Quality-Aware Transcoding Task Allocation Under Limited Power in Live-Streaming Systems

Dayoung Lee, *Member, IEEE*, and Minseok Song, *Member, IEEE*

***Abstract*—Transcoding in video live-streaming systems requires a lot of computation and, hence, a lot of power. Putting a limit on the power drawn by each of the transcoding processors in a server reduces the overall power consumption, but it also hinders the efficient allocation of transcoding tasks. We address this with a dynamic programming algorithm, together with a heuristic, which maximizes total processing capacity while limiting power consumption in a server with heterogeneous processors. A further greedy algorithm determines the bitrates, at which content is transcoded for each channel, and allocates transcoding tasks to processors, while taking video quality, popularity, and workload balance into account. The initial assumption is that all contents are transcoded to all bitrates for every channel. Then, the algorithm gradually reduces the number of versions to be produced by transcoding, while minimizing the consequent reduction in popularity-weighted video quality, as well as balancing the workload across processors. Experimental results show that our scheme improves aggregate popularity-weighted video quality under a power constraint by 3.82%–39.12%, compared to benchmark methods.**

***Index Terms*—Energy efficiency, multimedia computing, multimedia systems, streaming media, transcoding.**

I. Introduction

Dynamic adaptive streaming over hypertext transfer protocol (DASH), the *de facto* standard adopted by major video streaming companies such as YouTube and Netflix, requires a lot of transcoding tasks to run in parallel, generating various versions of many videos with different bitrates [1], [2]. For example, the transmission of a single Netflix video may involve 120 transcoding operations, and there are at least 20 different versions of every YouTube video [1], [3]. Meeting these requirements needs many transcoding servers, each with many processor nodes to perform a vast number of tasks in parallel, and these nodes may have different performance and power characteristics.

The price paid by a data center for electricity is typically calculated by peak power consumption, so it is important to limit the peak power requirement [4]. A cost-effective method for capping this requirement is to limit the power allocated to each server individually, but this reduces the scope for optimizing server performance under local power constraints, reducing total throughput.

Live-streaming services provide online games and coverage of events such as sports matches. While the transcoding tasks required for video on demand (VoD) can be performed offline, live streaming (which also uses DASH) requires immediate online transcoding [3], [5]. An example is the gaming channel Twitch.tv, which accounts for 1.8% of peak U.S. Internet traffic [6]. Twitch offers different levels of service to the broadcasters it hosts. The content of broadcasters without a premium package can be transmitted at bitrates that do not match all their users' devices, so some users are likely to receive a poor quality of experience (QoE) [3], [5], [7], [8].

Since the power consumed by a server depends on its central processing unit (CPU) utilization [9], [10], power consumption can be limited by reducing that utilization, but this obviously reduces the number of transcoding tasks that can be performed. In practice, it commonly leads to reduced QoE as fewer versions are generated, and versions with lower bitrates have to be substituted. This tradeoff between power consumption and QoE is very significant to both service providers and broadcasters.

Some authors [11], [12] have presented scheduling schemes for transcoding servers, which specifically aim to reduce peak power consumption. Others [13], [14] have extended this to include dynamic voltage and frequency scaling (DVFS), allocating transcoding tasks to processors running at different frequencies, with the same aim of capping peak power. However, these studies have been focused on offline transcoding in VoD systems, and not the ongoing allocation of computing resources required for live streaming.

We propose a new scheme that aims at limiting peak power consumption while taking into account the popularity of each item of content and its QoE at each bitrate. We begin by introducing algorithms to determine the maximum CPU utilization at each processor with the aim of maximizing total processing capacity of a transcoding server with heterogeneous processors. Then, we present an algorithm that determines a set of transcoded versions of each item of content for each live channel, while allocating each transcoding task to a processor with the aim of maximizing the popularity-weighted quality of each version under the power limitation. The algorithm starts from the assumption that version at every bitrate is transcoded for each channel and then greedily removes transcoded versions with the aim of minimizing the concomitant reduction in

Manuscript received 27 February 2021; revised 1 June 2021; accepted 31 July 2021. Date of publication 30 August 2021; date of current version 26 August 2022. This work was supported in part by the National Research Foundation of Korea (NRF), in part the Next-Generation Information Computing Development Program funded by the Ministry of Science, ICT, under Grant 2017M3C4A7080248, and in part by the Basic Science Research Program funded by the Ministry of Education under Grant NRF-2020R1F1A1068984. *(Corresponding author: Minseok Song.)*

The authors are with the Department of Computer Engineering, Inha University, Incheon 22212, South Korea (e-mail: dayoung08@inha.edu; mssong@inha.ac.kr).

