vCache: Supporting Cost-Efficient Adaptive Bitrate Streaming via NFV-Based Virtual Caching

Guanyu Gao, Yonggang Wen, and Jianfei Cai
School of Computer Science and Engineering
Nanyang Technological University
Email: {ggao001, ygwen, asjfcai}@ntu.edu.sg

***Abstract*—Each video must be transcoded into multiple representations in Adaptive Bitrate Streaming (ABR). Transcoding and caching videos consume tremendous resources, however, only a small percentage of video chunks are frequently requested. Thus a question arises: is it necessary to pre-transcode each video and cache all video chunks? To answer this, we design a Network Functions Virtualization (NFV) based virtual cache (vCache). In vCache, video chunks have two mutually-exclusive caching states: physically cached and virtually cached. A physically cached video chunk can be directly read from storage, and it consumes storage resources. A virtually cached video chunk will be transcoded online when being requested, and it consumes computing resources. With NFV, vCache can dynamically manage video chunks to achieve cost-efficiency, and intelligently provision resources to guarantee transcoding delays not to affect streaming services. The experiment results show that vCache can greatly reduce the operational cost for ABR.**

***Index Terms*—Video streaming, video caching, ABR, NFV**

I. Introduction

Video streaming dominates Internet traffic, accounting for more than 70% of North American downstream traffic at peak time [1]. However, limited bandwidth capacity, unstable network condition, and diverse viewing devices inherently deteriorate user experiences, triggering a tussle between the growing demand of video traffic and quality of viewing experiences [2]. ABR is a widely adopted solution for improving viewing experiences under such a condition.

The video processing flow for ABR is illustrated in Fig. 1. Each video must be transcoded into multiple representations, and then cached in streaming servers. A Media Presentation Description (MPD) file is required to manifest the available representations for a video [3]. When starting a video session, the video player first obtains the MPD file of a video, and then selects the best possible quality representations according to the current network condition and device capacity. However, transcoding is compute intensive and consumes tremendous resources. Caching multiple representations of a video consumes several times of storage space. Thus, streaming videos in ABR can greatly increase the operational cost.

In contrast to the tremendous resource consumption of ABR, it is observed that only a small percentage of video chunks are frequently requested by users. Specifically, the top 10% of the most popular videos account for almost 80% of total views [4], [5]; for 60% of video sessions, only less than 20% of the durations are viewed, and most of users abort viewing within 40 seconds [6], [7]. These user viewing patterns reveal

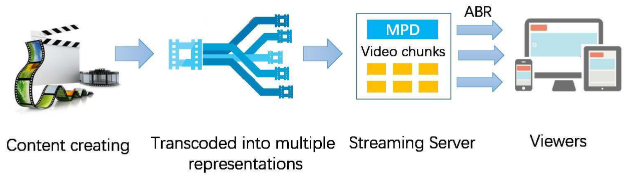


Fig. 1. The video processing flow for ABR. Videos are transcoded into multiple representations and cached in streaming servers for delivery.

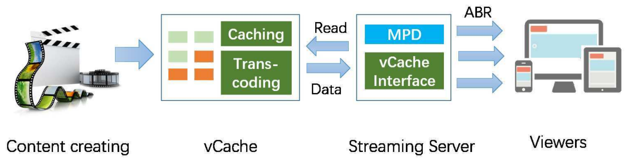


Fig. 2. The video processing flow with vCache. Videos are dynamically transcoded and cached to be delivered in ABR.

that users consume only a small fraction of video chunks. Thus a question arises: considering the tremendous computing and storage resource consumption for transcoding and caching videos, is it necessary to pre-transcode each video and always keep the video chunks of all representations in storage?

To answer this question, we design *vCache*, a NFV-based virtual caching scheme, to manage videos for ABR to minimize overall operational cost. Our design is based on the user viewing patterns observed in many online video services. For seldom requested video chunks, it is more cost-efficient to generate them on the fly, rather than always caching them. In our design, a video chunk in vCache can be in one of the two states. *Physically Cached:* the video chunk is cached, and can be directly read from storage. *Virtually Cached:* only the metadata of the video chunk is cached. The metadata contains the location of the source version of this video chunk and the corresponding transcoding parameters. The metadata is used for transcoding a virtually cached video chunk on the fly.

