IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 23, 2019, accepted August 17, 2019, date of publication September 2, 2019, date of current version September 17, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2939007

Video Quality Adaptation for Limiting Transcoding Energy Consumption in Video Servers

DAYOUNG LEE[1], JUNGWOO LEE[2], AND MINSEOK SONG[1], (Member, IEEE)
[1]Department of Computer Engineering, Inha University, Incheon 22212, South Korea
[2]TmaxSoft, Seoul 13613, South Korea

Corresponding author: Minseok Song (mssong@inha.ac.kr)

This work was supported in part by the Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT, under Grant 2017M3C4A7080248, in part by the Basic Science Research Program through the NRF funded by the Ministry of Education under Grant NRF-2018R1D1A1B07050614, and in part by the Inha University Research Grant.

ABSTRACT Dynamic adaptive streaming over HTTP (DASH) inherently requires many transcoding operations to produce various bit-rate versions. However, transcoding is highly computationally intensive, resulting in significant power consumption in transcoding servers. To limit the amount of energy consumption in transcoding servers, some transcoding tasks can be skipped and executed later, but this poses several questions, such as (1) which versions can be transcoded before an actual streaming session, (2) how to limit total amount of transcoding energy consumption, and (3) how to handle requests to the untranscoded versions. To address this, we first introduce the concept of transcoding gain to express the overall video quality assigned to each transcoded version by considering segment popularity and go on to propose an algorithm that determines when and which versions should be transcoded with the aim of maximizing the overall transcoding gain by limiting energy consumption. To achieve this, it allows lower bit-rate versions to be streamed than are requested, at the cost of minimal video quality degradation by considering the overall video quality. The proposed scheme was implemented on a clustered system where various issues were resolved. The experimental results show that the transcoding selection algorithms can effectively control the amount of power consumption, minimizing the effect of overall video quality degradation. For example, it can save up to 65% energy when video quality degradation is allowed to be 4.3% at maximum compared with the scheme that transcodes all the versions.

INDEX TERMS Transcoding, energy consumption, streaming media, multimedia systems.

I. INTRODUCTION

Today, because of the growth of network and multimedia technologies, wireless video streaming has become popular, which makes it essential to cope with various network conditions effectively. To address this, dynamic adaptive streaming over HTTP (DASH) techniques, which are used by the major streaming companies including YouTube and Netflix, divide each video into segments that are further encoded into various bit-rate versions [1]. This allows the client device to choose the most appropriate bit-rate segment to match the current network bandwidth, providing consistent wireless streaming environments under the time-varying network condition [2].

The associate editor coordinating the review of this article and approving it for publication was Usama Mir.

Transcoding is essential in DASH-based streaming systems [2]. For example, it was reported that 120 transcoding operations may be needed for Netflix streaming [3]. To support such high demands for transcoding, many cloud-based transcoding services have recently begun to provide streaming service providers with cost-effective transcoding service [4]. For example, Amazon Elastic Transcoder and Google Cloud Platform have started to support cloud transcoding options. However, transcoding operations are inherently computationally intensive, requiring significant CPU power consumption, which imposes a significant operational cost on service providers.

Energy efficiency in a data center is one of the most important issues for cloud service providers. For example, a data center with a provisioned 10-MW peak power capacity

requires an annual bill between $100 to $200 million [5]. In particular, although energy costs may vary depending on the workloads, the cost of the data center building is a fixed factor that depends on its peak capacity [5]. However, it was observed that 3000 Gbps is required for transcoding at peak hours, based on Akamai's workload traces [6]. It is therefore essential to limit peak transcoding power; otherwise, over-provision is required to prepare for peak workloads, adding significantly to the power cost of the data center [5], [6].

Much work has been performed on the management of computing resources in transcoding servers [7]. For example, some works tackled the tradeoff between storage and computation requirements for transcoding [4], [8], some works handled the issue of power management for transcoding [9], [10], and several works presented cost optimization algorithms for transcoding cloud environments. However, all of these works have attempted to transcode all the possible bit-rate versions, requiring significant computational overhead. To reduce such overhead, some recent works provide guidelines for the selection of transcoding parameters under various network characteristics [1], [3], but they did not consider the power issues required for transcoding. To the best of our knowledge, there is no previous work that aims to cap power consumption by considering the video quality.

We present a new scheme that aims at limiting power consumption in transcoding servers by considering video quality factors. We first introduce the concept of transcoding gain to express the overall video quality achieved as a result of transcoding by considering video popularity. Based on this, we propose a new algorithm that determines 1) which versions should be transcoded and 2) when they are transcoded with the aim of maximizing overall transcoding gain subject to total energy limit to balance transcoding power with video quality. We finally present implementation details for practical use of our scheme.

The rest of this paper is organized as follows. Section II reviews related work, and Section III presents the basic concept. Section IV proposes a new transcoding parameter selection scheme. Section V provides the experimental results, and Section VI finally concludes the paper.

II. RELATED WORK

Transcoding is highly computationally intensive, so its effective management has received much attention from research communities as DASH becomes common. For example, several works have focused on the optimization issues for computing resource management for quality of service (QoS) improvement [8], [11]–[13]. Gao et al. [12] classified transcoding workloads into two categories, namely, real-time and deferrable workloads, and proposed a scheme that makes use of QoS heterogeneity of transcoding operations with the aim of maximizing resource utilization, by allowing preemptive transcoding operations. They also improved it to guarantee QoS statistically; to keep the QoS loss probabilities within the prescribed bounds, an algorithm was proposed to determine the minimum required capacity by learning content-arrival distributions [11]. They tackled resource-provisioning issues for a transcoding cloud with the aim of maximizing financial profit accrued to cloud service providers.

Wang et al. [13] developed an algorithm to determine which edge servers perform transcoding operations for several transcoding scheduling policies with the aim of improving streaming quality. Li et al. [14] presented a self-configurable transcoder on heterogeneous cloud platforms, in which transcoding tasks are dynamically mapped to virtual machines based on the affinity of the transcoding tasks and QoS of requests. Ma et al. [15] presented a priority-based scheduling algorithm that dynamically assigns transcoding jobs to each processor based on workloads observed with the aim of maximizing transcoding throughput. Zhao et al. [16] presented a scheduling method that splits the video into multiple segments to allow parallel transcoding in MapReduce clusters. To maximize parallelism while balancing workloads among clusters, this scheme is based on task locality, minimizing unnecessary overhead such as large-scale data movement and data transfer during the mapping phase. These works, however, did not consider transcoding power issues.

Transcoding operations can be done during an offline transcoding session or an actual streaming session, and there exist two mechanisms that manage storage and computation costs for transcoding operations: online transcoding serves each request in a just-in-time way without storing transcoded versions offline, resulting in significant CPU computation in the streaming session, whereas offline transcoding stores all the transcoded versions in the offline transcoding session, resulting in a lot of storage space [6]. To address this tradeoff, several schemes were developed. For example, Shen et al. [7] provided several strategies to combine online with offline transcoding schemes at the network edges and compared them in terms of caching hit ratio and initial delay. Krishnappa et al. [6] presented a Markov prediction model for future transcoding requests to validate the efficacy of online transcoding. Song et al. [4] presented an algorithm to determine which versions should be stored on disk with the aim of minimizing CPU workloads by considering video popularity. Zhao et al. [8] developed a divide-and-conquer algorithm that minimizes the cloud combined cost of storage and CPU resources by considering high popularity of initial segments of video clips.

Video popularity follows a long-tail distribution, indicating that many versions may be requested only once or not at all [6]. However, all of the works described above attempt to transcode all the possible bit-rate versions without considering these characteristics. To address this, Toni et al. [3] provided a transcoding parameter-selection guideline based on network characteristics, and Aparicio-Pardo et al. [1] improved it with the aim of maximizing average quality-of-experience (QoE) specifically for live adaptive streaming. Kreuzberger et al. [17] assessed how the bit-rate and resolution of videos affect user satisfaction,

and provides a guideline to select the bit-rates of segments. Li et al. [18] proposed an algorithm that chooses the bit-rate versions of the DASH server in order to minimize the resource consumption while maintaining a high QoE for the users by taking delay and rate-distortion constraints at the server.

