2021 19th International Symposium on Modeling and Optimization in Mobile, Ad hoc, and Wireless Networks (WiOpt)

Decentralized Collaborative Video Caching in 5G Small-Cell Base Station Cellular Networks

Shadab Mahboob, Koushik Kar
Electrical, Computer and Systems Engineering
Rensselaer Polytechnic Institute
Troy, NY 12180.
Email: {mahbos, kark}@rpi.edu

Jacob Chakareski
College of Computing
New Jersey Institute of Technology
Newark, NJ 07102.
Email: jacob.chakareski@njit.edu

***Abstract*—We consider the problem of video caching across a set of 5G small-cell base stations (SBS) connected to each other over a high-capacity short-delay back-haul link, and linked to a remote server over a long-delay connection. Even though the problem of minimizing the overall video delivery delay is NP-hard, the Collaborative Caching Algorithm (CCA) that we present can efficiently compute a solution close to the optimal, where the degree of sub-optimality depends on the worst case video-to-cache size ratio. The algorithm is naturally amenable to distributed implementation that requires no explicit coordination between the SBSs, and runs in $O(N + K \log K)$ time, where N is the number of SBSs (caches) and K the maximum number of videos. We extend CCA to an online setting where the video popularities are not known a priori but are estimated over time through a limited amount of periodic information sharing between the SBSs. We demonstrate that our algorithm closely approaches the optimal integral caching solution as the cache size increases. Moreover, via simulations carried out on real video access traces, we show that our algorithm effectively uses the SBS caches to reduce the video delivery delay and conserve the remote server's bandwidth, and that it outperforms two other reference caching methods adapted to our system setting.**

I. Introduction

Recently, the Internet has witnessed deployment of a variety of video streaming applications, as well as tremendous growth in video traffic. Applications such as YouTube, Netflix, Amazon Prime Video, Hulu, and Sling TV are contributing to a large part of our daily Internet bandwidth consumption. Live video streaming and video-on-demand (VoD) services are growing, sharing of video news and messages through social networking applications like Facebook and WhatsApp is increasing steadily, and news readers are increasingly utilizing video feeds for their daily news. A recent Cisco study [1] finds that Internet video traffic has been growing annually at 33% and will constitute about 82% of all IP traffic by 2022.

The growth in video traffic is also accompanied by significant changes in video access patterns in recent years. Firstly, the quality and bit-rate of videos are steadily increasing, with growing availability of Ultra High Definition (UHD) or 4K video streams that occupy more than double the HD video bit rate. Secondly, the difference between 'live' and 'stored' video is blurring. As more users expect VoD capability for TV shows – where Internet-based delivery mechanisms are making steady inroads – access to TV shows gets staggered over time. Nevertheless, the access patterns of a show are often highly correlated temporally, i.e., close to each other in time, based on when it is posted online. Finally, there is considerable growth in video viewing over wireless, over both WiFi and cellular technologies. Emerging super-fast access technologies such as 802.11ac/802.11ax and 5G are accelerating this growth. There is also increased video viewing on mobile devices such as smartphones and tablets, which are expected to contribute to about 50% of the Internet traffic in a few years [1].

These trends indicate an increased importance of caching videos close to the users, to reduce the access delay and network/server congestion. Increasing deployment of 5G access points is expected over the next decade, making these access points (or *Small-cell Base Stations (SBS)*) natural candidates for hosting such caches. The range (coverage area) of these SBSs, and the cache sizes that can be included in them, are expected to be small. At the same time, in many of the deployment scenarios (malls, office buildings, campuses etc.), such SBSs may be deployed in large numbers, and be connected with each other other over a fast local area network such as high-speed Ethernet. This motivates pooling resources of multiple such SBSs, and using them to collaboratively cache videos for access by users covered by a cache pool.

In this paper, we consider the problem of video caching in a wireless edge network comprising multiple SBSs linked to each other over a high-capacity low-delay local area network. The SBSs host small video caches but can exchange videos with each other over the local network; or they can fetch videos from a remote server over a long-delay Internet path. In this setup, the problem of minimizing the overall video playout delay is NP-hard due to packing-type integrality constraints; even if the integrality constraints are relaxed, the problem is a concave minimization problem which could be NP-hard in general. Despite these facts, we utilize the specific structure of our problem to develop an efficient algorithm that computes a close-to-optimal solution, where the degree of sub-optimality depends on the worst case video-to-cache size ratio. More specifically, our algorithm is naturally amenable to distributed implementation and runs in $O(N + K \log K)$ time, where N is the number of SBSs (caches) and K the maximum number of videos. We also extend this algorithm to a dynamic setting where the video popularities are not known a priori but are estimated over time. The distributed, online implementation *does not require any explicit coordination between the SBSs*, as long as an ordering (tie-breaking) rules between the caches and the videos are pre-determined, and information on the

ISBN 978-3-903176-37-9 © 2021 IFIP

video requests from users is periodically shared between the SBSs. We show via numerical experiments on a small number of caches that our algorithm approaches the optimal (integral) caching solution when the cache size is large relative to the individual video sizes. Simulations conducted on real video access traces demonstrate the performance trade-offs between video playout delay and local and remote bandwidth used, and the impact of popularity estimation parameters and re-optimization intervals. We also show that when there is significant temporal correlation in the video access patterns across the caches, our algorithm is able to effectively use collaboration between the SBS cache pool to reduce the video playout delay and conserve the remote server's bandwidth.

