Asynchronous Coded Caching

Hooshang Ghasemi and Aditya Ramamoorthy
Department of Electrical and Computer Engineering, Iowa State University, Ames IA 50011 U.S.A.
Email: {ghasemi,adityar} @iastate.edu

***Abstract*—Coded caching is a technique that promises huge reductions in network traffic in content-delivery networks. However, the original formulation and several subsequent contributions in the area, assume that the file requests from the users are synchronized, i.e., they arrive at the server at the same time. In this work we formulate and study the coded caching problem when the file requests from the users arrive at different times. We assume that each user also has a prescribed deadline by which they want their request to be completed. In the offline case, we assume that the server knows the arrival times before starting transmission and in the online case, the user requests are revealed to the server over time. We present a LP formulation for the offline case that minimizes the overall rate subject to constraint that each user meets his/her deadline. While the online case is much harder, we demonstrate that in the case when the server wishes to minimize the overall completion time, the online solution can be as good as the offline solution. Our simulation results indicate that in the presence of mild asynchronism, much of the benefit of coded caching can still be leveraged.**

I. Introduction

Large scale content delivery over the Internet is often facilitated by the usage of caching. Conventional caching typically relies on placing popular content closer to the end users. The work of [1] considered the usage of coding in the caching problem and demonstrated that significant reductions in the induced network traffic were possible. While this is a significant result, the original formulation of the coded caching problem assumes that the user requests are synchronized, i.e., all file requests from the users arrive at the server at the same time. From a practical perspective, it is important to consider the case when the requests of the users are not synchronized; we refer to this as the asynchronous coded caching problem. In this case, a simple strategy would be to wait for the last request to arrive and then apply the scheme of [1]. Such a strategy will be quite good in terms of the overall rate of transmission from the server. However, this may be quite bad for an end user's experience, e.g., the delay experienced by the users will essentially be dominated by the arrival time of the last request.

In this work we formulate and study the coded caching problem when the user requests arrive at different times. Moreover, each user has a specific deadline by which his/her demand needs to be satisfied. We examine both the offline and online versions of this problem. In the offline version, the server knows the arrival times and deadlines of all users before starting transmission. In the online case, the server is revealed information about the arrival times and deadlines as time progresses.

This work was supported in part by the National Science Foundation by grants CCF- 1320416 and CCF- 1149860.

1) Main contributions:

- *Linear programming (LP) formulation in the offline case.* We propose a LP that determines a schedule for the equations transmitted from the server. If feasible, the schedule is such that each user meets its deadlines and the rate of transmission from the server is minimized. This allows us to study the effect of asynchronism on the coded caching rate.
- *Specific scenarios in the online case.* While the online case is much harder, we demonstrate that in general coding within subfiles of the same user is essential. Furthermore, we show that if the deadlines are loose enough, there is no difference between the rates in online and offline cases. In particular, the practically important scenario where users do not have deadlines, but wish to minimize the overall completion time by which each user is satisfied would be an instance of this.

We also present extensive simulation results that indicate that the coded caching rate degrades quite gracefully in the presence of asynchronism. Under mild asynchronism, much of the benefits of coded caching can still be leveraged.

2) Related Work: The delay sensitive coded caching problem was first studied in [2]. They considered the decentralized coded caching model, and considered a situation where each subfile has a specific deadline. Only the online case was considered and heuristics for transmission from the server were proposed. The heuristics are found to have good performance. However, the transmission time for each packet was not considered in their formulation. Our LP formulation can be viewed as a bound on the possible performance of any online scheme. The work of [3] investigated the problem of updating the cache content in the coded caching context; however synchronous file requests were considered.

II. Problem Formulation

A coded caching system contains a server with N files, denoted W_i, $i = 1, \ldots, N$, each of size F subfiles, where a subfile is a basic unit of storage. The system also contains K users each connected to the server through an error free, broadcast shared link. Each of the users is equipped with a local cache of size MF subfiles; we denote cache content of user i by Z_i which is a function of $W_1, \ldots, W_N$. The system operates in two distinct phases. In the *placement phase* the content of the caches is populated by server. This phase does not depend on the future requests of the users which are assumed to be arbitrary. In the *delivery phase* each user makes a request and the server transmits potentially coded signals

to satisfy the requests of the users. In the original work, the requests are assumed to arrive at the server at the same time.

