Transcoding based Video Caching Systems: Model and Algorithm*

Hongna Zhao[1], Chunxi Li[1], Yongxiang Zhao[1], Baoxian Zhang[2], and Cheng Li[3]

[1] Beijing Jiaotong University, No. 3 Shangyuancun, Beijing 100044, China
{17120196, chxli1, yxzhao}@bjtu.edu.cn
[2] University of Chinese Academy of Sciences, No.19A Yuquan Road, Beijing 100049, China
bxzhang@ucas.ac.cn
[3] Memorial University of Newfoundland St. John's, Newfoundland, A1B 3X5, Canada
licheng@mun.ca

Abstract. The explosive demand of online video watching brings huge bandwidth pressure to cellular networks. Efficient video caching is critical for providing high-quality streaming Video-on-Demand (VoD) services to satisfy the rapid increasing demands of online video watching from mobile users. Traditional caching algorithms typically treat individual video files separately and they tend to keep the most popular video files in cache. However, in reality, one video typically corresponds to multiple different files (versions) with different sizes and also different video resolutions. Thus, caching of such files for one video leads to a lot of redundancy since one version of a video can be utilized to produce other versions of the video by using certain video coding techniques. In this paper, we study transcoding based video caching in cellular networks where cache servers are deployed at the edge of cellular network for providing improved quality of online VoD services to mobile users. By using transcoding, a cached video can be used to convert to different low quality versions of the video as needed by different users in real time. We first formulate the transcoding based caching problem as integer linear programming problem. Then we propose a Transcoding based Caching Algorithm (TCA), which iteratively finds the placement leading to the maximal delay gain among all possible choices. We deduce the computational complexity of TCA. Simulation results demonstrate that TCA significantly outperforms traditional greedy caching algorithm with a decrease of up to 40% in terms of average delivery delay.

Keywords: Online Video-on-Demand · Transcoding · Distributed Caching.

* Supported in part by the National Science Foundation of China under Grant Nos. 61572071, U1534201, 61531006, and 61471339, the Natural Sciences and Engineering Research Council (NSERC) of Canada (Discovery Grant RGPIN-2018-03792), and the InnovateNL SensorTECH Grant 5404-2061-101.

1 Introduction

The explosive demand of online video watching from mobile users brings huge bandwidth pressure to cellular networks. A common way to relieve such pressure is to deploy cache servers close to the end users to help video diffusion [1, 2]. Caching algorithms can be categorized into two types: online algorithms and offline algorithms, which typically operate at small and large time scales, respectively. Typical online algorithms include Least Recently Used (LRU) [3] and Least Frequently Used (LFU) [4], both of which make caching decisions based on dynamically arrived user requests. In contrast, offline caching algorithms make caching decisions for all videos: caching each of them or not based on their historical data of user fetching without consideration of users' real-time requests. In this paper, we will focus on design of offline caching algorithm. Unless otherwise specified, the term caching algorithm in this paper means offline caching algorithm.

Traditional offline caching algorithms (e.g., [5, 6]) assume that all cached video files are independent and treat them separately. They compute each file's popularity by counting the time the file has been accessed in the past, and cache the most popular files. However, different users may request different versions (also with different resolutions) of a video, and the contents of different versions of a video are not independent but relevant. Thus, caching of files in such cases may contain a lot of redundancy since one version of a video can be utilized to produce some other versions of the video by using certain video encoding techniques such as scalable video coding (SVC) [7] and transcoding [8].

SVC has been used to improve caching performance [9]. SVC encodes a video into different layers. With SVC, one version of a video contains one or more video layers, and different quality versions of a video share certain low video layers. Based on this observation, the popularity can be tuned from a per-file perspective to a per-video-layer perspective. With such an understanding, [9] proposed a caching algorithm for providing video services layered by SVC so that caching decision is made on a per layer level instead of a per video level. However, SVC based caching has the following disadvantages. First, it requires SVC-encoded video files stored on cache servers and also SVC-decoding capability at mobile terminals; However, due to the high decoding complexity and excessive overhead [7], SVC is not widely deployed in online VoD. This largely limits the wide application of SVC based caching algorithm in reality. Second, the number of the quality versions supported by a SVC-encoded video exactly equals the number of layers in the video. This largely restricts the granularity of services that a SVC-encoded service platform can provide to users, who may desire various quality versions of a video.