II. Related Work

The general caching problem has attracted considerable attention before due to the emergence of content-centric networks [2, 3]. These studies mainly focused on either lowering the bandwidth consumption during the peak traffic times or the content delivery delay by developing efficient algorithms. With the advent of cellular edge networks, SBSs have become the most suitable candidates for caching with reduced latency, cost and energy consumption [4]. In [5], the content delivery delay is minimized for a single SBS by formulating this is as a knapsack problem [6] based on derived content popularities. [7] analyzes "femtocaching", where small wireless caching helpers with limited size and coverage area reduce the content delivery delay. Hierarchical coded caching in networks with multiple layers of caches is introduced in [8]. As the SBSs are deployed far from each other in traditional wireless cellular networks, these schemes mainly focus on caching optimization on a scale of a single SBS without considering possible collaboration among SBSs. Densely located SBSs in 5G networks make collaborative caching among SBSs feasible.

To this end, a number of cooperative caching strategies for cellular networks have been proposed using primarily optimization approaches, considering different objective functions such as overall delay, cost and revenue, together with cache capacity constraints. The study in [9] maximizes a total reward objective for an ISP in a collaborative manner, given limited caching space at each SBS, and formulates strong approximation algorithms for both, coded and uncoded data cases. In a follow-up study, joint caching, routing, and channel selection is investigated using large-scale column generation optimization with tight approximation guarantees [10]. In [11], the aggregated storage and download cost for caching is minimized for both limited and unlimited caching spaces by devising a near optimal greedy algorithm. However, implementing complex centralized algorithms may be infeasible in practice, and a simple distributed, adaptive algorithm that requires minimal coordination between the SBSs is what is practically desirable. This is the objective we pursue here.

III. Model and Formulation

We consider an edge network comprised of a set of N SBSs, indexed as $i = 1, \cdots, N$. There is a set of K videos, indexed as $k = 1, \cdots, K$, which may be downloaded from one or more remote servers. SBS i is associated with cache space C_i, which it uses to selectively cache some of the videos. The SBSs are connected to a high-speed local network over which they can exchange videos with each other. Each end-user is assumed to be associated with one of the SBSs at any given time, although that association may change over time due to user mobility. If the requested video is available at the SBS (cache) that the user is associated with, the video is served to the user with minimal playout delay.[1] Otherwise, the video is either obtained from one of the other SBSs in the local network, or is downloaded from the remote server(s) if the video is not available in the local caches. If the video is present in one of the other local SBS caches, we assume an average playout delay of d; otherwise (i.e., if the video is to be fetched from the remote server(s)), the average playout delay is $D > d$.[2] Henceforth, the term 'delay' refers to the video playout delay, unless mentioned otherwise. Our system model is illustrated in Figure 1.

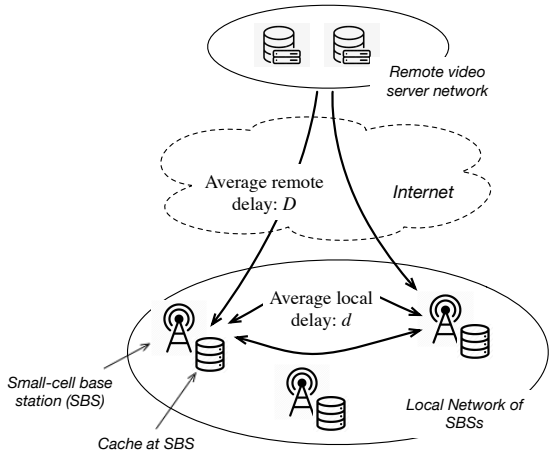

Fig. 1. Illustration of our System Model.

Let the popularity and size of video k be represented by π_k and s_k, respectively. Note that the video popularity is considered independent of the SBS (cache) k. Since users will typically be mobile across the coverage area of the SBSs, we reasonably assume that the same popularity vector (which represents the access rates of the videos across the entire population of served users) would apply to all SBSs. Given $\pi = (\pi_k, k = 1, \cdots, K)$, and letting $x_{i,k}$ be a binary variable indicating if video k is cached at SBS i, our goal is to minimize the overall video playout delay, expressed as

$$\sum_k \pi_k \left[\sum_i \left(\left(\max_{i'} x_{i',k} - x_{i,k} \right) d \; + \left(1 - \max_{i'} x_{i',k} \right) D \right) \right].$$

[1]For ease of exposition, we take this delay to be zero. There is no loss of generality here, as assuming that this delay on an average is a positive number δ does not affect our algorithm or its analysis.

[2]Note that we consider video *playout* delay and not the video delivery delay, and therefore this delay is assumed independent of the video size. The video playout delay, one of the most important factors affecting Quality of Experience (QoE), is primarily a function of the quality of the connection (path) (such as bandwidth, round trip time) over which the video is delivered.

ISBN 978-3-903176-37-9 © 2021 IFIP

Note that $(\max_{i'} x_{i',k} - x_{i,k})$ equals 1 only when video k is cached locally (in one of the SBS caches) but not at cache i, and zero otherwise. Therefore, the term $(\max_{i'} x_{i',k} - x_{i,k})\, d$ represents the additional video delay incurred if requested video k is not cached at SBS i but has to be fetched from one of the other SBSs. in On the other hand, the term $(1 - \max_{i'} x_{i',k})$ is 1 only if video k is not cached locally at all (and therefore must be fetched from a remote server incurring an additional average delay cost of D), and zero otherwise. Since the above objective is equivalent to maximizing $\sum_k \pi_k \Big[d \sum_i x_{i,k} \;+\; (D-d) \sum_i \max_{i'} x_{i',k} \Big]$, we define the *Collaborative Caching Problem (CCP)* as

$$\text{maximize} \;\; \sum_k \pi_k \Big[d \sum_i x_{i,k} \;+\; (D-d) \sum_i \max_{i'} x_{i',k} \Big], \quad (1)$$

$$\text{subject to} \;\; \textstyle\sum_k x_{i,k} s_k \leq \; C_i, \quad \forall i, \quad (2)$$

$$x_{i,k} \in \; \{0,1\}, \quad \forall i, \forall k. \quad (3)$$

Next, we mention a few points about the structure and complexity of the CCP formulation described in (1)-(3), towards motivating the solution approach that we will present in the next section. The complexity of the problem comes from two aspects: (a) the non-linearity (non-convexity) of the objective function (1); and (b) the integrality (binary nature) of the optimization variables in (3). The constraints (2)-(3) represent integral packing type constraints which make the problem NP-hard. For a single SBS, the problem reduces to the 0-1 knapsack problem [6]. While the binary knapsack problem is NP-hard, it can be solved in pseudo polynomial time using dynamic programming; several heuristic approaches are also known to work well in practice. However, most of these approaches are not easily amenable to distributed implementation with very low coordination and message exchange complexity, which is highly desirable in our setting.