We assume that a specific uncoded placement scheme is being used by the coded caching system. It is well recognized that the delivery phase in this case corresponds to an index coding problem [4]. While the optimal solution for an arbitrary index coding problem is known to be hard [4], techniques such as clique cover on the side information graph are well-recognized to have good performance [4]. In fact the delivery phase in [1] is precisely a clique cover on their side information graph, assuming worst case demands. Each transmitted equation is such that a certain number of users benefit from it simultaneously. We assume that our delivery phase in the asynchronous setting also transmits equations of this type.

Under these assumptions, we formulate and study the coded caching problem when the file requests come at different times. Furthermore, each user specifies a deadline by which he/she expects the request to be satisfied. We assume that transmitting a single packet over shared link takes a certain number of time slots. We study the rate gains of coded caching under this setup, i.e., among the class of strategies that allow the users to meet their deadlines, we attempt to determine those where the server transmits the fewest number of packets.

While our approach applies for general placement schemes, in this work we will describe our approach by assuming that the system works with the placement and delivery scheme of [1]. In particular, we assume a system with K users where each user has a cache of size MF subfiles. The server contains $N \geq K$ files[1] denoted $W_n, n = 1, \ldots, N$. Let $t = KM/N$ be an integer. For this system, $F = \binom{K}{t}$, i.e., each file is divided in $\binom{K}{t}$ subfiles corresponding to t-sized subsets of $[K]$. Thus file $W_n = \{W_{n,\mathcal{A}} : \mathcal{A} \subset [K], |\mathcal{A}| = t\}$. User i caches all subfiles $W_{n,\mathcal{A}}$ where $i \in \mathcal{A}$.

We assume that time $\tau \geq 0$ is slotted. Let $[n]$ denote the set $\{1, \ldots, n\}$ and the symbol $\oplus$ represent the XOR operation. We say that an equation E is of type *all-but-one* if $E = \oplus_{l=1}^{\ell} W_{d_{i_l}, \mathcal{A}_{i_l}}$ where for each $l \in [\ell]$, we have $i_l \notin \mathcal{A}_{i_l}$ and $i_l \in \mathcal{A}_{i_j}$ for all $j \in [\ell] \setminus \{l\}$. It is easy to see that the equations transmitted in the delivery phase in [1] are of the all-but-one type. With this specified placement, the asynchronous coded caching problem with deadlines can be formulated as follows.
Inputs.

- *User requests.* User i requests file W_{d_i}, with $d_i \in [N]$. User i's request arrives at the server at time T_i.
- *Deadlines.* The i-th user needs to be satisfied by time $T_i + \Delta_i$, where Δ_i is a positive integer.
- *Transmission delay.* We assume that the size of each subfile is such that it needs r time-slots to be transmitted over the shared link, i.e., each subfile can be treated as equivalent to r packets, where each packet can be transmitted in one time slot.

As the problem is symmetric with respect to users, w.l.o.g. we assume that $T_1 \leq T_2 \leq \ldots \leq T_K$. Let $T_{\max} = \max_i T_i + \Delta_i$. Note that there are upon sorting the set arrival times and deadlines, i.e., $\cup_{i=1}^{K} \{T_i, T_i + \Delta_i\}$, we can divide the interval $[T_1, T_{\max})$ into *at most* $2K - 1$ intervals. Let the integer β, where $1 \leq \beta \leq 2K - 1$ denote the number of intervals. Let $\Pi_1, \ldots, \Pi_\beta$ represent the intervals where Π_i appears before Π_j if $i < j$. The intervals are left-closed and right-open. An easy to see but very useful property of the intervals that we have defined is that for a given i, either $[T_i, T_i + \Delta_i) \cap \Pi_\ell = \Pi_\ell$ or $[T_i, T_i + \Delta_i) \cap \Pi_\ell = \emptyset$ (see Fig. 1 for an example). We define

$$U_\ell = \{i \in [K] : \ [T_i, T_i + \Delta_i) \cap \Pi_\ell = \Pi_\ell\}, \text{and}$$
$$D_\ell = \{d_i \in [N] : \ i \in U_\ell\}.$$

Thus, U_ℓ is the set of active users in time interval Π_ℓ and D_ℓ is the corresponding set of active file requests.
Outputs.