Several works have dealt with transcoding optimization issues in cloud-based systems. For example, Wei et al. [19] proposed a cloud-based online video transcoding system for live video streaming by optimizing the number of CPU cores and minimizing transcoding time based on the profiling results of transcoding parameters. Jin et al. [20] proposed a media cloud system composed of edge servers with the aim of minimizing total operational costs based on the optimal transcoding configuration and caching space allocation. Some studies have used edge servers or base stations to reduce the delay perceived by the users in streaming environments. For example, Dutta et al. [21] presented a system that reduces user latency while maintaining video quality through real-time transcoding on edge servers. Colonnese et al. [22] proposed a bandwidth allocation scheme that considers encoding-rate changes to minimize user-perceived latency. However, all of these works took no account of energy issues.

Transcoding requires high computation resulting in high energy consumption, but all of the works described above did not consider energy issues. To address this, Zhang et al. [10] proposed a job dispatching algorithm to consider different CPU power characteristics and provided an analytical model of dynamic voltage scaling for transcoding servers. Song et al. [9] proposed an algorithm that allocates a frequency and a workload to each CPU with the aim of minimizing power consumption while meeting all transcoding deadlines. Li et al. [23] proposed a joint caching and transcoding scheduling scheme for DASH with the aim of minimizing energy consumption in a transcoding proxy server. based on three types of energy consumption at the proxy: transcoding, caching and transport energy. Liu et al. [24] developed a video transcoder cluster using low-cost embedded board (the Raspberry Pi Model B). Although its transcoding power can be reduced greatly, their transcoding speed is 5.5× slower than Intel i5 CPU, making its use for large-scale servers questionable.

All of power management studies above attempted to transcode as many versions as possible, requiring significant power, so no power-capping issues were considered. Lee et al. [25] were the first to address the tradeoff between video quality and transcoding power by considering segment popularity, but this work attempted to limit power consumption during the offline transcoding session only, and did not consider transcoding power issues in the actual streaming session. They also did not consider implementation issues. To the best of the authors' knowledge, this paper reports the first approach that optimizes video quality to achieve power capping by combining offline and online transcoding operations.

III. BASIC IDEA AND PROBLEM FORMULATION

A. SYSTEM DESIGN AND WORKFLOW

Each video clip in a video-on-demand (VoD) system goes through two sessions: transcoding and streaming sessions. Before actual streaming starts, it first undergoes the transcoding session during which its video segments are transcoded and stored on storage if necessary. After this transcoding session ends, the streaming session for each video can be started to deliver its video segments to the clients [6].

In offline transcoding, all bit-rate versions are transcoded in the transcoding session, whereas in online transcoding, no transcoding is needed in the transcoding session because all the bit-rate versions are transcoded in the streaming session on-the-fly [6]. Regardless of the length of the segment, the time it takes to transcode the segment is known to be less than the segment length, so online transcoding can be a feasible solution, which allows videos to be delivered on time, even though segments are transcoded in the streaming session [6]. The transcoded versions generated by the online transcoding operations can be stored on storage, which eliminates further transcoding operations for the same versions afterwards.

It is important to limit energy consumption in transcoding servers to prevent over-provision of server resources [5], [6]. For this purpose, unlike the previous research, we make the following contributions:

1) Determination of transcoding parameters: The popularity of video is known to follow a heavily long-tail distribution where many videos are not requested at all or requested only once; this also means that unpopular videos may not need to be transcoded [6]. Based on this popularity behavior, our scheme skips transcoding operations selectively to limit transcoding power consumption, but in this case, a lower bit-rate version may be delivered in the streaming session if the requested version is not stored on the storage, which makes video quality degradation inevitable. To address this, we develop a transcoding parameter selection (TPS) algorithm to determine which versions are transcoded in transcoding session by considering video popularity and quality, with the aim of minimizing video quality degradation in the streaming session while keeping energy consumption below the specified limit. Unlike previous studies, our scheme combines online with offline transcoding to determine which versions can be transcoded in each transcoding and streaming session.
2) Transcoding model: If a version with the same or lower bit-rate than the requested version is not stored on the server, then streaming to match network bandwidth becomes impossible, resulting in stalls or hiccups at the client side. To avoid this, the lowest bit-rate version must be transcoded and stored on disk in a transcoding session; however, this requires transcoding of the lowest bit-rate versions of all video segments, which can be wasteful especially for unpopular video clips.

To address this, in our scheme, transcoding operations for the lowest bit-rate versions are optional in a transcoding session, (not mandatory), so the lowest bit-rate version may need to be transcoded and stored on disk even in an actual streaming session when a version with the same or lower bit-rate than the requested version is not stored on the storage in the transcoding session. Our TPS algorithm also considers this effect by predicting and reserving the transcoding workloads in this case.

3) Implementation: We implemented our scheme in a two-tier transcoding server architecture and solved a number of issues such as workload allocation and transcoding job scheduling. We also performed actual power measurements to determine if our algorithm could actually limit the transcoding energy as specified.

The basic flow of our scheme can be summarized as follows:

1) A TPS algorithm is executed to determine which versions can be transcoded in transcoding session for each video segment.
2) Each video then goes through a transcoding session to transcode the versions determined to be transcoded by the TPS algorithm.
3) After all the transcodable versions of each video clip are transcoded, the streaming session for each video can be started to deliver its video segments to the clients. In the streaming session, online transcoding may be required for the segment depending on the result of the TPS algorithm as follows:
   - Case 1 (no online transcoding is required): If the lowest bit-rate version is available, then no transcoding operation is required in the streaming session because the highest available bit-rate version lower than the requested one can be delivered to the client although the requested bit-rate version is not found.
   - Case 2 (online transcoding may be required): If the lowest bit-rate version is not available, then the same or lower version of the request may not be found in the storage. In this case, the lowest bit-rate version needs to be transcoded. After online transcoding, the lowest bit-rate version is stored on the storage device to eliminate online transcoding for that segment afterwards. Fig. 1 shows the architecture of our transcoding server.

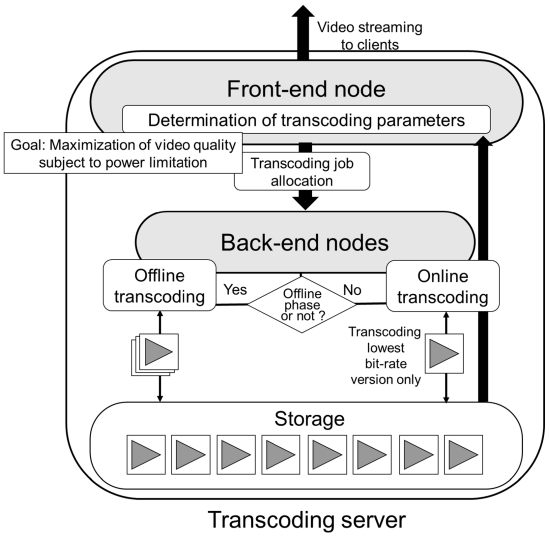

FIGURE 1. An architecture of the transcoding server.

TABLE 1. Symbols used in this paper.