Towards addressing (a), we note that since $D > d$, when the integrality constraints (3) are relaxed, the CCP posed in (1)-(3) represents a *concave minimization* problem over a polyhedral set [12]. While such problems are NP-hard in general, the $\max$ terms in the objective function can be replaced by a set of linear terms and additional linear constraints. However, this not only results in additional variables and constraints, it does not help towards developing distributed solutions with low coordination message complexity.

In our algorithm that we describe in Section IV, both (a) and (b) are addressed, and the resulting solution closely approximates the optimal integral solution of the problem, when the SBS cache capacities are sufficiently large compared to the individual video sizes (or units in which the videos are cached)[3]. Further, the algorithm can be implemented in a distributed manner with *no explicit coordination* between the SBSs, as long as certain tie-breaking rules are agreed upon in advance, and information about video requests from users is periodically shared between the SBSs to enable estimation of the popularity vector π.

[3]If video sizes are large (like HD/UHD quality movies), caches may store the beginning few minutes of each video (instead of the entire video, which may be large), seeking to reduce the initial playout delay, while the rest of the video is streamed directly from the server to the user. In other cases where the video size is large, such as in 360-degree videos, the video can be cached in units of small-size tiles [13]. Therefore, our assumption on the ratio of the video caching unit to the cache size being small generally holds true.

IV. Algorithm and Analysis

Towards developing intuition behind the algorithm and its optimality, we first present the *Collaborative Caching Algorithm (CCA)* for the special case of unit size videos, with the cache sizes (possibly different from each other) being integral multiples of this unit video size. This special case admits a simpler algorithm that can be described and illustrated easily, as well as a simpler optimality proof that still captures the essence of the argument. We then extend this algorithm and analysis to the general case of arbitrary video sizes (in addition to cache sizes being arbitrary), for which we argue that our algorithm is optimal in an approximate (asymptotic) sense.

A. Collaborative Caching for Unit Video Sizes

In this special case, $s_k = 1 \;\forall k$; also, $C_i = m_i$ for some positive integer $m_i \;\forall i$. This implies that there is no loss due to fragmentation of the videos. The formulation of CCP remains the same as that in (1)-(3) except that (2) reduces to $\sum_{k \in \mathcal{K}} x_{i,k} \leq \; C_i \;\forall i$.

Without loss of generality, we assume that the videos are indexed in decreasing order of their popularity values (ties broken arbitrarily), i.e., $\pi_k \geq \pi_{k'}$ when $k < k'$. Let the number of cached copies of video k during the run of the algorithm be denoted by n_k. The algorithm works in two phases:

Phase I (Greedy Filling): In this phase, each cache is filled up greedily and independently with videos in the order $1, 2, 3, \cdots$ (decreasing order of popularity) up to the cache size limit. For the case of unit video sizes, this simply means that cache i caches up to video with index $C_i = m_i$. Note that at the end of this phase, the highly popular videos may be cached in multiple (possibly all) caches, whereas videos with low popularity may not be cached at all. In general, a video k is cached in any cache i such that $k \leq m_i$. Let $S_{>1}, S_1$, and S_0 respectively denote the sets of videos that have been cached at multiple caches, at a single cache, or none at all. The greedy filling step is illustrated in Figure 2(a).

Phase II (Compare and Replace): In the second phase, the n_k values (and accordingly, the $S_{>1}, S_1$, and S_0 sets) are altered as some of the videos from the set S_0 are cached in one of the caches (and therefore move to set S_1) replacing videos that are present in more than one cache. This replacement happens if two videos $k_1 \in S_{>1}$ and $k_2 \in S_0$ satisfy a "popularity ratio test": $\pi_{k_2}/\pi_{k_1} > d/(ND-(N-1)d)$.[4] Thus, in this phase, the n_k values of the videos in $S_{>1}$ may reduce, and the set $S_{>1}$ may shrink as well, as some of the videos in $S_{>1}$ may just have one copy left and therefore move to the set S_1. Further, no video in S_0 or S_1 is cached multiple times in this phase, so no video is added to $S_{>1}$. The set S_0

[4]It will be observed later in our analysis that replacement according to this popularity ratio test improves the objective function in (1).

ISBN 978-3-903176-37-9 © 2021 IFIP

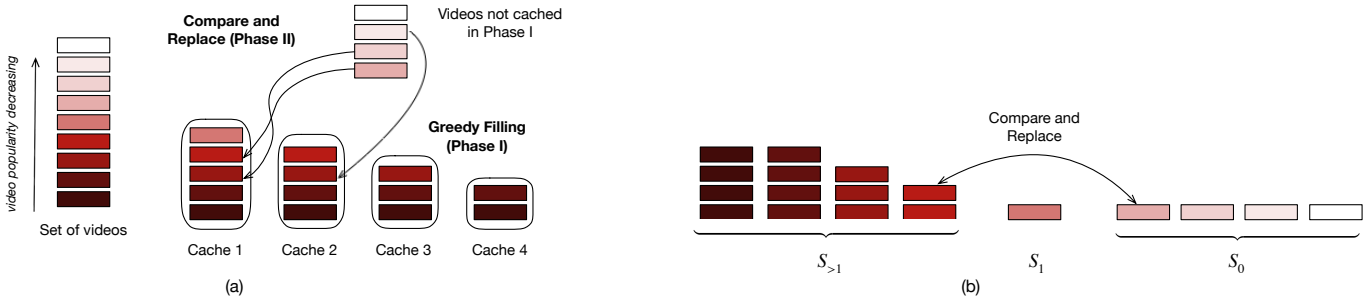

Fig. 2. Illustration of CCA for unit video sizes.