- *Transmissions at each time slot.* If the problem is feasible, the schedule specifies which equations need to be transmitted at each time. The equations need to be of the all-but-one type. The schedule is such that each user can recover all its missing subfiles within its deadline. The equations transmitted at time $\tau \in \Pi_\ell$ only depend on D_ℓ.

There is another constraint that can be modeled within this framework which has to do with integral vs. fractional solutions. In the fractional solution we assume that each packet that is transmitted over the shared edge can be subdivided as finely as needed. Thus, in each time slot we could transmit multiple equations that may serve potentially different subsets of users. This assumption is reasonable if the underlying subfiles and hence the packets are quite large. On the other hand, in the integral case, we assume that at each time slot, at most one equation can be transmitted. Thus, either the server transmits a certain packet entirely or does not transmit it; subdividing packets is disallowed. In what follows, we consider two versions of the above problem.

- *Offline version.* In the offline version, we assume that the server is aware of $\{T_i, \Delta_i, d_i\}_{i=1}^{K}$ at $\tau = 0$. However, at time $\tau \in \Pi_\ell$ the transmitted equation(s) will only depend on D_ℓ, i.e., the server cannot start sending missing subfiles for a given user until its request arrives.
- *Online version.* In the online version, information about the file requests are revealed to the server as time progresses. At each time τ, it only has information about the requests that have arrived until time τ.

III. Offline asynchronous coded caching

We present the following example to clarify the problem.

Example 1: Consider a scenario with $N = K = 3$, $M = 1$, and $r = 1$. We assume that $T_i = i$ and $\Delta_i = 2$ for $i \in \{1, 2, 3\}$. The server contains three files A, B, and C. According to placement scheme in [1], each file is divided to $\binom{K}{t} = 3$ subfiles and user i caches $Z_i = \{A_i, B_i, C_i\}$. We assume that the first user requests file A, the second user requests file B, and the third user requests file C.

An offline solution for this problem is shown in Fig. 1, where the transmitted equation in each time slot appears above the timeline. It can be observed that each user can recover the missing subfiles that they need.

[1]We assume that $N \geq K$ as it corresponds to the worst case rate and is also the more practical scenario.

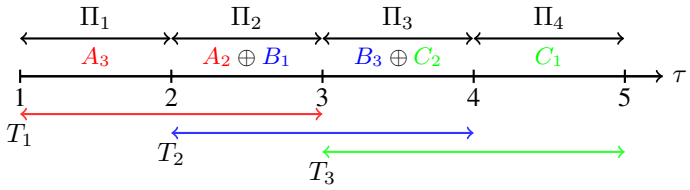

Fig. 1: Offline solution corresponding to the Example 1. The double-headed arrows show the active time slots for each user. The transmitted equations are shown above the timeline.

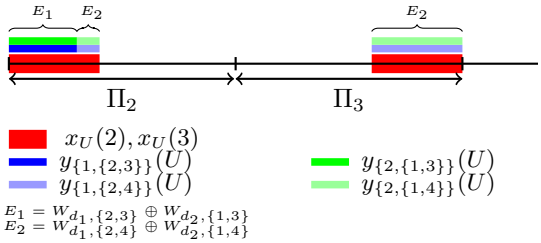

Fig. 2: Equations corresponding to the feasible solution discussed in Example 2.

A. Linear programming formulation

For ease of subsequent presentation we first define the following sets. We let $\Omega = \{\mathcal{A} : \mathcal{A} \subset [K], |\mathcal{A}| = t\}$, so that it represents the indices of all the subfiles. Let $\Omega^{(i)} = \{\mathcal{A} : \mathcal{A} \in \Omega, i \notin \mathcal{A}\}$ represent the indices that are not cached by user i. Recall that U_ℓ is the set of active users in time interval Π_ℓ and D_ℓ represents their file requests. Let $\mathcal{U}_\ell$ be set of all nonempty subsets $U \subseteq U_\ell$ with $|U| \leq t+1$. In the subsequent discussion we call such a U, a user group. In time interval Π_ℓ, a user group $U \in \mathcal{U}_\ell$ represents a collection of users that can be serviced simultaneously by the server. For any $U \subseteq [K]$, let $\mathcal{I}_U$ be the set of indices of all times intervals where users in U are simultaneously active, i.e., $\mathcal{I}_U = \{\ell : [T_i, T_i + \Delta_i) \cap \Pi_\ell \neq \emptyset, \ \forall\ i \in U\}$.