We illustrate the video processing flow for ABR with vCache in Fig. 2. The MPD files are cached in streaming servers for manifesting the available representations of each video. The video player obtains the MPD file from the streaming server when starting a new session, and then requests video chunks of the appropriate representations. The streaming server will read the requested video chunks in vCache, and the video chunks in vCache are dynamically cached or transcoded on the fly for serving user requests.

vCache is different from traditional methods in two perspectives. First, a video will not be pre-transcoded into multiple representations, because many videos, especially some representations, may seldom be requested. Pre-transcoding a video and caching all representations waste tremendous resources if seldom requested. In our design, all representations of a video are only virtually cached when the source video file is initially ingested into vCache. The transcoding for a video chunk is postponed to the time when it is being requested. Second, vCache manages video chunks dynamically rather than statically to minimize overall cost. Because video popularity changes over time [4], the caching decisions are made dynamically to capture time-varying video popularity.

We leverage the NFV framework to implement vCache as a virtualized network function. Streaming servers can access video chunks in vCache as accessing a common storage, while the underling mechanisms of vCache remain transparent to other applications. As transcoding video chunks on the fly incurs delays, vCache dynamically provision resources to ensure transcoding delays are within an acceptable range. vCache strikes a tradeoff between storage cost and computing cost, and it can reduce the operational cost for ABR.

This work is not the first to consider leveraging SDN/NFV and transcoding to improve the performance of ABR. The work in [8] introduced a Virtual Network Function (VNF) transcoding method for reinstating the QoE level of video streaming when network congestion occurs. The work in [9] proposed a framework for provisioning and programming of acceleration hardware for VNFs, and transcoding can be speeded up by acceleration hardware. The difference compared with [8], [9] is that this work studies how to manage video files for ABR cost-efficiently. We proposed a resource provisioning method for transcoding in Information Centric Networking (ICN) in [10], yet in [10] we did not consider video management for ABR. The work in [11] and our work [12] considered the file-level and segment-level trade-off between computing cost and storage cost for video caching. The main differences of this work compared with [11], [12] are that we consider how to control transcoding delays by dynamically provisioning computing resources, and how to implement the mechanism for ABR under the NFV architecture.

The rest of this article is organized as follows. Section II presents the system design. Section III presents the system models. Section IV presents the dynamic control policies for content management and resource provisioning. Section V presents some practical considerations for improving the performance. Section VI concludes this article.

II. System Design

A. Framework

The framework of vCache is illustrated in Fig. 3. The system mainly consists of the following modules.

Resource Virtualization Module: The hardware resources in NFV infrastructure consist of storage, computing, and network devices. The virtualization layer decouples the virtualized network functions in the upper layer from the underlying hardware [13]. It provides the capacity of underlying hardware as virtual computing, storage, and network resources.

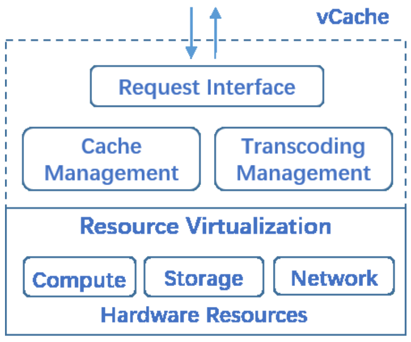


Fig. 3. The framework of vCache. The system consists of the resource virtualization module, the request interface module, the cache management module, and the transcoding management module.

Request Interface Module: The request interface module processes the requests from streaming servers. When receiving a request, the request interface module first locates the video chunk in storage. If the video chunk is physically cached, it will be read directly. Otherwise, the request interface module will initiate a transcoding request. A transcoding operation will be performed to transcode the video chunk of the source version into the user requested representation, and the transcoded video chunk will be rendered to the streaming server.

Cache Management Module: The cache management module dynamically determines whether a video chunk should be physically cached or virtually cached. The source file of each video is always cached for generating other representations on the fly. The cache management module analyzes the request information of each video chunk to make caching decisions for reducing the overall cost.

Transcoding Management Module: A virtually cached video chunk will be transcoded on the fly when being requested, and this incurs delays. As the transcoding workload is time-varying, the transcoding management module dynamically provisions resources according to the transcoding workload to ensure that transcoding delays are within an acceptable range.