Symbols	Meaning
	Problem formulation (Section III)
N^{seg}	Number of total video segments
N^{ver}	Number of versions in a video
$V_{i,j}$	jth version of video i
N^{tran}	Number of total possible transcoded versions
$Q_{i,j}$	Video quality index for version $V_{i,j}$
$p_{i,j}$	Access probability of version $V_{i,j}$
λ	Total segment request rate to a server
$N_{i,j}$	Number of requests for version $V_{i,j}$ in streaming session
$X_{i,j}$	Parameter, indicating if $V_{i,j}$ is transcoded
$I_{i,j}$	Actual version that is delivered to the client for version $V_{i,j}$
$G_{i,j}$	Overall video quality achieved for the request to version $V_{i,j}$
$C_i^{j\to k}$	Time required to transcode version $V_{i,j}$ to $V_{i,k}$
$H_{i,j}$	Source version of transcoding to produce version $V_{i,j}$
$C_{i,j}^{\text{offline}}$	Transcoding time for version $V_{i,j}$ in transcoding session
$C_{i,j}^{\text{online}}$	Transcoding time for version $V_{i,j}$ in streaming session
$C_{i,j}^{\text{total}}$	$C_{i,j}^{\text{total}} = C_{i,j}^{\text{online}} + C_{i,j}^{\text{offline}}$
L_i	$\min_{k=1,\ldots,N^{\text{tran}}}\{k \mid X_{i,k} = 1\}$
E^{limit}	Energy limit for the entire transcoding workloads
P^{trans}	Power required for transcoding
C^{limit}	Total transcoding time limit
	Algorithm (Section IV)
S_i	Set of all possible combinations of versions for segment i
$S_{i,m}$	mth combination in set S_i
$G_{i,m}^{\text{sum}}$	Sum of the transcoding gains in set $S_{i,m}$
$C_{i,m}^{\text{sum}}$	Sum of the transcoding times in set $S_{i,m}$
Y_i	Selected set index for S_i
N^{set}	Number of elements in set S_i
$R_{i,m}$	Ratio parameter value used for $S_{i,m}$
t_i	Variable used to store temporary set index for S_i
A	Array of ratio parameters, $R_{i,m}$'s

B. PROBLEM FORMULATION

To maximize video quality within energy limits, we formulate the optimization problem in the following steps: 1) overall video quality calculation, 2) transcoding time calculation and 3) problem formulation. Let N^{seg} be the number of segments in a transcoding server. Table 1 summarizes important symbols used in this paper. Each video segment i, ($i = 1, \ldots, N^{\text{seg}}$) has N^{ver} versions from the original highest version, $V_{i,N^{\text{ver}}}$ to transcoded versions ($V_{i,1}$,…,$V_{i,j}$,…,$V_{i,N^{\text{ver}}-1}$), where the values of $V_{i,j}$ are sorted in an ascending order of their bit rates. Let N^{tran} be the number of transcoded versions between $V_{i,1}$ and $V_{i,N^{\text{ver}}-1}$; therefore, $N^{\text{tran}} = N^{\text{ver}} - 1$.

1) OVERALL VIDEO QUALITY CALCULATION

Let $p_{i,j}$ be the access probability of version $V_{i,j}$, ($i = 1, \ldots, N^{\text{seg}}$ and $j = 1, \ldots, N^{\text{ver}}$), where $\sum_{i=1}^{N^{\text{seg}}} \sum_{j=1}^{N^{\text{ver}}} p_{i,j} = 1$. Then the popularity of segment i,

p_i can be calculated as: $\sum_{j=1}^{N^{\text{ver}}} p_{i,j}$. Let $Q_{i,j}$ be the video quality index for version $V_{i,j}$. The values of $Q_{i,j}$ can be easily derived by video quality measurement tools such as an SSIM tool [26].

Let $N_{i,j}$ be the number of requests for version $V_{i,j}$ for the length of total streaming session, L^{total}. Let λ be the total request rate to all the segments in a server. $N_{i,j}$ can be then calculated as follows:

$$N_{i,j} = \lambda p_{i,j} L^{\text{total}}.$$

Let $X_{i,j}$, $(i = 1, \ldots, N^{\text{seg}}$, and $j = 1, \ldots, N^{\text{tran}})$ be the parameter, indicating whether $V_{i,j}$ is transcoded or not in a transcoding session; if $X_{i,j} = 1$, then version $V_{i,j}$ is transcoded and stored so that it can be delivered to clients in a streaming session; otherwise, it is not transcoded, so its alternate version should be delivered to clients.

We assume that $\forall i,\ X_{i,N^{\text{ver}}} = 1$, because the original version is automatically accessible without transcoding. When $X_{i,j} = 0$, so the requested version $V_{i,j}$ is not available, the highest among available versions $V_{i,k}$, where $X_{i,k} = 1$ and $k \leq j$, needs to be delivered to clients instead of version $V_{i,j}$. In addition, if the lower bit-rate version than requested does not exist, then online transcoding may be required to produce the lowest bit-rate version, which is delivered to clients. To represent this, we introduce a new index variable that represents an actual version that is delivered to the client for the requested version $V_{i,j}$, $I_{i,j}$ as follows:

$$I_{i,j} = \begin{cases} \max_k \{k | k \leq j \text{ and } X_{i,k} = 1\} & \exists k, k \leq j, X_{i,k} = 1 \\ 1 & \text{otherwise.} \end{cases}$$

Then, the transcoding gain, $G_{i,j}$ can be defined to express the overall video quality achieved for the request to version $V_{i,j}$ as follows:

$$G_{i,j} = \lambda_{i,j} Q_{i,I_{i,j}}.$$

2) TRANSCODING TIME CALCULATION

Let $C_i^{j \to k}$ be the CPU transcoding time required to transcode version $V_{i,j}$ to $V_{i,k}$, $(j > k)$. In video transcoding, a higher bit-rate version can be transcoded to a lower bit-rate version, but the opposite conversion is not allowed. In addition, if $k < m$, then $C_i^{k \to j} < C_i^{m \to j}$; therefore, to transcode a version $V_{i,j}$, the lowest bit-rate version k, where $k > j$ (say, $H_{i,j}$), needs to be chosen as a source of transcoding to minimize computation overhead as follows:

$$H_{i,j} = \min_{k=1,\ldots,N^{\text{tran}}} \{k | k > j \text{ and } X_{i,k} = 1\}.$$

Let $C_{i,j}^{\text{offline}}$ be the transcoding time required to transcode version $V_{i,j}$ in the transcoding session which can be calculated as follows:

$$C_{i,j}^{\text{offline}} = X_{i,j} C_i^{H_{i,j} \to j}.$$

Let $C_{i,j}^{\text{online}}$ be the transcoding time required to transcode version $V_{i,j}$ in the streaming session. Since only the lowest version can be transcoded in the streaming session, $\forall j > 1$, $C_{i,j}^{\text{online}} = 0$. Suppose that L_i is the index for the lowest version for segment i transcoded in the transcoding session; thus, $\min_{k=1,\ldots,N^{\text{tran}}}\{k |\ X_{i,k} = 1\}$. If $\sum_{\forall j < L_i} N_{i,j} \geq 1$, then the probability that a lower version than L_i is requested is higher than 1, which means that the lowest version, $V_{i,1}$ needs to be transcoded in the streaming session once; otherwise, the expected transcoding time is calculated as $C_i^{H_{i,1} \leftarrow 1} \sum_{\forall j < L_i} N_{i,j}$. Thus, $C_{i,1}^{\text{online}}$ can be calculated as follows:

$$C_{i,1}^{\text{online}} = \begin{cases} C_i^{H_{i,1} \to 1} & \sum_{\forall j < L_i} N_{i,j} \geq 1 \\ C_i^{H_{i,1} \to 1} \sum_{\forall j < L_i} N_{i,j} & \text{otherwise.} \end{cases}$$

Based on this, we introduce the total transcoding time required for version $V_{i,j}$, $C_{i,j}^{\text{total}}$ as follows:

$$C_{i,j}^{\text{total}} = C_{i,j}^{\text{offline}} + C_{i,j}^{\text{online}}.$$

3) OPTIMIZATION PROBLEM FORMULATION

Let E^{limit} be the energy limit assigned to the entire transcoding operation. If P^{trans} is the transcoding power, then to cap the energy consumption below E^{limit}, the transcoding time should not exceed $C^{\text{limit}} = \frac{E^{\text{limit}}}{P^{\text{trans}}}$. We then define an optimization problem, called the transcoding selection problem ($\mathcal{TSP}$) that selects $X_{i,j}$ for segment i, with the aim of maximizing the overall video quality during the streaming session while limiting total transcoding time by C^{limit} in both online and offline phases as follows:

$$\text{Maximize} \sum_{i=1}^{N^{\text{seg}}} \sum_{j=1}^{N^{\text{ver}}} X_{i,j} G_{i,j}$$

$$\text{Subject to} \sum_{i=1}^{N^{\text{seg}}} \sum_{j=1}^{N^{\text{ver}}} X_{i,j} C_{i,j}^{\text{total}} \leq C^{\text{limit}},$$

$$X_{i,j} \in 0, 1 (i = 1, \ldots, N^{\text{seg}}, j = 1, \ldots, N^{\text{tran}})$$

IV. TRANSCODING PARAMETER SELECTION ALGORITHM

$\mathcal{TSP}$ is NP-hard because it includes a special case of the knapsack problem [27]. For example, if $N^{\text{tran}} = 1$ so that there is only one transcoded version, then the problem can be reduced to determine whether each version can be transcoded with the aim of maximizing the overall transcoding gain, which corresponds to a 0/1 knapsack problem.