Digital Object Identifier 10.1109/JSYST.2021.3103526

1937-9234 © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Inha University. Downloaded on September 21,2022 at 01:41:37 UTC from IEEE Xplore. Restrictions apply.

popularity-weighted quality. The whole process is subject to a total processing capacity constraint.

The rest of this article is organized as follows. We review related work in Section II and then provide a system model in Section III. We present an algorithm that limits peak power consumption in Section IV and an algorithm that determines transcodeable versions and allocates them to heterogeneous processors in Section V. We provide experimental results in Section VI. Section VII concludes this article.

II. Related Work

Many schemes have been proposed for distributing and scheduling workloads for transcoding servers. For example, Gao *et al.* [15] proposed a provisioning method to allocate computing resource in transcoding servers to satisfy requests with different delay requirements. They [16] also presented a cloud transcoding system, aiming at reducing the cost of building servers, using a probabilistic model of changing workloads. Li *et al.* [17] proposed a virtual machine allocation scheme for heterogeneous self-configurable cloud platforms, which accounts for quality-of-service requirements. Ma *et al.* [18] introduced a method of predicting transcoding times on which they constructed a priority-based scheduling task allocation (TA) scheme to maximize the throughput. However, none of these works meet the requirements of live-streaming systems.

Online transcoding is essential in live streaming, and several authors have addressed the tradeoff between storage and computation costs in online transcoding schemes. Krishnappa *et al.* [1] used a Markov model to predict the workload imposed by online transcoding in a content delivery network. They reduced the initial latency of streaming by offline transcoding of video prefixes. Jokhio *et al.* [19] balanced the costs of storage and computation in an online transcoding scheme while taking into account the length and frequency of different transcoding operations. Zhang *et al.* [20] proposed a method that reduces the storage overhead required to store transcodeable versions on the solid-state drive by allowing low-overhead online transcoding, which leverages the characteristics of video coding and flash memory device. However, issues related to video quality improvement were not considered in these studies.

Several cloud-based transcoding systems have been presented to provide online transcoding required for live-streaming services. Panarello *et al.* [21] presented an online transcoding scheme for a federated cloud server, which reduces processing time by cooperation among clouds. Zheng *et al.* [22] showed that the operating cost incurred by a service provider could be reduced by allocating an appropriate data center resource to each user. Mada *et al.* [23] designed a transcoding architecture that could scale up and down virtual machines used within the computing limits of various cloud providers. All of these schemes focused primarily on virtual machine allocation issues to reduce cloud costs.

Several studies have been presented to leverage computing facilities in edge systems for online transcoding. He *et al.* [24], [25] proposed replacing individual servers with online transcoding systems based on fog computing, in which edge devices and servers collaborated locally. Zhang *et al.* [26] also tried to maximize the QoE in an online transcoding scheme for a graphical-processing-unit-based server that makes an advantageous tradeoff between streaming quality and delay. Wang *et al.* [27] introduced an algorithm for minimizing network traffic by grouping users requiring similar bandwidth, which allows a single version of video content to be transcoded and sent to each group. Zhu *et al.* [28] took a similar approach, in which users' devices transcode incoming video data, so as to reduce the computation cost and concomitant delays at transcoding servers. However, none of these schemes took into account video quality issues.

Several authors have suggested ways to improve the QoE in online transcoding systems. Aparicio-Pardo *et al.* [3] maximized the average QoE of live streaming based on measurements of the computational requirements of transcoding processes. Kim *et al.* [29] enhanced the resolution of a video using deep learning techniques to remedy the low quality that can result from a streamer's limited upstream bandwidth and computational capacity limitations. Liu *et al.* [30] proposed a heuristic approach for media clouds by assigning channels to servers to improve the QoE while meeting CPU usage and bandwidth constraints. Lee and Song [31] introduced preliminary results for determining transcodeable bitrate versions by considering video quality in a live-streaming system. However, none of these schemes considered power issues for transcoding servers.