For each missing subfile $W_{d_i,\mathcal{A}}$ (where $\mathcal{A} \in \Omega^{(i)}$) we let $\mathcal{U}_{\{i,\mathcal{A}\}}$ be the set of user groups where it can be transmitted, i.e., $\mathcal{U}_{\{i,\mathcal{A}\}} = \{U \in \cup_{\ell=1}^{\beta} \mathcal{U}_\ell : \ i \in U, \ \ U \setminus \{i\} \subseteq \mathcal{A}\}$. We note here that for a fixed i, there are potentially multiple sets $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_l \in \Omega^{(i)}$ such that $U \in \mathcal{U}_{\{i,\mathcal{A}_j\}}$ for $j = 1, \ldots, l$. For example, suppose that $K = 4, t = 2$ and let $U = \{1, 2\}$. In this case $U \in \mathcal{U}_{1,\{2,3\}}$ and $U \in \mathcal{U}_{1,\{2,4\}}$. Thus, user group U can be used to potentially transmit different missing subfiles needed by user i.

We let $|\Pi_\ell|$ denote the length of the time interval Π_ℓ. For each time interval Π_ℓ with $\ell = 1, \ldots, \beta$ and for each $U \in \mathcal{U}_\ell$ we define variable $x_U(\ell) \in [0, |\Pi_\ell|]$ that represents the portion of time interval Π_ℓ that is allocated to an equation that benefits user group U. The actual equation will be determined shortly.

For each missing subfile $W_{d_i,\mathcal{A}}$ and each $U \in \mathcal{U}_{\{i,\mathcal{A}\}}$ we define variable $y_{\{i,\mathcal{A}\}}(U) \in [0, r]$ that represents the portion of the missing subfile $W_{d_i,\mathcal{A}}$ transmitted within some or all of the equations associated with $x_U(\ell)$ for $\ell \in \mathcal{I}_U$. As pointed out before, for a fixed i, U can be used to transmit different missing subfiles needed by user i. However, a single equation can only help recover one missing subfile needed by i. Thus, $\sum_{\ell \in \mathcal{I}_U} x_U(\ell)$ must be shared between the appropriate $y_{\{i,\mathcal{A}\}}(U)$'s. Accordingly, we need the following constraint for i and a user group U which contains i.

$$\sum_{\mathcal{A}:\ i \notin \mathcal{A},\ U \setminus \{i\} \subseteq \mathcal{A}} y_{\{i,\mathcal{A}\}}(U) \leq \sum_{\ell \in \mathcal{I}_U} x_U(\ell).$$

In addition, at time interval Π_ℓ at most $|\Pi_\ell|$ packets can be transmitted, so that $\sum_{U \subseteq \mathcal{U}_\ell} x_U(\ell) \leq |\Pi_\ell|$. To ensure that each missing subfile $W_{d_i,\mathcal{A}}$ is transmitted in exactly r time slots we have $\sum_{U \in \mathcal{U}_{\{i,\mathcal{A}\}}} y_{\{i,\mathcal{A}\}}(U) = r$. Finally, we have the following LP that minimizes the overall rate of transmission from the server.

$$\min \sum_{\ell=1}^{\beta} \sum_{U \in \mathcal{U}_\ell} x_U(\ell) \tag{1}$$

$$\text{s.t.} \sum_{U \in \mathcal{U}_\ell} x_U(\ell) \leq |\Pi_\ell|, \qquad \forall\ \ell = 1, \ldots, \beta,$$

$$\sum_{\mathcal{A} \in \mathcal{B}_{\{i,U\}}} y_{\{i,\mathcal{A}\}}(U) \leq \sum_{\ell \in \mathcal{I}_U} x_U(\ell), \quad \forall\ U \in \mathcal{V}_i,\ \forall i \in [K],$$

$$\sum_{U \in \mathcal{U}_{\{i,\mathcal{A}\}}} y_{\{i,\mathcal{A}\}}(U) = r, \qquad \forall\ \mathcal{A} \in \Omega^{(i)},\ \forall i \in [K],$$

$$0 \leq x_U(\ell) \leq |\Pi_\ell|, \text{ and}$$

$$0 \leq y_{\{i,\mathcal{A}\}}(U) \leq r.$$

where $\mathcal{V}_i = \{U \in \cup_{\ell=1}^{\beta} \mathcal{U}_\ell : \ i \in U \text{ and } |U| \leq t+1\}$ and $\mathcal{B}_{\{i,U\}} = \{\mathcal{A} : \mathcal{A} \in \Omega^{(i)}, U \in \mathcal{V}_i, U \setminus \{i\} \subseteq \mathcal{A}\}$.