Since $\mathcal{TSP}$ is NP-hard and the server has a very large number of video segments, it is almost impossible to derive the optimal transcoded set in reasonable time. We therefore propose an algorithm called transcoding parameter-selection based on a heuristic (TPS-H) algorithm with polynomial-time complexity, allowing values of $X_{i,j}$ to be obtained at small overhead.

A. ALGORITHM DESCRIPTION

Let S_i be the set of all the possible combinations for the transcoded versions of each video segment i, $(i = 1, \ldots, N^{\text{seg}})$, and there exist $2^{N^{\text{tran}}}$ possible sets for S_i. For example, suppose that the elements of S_i, $S_{i,m}$,

($m = 1, \ldots, 2^{N^{\text{tran}}}$) are sorted in ascending order of their transcoding times. If $N^{\text{tran}} = 2$ so that $V_{i,2}$ is the highest bit-rate version transcoded, then $S_i = \{S_{i,1} = \phi, S_{i,2} = \{V_{i,1}\}, S_{i,3} = \{V_{i,2}\}, S_{i,4} = \{V_{i,1}, V_{i,2}\}\}$, where ϕ means that there is no transcoded version. Here, let N^{set} be the number of elements in set $S_{i,m}$, so that $N^{\text{set}} = 2^{N^{\text{tran}}}$.

Let $G_{i,m}^{\text{sum}}$ be the sum of transcoding gains for set $S_{i,m}$, $G_{i,m}^{\text{sum}}$ is calculated as follows:

$$G_{i,m}^{\text{sum}} = \sum_{\forall V_{i,j} \in S_{i,m}} G_{i,j}.$$

Let $C_{i,m}^{\text{sum}}$ be the sum of transcoding times for set $S_{i,m}$, $C_{i,m}^{\text{sum}}$ is calculated as follows:

$$C_{i,m}^{\text{sum}} = \sum_{\forall V_{i,j} \in S_{i,m}} C_{i,j}^{\text{total}}.$$

Let Y_i be the selection index to represent that the Y_ith element in set $S_{i,m}$ is chosen, ($Y_i = 1, \ldots, N^{\text{set}}$). To find the value of Y_i for each video segment i, TPS-H defines a series of ratio parameters for each segment, $R_{i,m}$,($i = 1, \ldots, N^{\text{seg}}$ and $m = 1, \ldots, N^{\text{set}} - 1$) as follows:

$$R_{i,m} = \frac{G_{i,m}^{\text{sum}} - G_{i,m-1}^{\text{sum}}}{C_{i,m}^{\text{sum}} - C_{i,m-1}^{\text{sum}}},$$

where the numerator and denominator respectively represent the transcoding gain and time difference between two successive sets. All the values of Y_i are initialized to 1 so that no version is transcoded. Then TPS-H increases the values of Y_i in a greedy way and chooses the most profitable change first to balance the transcoding time with gain; to maximize the transcoding gain (numerator) with the shortest incremental cost of transcoding time (denominator), it repeatedly finds the highest value of $R_{i,m}$ to increase the value of Y_i within the energy limit. Based on this, the TPS-H in Fig. 2 has the two steps as follows:

1) Initialization phase: We introduce variables of t_i for each segment i, which signify that the t_i^{th} element in the set of S_i is selected for segment i. t_i is initialized to 1, so that no version is transcoded, which corresponds to no transcoding computation but the lowest transcoding gain (line 2).
2) Greedy search phase: t_i may be then increased to increase the total transcoding gain, but this should not exceed the transcoding time limit, C^{limit}. For this purpose, the highest value of $R_{i,high}$ in the set A needs to be selected to maximize overall transcoding gain, so that the value of t_i is increased to *high* until the transcoding computing constraint can be satisfied (lines 3-13).

B. ALGORITHM IMPLEMENTATION

We have implemented our scheme on a transcoding server composed of two types of nodes: front-end and back-end nodes [9], [15]. The front-end node allocates each transcoding job to an appropriate back-end node that performs actual transcoding. A TPS algorithm needs to be executed on the front-end node to determine which versions can be transcoded for each video segment.

1: Temporary variable: $C^{\text{temp}} \leftarrow 0$;
2: $\forall\ i$, $Y_i \leftarrow 1$;
3: **while** $C^{\text{temp}} \leq C^{\text{limit}}$ **do**
4: Choose the highest value of $R_{i,high}$ in the set A and removes $R_{i,high}$ from A;
5: **if** $high > Y_i$ **then**
6: $C^{\text{temp}} \leftarrow C^{\text{temp}} + C_{i,high}^{\text{sum}} - C_{i,t_i}^{\text{sum}}$;
7: $Y_i \leftarrow high$;
8: **end if**
9: **if** $A = \phi$ **then**
10: Break the loop;
11: **end if**
12: **end while**
13: $\forall\ i$, find the values of $X_{i,j}$ for set S_{i,Y_i};

FIGURE 2. A transcoding parameter-selection algorithm (TPS-H).

We resolved several issues for the implementation as follows:

1) Workload allocation: To balance transcoding workloads among back-end nodes, the front-end node calculates the total transcoding time for each back-end node, and assigns transcoding jobs to the back-end node using the current shortest total transcoding time.
2) Scheduling transcoding jobs for versions of a single segment: Among the transcoding jobs for a single segment, the higher priority is given to the job of transcoding higher bit-rate versions, because the transcoded versions can be used as a source of transcoding for a lower bit-rate version than the transcoded versions. For example, consider the transcodable set, $\{V_{i,1}, V_{i,2}, V_{i,3}\}$ for the original version $V_{i,4}$. In this case, $V_{i,3}$ is transcoded first from $V_{i,4}$, $V_{i,2}$ from $V_{i,3}$ next and finally $V_{i,1}$ from $V_{i,2}$ to minimize computational overhead for overall transcoding.
3) Communication between front-end and back-end nodes: Two mechanisms (socket and shared files) can be used for communication between front-end and back-end nodes. The back-end nodes stay in waiting states if there is no job. They can be awakened by the job assignment signals delivered from the front-end node. Then, the front-end node writes detailed transcoding information to the shared file for the back-end node to read.
4) Handling of online transcoding: Each back-end node has a separate queue for online transcoding processing. Whenever there exist transcoding requests in a streaming session, the front-end node assigns these jobs to the back-end node with the shortest total transcoding time. On each back-end node, the job queue for online transcoding has a higher priority than that for offline transcoding, which allows prompt processing of online transcoding.

TPS algorithms take an energy limit parameter (E^{limit}) and video segment parameters ($C_{i,j}^{\text{total}}$ and $G_{i,j}$) to determine

TABLE 2. Mapping between SSIM and video quality scores.

$SSIM_{i,j}$	$Q_{i,j}$	Meaning
$SSIM_{i,j} \geq 0.99$	5	excellent
$0.95 \leq SSIM_{i,j} < 0.99$	$25SSIM_{i,j} - 19.75$	good
$0.88 \leq SSIM_{i,j} < 0.95$	$14.29SSIM_{i,j} - 9.57$	fair
$0.5 \leq SSIM_{i,j} < 0.88$	$3.03SSIM_{i,j} + 0.48$	poor
$SSIM_{i,j} < 0.5$	1	bad

TABLE 3. Transcoding times for five sample video clips (seconds).