Several authors have addressed the high power consumption of transcoding servers. Li *et al.* [11] presented an algorithm to determine transcoding parameters that reduce delay and power consumption in DASH environments. Lee *et al.* [12] presented a scheme that delays the transcoding of the unpopular contents to improve the overall quality of streaming video while limiting the energy consumed by offline transcoding. Costero *et al.* [32] used a reinforcement learning technique to determine encoding and frequency parameters for high-efficiency video coding streaming. Song *et al.* [13] proposed a DVFS scheme, in which a CPU frequency and a processor are allocated to each transcoding task, with the aim of reducing energy consumption while meeting the deadlines of all transcoding tasks. Zhang *et al.* [14] provided a task dispatching algorithm based on an analytical model for relating CPU power consumption to transcoding delay when using DVFS. Li *et al.* [33] tried to reduce the cumulative energy requirements of transcoding, caching, and transmission in transcoding edge servers. Liu *et al.* [34] introduced a low-cost embedded board (a Raspberry Pi Model B) into a transcoding cluster and showed that it reduced power consumption.

Most of these energy-saving techniques have been developed for offline transcoding systems based on a single CPU or homogeneous processors, in which the reduction of energy in VoD systems is the main objective. Online transcoding systems require continuous use of CPU power in streaming sessions, where effective control of CPU usage is essential. Although power consumption can be effectively limited by controlling the CPU resource from the available processors, none of the work described above addressed this issue. To the best of our knowledge, this article is the first attempt to address power capping issues by controlling CPU utilization in a transcoding server

Authorized licensed use limited to: Inha University. Downloaded on September 21,2022 at 01:41:37 UTC from IEEE Xplore. Restrictions apply.

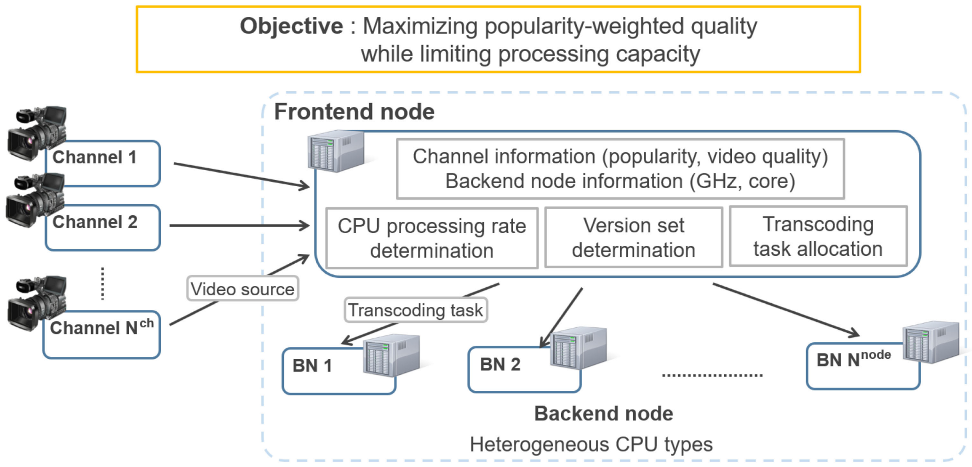
Objective : Maximizing popularity-weighted quality while limiting processing capacity
Frontend node
Channel 1
Channel 2
Channel N^ch
Video source
Channel information (popularity, video quality)
Backend node information (GHz, core)
CPU processing rate determination
Version set determination
Transcoding task allocation
Transcoding task
BN 1
BN 2
BN N^node
Backend node
Heterogeneous CPU types

Fig. 1. Transcoding server architecture.

consisting of heterogeneous processors running live-streaming applications.

III. System Model and Power Management Concept

A. System Model

In a typical live-streaming scenario, several channels of video are simultaneously captured, encoded, and uploaded to a live-streaming system with two servers: a transcoding server that transcodes the video stream into versions with multiple bitrates, and a streaming server that delivers the appropriate version to each viewer. The processing capacity of the CPU in the transcoding server must be divided between the channels being broadcast.