1) Interpretation of LP solution: We claim that the optimization problem in (1) gives us a fractional solution for the offline problem. Towards this end, we demonstrate that a feasible solution can be used to determine the appropriate equations that need to be transmitted at each time. The equations are such that the users can recover all their missing subfiles within their deadlines. For $i \in U$, suppose that $\mathcal{A}_1, \ldots, \mathcal{A}_l \in \mathcal{B}_{\{i,U\}}$ are such that $y_{\{i,\mathcal{A}_j\}}(U) \neq 0$ for $j = 1, \ldots, l$. This implies that missing subfiles $W_{d_i,\mathcal{A}_1}, \ldots, W_{d_i,\mathcal{A}_l}$ will be transmitted in part by user group U. Since, $\sum_{j=1}^{l} y_{\{i,\mathcal{A}_j\}}(U) \leq \sum_{\ell \in \mathcal{I}_U} x_U(\ell)$, it is possible to assign the $y_{\{i,\mathcal{A}_j\}}(U)$'s to the available (strictly) positive $x_U(\ell)$'s, such that there is no overlap between them. This assignment is in general not unique, but non-uniqueness does not affect our overall solution. This assignment process is repeated for each member of U. Example 2 discusses this in more detail.

At the end of this process, for a fixed user group U, we obtain an assignment of the different $y_{\{i,\mathcal{A}\}}(U)$ (for appropriate $\{i, \mathcal{A}\}$) to the corresponding $x_U(\ell)$'s. The equation transmitted on the particular interval is simply the XOR of subfile indices that map to that interval. This is also shown in Fig. 2. The equation is definitely valid since the missing subfile $W_{d_i,\mathcal{A}_j}$ is such that $U \setminus \{i\} \subseteq \mathcal{A}_j$, thus each user $l \in U \setminus \{i\}$ has this missing subfile in its cache.

The entire solution can be constructed by repeating this process for each user group U. According to the third constraint $\sum_{U \in \mathcal{U}_{\{i,\mathcal{A}\}}} y_{\{i,\mathcal{A}\}}(U) = r$. This ensures that all missing subfiles are transmitted.

The following example demonstrates how equations corresponding to a feasible solution can be determined.

Example 2: We assume that the solution is such that for $U = \{1, 2\}$ with $\mathcal{I}_U = \{2, 3\}$ we have $x_U(2) = x_U(3) = 0.4$, $y_{\{1,\{2,3\}\}}(U) = y_{\{2,\{1,3\}\}}(U) = 0.3$, and $y_{\{1,\{2,4\}\}}(U) = y_{\{2,\{1,4\}\}}(U) = 0.5$.

Fig. 2 visually demonstrates the equation for user group U. Portion of time-intervals Π_2 and Π_3 that are allocated to user group U are shown in Fig. 2 by thick red lines. Then, for $i = 1 \in U$ we allocate $y_{\{1,\{2,3\}\}}(U) = 0.3$ to $x_U(2)$, i.e., missing subfile $W_{d_1,\{2,3\}}$ will be transmitted in part in this time (see the dark blue line in Fig. 2). Furthermore, we allocate remaining part of $x_U(2)$ and the entire $x_U(3) = 0.4$ to $y_{\{1,\{2,4\}\}}(U) = 0.5$, i.e., missing subfile $W_{d_1,\{2,4\}}$ will be transmitted in part in this time (light blue line in Fig. 2).