B. Workflow

We illustrate the workflow for fulfilling a user request in Fig. 4. When receiving a request for a video chunk, vCache first lookups the requested video chunk. A physically cached video chunk can be read and delivered directly. For a virtually cached video chunk, a transcoding request will be sent to the transcoding cluster. The transcoding cluster consists of many homogeneous virtual machines (VMs). The dispatcher equally dispatches transcoding requests among the active VMs for load balancing. A transcoding server will transcode the requested video chunk on the fly when receiving a request.

C. Incorporated with ABR

vCache can be easily incorporated with current ABR solutions. As illustrated in Fig. 5, it can be implemented as a

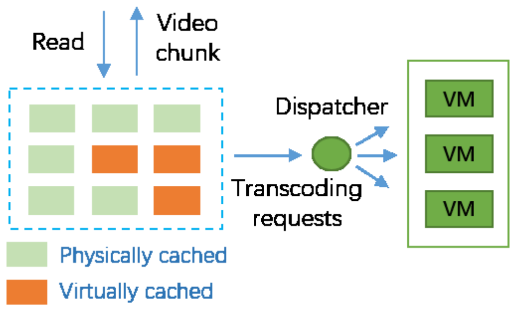


Fig. 4. The workflow for fulfilling a user request. If the requested video chunk is physically cached, it will be read and delivered directly. If the requested video chunk is virtually cached, it will be transcoded on the fly.

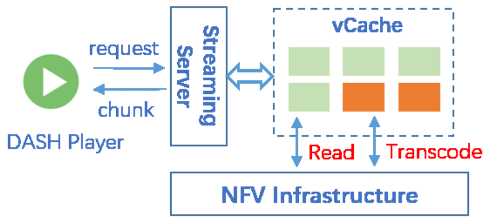


Fig. 5. Incorporate vCache with ABR. vCache can be implemented as a virtualized network function while remaining transparent to other applications.

virtualized network function by leveraging the NFV infrastructure. A video player can parse the MPD files and requests video chunks from the streaming server. The streaming server will read the video chunks from vCache. Because the underlaying caching and transcoding mechanisms of vCache are transparent to other applications, vCache can serve the streaming servers and other applications like common storage systems, e.g., hard disks, distributed file systems, etc. Thus, vCache will not affect the current implementation of ABR.

III. System Modeling and Problem Formulation

The system can typically make caching decisions each week, and make resource scaling decisions each hour.

A. Operational Cost

The operational cost consists of storage cost and computing cost. The storage cost is incurred by physically caching video chunks, and the computing cost is incurred by transcoding video chunks on the fly. For video chunk i, we denote the storage cost for physically caching it during a time period as s_i. We assume that the estimated request frequency of video chunk i during time period T is $f_i(T)$, and the computing cost for transcoding video chunk i on the fly for one time is c_i. If video chunk i is virtually cached during time period T, the computing cost for transcoding it on the fly during time period T can be estimated as $f_i(T)c_i$.

B. Processing Delay

The processing delay of a transcoding request consists of the queueing time and the transcoding time for the video chunk. Theoretically, we can adopt the M/G/1 or G/G/1 model to analyze the processing delay in a VM under varying transcoding request arrival rates. In practice, we can adopt some empirical methods to learn the relation between the transcoding request processing delay and the transcoding request arrival rate in a VM. To guarantee that transcoding delays are within an acceptable range, the system must ensure that the transcoding request arrival rate in each VM is less than a preset threshold.

C. Problem Formulation

Our objective is to minimize the overall operational cost while guaranteeing that transcoding delays incurred by virtual caching are within an acceptable range. At the beginning of each time period T, the system determines whether a video chunk should be physically cached or virtually cached for minimizing the overall cost, which can be presented as follow,

$$\underset{\vec{b}}{\text{minimize}} \quad \sum_{i \in K} \mathbb{E}\{s_i b_i(T) + (1 - b_i(T)) f_i(T) c_i\}, \tag{1}$$

where K is the set of existing video chunks, and $b_i(T)$ is a binary decision variable for video chunk i. Video chunk i will be physically cached at time period T if $b_i(T)$ equals one; and it will be virtually cached if $b_i(T)$ equals zero.

For a video chunk that was physically cached during time period $T-1$, it will be removed from storage if its binary decision variable equals zero at time period T. However, for a video chunk that was virtually cached during time period $T-1$, it will be transcoded on the fly when being requested for the first time during time period T. The transcoded video chunk will be physically cached if its binary decision variable equals one at time period T; or it will be discarded after a single use if its binary decision variable equals zero.