A transcoding server consists of a frontend node (FN) and backend nodes (BNs). The FN determines the bitrate versions to which content is transcoded for each channel and then allocates these transcoding tasks to appropriate BNs, which perform the transcoding. Fig. 1 shows the architecture of a transcoding server, where N^{BN} is the number of BNs. Table I summarizes the symbols used in this article.

Let N^{ch} be the total number of channels. We introduce $V_{i,k}$ ($i = 1, \ldots, N^{\mathrm{ch}}$ and $k = 1, \ldots, N^{\mathrm{br}}$) to express the kth bitrate version required for channel i, where N^{br} is the number of total bitrate versions. Obviously, the original version does not need to be transcoded. We define the processing rate required to complete a transcoding task in the time available as the average frequency at which the CPU has to run over that period. This rate is a tool for analysis and is unlikely to correspond to any available clock frequency. A model of required processing rate can be established as an expression of the form $C^{\mathrm{ch}}_{i,k}$ as follows:

$$C^{\mathrm{ch}}_{i,k} = a r^{b}_{i,k} \tag{1}$$

where $C^{\mathrm{ch}}_{i,k}$ is the processing rate, $r_{i,k}$ is the bitrate of the transcoded video in Mb/s for $V_{i,k}$, and a and b are parameters that can obtained from real measurements and can vary depending on the selected bitrate [3].

TABLE I
Symbols Used in This Article

Symbol	Meaning
N^{ch}	Number of channels
N^{BN}	Number of backend nodes
N^{br}	Number of bitrate versions
$V_{i,k}$	kth bitrate version for channel i
$C^{\mathrm{ch}}_{i,k}$	CPU processing-rate for version $V_{i,k}$
$r_{i,k}$	Bitrate of version $V_{i,k}$ in Mbps
C^{BN}_{j}	Maximum CPU processing-rate provided by BN j
$P_j(u)$	Total power drawn by BN j when CPU utilization is u
$\alpha_j(u)$	Active power coefficient for BN j when CPU utilization is u
β_j	Idle power coefficient for BN j
P^{limit}	Maximum total power that can be supplied to BNs
X_j	Maximum usable processing-rate for BN j
$D^{\mathrm{rate}}_{j,n}$	Maximum total processing-rate at power limit n
$D^{\mathrm{back}}_{j,n}$	CPU processing-rate for BN j at power limit n
$R_{j,m}$	Parameters used in a CPU processing-rate determination heuristic
S^{ratio}	List of $R_{j,m}$ parameters
$p_{i,k}$	Probability that version $V_{i,k}$ is accessed by viewers
$Q_{i,k}$	Video quality index for version $V_{i,k}$
$G_{i,k}$	Popularity-weighted video quality of version $V_{i,k}$
S^{set}_{i}	Set of all combinations of transcoded versions for channel i
N^{set}	Number of elements in S^{set}_{i}
$S^{\mathrm{set}}_{i,m}$	mth element in S^{set}_{i}
$C^{\mathrm{set}}_{i,m}$	Total CPU processing-rate for all versions in set $S^{\mathrm{set}}_{i,m}$
$G^{\mathrm{set}}_{i,m}$	Total popularity-weighted video quality of all versions in $S^{\mathrm{set}}_{i,m}$
Y_i	Selected element index in powerset S^{set}_{i}
S^{ver}	Set of transcoded versions chosen by version-determination algorithm
$Z_{i,k}$	Backend node index for task to transcode $V_{i,k}$, $(V_{i,k} \in S^{\mathrm{ver}})$
$I_{i,m}$	Parameters used for version-determination algorithm
A	List of $I_{i,m}$ parameters
P^{ratio}	Ratio of power consumption compared to when all BNs are fully utilized

The number of CPU cycles required to transcode a video segment to different bitrate versions can be measured using a program such as the Linux perf tool, and the processing rate can be calculated by dividing the number of CPU cycles by the playback time of each segment [3]. For example, Table II tabulates these parameters calculated by fitting a curve to the data obtained when using an Intel Xeon E5640 CPU with 2.66 GHz and four cores [3].

Each BN may have different power characteristics. We use the term active power to describe the power consumed by the BN while it is performing computation, which increases nonlinearly with CPU utilization [9], [35]. When no computation is being done, the server still draws some power, which we call the idle

Authorized licensed use limited to: Inha University. Downloaded on September 21,2022 at 01:41:37 UTC from IEEE Xplore. Restrictions apply.