In a similar manner, for $i = 2 \in U$ we allocate portions of $x_U(2), x_U(3)$ to $y_{\{2,\{1,3\}\}}(U)$ and $y_{\{2,\{1,4\}\}}(U)$ (green and light green lines in Fig. 2 respectively). The actual equation transmitted at any time is simply the XOR of the missing subfiles. For example, in the first 0.3 units if time-interval Π_2, the server transmits $W_{d_1,\{2,3\}} \oplus W_{d_2,\{1,3\}}$ while in the next 0.1 units, it transmits $W_{d_1,\{2,4\}} \oplus W_{d_2,\{1,4\}}$. We emphasize that the assignment for user $2 \in U$ can be done independently of the assigment for user $1 \in U$. The transmitted equations would change but the solution will still satisfy each user.

2) Complexity analysis: It is easy to see that $\mathcal{U}_\ell \subseteq \{U \subseteq [K] : |U| \leq t + 1\}$ thus $|\mathcal{U}_\ell| \leq \sum_{j=0}^{t+1} \binom{K}{j}$. Hence, there are at most $(2K-1)\sum_{j=0}^{t+1}\binom{K}{j}$ variables of the form $x_U(\ell)$. Furthermore, as there are $K\binom{K-1}{t}$ missing subfiles, thus there are at most $K\binom{K-1}{t}\sum_{j=0}^{t+1}\binom{K}{j}$ variables of the form $y_{\{i,\mathcal{A}\}}(U)$'s. Therefore, the total number of variables in (1) is at most $\{K\binom{K-1}{t} + 2K - 1\}\sum_{j=0}^{t+1}\binom{K}{j}$ variables.

It is not difficult to see that there are at most $2K - 1 + (5K-2)\sum_{i=1}^{t+1}\binom{K}{i} + K\binom{K-1}{t}(2\sum_{j=0}^{t+1}\binom{K}{j} + 1)$ constraints.

3) Discussion: The complexity of our solution does not have any dependence on the arrival times T_i's and the deadlines Δ_i's. Our formulation of the LP in terms of the intervals allows us to circumvent this potential dependence. A straightforward formulation of the above problem would assign variables for all possible equations in the different intervals (or time-slots). In particular, for a given user group U, we will need to consider $\left[\binom{K-|U|}{t+1-|U|}\right]^{|U|}$ possible equations. It can be seen that this will require much more variables and the corresponding problem will explode in size. We work with user groups in our formulation and determine the actual equations by interpretation of the feasible solution. This reduces the complexity significantly with respect to a straightforward formulation. Reference [2] considers the case when each missing subfile has a prescribed deadline. Our LP above can be modified easily to incorporate this aspect, though the number of intervals in the worst case will increase a lot. We will need to introduce variables of the form $y^{\ell}_{i,\mathcal{A}}$ that are non-zero at time interval Π_ℓ only if subfile $W_{d_i,\mathcal{A}}$ is active at time interval Π_ℓ and modify some constraints appropriately.

IV. Online asynchronous coded caching

The online problem appears to be much harder to reason about conceptually than the offline case. We make some partial progress on a specific instances of this problem in our work.

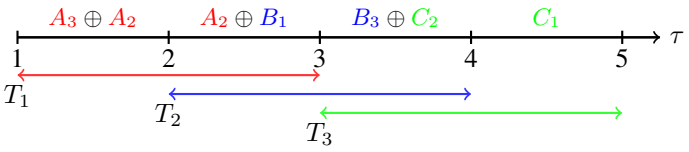

Fig. 3: Online solution corresponding to the Example 1. Note that the server is forced to transmit $A_2 \oplus A_3$ at $t = 1$.

A. Necessity of coding across missing subfiles of the same user

Consider Example 1 from Section III, but now consider the online scenario. Suppose that there is an adversary that makes decisions on when a particular user request arrives. Furthermore, we assume that the adversary can see the decisions made by the server. Suppose that the server does not code across any user's missing subfiles. At $\tau = 1$, it has the choice to transmit A_2 or A_3. We emphasize that it has to transmit either of these as the deadline for user 1 is $T_1 + \Delta_1 = 3$. If the server transmits A_3, then the adversary can force the arrival of the third user with $(T_3, \Delta_3) = (2, 2)$ and subsequently the arrival of the second user with $(T_2, \Delta_2) = (3, 2)$. In this case, the server is forced to transmit A_2 at $\tau = 2$, which implies that user 3 misses its deadline. In a similar manner, if the server transmits A_2, the adversary can easily generate an arrival pattern so that user 2 misses its deadline.