The transcoding management module scales the computing capacity of the transcoding cluster at a higher frequency. We assume the scaling decision is made at each time slot t, where the duration of each time slot t is much shorter than that of T. The transcoding management module estimates the transcoding request arrival rate at the beginning of each time slot t. It provisions the right number of VMs so that the transcoding delays in each VM will not exceed the preset threshold to guarantee the delay requirement,

$$g(\hat{\lambda}(t)/n(t)) < \delta, \tag{2}$$

where $\hat{\lambda(t)}$ is the estimated transcoding request arrival rate during time slot t, $n(t)$ is the number of provisioned VMs at time slot t, $g(\cdot)$ is a function that describes the relation between the processing delay and the transcoding request arrival rate in a VM, and δ is the preset maximum acceptable processing delay for a transcoding request.

IV. Dynamic Control Policies

A. Dynamic Caching Policy

The dynamic caching policy is implemented in the cache management module, and it determines whether a video chunk should be physically cached or virtually cached during each time period. The dynamic caching policy makes caching decision for a video chunk based on the request frequency,

TABLE I
THE STORAGE COST AND COMPUTING COST

Cost (10^{-4} USD)	720p	480p	360p	240p
Storage Cost	0.579	0.396	0.312	0.282
Computing Cost	0.146	0.131	0.114	0.108

$f_i(T)$, to minimize the cost. We estimate the request frequency of a video chunk as its request frequency during the last time period. The cache management module updates the caching state (physically cached or virtually cached) of each video chunk at the beginning of each time period T. We can make the caching decision for each video chunk independent of the others to minimize the overall cost. Therefore, the algorithm complexity for making caching decisions for all video chunks is $O(n)$, where n is the number of video chunks.

We conduct the simulation with the real-world dataset provided in [14]. The dataset contains video viewing information collected from YouTube over 21 weeks, and the video request rate information is collected once a week. We adopt the request rate for a video in the past week to predict the request rate in the next week. Most of the users just view the first several video chunks before aborting a video session. We calculate the probability of that a user would watch a video up to a specified duration according to the data provided in [6].

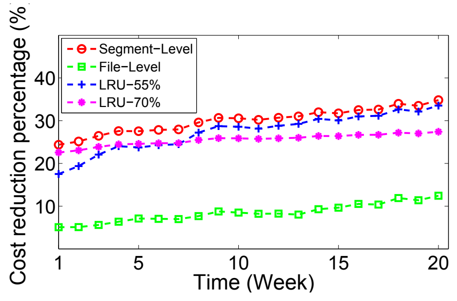
We randomly select 1000 videos in the dataset for the experiments. Each source video file is transcoded into four representations, namely, 720p, 480p, 360p, and 240p, and each representation is equally requested. Table I illustrates the storage cost for caching a video chunk of each representation for one month and the computing cost for transcoding a video from the source version to each representation for one time.

We evaluate the cost reduction percentage of our method and compare it with some baseline methods. The cost reduction percentage is calculated by comparing with the method of physically caching all the video chunks. We calculate the cost reduction percentage using the following equation,

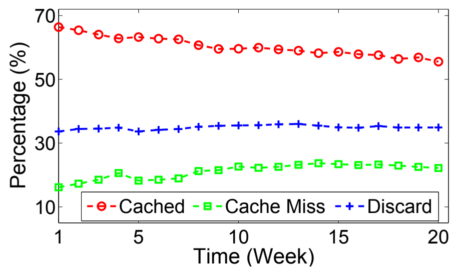
$$Cost\ Reduction\ Percentage = \frac{C_A - C_O}{C_A} * 100\%, \quad (3)$$

where C_A is the cost for physically caching all video chunks, and C_O is the cost for our method. We compare our method (*Segment-Level*) with the following baseline methods: *File-Level*, the video chunks of a video file are either all physically cached or all virtually cached; *LRU-55%*, caching 55% of video chunks with Least Recently Used (LRU) based video chunk replacement policy; *LRU-70%*, caching 70% of video chunks with LRU-based video chunk replacement policy. Fig. 6(a) shows that our method can save about 30% of the cost, and the cost reduction percentage increases over time. This is because most of videos gain more views when they are initially released, but the video popularity decreases over time.