Source (Resolution, kbps)	Target (Resolution, kbps)	Video type 1	2	3	4	5
(1920 × 1080, 6000)	(1280 × 720, 4000)	4.1	3.5	4.5	3.9	4.3
	(854 × 480, 2000)	3.2	2.9	3.6	2.9	3.3
	(640 × 360, 1000)	2.7	2.1	3.1	2.2	2.6
	(426 × 240, 700)	2.5	2.1	2.9	1.9	2.3
(1280 × 720, 4000)	(854 × 480, 2000)	1.9	1.8	1.8	1.8	2.2
	(640 × 360, 1000)	1.4	1.4	1.4	1.4	1.5
	(426 × 240, 700)	1.2	1.3	1.3	1.2	1.3
(854 × 480, 2000)	(640 × 360, 1000)	1.0	1.1	1.1	1.0	1.0
	(426 × 240, 700)	0.6	0.8	0.8	0.7	0.8
(640 × 360, 1000)	(426 × 240, 700)	0.6	0.6	0.6	0.5	0.6

transcodable versions for each segment. When new video clips are released or uploaded so that workloads in transcoding session change, TPS is executed to find the best transcoding parameters for the segments in a current job queue. The value of E^{limit} can be set to an energy value that should be limited for the current transcoding workloads.

V. EXPERIMENTAL RESULTS

A. SIMULATIONS

We performed simulations to examine the effect of our scheme on overall video quality and energy consumption. For simulation parameters, we profiled video quality indices for each version of the five sample videos using an SSIM tool [26], and then the SSIM value, $SSIM_{i,j}$ are chosen randomly from version j in these sample clips. However, video quality scores are generally expressed as mean opinion scores (MOS) between 1 (bad) and 5 (excellent) [28], so the values of SSIM are converted to those of MOS to derive the values of $Q_{i,j}$, $1 \leq Q_{i,j} \leq 5$ as tabulated in Table 2 [28]. P^{tran} is set to 93 W, as measured [9].

Unless otherwise stated, the length of a segment is set to 6 seconds. We profiled the transcoding times required for each segment of five sample videos (1: Spiderman, 2: The Mummy, 3: Transformers, 4: War for the Planet of the Apes, 5: Wonder Woman), as tabulated in Table 3 [25]. From this table, we observe that the required transcoding time is quite different depending on the resolution of the source version. To reflect this, the transcoding time of each version is chosen randomly between the maximum and minimum transcoding times of these sample versions, which is dependent on the source version. For example, transcoding times from the source with resolution of 1920 × 1080 to the target with resolution of 1280 × 720 can be selected randomly between 3.5s and 4.5s. Unless otherwise stated, λ is set to 100 requests per second.

The access probability of each segment follows a Zipf distribution, where θ was set to 0.271, as profiled in a real VoD application [29]. Typically, the previous segment of the video clip has a higher priority than the latter segment [30], [31], so the popularity of segments in a single video can be modeled by a Zipf distribution with $\theta = 0.2$ (see [31]). To express various version popularities, the following workloads are considered:

- HVP: High bit-rate versions are popular ($\forall\ i$, $p_{i,1} = 0.1p_i$, $p_{i,2} = 0.1p_i$, $p_{i,3} = 0.2p_i$, $p_{i,4} = 0.3p_i$, $p_{i,5} = 0.3p_i$).
- MVP: Medium bit-rate versions are popular ($\forall\ i$, $p_{i,1} = 0.1p_i$, $p_{i,2} = 0.2p_i$, $p_{i,3} = 0.3p_i$, $p_{i,4} = 0.3p_i$, $p_{i,5} = 0.1p_i$).
- LVP: Low bit-rate versions are popular ($\forall\ i, p_{i,1} = 0.3p_i$, $p_{i,2} = 0.3p_i$, $p_{i,3} = 0.2p_i$, $p_{i,4} = 0.1p_i$, $p_{i,5} = 0.1p_i$).
- RVP: Version popularity is chosen randomly.

1) COMPARISON BETWEEN TPS-H AND OPTIMUM

To verify the efficacy of TPS-H, we developed a dynamic programming algorithm called transcoding parameter selection algorithm using dynamic programming (TPS-DP) that can derive an optimal solution for only small values of N^{seg} and C^{limit} due to its high time complexity. TPS-DP has the three steps as follows:

1) Initialization:
   a) Let $D^{\text{gain}}_{i,n}$ be the maximum transcoding gain when i is a video segment index and n is the value of transcoding time limit ($i = 1, \ldots, N^{\text{seg}}$ and $n = 0, \ldots, C^{\text{limit}}$). Let $D^{\text{back}}_{i,n}$ be the set index of transcoded versions for a segment i, to achieve this maximum transcoding gain, $D^{\text{gain}}_{i,n}$.
   b) $\forall\ i$ and n ($i = 1, \ldots, N^{\text{seg}}$ and $n = 0, \ldots, C^{\text{limit}}$), the values of $D^{\text{gain}}_{i,n}$ and $D^{\text{back}}_{i,n}$ are all initialized to 0.
   c) $D^{\text{gain}}_{1,n}$ and $D^{\text{back}}_{1,n}$ are initialized to the maximum transcoding gain at each time limit.
2) Recurrence establishment: At each iteration for each segment index i ($i = 2, \ldots, N^{\text{seg}}$), the value of $D^{\text{gain}}_{i,n}$ are all calculated using recurrence relationship as follows:

$$D^{\text{gain}}_{i,n} = \max(D^{\text{gain}}_{i-1,n}, \max_{m=1,\ldots,N^{\text{set}}}(D^{\text{gain}}_{i-1,n-C^{\text{sum}}_{i,m}} + G^{\text{sum}}_{i,m})).$$

   Then, using this recurrence, all the values of $D^{\text{gain}}_{i,n}$ and $D^{\text{back}}_{i,n}$ can be found.
3) Backtracking: $D^{\text{gain}}_{N^{\text{seg}},C^{\text{limit}}}$ corresponds to the optimal value of overall transcoding gain. Therefore, from the segment index, N^{seg} and the transcoding time index, C^{limit}, the backtracking phase starts to find the values of Y_i by replacing Y_i by $D^{\text{back}}_{i,n}$ values stored in the recurrence establishment phase.

Because TPS-DP may not produce results within a reasonable time, we compared TPS-DP with TPS-H to demonstrate

TABLE 4. Video quality scores of TPS-H and TPS-DP against the values of E^{limit}.

Algorithm	Workload	E^{limit} values			
		406.3 Wh	580.4 Wh	754.5 Wh	906.6 Wh
TPS-H	HVP	4.315	4.454	4.503	4.523
	MVP	4.102	4.273	4.331	4.354
	LVP	4.005	4.106	4.133	4.143
	RVP	4.181	4.302	4.340	4.355
TPS-DP	HVP	4.318	4.457	4.503	4.523
	MVP	4.107	4.276	4.331	4.354
	LVP	4.006	4.106	4.133	4.143
	RVP	4.184	4.303	4.341	4.355

TABLE 5. Algorithm execution time (seconds).

Algorithm	Workload	E^{limit} values			
		406.3 Wh	580.4 Wh	754.5 Wh	906.6 Wh
TPS-H	HVP	0.019s	0.023s	0.039s	0.044s
	MVP	0.018s	0.023s	0.043s	0.047s
	LVP	0.018s	0.028s	0.044s	0.046s
	RVP	0.016s	0.027s	0.037s	0.048s
TPS-DP	HVP	12.60s	44.78s	74.20s	123.65s
	MVP	12.46s	44.95s	74.89s	125.42s
	LVP	12.41s	44.73s	71.73s	127.06s
	RVP	12.61s	43.41s	71.30s	131.00s

the efficacy of the heuristic algorithm only for small transcoding workloads, ($N^{\text{seg}} = 583$ and the length of a segment is 1 minute). Table 4 tabulates how the video quality scores of TPS-H and TPS-DP vary for different values of E^{limit}, which shows that TPS-H provides a nearly optimal solution for $\mathcal{TSP}$, by exhibiting between 0.001% to 0.121% less QoE scores than the optimal value obtained from TPS-DP.

Table 5 tabulates the times taken to run the algorithms for different values of E^{limit} on a machine with an Intel i7-4790. It takes between 0.016s and 0.048s for TPS-H, but between 12.405s and 131.001s for TPS-DP. In particular, in TPS-DP, the execution time increases significantly as E^{limit} increases, which indicates that TPS-DP has a scalability problem, making it impossible to use it for large-scale streaming systems.