Transcoding has also been used to improve the caching performance of an online VoD system. Different from SVC encoding/decoding, transcoding has been a mature technology for converting a video from high-quality to any low-qualities [8]. The conversion can be easily done at cache servers without involvement of mobile clients. Compared with SVC based caching, transcoding based caching has two advantages. First, video quality transforming can be conducted on much

finer granularity to meet diverse user requirements. Second, transcoding is compatible with existing video formats, without need to upgrade the software at mobile terminals (client side) and video source (source side). In [12], Shen et al. proposed a transcoding based online caching algorithm, with which a caching system transcodes the cached videos according to real time user requests and makes cache replacement with LRU algorithm. However, this algorithm is just a simple combination of LRU and transcoding, which largely restricts its caching performance. Moreover, the use of LRU in [12] cannot ensure each video has at most one quality version in a cache server, and thus much redundancy still remains among cached files.

In this paper, we focus on studying transcoding enabled caching. The objective is to enable cache servers to keep most valuable video versions so as to minimize the video delivery delay for video requests from all users subject to constraints of cache sizes and limited bandwidth on links between different base stations (for cooperative caching). We first formulate the transcoding based caching problem as integer linear programming problem. Then we propose a Transcoding based Caching Algorithm (TCA), which iteratively finds the placement leading to the maximal delay gain for video fetching among all possible choices by considering request arrival rates of videos, file sizes of different video versions, and also delivery delays between different cache servers. To reduce content redundancy, TCA restricts each cache server to keep at most one quality version of a video. We deduce the computational complexity of TCA. Simulation results demonstrate that our TCA algorithm can significantly reduce the video transmission delay by up to 40% compared with traditional offline greedy algorithm in [5].

The rest of the paper is organized as follows. Section 2 gives a brief review of related work. Section 3 describes the caching system model and formulates optimal transcoding based independent caching and cooperative caching problems, respectively. In Section 4, we propose the TCA caching algorithm. In Section 5, we perform simulations to evaluate the performance of TCA. We conclude this paper in Section 6.

2 Related Work

Traditional offline caching algorithms usually treat individual video files separately, and tend to keep the most popular video files in cache [5, 6]. Ref. [5] proposed a cooperative caching algorithm for a distributed cache system. Staring with an all zero caching vector, this algorithm then iteratively updates the caching vector by placing a file to a cache server such that this placement leads to the maximum performance improvement, which is computed based on the popularity of each video. Ref. [6] proposed a caching algorithm which makes use of both user interests and video popularities, in order to achieve a good balance between cache efficiency and user preference.

However, in the traditional caching algorithms, several files kept at a cache server may belong to the same video but having different qualities, and thus lead

to a lot of redundancy since one version of a video can actually be used to produce some other versions of the video by using certain video encoding/decoding technique, such as SVC [7] and transcoding [8].

SVC encodes a video into different layers, and different quality versions of a video share certain low layers. Thus, for SVC encoded videos, the popularity calculation is changed from a per-video perspective to a per-layer perspective. Following this direction, in [9], a SVC-based caching algorithm was proposed to improve the caching performance. However, due to the high decoding complexity and the additional overhead [7, 10], SVC is not widely deployed in online VoD in particular for resource-limited mobile terminals. Moreover, the amount of quality versions that a SVC based video can be transformed is quite limited and it equals the number of its encoded layers. This largely restricts the service granularity that a SVC-based encoding system can provide.

Transcoding has been a mature technology, which can convert a high-quality video to a low-quality video in real time [11]. By applying transcoding at cache servers, video quality transforming can be conducted on much finer granularity to meet diverse user requirements, and it is compatible with existing video formats without need to upgrade the software at mobile clients or video sources.

In this aspect, [12] designed an online caching system, which uses LRU to make cache replacement and uses transcoding to covert cached video files. Reference [13] designed an augmented radio access network model to evaluate the performance of such transcoding based online caching systems and showed that the caching performance can be further improved. Ref. [14] studied a multiple-parameter optimal model subject to the cache storage and transcoding capacity constraints, and proposed a cooperative LRU-based online video caching algorithm (online JCCP), which uses the collaboration among the cache servers to further improve cache performance. However, in these studies, LRU is just simply combined with transcoding, and the caching or replacing of a file is still based on LRU itself, which cannot ensure each video has at most one quality version cached in individual cache server. For example, when a high version needs to be cached, removing the low version of the file from the cache (if any) should be a better choice. Howerver, the use of LRU in [12–14] fails to incooperate such operation. Thus, redundancy still remains in caches

Transcoding also causes certain extra cost. Ref. [15] pointed out this problem and proposed an online partial transcoding caching scheme used in cloud computing nework with an aim to minimize the total extra cost caused by the transcoding and storage. Ref. [16] in 2018 proposes a cloud-based architecture to allocate transcoding tasks among virtual machines in a cache system to decrease the cost of streaming service provider, and pointed out transcoding based video caching is worth to be further studied. Moreover, due to the significant improvement of capability of mobile edge computing [17, 18] and the appearance of new transcoding technologies [19, 20], it is now feasible to conduct full transcoding in caching systems at mobile edge.