Compared with *File-Level*, our method save more cost because different parts of a video have different popularity, and most of users only view the initial part of a video, thus it is not cost-efficient to cache a whole video file. Our method also saves more cost than the LRU-based methods, because our method can dynamically determine the percentage of video chunks that should be physically cached by analyzing request



(a) The cost reduction percentage of different methods.



(b) The performance of vCache over time.

Fig. 6. The performance of the dynamic caching policy.

frequency, and the LRU-based methods do not differentiate video chunks of different representations, yet different video chunks incur different cost for transcoding and caching.

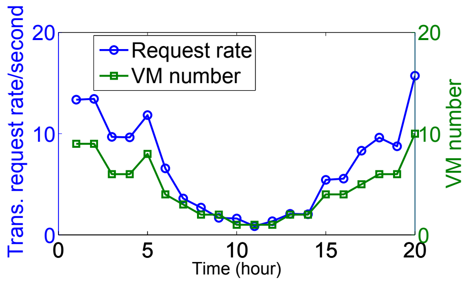
In Fig. 6(b), we illustrate the performance of vCache over time, including the percentage of physically cached video chunks (*Cached*), the percentage of requests incurring cache miss (*Cache Miss*), and the percentage of video chunks that have been transcoded on the fly but discarded right after a single use (*Discard*). It can be observed that the percentage of physically cached video chunks decreases over time. This is because video popularity generally decreases over time. The seldom requested video chunks will be virtually cached for reducing storage cost, thus the cache miss ratio increases slightly. About 30% video chunks are transcoded on the fly but discarded right after a single use, because caching these seldom requested video chunks will incur more cost.

B. Dynamic Scaling Policy

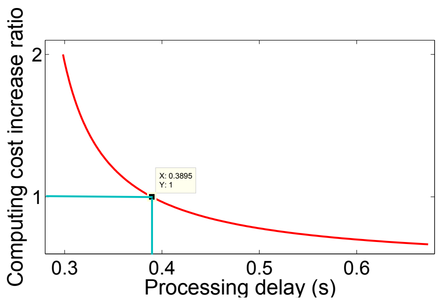
The dynamic scaling policy is implemented in the transcoding management module, and it controls the number of provisioned VMs in the transcoding cluster. The transcoding request arrival rate is affected by many factors, including the percentage of physically cached video chunks, the user request rate, the number of new released videos, etc. Video streaming is very sensitive to delays. It may greatly deteriorate user experience if transcoding delays are not well controlled.

To address this problem, we adopt a simple and robust strategy for scaling the transcoding cluster to accommodate to the time-varying workloads. To make the performance robust to the fluctuation of the workloads, the system observes the transcoding request arrival rate at the beginning of each time slot and over-provisions resources by a certain percentage,

$$\hat{\lambda}(t) = \alpha\lambda(t), \alpha > 1, \quad (4)$$



(a) The number of provisioned VMs over time.



(b) Computing cost increase ratio under different delay constraints.

Fig. 7. The performance of the dynamic scaling policy.

where $\lambda(t)$ is the observed transcoding request arrival rate, α is a factor larger than one, and $\hat{\lambda}(t)$ is the estimated transcoding request arrival rate for avoiding resource under-provisioning. The system provisions resources according to the estimated transcoding request arrival rate $\hat{\lambda}(t)$.

To guarantee transcoding delays not to exceed δ, the transcoding request arrival rate in each VM cannot exceed the maximum allowed threshold. We can apply Eq. (2) to derive the maximum allowed transcoding request arrival rate for each VM in real environments. We let the maximum allowed transcoding request arrival rate in each VM be λ_m, and the minimum required VM number can be calculated as

$$n(t) = \lceil \hat{\lambda}(t)/\lambda_m \rceil, \tag{5}$$

where $n(t)$ is the number of active VMs provisioned for transcoding virtually cached video chunks on the fly. A more general case is to impose a percentile delay bound on transcoding delays, for instance, 99% of transcoding delays are less than δ. In this case, $g(\cdot)$ should map the transcoding request arrival rate in a VM to the 99th percentile of delay.