2) VIDEO QUALITY AND ENERGY COMPARISON BETWEEN TPS-H AND A SCHEME THAT TRANSCODES ALL BIT-RATE VERSIONS

For comparison, we consider 3000 videos with lengths between 1 and 3 hours. We profiled overall video quality delivered to the clients for 72 hours when $\lambda = 100$ requests/s. We here compared TPS-H with a scheme that transcodes every bit-rate version, which we call TPS-ALL. Fig. 3 shows the average amount of energy consumed by both schemes for different values of E^{limit}, and Table 6 tabulates video quality scores against the values of E^{limit}. From these results, we made the following observations:

1) TPS-H always keeps total energy consumption below E^{limit}. This is because TPS-H attempts to find the best parameter configuration subject to E^{limit}. However, the energy used for TPS-ALL is 701.5 kWh.
2) The overall video quality score increases with the values of E^{limit} because more versions can be transcoded at a higher energy budget in TPS-H.

TABLE 6. Video quality scores of TPS-H and TPS-ALL against E^{limit}.

Workload	E^{limit} values				
	TPS-ALL	TPS-H			
	701.5 kWh	245.5 kWh	350.7 kWh	456.0 kWh	561.2 kWh
HVP	4.378	4.225	4.331	4.362	4.375
MVP	4.168	3.987	4.113	4.150	4.165
LVP	3.978	3.884	3.953	3.971	3.977
RVP	4.179	4.052	4.140	4.166	4.176

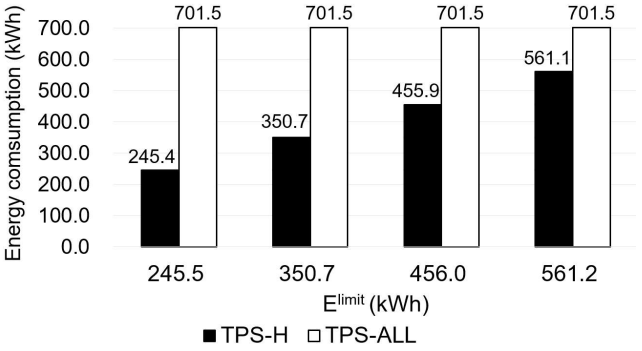

FIGURE 3. Average energy consumption of TPS-ALL and TPS-H against the values of E^{limit}.

3) TPS-ALL exhibits better video quality than TPS-H because all the versions can be provided to users as requested. However, the difference is likely to be very small, ranging from 0.033% and 4.361%.
4) Energy differences between TPS-H and TPS-ALL range between 20% and 65%. In particular, even when the energy difference is 65%, the difference of average video quality is only 3.31%, which clearly shows that TPS-H can be effectively used to limit energy consumption at a small cost of video quality degradation.

3) COMPARISON BETWEEN TPS-H AND STANDARD ALGORITHMS

We also compared TPS-H with three standard algorithms that can be derived to solve $\mathcal{TSP}$ as follows:

- Pop-video: All the transcoding versions in the most popular videos are transcoded first subject to transcoding time limit, C^{limit}.
- Pop-segment: All the transcoding versions in the most popular segments are transcoded first subject to transcoding time limit.
- Pop-version: The popularity of each individual version is considered, so that the most popular versions are transcoded first subject to the transcoding time limit.

For fair comparison, similar to the TPS-H, all other algorithms above transcode the lowest version on-the-fly when a version lower than the requested version is not stored. Fig. 4 shows average amount of energy consumed, while Fig.5 shows video quality scores of each algorithm against the values of E^{limit}. We then made the following observations:

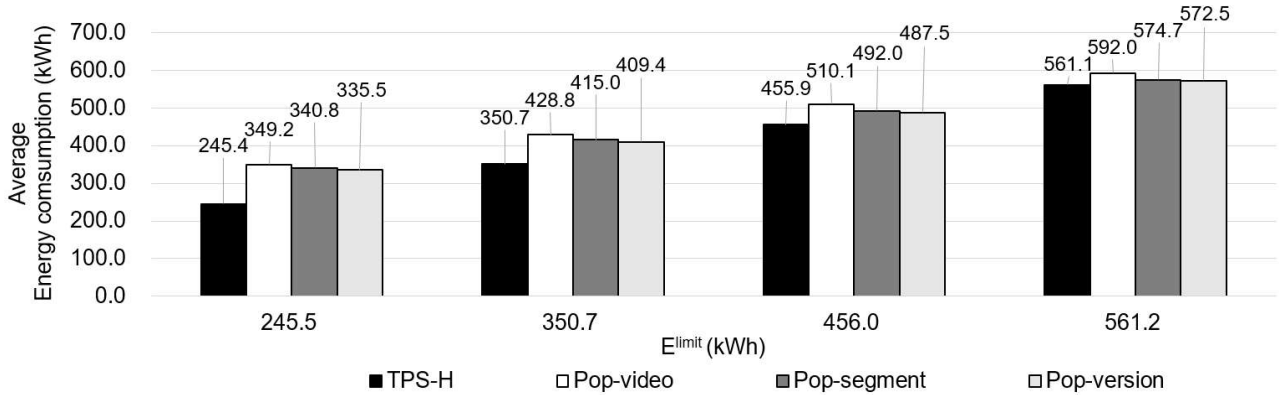

FIGURE 4. Average energy consumption of each algorithm against E^{limit}.

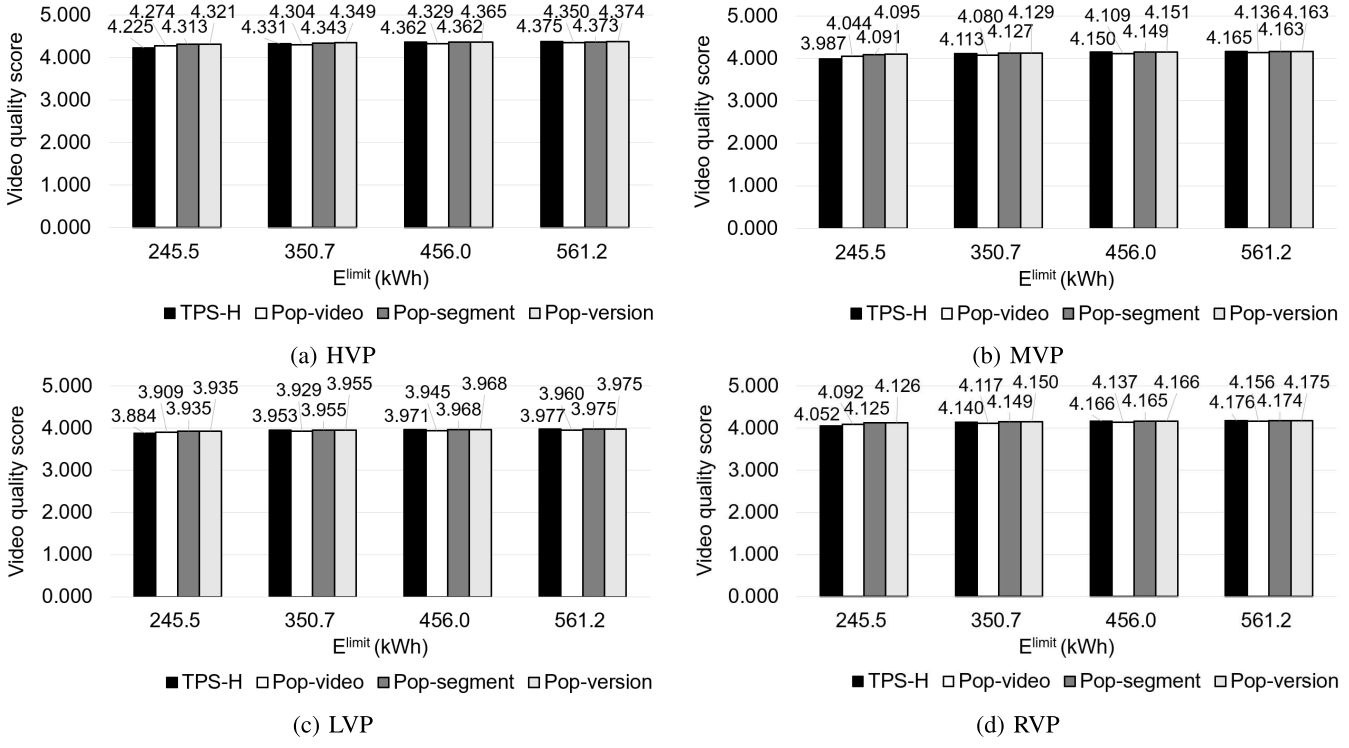

FIGURE 5. Video quality scores of each algorithm against the values of E^{limit}.