In this paper, we focus on studying transcoding based offline caching algorithm. Our algorithm in this paper differs from the above work in the following

ways. First, it restricts one cache server to keep at most one quality version of a video in order to remove unnecessary redundancy. Second, it made caching decision on the quality version of a video to be cached based on all the requests for different versions of the video.

3 Problem Statement

In this section, we first give an overview of transcoding based video caching system for providing streaming services at mobile edges. Then, we illustrate the key idea for performing transcoding based caching. Finally, we formulate the optimal transcoding based caching problem.

3.1 System Model

We first introduce the caching system under study. As Fig. 1 shows, there are N cache servers in the system, which are geographically distributed at the edge of the cellular network to provide online VoD services to mobile users covered by the base stations. We assume each cache server is attached to a base station via a short link with unlimited bandwidth, and for simplicity we assign the same sequence number to a cache server and its attached base station. Each cache server can transcode a cached video from a high quality version to a low quality version in real time. These cache servers are connected to a remote video source server, which has all the quality versions of all the videos that users may request, via the back-haul network. In this paper, each user is assumed to be covered by one base station, and can fetch video data from its local cache server, the video source server, or another local cache server (in the case of cooperative caching).

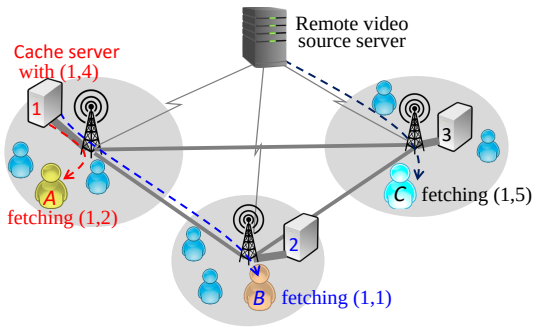

Fig. 1. Example illustrating the application scenario with three cellular cells. In this figure, combination of two figures, in the form of (x, y), represents video x's y version.

We next introduce a typical application scenario as shown in Fig. 1 to illustrate the operation of vide fetching in such a system. As the figure shows, a user

(say A) sends a request for a video quality version labeled by $(1, 2)$, i.e., a quality version 2 of video 1, to its local cache server 1, which just has a video quality version $(1, 4)$. Therefore, cache server 1 can satisfy the request by transcoding $(1, 4)$ to $(1, 2)$ since the locally cached version is higher than the requested version. We assume the transmission rate between a mobile user and its local base station is high enough such that the delivery delay for fetching a video from local cache server to its users is negligible, and we further assume transcoding can be performed in real time on a cache server such that we do not consider the transcoding delay.

Furthermore, assume the $N = 3$ cache servers in the system work cooperatively. A user, say B in Fig. 1, who requests $(1, 1)$ but cannot get it directly from its local cache server 2, can fetch the video also from server 1, which transcodes $(1, 4)$ to $(1, 1)$ and send $(1, 1)$ to user B via base station 2. Along the path from server 1 to user B, we assume the link between base station 1 and base station 2 will allocate a certain bandwidth for each session for such video transmission and also cause certain delivery delay. Finally, if the request for $(1, 5)$ of a user C cannot be satisfied by any local cache servers, he/she needs to fetch it from the remote source server. Along the path from the remote video source server to user C, we assume the backhaul link will allocate a certain bandwidth for each session of such video transmission and also cause certain delivery delay.

Our objective here is to find a caching vector to place video files on different cache servers, while minimizing the average delivery delay of video transmissions for all users. Next, we will first define variables and parameters used and then formulate the problem.

3.2 Symbols and Parameters

Before formulating the problem under study, we list the symbols and parameters used hereafter in Table 1.

We denote the set of cache servers by $\mathcal{N} = \{n | n = 1, 2, \cdots, N\}$, where N is the total number of cache servers. We assume a cache server $n, n \in \mathcal{N}$ has a space of C_n bytes for caching video files.

Assume there are in total V videos in the system, each having Q different versions to be requested, and we use $\mathcal{V} = \{1, 2, \cdots, V\}$ and $\mathcal{Q} = \{1, 2, \cdots, Q\}$ to denote the set of all the videos and the set of all the video versions, respectively. Each quality version of a video, denoted by $(v, q), v \in \mathcal{V}, q \in \mathcal{Q}$, has a given size of S_{vq} bytes, and we assume $S_{v1} < S_{v2} < \cdots < S_{vQ}, \forall v$. The number of requests for each video version (v, q) from the users under base station n is denoted by λ_{nvq}, which is assumed to be known in advance like in [5, 9].