TABLE II
PARAMETERS a AND b FOR DETERMINING CPU PROCESSING RATE (GHZ) REQUIRED FOR TRANSCODING A 2750-KB/S VIDEO AT 1080P IN LIVE STREAMING [3]

Target resolution	a	b
224p	0.673091	0.024642
360p	0.827912	0.033306
720p	1.341512	0.060222
1080p	1.547002	0.080571

power. For BN j, we introduce the active power coefficient $\alpha_{j,u}$ to relate the power drawn to CPU utilization u, $(0 \leq u \leq 1)$ and idle power coefficient β_j to express power drawn when the CPU in idle. The total power $P_j(u)$ drawn by BN j can then be expressed as follows [9], [35]:

$$P_j(u) = \alpha_{j,u} u + \beta_j (1 - u) \tag{2}$$

where the values of $\alpha_{j,u}$ and β_j can be determined from measurements such as those collected by the SPEC Power Committee [35].

Equation (2) represents the power model specified by the SPEC Power Committee, an industry standard organization for evaluating the power and performance characteristics of server-class computers. This organization provides strict guidelines for measuring the power and performance of server-class computer equipment and collects measured power data to disclose accurate power model for commercial servers. This committee's actual power measurements show that the active power increases non-linearly with CPU utilization [9], [10], [35].

We consider a transcoding server with heterogeneous BNs, in which each BN j can provide a number of different processing rates. We introduce C_j^{BN} to represent the maximum CPU processing rate provided by BN j. Then, the CPU utilization at BN j can be limited by controlling the effective processing rate, which is the sum of the rates used by all the transcoding tasks running at BN j. For this purpose, we introduce the maximum usable processing rate X_j for each BN j, which has a value between 0 and C_j^{BN}. For example, consider a BN for which $C_j^{\mathrm{BN}} = 200$ GHz. If $X_j = 100$ GHz, then 50% of the total processing capacity can be used, resulting in a CPU utilization of 0.5. The power consumption of BN j can then be expressed using X_j parameters as $P_j(\frac{X_j}{C_j^{\mathrm{BN}}})$.

B. Power Management Concept

The objective of our scheme is to determine which transcoded versions should be produced for each channel, in order to maximize the aggregate video quality delivered to viewers, while limiting the total peak power consumption of a transcoding server with heterogeneous BNs in a live-streaming system. We achieve this by means of a process with two steps: first, we determine an actual processing rate for each BN, which keeps the total power consumption under the limit, and then, we determine which transcoded versions should be produced for each channel within these the processing rates by taking channel popularity into account.

Because each BN has different characteristics in terms of power and processing rate, we need to determine the processing rate used by each BN, with the aim of maximizing the total processing rate subject to the power limit. We address this with a dynamic programming (DP) technique, which theoretically provides an optimal solution. It is also implemented using a heuristic, which reduces the time complexity at the cost of a suboptimal solution.

In the second step, the algorithm first determines the bitrate, at which versions should be transcoded for each channel, based on the processing rates at each BN that we found in the first step, together with channel popularity and video quality. Then, each transcoding task is allocated to an appropriate BN to make effective use of available processing rates.

IV. DETERMINING THE MAXIMUM USABLE CPU PROCESSING RATE

A. Problem Formulation

Let P^{limit} be the maximum total power that can be supplied to the BNs. Then, the total power consumption over every BN must not exceed P^{limit} as follows:

$$\sum_{j=1}^{N^{\mathrm{BN}}} P_j \left(\frac{X_j}{C_j^{\mathrm{BN}}} \right) \leq P^{\mathrm{limit}}. \tag{3}$$

Based on this, we set up the maximum usable CPU processing-rate determination problem (MUPD), which determines X_j for BN j, $(X_j = 0, \ldots, C_j^{\mathrm{BN}})$, as follows:

$$\text{Maximize} \sum_{j=1}^{N^{\mathrm{BN}}} X_j$$

$$\text{subject to} \sum_{j=1}^{N^{\mathrm{BN}}} P_j \left(\frac{X_j}{C_j^{\mathrm{BN}}} \right) \leq P^{\mathrm{limit}}.$$

We address MUPD by selecting successive usable processing rates, X_j for each BN j, with the aim of maximizing total processing rate, $\sum_{j=1}^{N^{\mathrm{BN}}} X_j$, while limiting the power consumption. This is a variant of the multiple-choice knapsack problem (MCKP), which is NP-hard [36]. We propose two algorithms for MUPD using DP and greedy techniques.