1) TPS-H keeps total energy consumption below E^{limit}. However, the actual energy usage of other standard algorithms exceeds the E^{limit}.
2) TPS-H always consumes the lowest energy compared with three other algorithms. For example, TPS-H saves more energy between 5.19% and 30.23% than Pop-video, between 2.1% to 28.66% than the Pop-segment, and between 1.68% and 28.92% than Pop-version.
3) The Pop-version scheme consumes the lowest energy among three standard algorithms but the difference is not high.
4) Video quality improves with the energy budget, because more versions can be stored at high energy budget.
5) Fig. 5 shows that TPS-H has the lowest video quality scores when $E^{\text{limit}} = 245.5$ kWh, exhibiting 0.096 less video quality score than Pop-version at maximum for HVP workloads. This is because TPS-H keeps the energy usage below 245.5 kWh, whereas other schemes cannot. However, as E^{limit} increases, the difference in QoS score decreases, and finally TPS-H shows better video quality than other techniques. For example, TPS-H exhibits better video quality than Pop-video when $E^{\text{limit}} \geq 350.7$ kWh, than Pop-segment when $E^{\text{limit}} \geq 456.0$ kWh and Pop-version when $E^{\text{limit}} \geq 561.2$ kWh. These results clearly show that TPS-H can achieve the best video quality while capping the amount of energy by the specified limit.

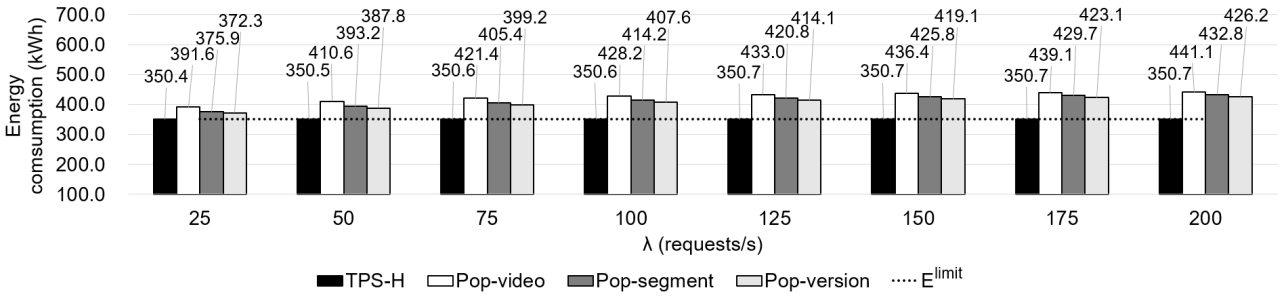

FIGURE 6. Energy comparison against the values of λ.

4) COMPARISON BETWEEN TPS-H AND STANDARD ALGORITHMS AGAINST THE VALUES OF λ

Request rates (λ) determine the load of the streaming servers. To consider various streaming environment, we examined how the values of λ affect energy consumption and video quality scores for RVP workloads when E^{limit} is 350.7 kWh. Fig. 7 shows how the amount of energy consumed varies against the values of λ. We then draw the following observations:

1) The amount of energy consumed increases with the values of λ for three standard algorithms, but TPS-H always limits the amount of energy used irrespective of λ values. From these results, we can easily see that the three standard schemes have scalability problems in terms of power consumption, whereas power is always capped in TPS-H.
2) TPS-H consumes the lowest energy. For example, it can save between 10.52% and 20.5% more energy than Pop-video, between 6.79% to 18.98% than Pop-segment, and between 5.9% to 17.72% than Pop-version.
3) The Pop-version scheme shows the best performance in terms of energy consumption among the three standard algorithms, because it considers version popularity for transcoding parameter selection, which reflects more accurate transcoding popularity than the other two techniques.
4) The request rate has no impact on video quality. For example, the average video quality scores were calculated as 4.14 for TPS-H, 4.117 for Pop-video, 4.149 for Pop-segment, and 4.150 for Pop-version, irrespective of the request rates. This is because the determinant of the video quality score is the popularity of video versions, which cannot be changed by the request rates.

B. POWER CAPPING RESULTS ON A REAL TESTBED

We also examined the power capping effect of our scheme on real transcoding testbed. To this end, we implemented a TPS-H algorithm on a testbed with a front-end node and four back-end nodes, each of which transcodes a total of 101 segments. The resolution of the original source video segments is set to 1920 × 1080, and the original files can be transcoded into four low-resolution versions of 1280 × 720, 854 × 480, 640 × 360 and 426 × 240.

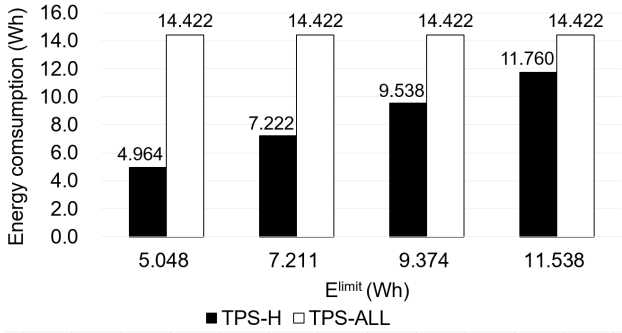

FIGURE 7. Energy of TPS-ALL and TPS-H against E^{limit} on a real testbed.

The TPS-H algorithm requires the length of the transcoding time parameter for each transcoding operation, for which the average transcoding times measured in four out of a total of 101 segments were used. For transcoding tasks, we used an FFmpeg tool [32] which can be executed using a system function call. We measured the amount of energy consumed in the transcoding session five times.

Fig. 7 shows the average energy in the transcoding session for different values of E^{limit} on a real testbed. We observe that TPS-H can accurately limit transcoding energy consumption. For example, the percentage difference between the energy budget (E^{limit}) and the actual energy consumption is between −1.66% and +1.93%, which clearly shows that our method can effectively limit energy usage as specified in the E_{limit} value.

VI. CONCLUSION

We proposed new transcoding parameter selection schemes with the aim of maximizing video quality while limiting the amount of transcoding power consumption. We introduced the concept of transcoding gain to express the video quality and version popularity, and formulated an optimization problem that determines 1) the proportion of offline and online transcoding and 2) the versions transcoded during the offline phase to maximize overall video quality subject to energy limit. We next went on to propose a transcoding parameter

selection algorithm and provide implementation details of the algorithm.

Experimental results showed that the proposed scheme can reduce transcoding power from 20% to 65% compared with conventional transcoding schemes that transcode all the bit-rate versions at the maximum degradation cost of 4.36% video quality. They also demonstrated that: 1) a heuristic algorithm provides near-optimal solution with reasonable complexity; 2) TPS-H exhibits better performance than standard algorithms, reducing transcoding energy consumption between 1.68% and 30.23%; 3) the energy gap between TPS-H and other standard algorithms increases with the loads imposed on streaming servers. Real measurements on a small testbed confirm that the proposed scheme can effectively limit transcoding power consumption.

Power capping is essential in transcoding servers to prevent excessive provision of server computing resources, but this involves efficient transcoding parameter selection and computing resource management. The results confirm that our schemes can make a contribution to the construction of power-capped transcoding servers by using video quality adaptation techniques.

The goal of this paper is to maximize video quality within energy limits, but QoE in video streaming environments can be affected by various factors including video quality, switching probability, transmission delay, jitter and rebuffering. As future work, we plan to extend our scheme to take into account various networking parameters with the aim of maximizing QoE. One way to improve QoE by reducing network latency is to take account of the relationship between video quality and network transmission delay when determining transcoding parameters.