Delivering a file may cause certain delivery delay. Specifically, delivering a video with a size of S_{vq} from the remote video source server to a user in the cell of base station n will cause a delay of $S_{vq}d_n$, where d_n is the delay for delivering one unit of video data from the source server to the user via base station n, due to the limited bandwidth assigned on backhaul link for the transmission. Furthermore, in the situation where these cache servers work cooperatively, delivering a video with a size of S_{vq} from a server n' to a user associated with base station n,

will cause a delay of $S_{vq}d_{nn'}$, where $d_{nn'}$ is the delay for delivering one unit of video data from cache server n' to the user via base station n, due to the limited bandwidth assigned on the link between base station n' and base station n for the transmission. We denote a caching vector by an set of integers $X = \{x_{nv}|n \in \mathcal{N}, v \in \mathcal{V}, x_{nv} \in \mathcal{Q}+\{0\}\}$, where each element x_{nv} represents the quality version of video v cached at server n, and specially, $x_{nv} = 0$ means server n does not have any version of video v. We use $X_n = \{x_{nv}|v \in \mathcal{V}\}$ to represent the caching vector for server n.

Table 1. Symbol Definition.

Symbols	Definition
$\mathcal{N}$	A set of all cache servers.
$\mathcal{V}$	A set of all videos.
$\mathcal{Q}$	A set of all quality versions of a video.
C_n	Cache space of server n.
S_{vq}	Size of file (v, q), i.e., quality version q of video v.
λ_{nvq}	Number of requests for file (v, q) from the users under base station n.
d_n	Unit data delivery delay from remote source server to a user under base station n.
$d_{nn'}$	Unit data delivery delay from server n to a user under base station n'
D_n	Delivery delay of server n for serving all its local requests in independent caching.
D^c	Delivery delay for serving all requests from all users in cooperative caching.
X	Caching vector, where each element x_{nv} equals the quality version of video v to be cached at server n while $x_{nv} = 0$ means no file related to video v is cached at server n.

3.3 Problem Formulation

A. Independent Caching with Transcoding. We first consider the situation that cache servers operate independently. In this situation, a user can only fetch a video file either from its local cache server (with high priority) or from the remote video source in the cloud (with low priority). Our goal is to find an optimal caching vector X_n for each individual cache server n ($n \in \mathcal{N}$) while minimizing the average delivery delay of all the video fetching of users covered by the base station n. Since the average delivery delay equals the total delivery delay of all the video fetching divided by the total number of requests, and the total number of requests is assumed to be a constant, the objective of minimizing the average delivery delay is equivalent to minimizing the total delivery delay

(denoted by D_n). The total delivery delay D_n can be computed as follows.

$$D_n = \sum_{v \in \mathcal{V}} \sum_{q \in \mathcal{Q}} \lambda_{nvq} S_{vq} d_n I^{x_{nv} < q}, \tag{1}$$

where $I^{\text{condition}}$ is an indicator function which equals 0 and 1, respectively, when the "condition" equals false and true. We explain (1) as follows. First, let us consider a given video quality version (v, q) requested by users under base station n. If the cached version for video v at server n can satisfy these requests, i.e., $x_{nv} \geq q$, we have $I^{x_{nv} < q} = 0$, and these users can fetch (v, q) from the local cache server directly without delay; Otherwise, we have $I^{x_{nv} < q} = 1$, and these users have to fetch it from the remote video source server with a delivery delay $\lambda_{nvq} S_{vq} d_n$, where λ_{nvq} is the total number of the requests for file (v, q), and $S_{vq} d_n$ is the video delivery delay for each such request. Consider all the versions of video v to be requested, we have the following total delivery delay (denoted by D_{nv}) for fetching all these versions.

$$D_{nv} = \sum_{q \in \mathcal{Q}} \lambda_{nvq} S_{vq} d_n I^{x_{nv} < q}. \tag{2}$$

Consider all the videos and all their quality versions to be requested, we have formula (1).

Finally, we formulate the optimal transcoding based independent caching at a cache server n as follows.

$$\begin{aligned} \underset{X_n}{\text{minimize}} \quad & D_n \\ \text{subject to} \quad & \sum_{v \in \mathcal{V}} \sum_{q \in \mathcal{Q}} S_{vq} I^{x_{nv} = q} \leq C_n \\ & x_{nv} \in \mathcal{Q} + \{0\}, \ \forall v \in \mathcal{V}. \end{aligned} \tag{3}$$

Equation (3) has two constraints. First, the size of the total cached files at a server n must not exceed the cache space C_n. Second, the value of x_{nv} must be an integer, and it indicates this is an integer programming problem, which is usually hard to be solved.

B. Cooperative Caching with Transcoding. We then consider the situation where the cache servers work cooperatively with each other for providing high-quality online VoD services. In this situation, among all the servers including the local cache servers and the remote video source server, a user can fetch a video file from a best server which can satisfy the request while leading to the minimal delivery delay. Our goal is to find a caching vector X for the local caching system while minimizing the average delivery delay of all users. Again, this objective is equivalent to minimize the total delivery delay (denoted by D^c) of all video fetching requests.