We illustrate the mechanism of dynamic scaling policy in Fig. 7(a). We adopt a workload trace which captures video requests to a streaming server in CDN, and we extract the user requests in the trace as the transcoding requests in vCache. In our simulation, α is 1.2 and λ_m is 2. The system dynamically controls the number of active VMs to ensure that the transcoding request arrival rate is less than λ_m in each VM for guaranteeing the transcoding performance.

A system may set λ_m according to the delay requirement. A smaller λ_m requires more computing resources for serving transcoding requests, but will incur fewer delays. We illustrate the relation between the processing delay threshold and the computing cost increase ratio in Fig. 7(b). The computing cost increase ratio is compared with $\lambda_m = 2$ and the corresponding delay threshold is 0.3895s. We can observe that with more stringent processing delay requirement, it will incur more computing cost for reducing the processing delays.

C. Throughput and Delay

We can set the appropriate value of δ according to the delay requirement. If the required computing resources and storage resources can be fully satisfied, the system performance can be guaranteed. In this sense, vCache will not affect the overall throughput of the streaming system. Another concern is the lower bound of δ. Transcoding incurs unavoidable delays. The transcoding time for a video chunk is affected by the duration of the video chunk, the CPU frequency of the server, and the target representation, etc. To reduce delays, we can segment videos into small-sized video chunks, or we can adopt some parallelization mechanisms so that the independent Group of Pictures (GoPs) in a video chunk can be processed in parallel.

V. Practical Consideration

We can adopt some other techniques in vCache for improving its performance and further reducing the operational cost.

Joint Optimization Method: One promising approach for improving the performance is to jointly optimize the caching module and resource scaling module. First, the caching module maintains the request information and the current state of each video chunk. This can provide the resource scaling module with a more precise prediction of the transcoding request rates. Second, the amount of idle computing resources is time-varying, and the caching module can strike a better trade-off between the computing cost and storage cost by considering the idle computing resource information. We can implement the two modules as two VNFs under a centralized control.

Pre-transcoding Subsequent Video Chunks: To further reduce transcoding delays, we can pre-transcode the subsequent several virtually cached video chunks of the currently being requested one. Because users usually watch videos sequentially, the subsequent video chunks have a higher probability to be requested. This can greatly reduce transcoding delays, however, it may waste computing resources if the user aborts the current video session or jumps to another part of the video. Another scenario is that a video may have the similar popularity changing patterns among different regions but with time-lags. In this case, when a video becomes popular in one region, we can pre-transcode the video and physically cache it in the other regions to eliminate the transcoding delays.

Matching Video Representations with Network Conditions: The current ABR solution requires that videos are pre-transcoded into several representations, and the parameters of each representation are pre-determined. However, this may lead to mismatch between the available representations and users' real network conditions. For instance, a video is mostly viewed on mobile devices, however, it was pre-transcoded into many representations in high resolutions. Videos are only virtually cached when they are initially ingested into vCache, therefore, vCache can dynamically determine the available representations of each video based on users' viewing patterns and network conditions. We can implement some representation selection algorithms in vCache so that it can provide users with the most matching representations.

Utilizing Idle Resources for Achieving a Better Trade-off: To improve resource utilization, vCache can utilize the idle resources in different regions for caching and transcoding to achieve a better tradeoff between computing resource consumption and storage resource consumption. For instance, if more idle computing resources are available in one region, vCache can use the idle computing resources to serve some transcoding-on-the-fly requests for another region. In contrast, if more idle storage resources are available, vCache can use the idle storage resources to physically cache more video chunks.

Cross-Region/-Datacenter Optimization: A video may be cached at the different regions or datacenters for reducing streaming delays. In this case, we may consider that the caching system of each region/datacenter is an instance of v-Cache, and each of the instances works independently. We can also design some sophisticated mechanisms so that different instances of vCache can work collaboratively. For example, when an instance of vCache receives a request for a virtually cached video chunk, the instance may request the video chunk from other instances which have the data; or the instance may offload the transcoding request to another instance which has abundant computing resources. In these cases, network delays and bandwidth consumption should also be considered.

VI. Conclusion

We design vCache to reduce the operational cost for ABR under the NFV infrastructure. Traditional video caching methods for ABR may waste tremendous resources, because each video has multiple representations cached in storage, yet only a small proportion are frequently requested. The design of vCache follows this principle: for the seldom requested video chunks, it is more cost-efficient to generate them on the fly, rather than always caching them. vCache makes a trade-off between storage cost and computing cost based on video popularity for minimizing the overall cost. To guarantee transcoding delays not to affect streaming services, vCache dynamically provisions computing resources to accommodate to transcoding workloads. vCache can greatly reduce the operational cost for caching adaptive videos, and can be easily incorporated with current ABR solutions.