may shrink as well, as some of its videos may be cached (in a single cache at most) and therefore move to S_1. The set S_1 can only expand, due to videos from both $S_{>1}$ and S_0 possibly moving to this set. See the illustration in Figure 2(b). The popularity ratio test is conducted by picking k_1 to be the video with the largest index in $S_{>1}$ and k_2 as the video with the smallest index in S_0; therefore the replacement step can be viewed as the boundary between S_1 and S_0 moving right by one step; and the boundary between $S_{>1}$ and S_1 staying the same, or moving left by one step. The replacement stops (and the algorithm ends) when the k_1, k_2 thus picked fails the ratio test, or either $S_{>1}$ or S_0 becomes empty (i.e., k_1 becomes 0 or k_2 becomes $N+1$).

The algorithm is described in Algorithm 1. Assuming the videos are already pre-sorted according to their π_k values, Phase II involves up to K compare and replace steps, and therefore has $O(K)$ complexity. The complexity of Phase II is $O(K)$ per cache, or $O(NK)$. Including the complexity of the sorting step (which is needed for Phase I as well), the complexity of CCA for unit size videos is $O(NK + K\log K)$.

Algorithm 1: Collaborative Caching Algorithm (CCA): *Unit Video Sizes* (**Input:** d, D, π)

Phase I (Greedy Filling):
for *each SBS (cache)* i **do**
  Fill up cache i with videos $1, 2, \cdots, m_i$.
for *each video* k **do**
  $n_k = \{i | k \le m_i\}$ // number of cached copies of k.
Set $S_{>1} = \{k | n_k > 1\}, S_1 = \{k | n_k = 1\}, S_0 = \{k | n_k = 0\}$.
Phase II (Compare and Replace):
Let k_1 = last index in $S_{>1}$; k_2 = first index in S_0.
while *($S_{>1}$ and S_0 are both non-empty) &&* $(\frac{\pi_{k_2}}{\pi_{k_1}} > \frac{d}{ND-(N-1)d})$ **do**
  Replace any one copy of video k_1 with k_2, and set $n_{k_1} = n_{k_1} - 1$ and $n_{k_2} = 1$.
  Set $S_0 = S_0 - \{k_2\}$, $S_1 = S_1 + \{k_2\}$, and $k_2 = k_2 + 1$.
  if $(n_{k_1} == 1)$ **then**
    Set $S_{>1} = S_{>1} - \{k_1\}; S_1 = S_1 + \{k_1\}$, and $k_1 = k_1 - 1$.

Our optimality claim for the special case of unit video sizes is stated in Theorem 1 (proof in [14]).

Theorem 1 *If $s_k = 1$ for all videos k, and all cache sizes C_i are integral, then Algorithm 1 computes an optimal solution of CCP as posed in (1)-(3).*

B. Collaborative Caching for Arbitrary Video Sizes

For arbitrary video sizes, the Collaborative Caching Algorithm (CCA) works along similar lines as Algorithm 1, but with the following differences. Firstly, in both Phase I (Greedy Filling) and Phase II (Compare and Replace), we ignore the integrality constraints, and allow videos to be filled or replaced *fractionally* if needed, so that no cache space is wasted. Secondly, popularity values π_k are replaced by *popularity density* values $w_k = \pi_k / s_k$, and the video indices are sorted in the decreasing order of the w_k values, i.e., $w_k \ge w_{k'}$ when $k < k'$. Finally, there is a final Rounding phase (Phase III) of the algorithm where the space allocated to videos in S_1 are reallocated to the same videos, but in a way that satisfies the integrality constraints. At the same time, any fractional videos included in $S_{>1}$ are dropped. Therefore, the rounding step ensures that the final solution contains only full videos. Further details on the three phases of the algorithm is provided next, and the Collaborative Caching Algorithm (CCA) for this general case is fully described in Algorithm 2.

Phase I (Greedy Filling (*fractional*)): Note that in this phase, cache i stores up to video m_i, where m_i satisfies $\sum_{k=1}^{m_i-1} s_k < C_i$ and $\sum_{k=1}^{m_i} s_k \ge C_i$. Thus videos $1, \cdots, m_i - 1$ are fully included, but only a part of video m_i may be included to fill up the cache space.

Phase II (Compare and Replace (*fractional*)): In this phase, we replace n_k by y_k, the aggregate size (including fractional) of video k as cached across all SBSs. Thus, if video k is cached fully in only one cache, $y_k = s_k$. The sets $S_{>1}, S_1$, and S_0 are defined accordingly, as shown in Algorithm 2. In Phase II, "extra" copies of video $k_1 \in S_{>1}$ are replaced by video $k_2 \in S_0$ (even though this replacement may only be fractional), if the popularity density values of videos k_1, k_2 satisfy a ratio test.

Phase III (Rounding): The rounding phase collects all the space allocated to videos in S_1 across the different caches, and reallocates those videos in that space sequentially. Note that the Phase II may have split the single copy of a video k (belonging to S_1) across multiple caches, and the rounding step collects all that and uses it to place video k in a single cache. At any point the next cache among those that have available space (given by the set R) is chosen. If this cache

ISBN 978-3-903176-37-9 © 2021 IFIP

(say i) does not have enough remaining space to accommodate the video (say κ), i.e., $c_i < \kappa$, then video k is dropped. This ensures only full size videos in S_1 remain in the final solution. Any fractional videos included in $S_{>1}$ and S_0 are also dropped. Therefore, the rounding phase ensures that the integrality constraints (3) are satisfied by the final solution.