REFERENCES

[1] R. Aparicio-Pardo, K. Pires, A. Blanc, and G. Simon, "Transcoding live adaptive video streams at a massive scale in the cloud," in *Proc. ACM Multimedia Syst. Conf.*, pp. 49–60, Mar. 2015.

[2] T. Stockhammer, "Dynamic adaptive streaming over HTTP–: Standards and design principles," in *Proc. ACM Int. Conf. Multimedia Syst.*, Feb. 2011, pp. 133–144.

[3] L. Toni, R. Aparicio-Pardo, G. Simon, A. Blanc, and P. Frossard, "Optimal set of video representations in adaptive streaming," in *Proc. ACM Multimedia Syst. Conf.*, Mar. 2014, pp. 271–282.

[4] M. Song, J. S. Sim, J. Go, B. Lee, and S. J. Park, "Balancing MPEG transcoding with storage in multiple-quality video-on-demand services," *ETRI J.*, vol. 31, no. 3, pp. 333–335, Jun. 2009.

[5] M. G. Khatib and Z. Bandic, "PCAP: Performance-aware power capping for the disk drive in the cloud," in *Proc. USENIX FAST Conf.*, Feb. 2016, pp. 227–240.

[6] D. K. Krishnappa, M. Zink, and R. K. Sitaraman, "Optimizing the video transcoding workflow in content delivery networks," in *Proc. ACM Multimedia Syst. Conf.*, Mar. 2015, pp. 37–48.

[7] B. Shen, S.-J. Lee, and S. Basu, "Caching strategies in transcoding-enabled proxy systems for streaming media distribution networks," *IEEE Trans. Multimedia*, vol. 6, no. 2, pp. 375–386, Apr. 2004.

[8] H. Zhao, Q. Zheng, W. Zhang, B. Du, and H. Li, "A segment-based storage and transcoding trade-off strategy for multi-version VoD systems in the cloud," *IEEE Trans. Multimedia*, vol. 19, no. 1, pp. 149–159, Jan. 2017.

[9] M. Song, Y. Lee, and J. Park, "Scheduling a video transcoding server to save energy," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 11, no. 2s, p. 45, Feb. 2015.

[10] W. Zhang, Y. Wen, J. Cai, and D. Wu, "Toward transcoding as a service in a multimedia cloud: Energy-efficient job-dispatching algorithm," *IEEE Trans. Veh. Technol.*, vol. 63, no. 5, pp. 2002–2012, Jun. 2014.

[11] G. Gao, H. Hu, Y. Wen, and C. Westphal, "Resource provisioning and profit maximization for transcoding in clouds: A two-timescale approach," *IEEE Trans. Multimedia*, vol. 19, no. 4, pp. 836–848, Apr. 2017.

[12] G. Gao, Y. Wen, and C. Westphal, "Dynamic resource provisioning with QoS guarantee for video transcoding in online video sharing service," in *Proc. ACM Multimedia Conf.*, Oct. 2016, pp. 868–877.

[13] Z. Wang, L. Sun, C. Wu, W. Zhu, Q. Zhuang, and S. Yang, "A joint online transcoding and delivery approach for dynamic adaptive streaming," *IEEE Trans. Multimedia*, vol. 17, no. 6, pp. 867–879, Jun. 2015.

[14] X. Li, M. A. Salehi, M. Bayoumi, N.-T. Tzeng, and R. Buyya, "Cost-efficient and robust on-demand video transcoding using heterogeneous cloud service," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 3, pp. 556–571, Mar. 2018.

[15] H. Ma, B. Seo, and R. Zimmermann, "Dynamic scheduling on video transcoding for MPEG DASH in the cloud environment," in *Proc. ACM Multimedia Syst. Conf.*, Jun. 2014, pp. 283–294.

[16] H. Zhao, Q. Zheng, W. Zhang, and J. Wang, "Prediction-based and locality-aware task scheduling for parallelizing video transcoding over heterogeneous MapReduce cluster," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 4, pp. 1009–1020, Apr. 2018.

[17] C. Kreuzberger, B. Rainer, H. Hellwagner, L. Toni, and P. Frossard, "A comparative study of dash representation sets using real user characteristics," in *Proc. Int. Workshop Netw. Operating Syst. Support Digit. Audio Video*, May 2016, p. 4.

[18] C. Li, L. Toni, J. Zou, H. Xiong, and P. Frossard, "Delay-power-rate-distortion optimization of video representations for dynamic adaptive streaming," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 7, pp. 1648–1664, Jul. 2018.

[19] L. Wei, J. Cai, C. H. Foh, and B. He, "QoS-aware resource allocation for video transcoding in clouds," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 27, no. 1, pp. 49–61, Jan. 2017.

[20] Y. Jin, Y. Wen, and C. Westphal, "Optimal transcoding and caching for adaptive streaming in media cloud: An analytical approach," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 12, pp. 1914–1925, Dec. 2015.

[21] S. Dutta, T. Taleb, P. A. Frangoudis, and A. Ksentini, "On-the-fly QoE-aware transcoding in the mobile edge," in *Proc. IEEE GLOBECOM*, Dec. 2016, pp. 1–6.

[22] S. Colonnese, F. Cuomo, T. Melodia, and I. Rubin, "A cross-layer bandwidth allocation scheme for HTTP-based video streaming in LTE cellular networks," *IEEE Commun. Lett.*, vol. 21, no. 2, pp. 386–389, Feb. 2016.

[23] Z. Li, R. Xie, Q. Jia, and T. Huang, "Energy-efficient joint caching and transcoding for HTTP adaptive streaming in 5G networks with mobile edge computing," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, May 2018, pp. 1–4.

[24] P. Liu, J. Yoon, L. Johnson, and S. Banerjee, "Greening the video transcoding service with low-cost hardware transcoders," in *Proc. USENIX Annu. Tech. Conf.*, Jun. 2016, pp. 407–419.

[25] J. Lee, H. Han, and M. Song, "Balancing transcoding against quality-of-experience to limit energy consumption in video-on-demand systems," in *Proc. IEEE Symp. Multimedia*, Dec. 2017, pp. 334–337.

[26] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004.

[27] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. Hoboken, NJ, USA: Wiley, 1990.

[28] T. Zinner, O. Hohlfeld, O. Abboud, and T. Hossfeld, "Impact of frame rate and resolution on objective QoE metrics," in *Proc. IEEE Int. Workshop Quality Multimedia Exper.*, Jun. 2010, pp. 29–34.

[29] A. Dan, D. Sitaram, and P. Shahabuddin, "Dynamic batching policies for an on-demand video server," *Multimedia Syst.*, vol. 4, no. 3, pp. 112–121, 1996.

[30] H. Yu, D. Zheng, B. Zhao, and W. Zheng, "Understanding user behavior in large-scale video-on-demand systems," in *Proc. 1st ACM SIGOPS/EuroSys*, Apr. 2006, pp. 333–344.

[31] S. Lim, Y. Ko, G. Jung, J. Kim, and M. Jang, "Inter-chunk popularity-based edge-first caching in content-centric networking," *IEEE Commun. Lett.*, vol. 18, no. 8, pp. 1331–1334, Aug. 2014.

[32] *FFmpeg*. Accessed: Sep. 5, 2019. [Online]. Available: http://ffmpeg.mplayerhq.hu/

DAYOUNG LEE received the B.S. and M.S. degrees in computer engineering from Inha University, South Korea, in 2016 and 2018, respectively, where she is currently pursuing the Ph.D. degree with the Department of Computer Engineering. Her current research interests include embedded software and multimedia systems.

JUNGWOO LEE received the B.S. and M.S. degrees in computer engineering from Inha University, South Korea, in 2016 and 2017, respectively. He is currently with TmaxSoft, South Korea. His current research interests include embedded software and ubiquitous computing.

MINSEOK SONG (M'07) received the B.S., M.S., and Ph.D. degrees, all in computer engineering from Seoul National University, South Korea, in 1996, 1998, and 2004, respectively. Since September 2005, he has been with the Department of Computer Engineering, Inha University, Incheon, South Korea, where he is currently a Professor. His research interests include embedded systems and multimedia systems.