The total delivery delay D^c can be computed as follows,

$$D^c = \sum_{\substack{n\in\mathcal{N}\\ v\in\mathcal{V}\\ q\in\mathcal{Q}}} \{\lambda_{nvq} S_{vq} d_n \prod_{n'\in\mathcal{N}} I^{x_{n'v}<q} + (1 - \prod_{n'\in\mathcal{N}} I^{x_{n'v}<q}) \lambda_{nvq} S_{vq} \min_{n'\in\mathcal{N}'} d_{nn'}\}, \tag{4}$$

where $\mathcal{N}' = \{n' | n' \in N, I^{x_{n'v}<q} = 0\}$ is the set of the cache severs which can satisfy the requests for a given video version (v, q).

We explain (4) as follows. Let us first compute the delivery delay for a given combination of n, v, and q, i.e., transmitting video (v, q) to satisfy all user requests under base station n. On one hand, if none of the cache servers can satisfy the user requests, we have $I^{x_{n'v}<q} = 1, \forall n' \in \mathcal{N}$, and thus have $\prod_{n'\in\mathcal{N}} I^{x_{n'v}<q} = 1$. Thus, the users have to fetch (v, q) from the remote video source server, which leads to a delivery delay of $\lambda_{nvq} S_{vq} d_n$. On the other hand, if at least one cache server n' can satisfy the user requests, we have $I^{x_{n'v}<q} = 0$ for each such server n', and thus have $(1 - \prod_{n'\in\mathcal{N}} I^{x_{n'v}<q}) = 1$. Then, among all such cache severs $\mathcal{N}'$, we choose the cache server with the minimal delay to base station n to satisfy the requests of video (v, q), and we then have the total delivery delay as $\lambda_{nvq} S_{vq} \min_{n'\in\mathcal{N}'} d_{nn'}$. Here, we assume the delay between a pair of cache servers is smaller than that between the remote source server and a local cache server. Consider all the versions of video v to be requested, we have the following total delivery delay (denoted by D_v^c) for fetching all these versions.

$$D_v^c = \sum_{\substack{n\in\mathcal{N}\\ q\in\mathcal{Q}}} \{\lambda_{nvq} S_{vq} d_n \prod_{n'\in\mathcal{N}} I^{x_{n'v}<q} + (1 - \prod_{n'\in\mathcal{N}} I^{x_{n'v}<q}) \lambda_{nvq} S_{vq} \min_{n'\in\mathcal{N}'} d_{nn'}\}. \tag{5}$$

Consider all the videos and all their quality versions to be requested, we have formula (4).

Finally, we formulate the optimal transcoding based cooperative caching problem as follows.

$$\begin{aligned} \underset{X}{\text{minimize}} \quad & D^c \\ \text{subject to} \quad & \sum_{v\in\mathcal{V}} \sum_{q\in\mathcal{Q}} S_{vq} I^{x_{nv}=q} \le C_n, \ \forall n \in \mathcal{N} \\ & X_{nv} \in \mathcal{Q} + \{0\}, \ \forall n \in \mathcal{N}, \forall v \in \mathcal{V}. \end{aligned} \tag{6}$$

The two constrains in (6) are similar to those in (3).

4 Transcoding based Cache Algorithm

In this section, we design a transcoding based caching algorithm (TCA).

4.1 Algorithm Overview

TCA works for generating a caching vector X in a greedy manner such that it iteratively assigns a video quality version to an available cache server, which

leads to the maximal performance gain among all choices, until no file can be placed.

TCA works as follows. First, it initializes an all zero caching vector X with $x_{nv} = 0, \forall n, v$. Then, it iteratively updates X by placing a video version on a cache server, one video version placed each time. Given a set of video versions to be considered for caching at a set of cache servers, there typically exist multiple placement choices in each iteration. For each possible placement (n, v, q) (i.e., placing video (v, q) on server n), we compute the delivery delay for video v (called after-placement delay) by using (2) or (5), based on whether independent caching or cooperative caching is used, under the current X. Moreover, we have a delivery delay for video v before the placement (called base delay). The reduced delay (i.e., the base delay minus the after-placement delay) is called the delay gain associated with the placement. Among all possible placement choices, TCA chooses the choice leading to the maximal delay gain, and accordingly updates the caching vector. This process is repeated until no more file can be cached at any of the cache servers or no more video version needs to be considered for caching.

4.2 Algorithm Design

The procedure of TCA is shown in Algorithm 1, which is described partially based on the code framework in [5].