The complexity of Phases I and II in Algorithm 2 is similar to those of Algorithm 1. The Rounding phase takes an additional $O(N + K)$ time. Therefore, the overall time complexity of CCA remains the same as that in the special case, i.e., $O(NK + K \log K)$.

Algorithm 2: Collaborative Caching Algorithm (CCA): *Arbitrary Video Sizes* (**Input:** d, D, π, s)

Define $w_k = \pi_k / s_k \ \forall k$, and reorder the video indices k in the decreasing order of the w_k values.

Phase I (Greedy Filling (*fractional*)):

for *each SBS (cache)* i **do**
  Fill up cache i with videos $1, 2, \cdots, m_i - 1$ (full), and m_i (possibly fractional).

for *each video* k **do**
  y_k = aggregate size of all cached copies of k (including fractional).

Set $S_{>1} = \{k | y_k > s_k\}, S_1 = \{k | y_k = s_k\}, S_0 = \{k | y_k < s_k\}$.

Phase II (Compare and Replace (*fractional*)):

Let k_1 = last index in $S_{>1}$; k_2 = first index in S_0.

while *($S_{>1}$ and S_0 are both non-empty) &&* $(\frac{w_{k_2}}{w_{k_1}} > \frac{d}{ND-(N-1)d})$ **do**
  Replace an amount r (from any cache(s)) of video k_1 with k_2, where $r = \min\{y_{k_1} - s_{k_1}, s_{k_2} - y_{k_2}\}$. Set $y_{k_1} = y_{k_1} - r$ and $y_{k_2} = y_{k_2} + r$.
  if *($y_{k_1} == s_{k_1}$)* **then**
    Set $S_{>1} = S_{>1} - \{k_1\}; S_1 = S_1 + \{k_1\}$, and $k_1 = k_1 - 1$.
  if *($y_{k_2} == s_{k_2}$)* **then**
    Set $S_0 = S_0 - \{k_2\}; S_1 = S_1 + \{k_2\}$, and $k_2 = k_2 + 1$.

Phase III (Rounding):

Remove any fractional videos from $S_{>1}$ and S_0.

Let c_i = the amount of space allocated to videos in S_1 in cache i after Phase II. Set $R = \{i | c_i > 0\}$.

Let κ_1 (κ_2) be the first (last) video in S_1.

while *($\kappa_1 \leq \kappa_2$)* **do**
  Pick the first cache i in R.
  if *($c_i > s_{\kappa_1}$)* **then**
    Cache video κ_1 in i, and set $c_i = c_i - s_{\kappa_1}$.
  else
    $R = R - \{i\}$.
  $\kappa_1 = \kappa_1 + 1$.

Our optimality claim for the general case (arbitrary video and cache sizes) is stated in Theorem 2 (proof in [14]).

Theorem 2 *Let $\epsilon = \frac{\max_k s_k}{\min_i C_i}$. Then Algorithm 2 computes a feasible solution to CCP as posed in (1)-(3), with an objective function value no less than $(1 - O(\epsilon))$ of the optimum.*

From Theorem 2, we note that the approximation factor $(1 - O(\epsilon))$, which approaches 1 when the cache sizes are sufficiently large compared to the individual video sizes (or the units in which the videos are cached), which is a reasonable assumption in practice. The proof of the result relies on two parts. In the first part, we show that the fractional solution computed at the end of Phase II is an optimal solution to CCP when the (3) is relaxed, i.e., Phases I and II of Algorithm 2 solves the fractional version of CCP optimally. The essence of the argument behind this claim is similar to that in the proof of Theorem 1. We then show that in Phase III, the rounding process and the dropping of fractional videos (needed to satisfy integrality constraints (3)) only adds an $O(\epsilon)$ sub-optimality gap. Note that a naive round down of the solution computed at the end of Phase II would result in a sub-optimality gap that grows as $O(K\epsilon)$, which is undesirable since the number of videos (K) can be large.

C. Distributed and Online Implementation

Algorithm 2 can be easily implemented in a distributed manner provided the SBSs (caches) agree upon (i) The video popularity vector π, and how ties are broken; (ii) An ordering between the caches. The video popularity vector will typically be estimated based on the user video requests at different caches. If the request data is periodically shared between caches, this estimation can be done independently at each cache. Any ties in the π_k values can be broken according to some predetermined rule, such as video id. The caches can be indexed simply in the decreasing order of their cache sizes, with ties broken according to any pre-determined manner.

Note that Phase I (Greedy Filling) is a fully decentralized process (requiring no coordination) which caches can carry out fully independently of each other. Phases II and III may appear centralized as they consider the set of all caches at the same time; however, under the conditions (i) and (ii) specified above, each cache can carry out the Phase II computations without any coordination with each other. Each cache will calculate the same overall solution from Phases II and III, but will only cache the videos that it itself is supposed to cache. In other words, to implement Algorithm 2 no explicit coordination or messaging is required between caches. Further, after the completion of the algorithm, each cache will also automatically know which videos are being cached in the other caches. This helps in requesting videos from those caches, as needed in a practical (online) implementation of the algorithm, as we describe next. Note that since Phase I can be run in parallel across the different caches, the time complexity of the distributed implementation of CCA is $O(N + K \log K)$.

