distribution – old & new," 2019. [Online]. Available: https://arxiv.org/pdf/1908.10024.pdf
[35] O. Gascuel and G. Caraux, "Bounds on expectations of order statistics via extremal dependences," *Statist. Probab. Lett.*, vol. 15, no. 2, pp. 143–148, Sep. 1992.
[36] M. Raab and A. Steger, "Balls into bins — A simple and tight analysis," in *Proc. Int. Workshop Randomization Approx. Techn. Comput. Sci.*, 1998, pp. 159–170.

Adeel Malik received the B.S. degree in Electrical (Telecommunication) Engineering from the COMSATS Institute of Information and Technology, Pakistan, in 2013. During 2014–2016, he worked as a research assistant with Dr. Jalaluddin Qureshi on Namal College funded research projects focusing on the construction of wireless transmission protocols. In 2018, he graduated with an M.Sc. in Computer Science and Engineering from Dankook University, South Korea. He received Ph.D. from Sorbonne University, France, in 2022, while working at EURECOM's Duality project under the supervision of Prof. Petros Elia.

Berksan Serbetci received the B.Sc. degree in Electrical and Electronics Engineering from Middle East Technical University in 2009, the M.Sc. degree in Electrical and Electronics Engineering from Bogazici University in 2012, and the Ph.D. degree in Applied Mathematics from the University of Twente in 2018. From 2018 to 2022 he held a postdoctoral researcher position at EURECOM. His research interests include caching, wireless networks, optimization theory, stochastic processes, stochastic geometry, information theory and machine learning.

Petros Elia received the B.Sc. degree from the Illinois Institute of Technology, and the M.Sc. and Ph.D. degrees in electrical engineering from the University of Southern California (USC), Los Angeles, in 2001 and 2006 respectively. He is now a professor with the Department of Communication Systems at EURECOM in Sophia Antipolis, France. His latest research deals with the intersection of coded caching and feedback-aided communications in multiuser settings. He has also worked in the area of complexity-constrained communications, MIMO, queueing theory and cross-layer design, coding theory, information theoretic limits in cooperative communications, and surveillance networks. He is a Fulbright scholar, the co-recipient of the NEWCOM++ distinguished achievement award 2008-2011 for a sequence of publications on the topic of complexity in wireless communications, the recipient of the ERC Consolidator Grant 2017-2022 on cache-aided wireless communications, and the recipient of the ERC-PoC 2022-2024.