First, a series of variables are initialized in lines 1–4, including i) $E = \{e_{nvq} | n \in \mathcal{N}, v \in \mathcal{V}, q \in \mathcal{Q}\}$ is initialized as a set containing all possible placement choices, where each element $e_{nvq} \in E$ represents a possible placement, i.e., it is still applicable to place (v, q) at server n, ii) the possible placements $E_n \subseteq E$ for each cache server n, iii) an all zero caching vector X, and iv) an all zero caching vector X_n for each cache server n.

Then, it iteratively updates X (see line 5 to line 24 in Algorithm 1). In each iteration, line 6 finds the best placement $e_{\alpha\beta\gamma}$ (i.e., placing the γ version of video β at server α) among all possible placements in E, where function $J_X(g)$ computes the delay gain of a possible placement g based on the current caching vector X. Following that, if the maximal delay gain equals zero, TCA will break the loop; Otherwise, it will continue the following operations. Line 10 adjusts cache size C_α, by removing the space to be occupied by the new video version (β, γ) and also retrieving the space occupied by the previously cached video version $(\beta, x_{\alpha\beta})$ (if any). Lines 11 and 12 update $x_{\alpha\beta} = \gamma$ in X and X_α, respectively, according to the new choice of $e_{\alpha\beta\gamma}$; Lines 13 and 14 removes placements $e_{nvq}, \forall n = \alpha, v = \beta, q \leq r$, from E and E_α, since all the user requests for fetching $(\beta, q), q \leq r$ from server α can be satisfied by this new placement and thus it no longer needs to cache any lower-version of the video α. Next, lines 15–20 remove those placements, which are impossible to be realized due to limited remaining cache space, from E and E_n. Following that, if E becomes empty (see line 21), TCA will break the loop. Finally, TCA outputs the final caching vector X.

The computational complexity of TCA can be easily deduced as follows. The initialization in lines 1–4 obviously takes $O(\mathcal{NVQ})$ time. In the "for" loop between line 5 and line 24, line 6 has the highest complexity of $O(\mathcal{NVQ})$ and the "for" loop takes at most $O(\mathcal{NVQ})$ time. Thus, the computational complexity of TCA is $O(\mathcal{N}^2\mathcal{V}^2\mathcal{Q}^2)$.

Algorithm 1 Procedure of TCA.

1: $E = \{e_{nvq} | \forall n \in \mathcal{N}, \forall v \in \mathcal{V}, \forall q \in \mathcal{Q}\}$
2: $E_n = \{e_{nvq} | \forall v \in \mathcal{V}, \forall q \in \mathcal{Q}\}, n \in \mathcal{N}$
3: $X \leftarrow \{0, 0, \cdots, 0\}$
4: $X_n \leftarrow \{0, 0, \cdots, 0\}, \forall n \in \mathcal{N}$
5: **for** $i = 1, 2, ..., N \times V \times Q$ **do**
6: $e_{\alpha\beta\gamma} = \text{argmax}_{g \in E} J_X(g)$
7: **if** $J_X(e_{\alpha\beta\gamma}) = 0$ **then**
8: **break**
9: **end if**
10: $C_\alpha \leftarrow C_\alpha - S_{\beta\gamma} + S_{\beta x_{\alpha\beta}}$
/*note: $S_{\beta x_{\alpha\beta}} = 0$ if $x_{\alpha\beta} = 0$*/
11: $x_{\alpha\beta}$ of $X \leftarrow \gamma$
12: $x_{\alpha\beta}$ of $X_\alpha \leftarrow \gamma$
13: $E \leftarrow E \backslash e_{\alpha\beta q}, \forall q \leq \gamma$
14: $E_\alpha \leftarrow E_\alpha \backslash e_{\alpha\beta q}, \forall q \leq \gamma$
15: **for** $e_{\alpha vq} \in E_\alpha$ **do**
16: **if** $C_\alpha + S_{v x_{\alpha v}} < S_{vq}$ **then**
17: $E \leftarrow E \backslash e_{\alpha vq}$
18: $E_\alpha \leftarrow E_\alpha \backslash e_{\alpha vq}$
19: **end if**
20: **end for**
21: **if** $|E| = 0$ **then**
22: **break**
23: **end if**
24: **end for**
25: Ouput X

5 Performance Evaluation

In this section, we perform simulations to evaluate the performance of TCA. We compare three instances of two caching algorithms, including two instances of TCA, which adopt independent caching (TCA-I) and cooperative caching (TCA-C), respectively, and the traditional greedy algorithm (Greedy) for cooperative caching [5].

5.1 Parameter settings

We set $N = 3$ cache servers for simulations, each having the same cache size. The default rate of transmitting videos between each of the servers and the remote source server is 2 Mbps, and the default rate (i.e., cooperative transmit rate) among the cache servers for video delivery in the mode of cooperative caching was set to 5 Mbps. The default cache size at a cache server is 400 GByte. We assume there are 1000 videos for caching, all of which have one-hour playback duration. According to the Youtube recommended resolutions and bitrates, each video supports 5 different quality levels of video resolutions, including 240p, 360p, 480p, 720p and 1080p, which correspond to video bitrates of 400, 750, 1000, 2500 and 4500kbps, respectively [21]. Thus, the total size of the 5000 video versions is close to 4 TB.