References

[1] *Global Internet Phenomena Report*, Sandvine, 2015.

[2] Y. Wen, X. Zhu, J. Rodrigues, and C. Chen, "Cloud mobile media: Reflections and outlook," *Multimedia, IEEE Transactions on*, vol. 16, no. 4, pp. 885–902, 2014.

[3] I. Sodagar, "The mpeg-dash standard for multimedia streaming over the internet," *IEEE MultiMedia*, no. 4, pp. 62–67, 2011.

[4] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "Analyzing the video popularity characteristics of large-scale user generated content systems," *IEEE/ACM Transactions on Networking (TON)*, vol. 17, no. 5, pp. 1357–1370, 2009.

[5] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Characteristics of youtube network traffic at a campus network–measurements, models, and implications," *Computer Networks*, vol. 53, no. 4, pp. 501–514, 2009.

[6] A. Finamore, M. Mellia, M. M. Munafò, R. Torres, and S. G. Rao, "Youtube everywhere: Impact of device and infrastructure synergies on user experience," in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM, 2011, pp. 345–360.

[7] L. C. Miranda, R. L. Santos, and A. H. Laender, "Characterizing video access patterns in mainstream media portals," in *Proceedings of the 22nd international conference on World Wide Web companion*. International World Wide Web Conferences Steering Committee, 2013, pp. 1085–1092.

[8] H. Koumaras, C. Sakkas, M. A. Kourtis, C. Xilouris, V. Koumaras, and G. Gardikis, "Enabling agile video transcoding over sdn/nfv-enabled networks," in *Telecommunications and Multimedia (TEMU), 2016 International Conference on*. IEEE, 2016, pp. 1–5.

[9] L. Nobach and D. Hausheer, "Open, elastic provisioning of hardware acceleration in nfv environments," in *Networked Systems (NetSys), 2015 International Conference and Workshops on*. IEEE, 2015, pp. 1–5.

[10] G. Gao, Y. Wen, and C. Westphal, "Resource provisioning and profit maximization for transcoding in information centric networking," *arXiv preprint arXiv:1605.05758*, 2016.

[11] A. Kathpal, M. Kulkarni, and A. Bakre, "Analyzing compute vs. storage tradeoff for video-aware storage efficiency." in *HotStorage*, 2012.

[12] G. Gao, W. Zhang, Y. Wen, Z. Wang, W. Zhu, and Y. P. Tan, "Cost optimal video transcoding in media cloud: Insights from user viewing pattern," in *2014 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2014, pp. 1–6.

[13] G. ETSI, "Network functions virtualisation (nfv): Architectural framework," *ETSI GS NFV*, vol. 2, no. 2, p. V1, 2013.

[14] X. Cheng, C. Dale, and J. Liu, "Statistics and social network of youtube videos," in *Quality of Service, 2008. IWQoS 2008. 16th International Workshop on*. IEEE, 2008, pp. 229–238.

Guanyu Gao received his Master's degree from University of Science and Technology of China (USTC) in 2012 and Bachelor's degree from University of Electronic Science and Technology of China (UESTC) in 2009. He is currently a Ph.D. student in the Interdisciplinary Graduate School at Nanyang Technological University (NTU). His research interests include multimedia communication, cloud computing, and video processing.

Yonggang Wen (S99-M08-SM14) is an associate professor with School of Computer Science and Engineering at Nanyang Technological University, Singapore. He received his PhD degree in Electrical Engineering and Computer Science (minor in Western Literature) from Massachusetts Institute of Technology (MIT), Cambridge, USA, in 2008. His research interests include cloud computing, green data center, big data analytics, multimedia network and mobile computing.

Jianfei Cai (S98-M02-SM07) received the Ph.D. degree from the University of Missouri-Columbia, Columbia, MO, USA. He is currently an Associate Professor and has served as the Head of Visual and Interactive Computing Division and the Head of Computer Communication Division, School of Computer Engineering, Nanyang Technological University, Singapore. His major research interests include visual computing and multimedia networking.