B. Dynamic Programming Algorithm

Let $D_{j,n}^{\mathrm{rate}}$ be the maximum value of the total processing rate, where j is index of a BN and n is the value of the power limit between 0 and P^{limit} ($j = 1, \ldots, N^{\mathrm{BN}}$ and $n = 0, \ldots, P^{\mathrm{limit}}$). Our algorithm builds a table of values of $D_{j,n}^{\mathrm{rate}}$, so that $D_{N^{\mathrm{BN}},P^{\mathrm{limit}}}^{\mathrm{rate}}$ represents the maximum value of $\sum_{j=1}^{N^{\mathrm{BN}}} X_j$ when inequality (3) is satisfied. To construct this table, a recurrence relation is established to obtain successive values of $D_{j,n}^{\mathrm{rate}}$. Finally, the algorithm backtracks to find the value of X_j for each BN j.

As each value of $D_{j,n}^{\mathrm{rate}}$ is updated, the algorithm records the value of the CPU processing rate for BN j at power limit n, $D_{j,n}^{\mathrm{back}}$, $(D_{j,n}^{\mathrm{back}} = 0, \ldots, C_j^{\mathrm{BN}})$, which allows BN j to achieve this value of $D_{j,n}^{\mathrm{rate}}$. The CPU processing-rate determination

Authorized licensed use limited to: Inha University. Downloaded on September 21,2022 at 01:41:37 UTC from IEEE Xplore. Restrictions apply.

algorithm using a DP technique (CPD-DP) is shown in Algorithm 1 and has the following three steps.

1) *Initialization:* $\forall\ j$ and n $(j = 1, \ldots, N^{\text{BN}}$ and $n = 0, \ldots, P^{\text{limit}})$, the values of $D^{\text{rate}}_{j,n}$ are all initialized to $-\infty$, and the values of $D^{\text{back}}_{j,n}$ are all initialized to 0 (lines 4–9).
2) *Recurrence Establishment:* The values, $D^{\text{rate}}_{1,n}$ and $D^{\text{back}}_{1,n}$ for the first BN are initialized (lines 10–14). Then, at each iteration for each index j $(j = 2, \ldots, N^{\text{BN}})$, the values of $D^{\text{rate}}_{j,n}$ are calculated using recurrence relationship as follows:

$$D^{\text{back}}_{j,n} = \max(D^{\text{rate}}_{j,n-1}, \max_{m=0,\ldots,C^{\text{BN}}_j}(D^{\text{rate}}_{j-1,n-P_j(\frac{m}{C^{\text{BN}}_j})} + m)). \quad (4)$$

Using this recurrence, all the values of $D^{\text{rate}}_{j,n}$ and $D^{\text{back}}_{j,n}$ can be calculated successively (lines 15–20).
3) *Backtracking:* $D^{\text{rate}}_{N^{\text{BN}},P^{\text{limit}}}$ corresponds to the maximum value of $\sum_{j=1}^{N^{\text{BN}}} X_j$ when inequality (3) is satisfied. Therefore, starting from the highest BN index, N^{BN}, and the corresponding power limit value, P^{limit}, the algorithm backtracks and replaces the values of X_j with the value of $D^{\text{back}}_{j,n}$ stored in phase 2 (lines 22–28).

The time complexity of the CPD-DP algorithm is $O(\max_{j=1,\ldots N^{\text{BN}}} C^{\text{BN}}_j N^{\text{BN}} P^{\text{limit}})$, which is determined by the two loops (lines 15–20).