Following typical settings used for empirical studies in VoD system [9, 22], the popularity of the 1000 videos is assumed to follow a Zipf distribution, i.e., the i^{th} most popular video will be requested at a rate in proportion to i^{-z}, where z is a shape parameter and was set to 0.8 unless otherwise specified. Furthermore, different versions of a video are assumed to be equally requested by users.

5.2 Numerical Results

Impact of cooperative transmit rate on delivery delay. For this test, we conducted a series of simulations with varying cooperative transmit rate ranging from 2 to 11 Mbps at 1 Mbps granularity. The results are shown in Fig. 2, where the x-axis is the cooperative transmit rate, and the y-axis is the average video transmission delay. The three curves correspond to TCA-C, Greedy, and TCA-I, respectively. The former two algorithms adopt cooperative caching and can reduce the delivery delay by increasing the cooperative transmit rate. The third algorithm uses independent local caching and thus has a constant performance irrelevant to cooperative transmit rate. The results in this figure clearly demonstrate that TCA-C has the best performance, since it makes full use of both transcoding and cooperative caching at different cache servers. The gains in terms of delivery delay is up to 40% and 38% as compared to TCA-I and Greedy, respectively.

Impact of cache size on delivery delay. In this test, we conduct a series of simulations with varying cache size ranging from 50 GB to 800 GB at 100 GB granularity. Fig. 3 shows the results. In Fig. 3, all the three curves show similar decreasing trends on average delivery delay with cache space increasing. Moreover, for each given cache size, the results show that TCA-C significantly outperforms TCA-I and Greedy, with a decrease of average delivery delay by up to 50% and 53%, respectively. The results indicate that TCA-C can efficiently utilize available cache space, so as to improve user experience of video fetching.

Impact of video popularity distribution on delivery delay. In this test, we conduct a series of simulations with varying shape parameter Zipf ranging from 0.4 to 1.2 at 0.1 granularity. Fig. 4 shows the results. Not surprisingly, all the

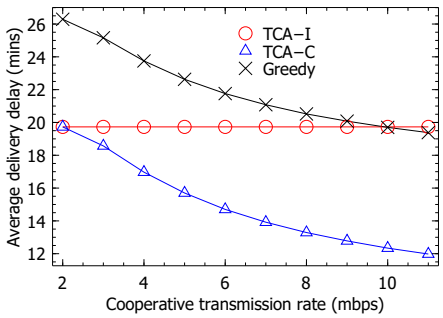

Fig. 2. Impact of cooperative transmit rate.

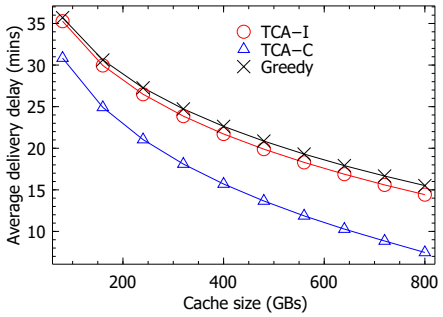

Fig. 3. Impact of cache size.

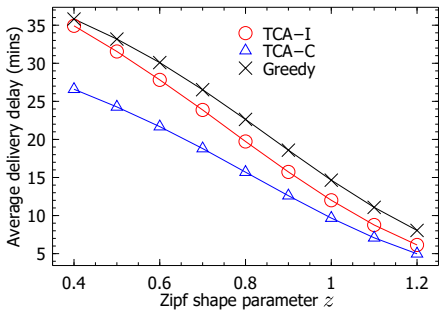

Fig. 4. Impact of Zipf shape parameter.

simulated algorithms lead to decrease on delivery delay with increasing of the shape parameter z. This is because a larger z indicates fewer videos are more popular such that the caching of them can satisfy a large proportion of all users' requests. Fig. 4 shows that TCA-C always has the best performance.

6 Conclusion

In this paper, we studied transcoding based caching for improving the performance of a distributed video caching system. We formulated optimal transcoding based video caching problem under two different modes (i.e., independent caching and cooperative caching) as integer linear programming problem. We then proposed a transcoding based caching algorithm TCA, which iteratively places a video version to a cache server, which leads to the maximal delay gain among all possible choices. Simulation results demonstrate that TCA (when working in cooperative caching mode) can significantly reduce the delivery delay compared with traditional greedy algorithm.