In a real-life scenario, the video population would not be fixed, and the relative popularities of the videos will also evolve over time. To adapt our algorithm to such scenarios, we can re-estimate the video popularity values periodically, and re-optimize the caching solution accordingly. In our performance evaluation as described in Section V, we re-estimate the popularity vector π periodically, by counting the number

ISBN 978-3-903176-37-9 © 2021 IFIP

of videos requested by users in the local network (across all SBSs) over a time window W ending at the current time. This requires the SBSs (caches) *to share with each other the list of videos requested, but no other information exchange or coordination is required*. Each cache then applies Exponential Weighted Moving Average (EWMA) to calculate the the popularity values to be used in the caching re-optimization. More specifically, the popularity value of video k at the beginning of the t^{th} window, π_k^t, is calculated as

$$\pi_k^t = (1-\alpha)\pi_k^{t-1} + \alpha \frac{n_k^{t-1}}{W}, \tag{4}$$

where n_k^t represents the number of video requests during the $(t-1)^{\text{th}}$ window (across all SBSs), and α, where $0 \leq \alpha \leq 1$ is an appropriately chosen weighting parameter.

Note that CCA needs to be re-run (and a new optimal solution needs to be re-computed) at the end of each window, assuming that the popularity vector π has changed significantly from last time. CCA then reoptimizes the caching solution, which happens in the background. In this reoptimization process, CCA tries to minimize the use of remote bandwidth: thus, if a cache needs to obtain a new video due to this reoptimization, it preferentially gets it from another cache, and contacts the server only if the video is not stored in the local network. We observe (see Section V) that the amount of bandwidth consumed (both locally and remotely) is only a very small fraction of the total bandwidth consumed, implying that this reoptimization process has very limited overhead.

V. Performance Evaluation

We evaluated the performance of CCA through simulations on real video request data traces from Netflix and YouTube. The nature of the results was largely similar between the two traces, but the effectiveness of caching was more pronounced with the Netflix dataset, due to its higher degree of temporal correlation across video requests, as intuitively expected. Due to limited space, we only present results for the Netflix dataset.

A. Dataset

We use the publicly available Netflix Prize dataset [15] for simulating and analyzing the performance of our algorithm. The trace includes the pattern of about 15.5 million video requests from a database of 17770 videos. From this we extract the request pattern for the most popular 3000 videos and use it in our simulations. Since video size information is not provided in the dataset, we assumed a uniform distribution over a range between 2.5 GB and 5 GB for estimating the size of the videos. This range of sizes correspond to about 90 mins to 180 mins of video, assuming about 27.77 MB per min of video. While we have experimented with different values (ratios) of playout delays d and D, the results presented in this paper are mostly for $d = 500$ ms, remote delay $D = 5$ s. These numbers correspond to about 13.5 secs of initial playout buffering of the video, assuming local network bandwidth of 100 Mbps and remote server bandwidth (i.e., the link bandwidth between the local network and the remote server) of 10 Mbps.

B. Comparison with the Optimal Solution

We compare CCA with the optimal solution (OPT), where OPT is calculated by solving CCP (as defined by (1)-(3)) exactly. Due to the high complexity of solving OPT (due to the integrality constraints), we limit this comparison study to the 20 highest-popularity videos in the dataset. We set the local delay $d = 500$ ms, remote delay $D = 2.5$ s and 5 s. We compare the average delay by varying cache size for each SBS from 5 GB to 50 GB with fixed number of SBSs, $N = 6$ (Figure 3), and varying the number of SBSs from 2 to 8 when the size of each cache is fixed at 25 GB (Figure 4).

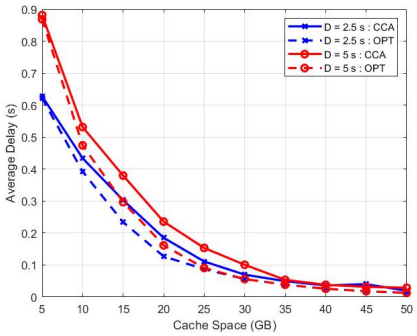

Fig. 3. Average Delay vs Cache Size at each SBS ($d = 500$ ms).

We can observe from Figure 3 that CCA closely approaches OPT at all cache space values. Note that when the cache space is very small, very few videos can be accommodated in the cache, and neither CCA nor OPT performs very well. On the other hand, for sufficiently large cache spaces, the entire video set can be stored in each cache (or at least in the local network), and both CCA and OPT perform very well. This explains why the the performance of CCA and OPT are very close at these two extremes. From Figure 4, we see that the ratio of the average delays under CCA and OPT decreases with increase in SBSs. From Figures 3 and 4, we see that even in the worst case (within the range of the settings simulated), this performance ratio is within a factor of 2.

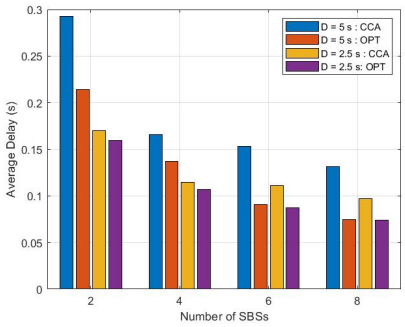

Fig. 4. Average Delay vs Number of SBSs ($d = 500$ ms in all cases).

C. Effect of Online Implementation Paramters

Next we implement the online version CCA, with the goal of evaluating the effect of algorithm parameters, W (window size) and α (weighting parameter) on the performance. The performance is measured both in terms of (i) average fetched bytes, (ii) average playout delay.

ISBN 978-3-903176-37-9 © 2021 IFIP

The average fetched bytes, which represent the total amount of bytes used by CCA in the local or remote network to serve a video request on an average, consist of two parts: (a) Delivery bytes: The bytes actually fetched from the remote server, or from another cache (SBS) in the local network, at the time of serving a video request; (b) Reoptimization bytes: The bytes transferred over the local network or over the link to the remote server(s) during the re-optimization process under CCA, which happens periodically (end of each time window of length W). Accordingly, we plot the average fetched bytes including and excluding reoptimization bytes, where the latter (i.e., when reoptimization bytes are excluded) only consist of the delivery bytes. We also distinguish the fetched bytes based on whether they were fetched over the local network (i.e., from another SBS cache) or from the remote server.