This issue can be circumvented if we transmit a linear combination of both A_2 and A_3 in the first time slot as shown in Fig 3. This example demonstrates that coding across missing subfiles of user 1 is strictly better than the alternative. Note that inn the synchronized model of [1], this is not needed.

B. Scenario where deadlines are loose enough

Let R^{online} and $R^{offline}$ denote the rate of transmission from the server in the online and offline cases respectively. Our next result shows that if the deadlines are loose enough, the overall rate of transmission from the server in the online case matches the offline case, i.e. $R^{\text{online}}/R^{\text{offline}} = 1$.

Theorem 1: Consider an asynchronous coded caching problem instance with parameters N, K, $M = tN/K$, r and $\{T_i, \Delta_i\}_{i=1}^K$ where $T_1 \leq T_2 \leq \cdots \leq T_K$. Suppose that the deadlines are such that $[i] \subseteq U_\tau$ whenever $i \in U_\tau$. Then, we have $R^{\text{online}}/R^{\text{offline}} = 1$.

Remark 1: Optimality of online solution for minimizing the overall completion time. Consider the important situation in practice where individual users do not have deadlines. However, they are interested in minimizing the overall completion time, i.e., the time by which each user is satisfied. In this case, there is an online solution that has the same completion time as the offline version, while having $R^{\text{online}}/R^{\text{offline}} = 1$.

Proof: For simplicity we assume that $r = 1$. The proof when $r > 1$ is similar. Suppose that there is an offline fractional solution to the problem which is given by a set of equations, denoted $\{\mathbb{E}(\tau)\}_{\tau=T_1}^{T_{\max}}$ that are transmitted within each time slot $[\tau, \tau+1)$. Each equation $E \in \mathbb{E}(\tau)$ is assumed to be of the all-but-one type. We will show that using the steps in Algorithm 1, we can modify it to an online solution with rate $(K-t)/(1+t)$.

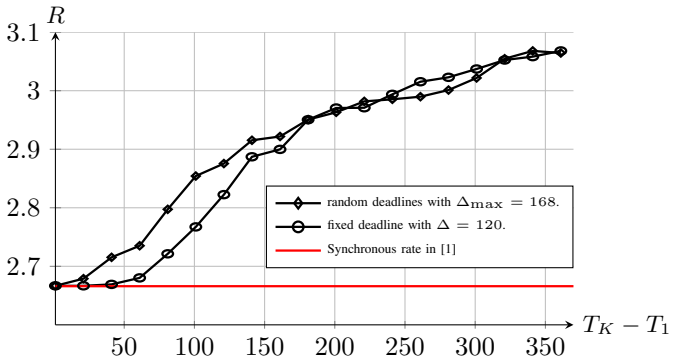

Fig. 4: Offline asynchronous coded caching simulation results for a system with $N = K = 10$, $t = 2$, and $r = 1$ with fixed and random deadlines.

In Algorithm 1 an equation which benefits $(t+1)$ users is called "good" and others "bad". In each iteration starting from $i = K$ and going backward, we keep converting bad equations into good equations. First note that this operation is always possible. For instance, if $i = K$, then all users are active in $[T_K, \ldots, T_K + \Delta_K]$, thus the replacement step (6) is valid. At an intermediate step $i < K$, it can be seen that any equation that benefits a user $j > i$ has already been transformed into a good one. Thus, the only bad equations under consideration when $i < K$ will be those that will benefit users $\leq i$. Again, these users are active at this stage, which implies that transformation of these equations into good equations is valid. Thus, our transformation ensures that users continue to meet their deadlines.

It is easy to see that the rate of transmission can only decrease with our modification. Next, we claim that at the end of the algorithm all transmitted equations are good. Assume otherwise, i.e., there is an equation E that benefits strictly less than $t+1$ users. Suppose that this equation results in (part of) missing subfile $W_{d_i,\mathcal{A}}$ being recovered by user $i \leq t$. However, $|\{i\} \cup \mathcal{A}| = t+1$, i.e., there is an index $k \geq t+1$ such that $k \in \mathcal{A}$. This implies that this missing subfile should have been removed by the algorithm when we running iteration k or higher. Thus, we arrive at a contradiction.