References

1. Q. Zhang, Z. Xiang, W. Zhu and L. Gao, "Cost-based cache replacement and server selection for multimedia proxy across wireless Internet," *IEEE Transactions on Multimedia*, vol. 6, no. 4, pp. 587–598, 2004.
2. H. Chen and Y. Xiao, "Cache access and replacement for future wireless Internet," *IEEE Communications Magazine*, vol. 44, no. 5, pp. 113–123, 2006.
3. R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Systems Journal*, vol. 9, no. 2, pp. 78–117, 1970.
4. A. V. Aho, P. J. Denning and J. D. Ullman, "Principles of optimal page replacement," *Journal of the ACM (JACM)*, vol. 18, no. 1, pp. 80–93, 1971.
5. K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch and G. Caire, "FemtoCaching: Wireless video content delivery through distributed caching helpers," *In: IEEE INFOCOM 2012*, pp. 1107–1115, Orlando, FL, USA, 2012.
6. L. E. Chatzieleftheriou, M. Karaliopoulosand and I. Koutsopoulos, "Caching-aware recommendations: Nudging user preferences towards better caching performance," *In: IEEE INFOCOM 2017*, pp. 1–9, Atlanta, GA, USA, 2017.
7. H. Schwartz, D. Marpe and T. Wiegand, "Overview of the scalable video coding extension of the H.264/AVC standard," *IEEE Transactions on Circuits and System for Video Technology*, vol. 17, no. 9, pp. 1103–1120, 2007.
8. A. Vetro, C. Christopoulos and H. Sun, "Video transcoding architectures and techniques: An overview," *IEEE Signal Processing Magazine*, vol. 20, no. 2, pp. 18–29, 2003.
9. K. Poularakis, G. Iosifidis, A. Argyriou, I. Koutsopoulos and L. Tassiulas, "Caching and operator cooperation policies for layered video content delivery," *In: IEEE INFOCOM 2016*, pp. 1–9, San Francisco, CA, 2016.
10. G. Zhang, "Computational complexity optimization on H.264 scalable/multiview video coding," *PhD thesis*, University of Central Lancashire, 2014.
11. B. Shen and S. Roy, "A very fast video special resolution reduction transcoder," *In: IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 1989–1992, Orlando, FL, USA, 2012.

12. B. Shen, S. J. Lee and S. Basu, "Caching strategies in transcoding-enabled proxy systems for streaming media distribution networks," *IEEE Transactions on Multimedia*, vol. 6, no. 2, pp. 375–386, 2009.
13. S. E. Elayoubi and J. Roberts, "Performance evaluation of video transcoding and caching solutions in mobile networks," *In: 27th International Teletraffic congress*, pp. 55–63. Ghent, Belgium, 2015.
14. T. X. Tran, P. Pandey, A. Hajisami and D. Pompili, "Collaborative multi-bitrate video caching and processing in mobile-edge computing networks," *In: IEEE WONS*, pp. 165–172, Jackson Hole, Wyoming, USA, 2017.
15. G. Gao, W. Zhang, Y. Wen, Z. Wang and W. Zhu, "Towards cost-efficient video transcoding in media cloud: insights learned from user viewing patterns," *IEEE Transactions on Multimedia*, vol. 17, no. 8, pp.1286–1296, 2015.
16. X. Li, M. A. Salehi, M. Bayoumi, N. Tzeng and R. Buyya, "Cost-efficient and robust on-demand video transcoding using heterogeneous cloud services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 3, pp. 556–571, 2018.
17. A. Albanese, P. S. Crosta, C. Meani and P. Paglierani, "GPU-accelerated video transcoding unit for multi-access edge computing scenario," *In: ICN 2017*, pp. 143–147, Venice, Italy, 2017.
18. S. Dutta, T. Taleb, P. A. Frangoudis and A. Ksentini, "On-the-Fly QoE-aware transcoding in the mobile edge," *In: IEEE Globecom 2016*, pp. 1–6, Washington, DC, USA, 2016.
19. K. Fung and W. Siu, "Low complexity H.263 to H.264 video transcoding using motion vector decomposition," *In: IEEE ISCAS 2005*, pp. 908–911, Kobe, Japan, 2005.
20. A. J. Daz-Honrubia, G. Cebrin-Mrquez, J. L. Martnez et al, "Low-complexity heterogeneous architecture for H.264/HEVC video transcoding," *Journal of Real-Time Image Processing*, vol. 12, no. 2, pp. 311–327,2016.
21. Youtube live encoder settings, https://support.google.com/youtube/answer/2853702?hl=en. Last accessed Apr. 18, 2018.
22. M. Hefeeda and O. Saleh, "Traffic modeling and proportional partial caching for Peer-to-Peer systems," *IEEE/ACM Transactions on Networking*, vol. 16, no. 6), pp. 1447–1460, 2008.