In the following simulations, we set the local delay d = 500 ms, remote delay = 5 s, Number of SBSs, N = 4, Cache space per SBS = 500 GB, and consider the most popular 3000 videos from the Netflix Prize Dataset. Figures 5-7 show the variation in fetched bytes (local and remote) and playout delay as the window length W varies, for a fixed $\alpha = 0.4$. For these figures, the window length is measured in terms of the total number of requests (across all SBSs), although it could equivalently interpreted in time units as well. We note that the local fetched bytes increase modestly as W increases, before almost flattening out. On the other hand, the remote fetched bytes attain a minimum at an intermediate value of W, which is in this case is about $30,000 - 40,000$. This is intuitively expected: when the window size W is very small, then the estimated π_k values do not reflect the true popularity values of the videos. On the other hand, when the window size is very large, any dynamic change in the true popularity values is not immediately reflected in the estimated π_k values. From Figure 7, we note that that the average playout delay for each video is also minimized at about the same window size. Comparing Figures 5-6, we see that the local fetched bytes are about 3-5 times the remote fetched bytes. However, since the local network bandwidth is expected to be much larger (at least an order more) than the bandwidth to the remote server(s), our primary network goal is to minimize remote fetched bandwidth (along with minimizing average video playout delay). This is attained at an intermediate value of the window size, which in practice can be determined by analysis of the timescale at which the video popularities change on an average.

The results with variation in α are omitted due to space limitations; however, the performance trends with respect to varying α were observed to be just the opposite of varying W, i.e., increasing α produces a trend in the fetched bytes (local and remote) and delay that are similar to those with decreasing W. This is expected from intuition: increasing α reduces the weight placed on historical requests, and decreasing W also has a similar effect. This implies that in practice, we can set one of the parameters to a reasonable value (set α to $0.3 - 0.5$, say), and choose the other (W, say) depending on the timescale of variation in request statistics.

From Figures 5-6, we also note that the amount of re-optimization bytes is fairly small compared to the delivery bytes, for both remote and local delay. In other words, the almost all of the fetched bytes are utilized in directly serving the video requests. This is certainly a desirable feature, and implies that the reoptimization process that is carried out in the background consumes very little additional bandwidth (both local and remote) but is nevertheless important for adapting the solution to changes in the video popularities.

Figure 8 illustrates the temporal variation in average fetched bytes (including reoptimization bytes), for $\alpha = 0.4$ and $W = 35,000$. Here, we assume a cold start, i.e, the caches do not have any cached videos initially. As the window number progresses, videos are cached, and popularity (π_k) estimates improve, resulting in a sharp drop of remote fetched bytes. Simultaneously, the local fetched bytes increase (finally reaching a steady state value), as most of the requested videos are fetched from a cache in the local network (instead of being fetched from the server), showing the effectiveness of collaborative caching.

D. Comparison with Other Caching Methods

In this section we compare the performance of CCA with two other popularly used caching algorithms, Least Recently Used (LRU) and Least Frequently Used (LFU), extended to include collaborative caching among the SBSs. In the collaborative version of LRU (LFU) that we compare CCA with, each cache implements LRU (LFU) individually, but preferentially obtains a requested video (that is not stored at the cache itself) from one of the other local SBS caches, before requesting the remote server. Also, to quantify the benefits of collaboration, we compare CCA with a version that does not include any collaboration, i.e., Algorithm 2 is run with Phase I (Greedy Filling) alone. Figures 9-10 shows the results. For CCA, we use $W = 35,000$ and $\alpha = 0.4$, as before.

From Figure 9, we see that compared to LRU and LFU, CCA saves significantly in terms of remote bandwidth. While this saving comes at the cost of increased local network bandwidth use, this is what we desire - since remote bandwidth is expected to be more costly (limited) compared to local bandwidth. Figure-10 shows that CCA provides considerable improvement in video playout delay compared to LRU and LFU, when all the algorithms have a "cold start". Furthermore, the average playout delay under CCA reduces quickly after start, and reaches a steady value. For the other two algorithms, the playout delay decreases initially, but then increases slowly but steadily, implying that these algorithms are not able to effectively adapt to changes in the video popularities as time progresses. We also note that CCA with no collaboration performs very close to LFU, as can be expected from intuition.

VI. Conclusion

In this paper we considered the problem of collaborative caching of videos across a set of local small-cell base stations (SBSs), with the goal of minimizing video playout delay. Despite the integrality constraints – and non-convexity of the problem even when the integrality constraints are relaxed

ISBN 978-3-903176-37-9 © 2021 IFIP

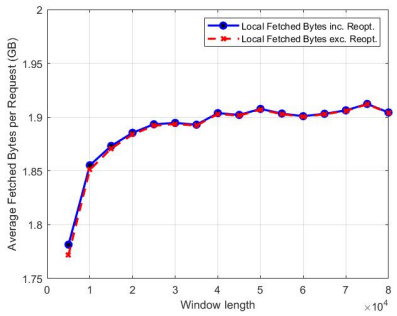

Fig. 5. Local Fetched Bytes vs Window Length.

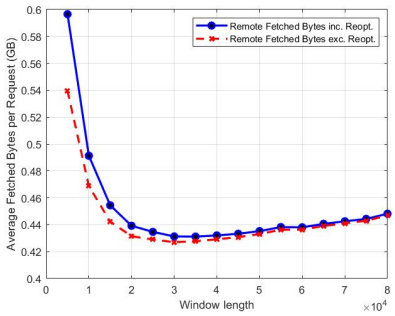

Fig. 6. Remote Fetched Bytes vs Window Length.