Algorithm 1: The Dynamic Programming Algorithm (CPD-DP), Which Determines CPU Processing Rates for BNs.

```
   Input: $P^{\text{limit}}$, $C^{\text{BN}}_j$ and $P_j(u)$, $(j = 1, ..., N^{\text{BN}})$ ;
   Output: $X_j$, $(j = 1, ..., N^{\text{BN}})$;
1  Temporary variables: $j$, $n$ and $m$;
2  /* Initialization */;
3  for $j = 1$ to $N^{\text{BN}}$ do
4      for $n = 0$ to $P^{\text{limit}}$ do
5          $D^{\text{rate}}_{j,n} \leftarrow -\infty$;
6          $D^{\text{back}}_{j,n} \leftarrow 0$;
7      end
8  end
9  /* Recurrence establishment */;
10 for $n = 0$ to $P^{\text{limit}}$ do
11     $D^{\text{rate}}_{1,n} \leftarrow \max_{\{m|P_1(\frac{1}{C^{\text{BN}}_1})\le n\}} m$;
12     $D^{\text{back}}_{1,n} \leftarrow arg\max_{m=0,\ldots,C^{\text{BN}}_j} D^{\text{rate}}_{1,n}$;
13 end
14 for $j = 2$ to $N^{\text{BN}}$ do
15     for $n = 0$ to $P^{\text{limit}}$ do
16         $D^{\text{rate}}_{j,n} \leftarrow \max(D^{\text{rate}}_{j,n-1}, \max_{m=0,\ldots,C^{\text{BN}}_j}(D^{\text{rate}}_{j-1,n-P_j(\frac{m}{C^{\text{BN}}_j})} + m))$;
17         $D^{\text{back}}_{j,n} \leftarrow arg\max_{m=0,\ldots,C^{\text{BN}}_j} D^{\text{rate}}_{j,n}$;
18     end
19 end
20 /* Backtracking */;
21 $j \leftarrow N^{\text{BN}}$;
22 $n \leftarrow P^{\text{limit}}$;
23 while $j > 0$ do
24     $X_j \leftarrow D^{\text{back}}_{j,n}$;
25     $j \leftarrow j - 1$;
26     $n \leftarrow n - P_j(\frac{X_j}{C^{\text{BN}}_j})$;
27 end
```

C. Greedy Heuristic

The CPD-DP algorithm produces an optimal solution, but take too long due to its high complexity. We, therefore, introduce a greedy heuristic algorithm (CPD-H), shown in Algorithm 2. A greedy algorithm based on profit to weight ratios is known to perform well on various types of MCKP [36]. We, therefore, introduce a series of ratios, $R_{j,m}$ $(m = 1, \ldots, C^{\text{BN}}_j)$, which relate processing rate to power consumption when m GHz is selected as the CPU processing rate at BN j as follows:

$$R_{j,m} = \frac{m}{P_j(\frac{m}{C^{\text{BN}}_j}) - P_j(0)}. \quad (5)$$

We introduce a temporary variable of X_j, X^{tmp}_j, for each BN j, which is initialized to 0. The values of X^{tmp}_j are then increased with the aim of maximizing the increase in the total processing rate. CPD-H finds the highest value from a list of $R_{j,m}$ values (say S^{ratio}), for which $j = M$ and $m = H$. This value of $R_{j,m}$ allows the processing rate (the numerator of the expression for $R_{j,m}$) to be increased the most for the smallest increase in power consumption (the denominator of $R_{j,m}$). The algorithm, therefore, increases the value of X^{tmp}_j to H. This step is repeated while inequality (3) is being satisfied. Finally, all the values of X_j are replaced by the values of X^{tmp}_j. CPD-H repeatedly finds the highest values of $R_{j,m}$ in S^{ratio}, which involves sorting the elements in this set. Thus, the time complexity for Algorithm 2 is calculated as $O(log(n(S^{\text{ratio}}))n(S^{\text{ratio}}))$ [37].

V. Version-Set Determination and Task Allocation Algorithm

A. Problem Formulation

We assume that the lowest bitrate version of the content broadcast over each channel is always transcoded. This guarantees that each viewer receives the transmission, albeit at a bitrate lower than that requested. Thus, two bitrate versions are always available: the original version and the version with the lowest bitrate version. That leaves $N^{\text{br}} - 2$ versions, which may or may not be transcoded, making $2^{N^{\text{br}}-2}$ possible combinations. Let S^{set}_i be a set of all combinations of bitrate versions for channel i, sorted in ascending order of total CPU processing rate required to transcode all the versions in that combination. Let $S^{\text{set}}_{i,m}$ be the mth element in S^{set}_i $(i = 1, \ldots, N^{\text{ch}}$ and $m = 1, \ldots, N^{\text{set}})$, where N^{set} is the number of elements in S^{set}_i, which is $2^{N^{\text{br}}-2}$.