Finally, as all equations are good, our rate matches the schemes of [1], i.e., it is exactly $(K-t)/(1+t)$. The new scheme can easily be implemented by an online algorithm as follows. The server simply waits until $t+1$ requests come in. Following this it greedily keeps transmitting equations that benefit $(t+1)$ users whenever it can. If it is unable to, it simply waits until another request comes in.

V. Simulation Results

We investigated the performance of the offline asynchronous coded caching system via simulations. The maximal spread in the arrival times is $T_K - T_1$. The remaining arrival times were generated according to a Poisson process model with parameter $\lambda = (T_K - T_1)/K$ and were quantized to the most recent integral value of τ. It is evident that the deadline of each user, Δ_i, has to be at least $r\binom{K-1}{t}$, i.e., the minimum number of time slots each user needs to receive its missing subfiles. We ran two types of simulations. The first set considers the

Algorithm 1 Modification of offline solution

Input: A solution, $\{\mathbb{E}(\tau)\}_{\tau=T_1}^{T_{\max}}$

1: Set $\mathcal{W} = \emptyset$. If an equation $E \in \mathbb{E}(\tau)$ for $\tau = T_1, \ldots, T_{\max}$ benefits $(t+1)$ users, tag it "good", otherwise "bad".
2: **for** $i = K, K-1, \ldots, t+1$ **do**
3: Set $\mathcal{X}_i = \{E : E \in \mathbb{E}(\tau)$ where $\tau = T_i, \ldots, T_i + \Delta_i$, E includes a missing subfile of user $i\}$.
4: **for** each "bad" equation $E \in \mathcal{X}_i$ **do**
5: Let $E = W_{d_i,\mathcal{A}} \oplus \ldots \oplus W_{d_\ell,\mathcal{A}_\ell}$
6: Replace E with $\oplus_{k\in S} W_{d_k,S/\{k\}}$ where $S = \mathcal{A} \cup \{i\}$,
7: $\mathcal{W} \leftarrow \mathcal{W} \cup \{W_{d_k,S/\{k\}} : k \in S\}$,
8: Tag E as "good",
9: **end for**
10: For each entry in $\mathcal{W}$, remove the corresponding missing subfiles from any "bad" equation in $\{\mathbb{E}(\tau)\}$ for $\tau = T_1 \ldots T_{\max}$.
11: **end for**

situation when $\Delta_i = \Delta$ for all users. In the second set the deadlines are random; we let $\Delta_i = r\binom{K-1}{t} + \ell_i$ where ℓ_i is chosen uniformly from $[\Delta_{\max}]$.

Fig. 4 shows the simulation results for a scenario with $N = K = 10$, $t = KM/N = 2$ and $r = 1$. The synchronous rate in this case is $(K-t)/(t+1) = 2.67$. As expected the asynchronous rate matches the synchronous rate when $T_K - T_1$ is small. Recall that even in the synchronous case, the arrival time of the last user T_K has to be before the deadline of the other users. In the random scenarios, when $\Delta_{\max} = 168$ we have that the expected value of the Δ_i's is 120. Thus, with high probability when the arrival spread is at least 120 (corresponding to the right part of the plot) the deadline for T_1 expires before T_K, i.e., an equation that benefits both user 1 and user K cannot be transmitted. However, even in this case the simulation indicates that the rate degrades to about 3.06, which is still quite good. When the arrival spread is small, having fixed deadlines rather than random ones appears to allow for a lower rate, though this gap reduces when the arrival spread increases.

Overall, the simulations allow us to conclude that in the offline case, the coded caching rate is quite robust to asynchronous arrivals, i.e., the degradation in the rate is gradual and most of the rate benefits can still be leveraged when the asychronism is mild.

References

[1] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Trans. on Info. Th.*, vol. 60, no. 5, pp. 2856–2867, May 2014.
[2] U. Niesen and M. A. Maddah-Ali, "Coded caching for delay-sensitive content," in *IEEE Intl. Conf. Comm.*, 2015, pp. 5559–5564.
[3] R. Pedarsani, M. A. Maddah-Ali, and U. Niesen, "Online coded caching," *IEEE/ACM Transactions on Networking*, vol. 24, no. 2, pp. 836–845, April 2016.
[4] Z. Bar-Yossef, Y. Birk, T. Jayram, and T. Kol, "Index coding with side information," vol. 57, no. 3, pp. 1479–1494, March 2011.