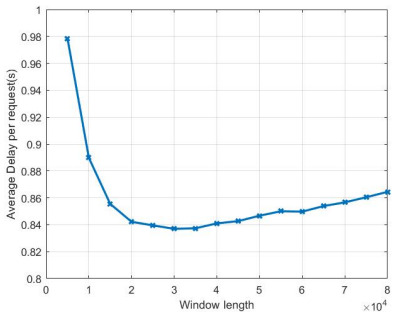

Fig. 7. Average Delay vs Window Length.

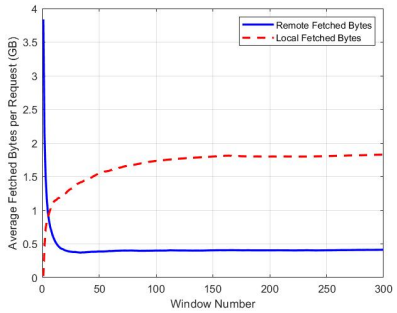

Fig. 8. Evolution of Total Fetched Bytes (including Reoptimization Bytes) with Window Number.

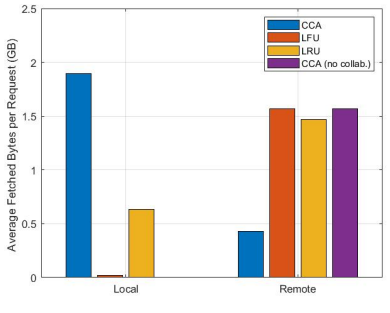

Fig. 9. Comparison in Total Fetched Bytes with other caching methods.

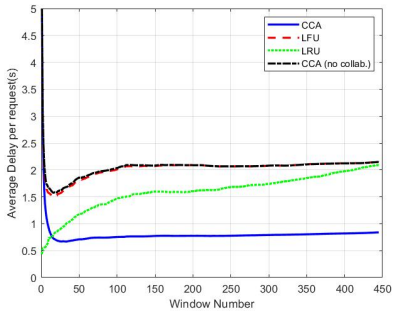

Fig. 10. Comparison in Average Delay with other caching methods.

– we provide an algorithm (CCA) that yields a close-to-optimal solution and is computationally efficient. CCA is also implementable in a distributed manner with very limited messaging between the SBSs, and without requiring any explicit coordination among them. Our simulations show that an online implementation of CCA is able to reduce remote bandwidth usage significantly through local sharing of videos among the SBSs. The comparison with collaborative versions of LRU and LFU show that CCA not only results in significantly lower delay, but also settles down (from a cold start) to lower steady-state playout delay values faster.

Two extensions of our algorithm may be worth exploring in the future. Firstly, we believe that CCA can be extended to the case where the average local and remote playout delays (d and D) are a function of individual video properties (such as the resolution, video type or size), albeit at an increased complexity of Phases II and III of the algorithm. This however needs to be explored further, and formally proven, in future work. Secondly, automated and online (re-) training of the window size parameter W – to match the timescale of variation of the video popularities – could be explored in future work.

Acknowledgment

The work of Jacob Chakareski was supported by NSF Awards CCF-1528030, ECCS-1711592, CNS-1836909, and CNS-1821875.

References

[1] Cisco Systems, "Cisco Annual Internet Report (2018–2023) White Paper," last accessed 31 August 2021. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[2] C. Bernardini, T. Silverston, and O. Festor, "MPC: Popularity-based caching strategy for content centric networks," in *2013 IEEE International Conference on Communications (ICC)*, 2013, pp. 3619–3623.

[3] Y. Li, T. Zhang, X. Xu, Z. Zeng, and Y. Liu, "Content popularity and node level matched based probability caching for content centric networks," in *2016 IEEE/CIC International Conference on Communications in China (ICCC)*, 2016, pp. 1–6.

[4] D. Liu and C. Yang, "Energy efficiency of downlink networks with caching at base stations," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 4, pp. 907–922, 2016.

[5] P. Blasco and D. Gündüz, "Learning-based optimization of cache content in a small cell base station," in *2014 IEEE International Conference on Communications (ICC)*, 2014, pp. 1897–1903.

[6] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. USA: John Wiley & Sons, Inc., 1990.

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

[8] N. Karamchandani, U. Niesen, M. A. Maddah-Ali, and S. N. Diggavi, "Hierarchical coded caching," *IEEE Trans. Information Theory*, vol. 62, no. 6, pp. 3212–3229, 2016.

[9] A. Khreishah and J. Chakareski, "Collaborative caching for multicell-coordinated systems," in *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2015, pp. 257–262.

[10] A. Khreishah, J. Chakareski, and A. Gharaibeh, "Joint caching, routing, and channel assignment for collaborative small-cell cellular networks," *IEEE J. Selected Areas in Communications*, vol. 34, no. 8, Aug. 2016.

[11] P. Ostovari, J. Wu, and A. Khreishah, "Efficient online collaborative caching in cellular networks with multiple base stations," in *2016 IEEE 13th Intl. Conf. on Mobile Ad Hoc and Sensor Systems (MASS)*, 2016.

[12] H. P. Benson, *Concave Minimization: Theory, Applications and Algorithms*. Boston, MA: Springer US, 1995, pp. 43–148.

[13] J. Chakareski, "Viewport-adaptive scalable multi-user virtual reality mobile-edge streaming," *IEEE Trans. Image Processing*, vol. 29, no. 1, pp. 6330–6342, Dec. 2020.

[14] S. Mahboob, K. Kar, and J. Chakareski, "Decentralized Collaborative Video Caching in 5G Small-Cell Base Station Cellular Networks," Full version, arXiv:2109.00369.

[15] J. Bennett and S. Lanning, "The Netflix Prize," in *Proceedings of the KDD Cup Workshop 2007*, Aug. 2007, pp. 3–6.

ISBN 978-3-903176-37-9 © 2021 IFIP