Let $p_{i,k}$ be the probability that version $V_{i,k}$ is accessed by a viewer $(i = 1, \ldots, N^{\text{ch}}$ and $k = 1, \ldots, N^{\text{br}})$, where $\sum_{i=1}^{N^{\text{ch}}} \sum_{k=1}^{N^{\text{br}}} p_{i,k} = 1$. The value of $p_{i,k}$ can be derived from the current view count of channel i. Let $Q_{i,k}$ be the video quality index for version $V_{i,k}$. We use the video multimethod

Authorized licensed use limited to: Inha University. Downloaded on September 21,2022 at 01:41:37 UTC from IEEE Xplore. Restrictions apply.

Algorithm 2: The Greedy Heuristic (CPD-H), Which Determines CPU Processing Rate for BNs.

```
   Input: $P^{\mathrm{limit}}$, $C_j^{\mathrm{BN}}$ and $P_j(u)$, $(j = 1, ..., N^{\mathrm{BN}})$ ;
   Output: $X_j$, $(j = 1, ..., N^{\mathrm{BN}})$;
1  Temporary variables: $X_j^{\mathrm{tmp}} \leftarrow 0$, $(j = 1, ..., N^{\mathrm{BN}})$;
2  List of $R_{j,m}$ parameters: $S^{\mathrm{ratio}}$;
3  while $S^{\mathrm{ratio}} \neq \phi$ do
4      Find the highest value of $R_{j,m} \in S^{\mathrm{ratio}}$, for which
         $j = M$ and $m = H$;
5      if $H > X_M^{\mathrm{tmp}}$ and $\sum_{j=1}^{N^{\mathrm{BN}}} P_j(\frac{X_j^{\mathrm{tmp}}}{C_j^{\mathrm{BN}}}) \leq P^{\mathrm{limit}}$ then
6          $X_M^{\mathrm{tmp}} \leftarrow H$;
7      end
8      $S^{\mathrm{ratio}} \leftarrow S^{\mathrm{ratio}} - \{R_{M,H}\}$;
9  end
10 for $j = 1$ to $N^{\mathrm{BN}}$ do
11     $X_j \leftarrow X_j^{\mathrm{tmp}}$;
12 end
```

assessment fusion (VMAF) index developed by Netflix, which is known to be more accurate than other popular video quality indices [38], [39]. Values of $Q_{i,k}$ can be derived using the VMAF measurement tool [38], [39]. We now introduce a metric $G_{i,k}^{\mathrm{ver}}$, which represents the popularity-weighted video quality (PWQ) of version $V_{i,k}$ when (and if) it has been transcoded. This is expressed as follows:

$$G_{i,k}^{\mathrm{ver}} = Q_{i,k} p_{i,k}. \tag{6}$$

The total CPU processing rate $C_{i,m}^{\mathrm{set}}$ required to transcode all the versions in set $S_{i,m}^{\mathrm{set}}$ can be determined from $C_{i,k}^{\mathrm{ch}}$, which is the CPU processing rate required for transcoding version $V_{i,k}$. This is expressed as follows:

$$C_{i,m}^{\mathrm{set}} = \sum_{\forall V_{i,k} \in S_{i,m}^{\mathrm{set}}} C_{i,k}^{\mathrm{ch}}. \tag{7}$$

In live streaming, each viewer requests the highest bitrate version that can be supported by the current network situation [1], [2]. If the delivered version is higher than the requested bitrate by the viewer, the QoE is likely to deteriorate due to rebuffering effect, which may even cause playback to stop altogether. Therefore, if any version k is not included in powerset $S_{i,m}^{\mathrm{set}}$, then the highest version lower than k in $S_{i,m}^{\mathrm{set}}$ needs to be delivered. So, we now introduce $V_{i,m,k}^{\mathrm{high}}$, which is the index of the version in set $S_{i,m}^{\mathrm{set}}$ with the highest bitrate, which is lower than or equal to that of version index k. The total PWQ $G_{i,m}^{\mathrm{set}}$ of set $S_{i,m}^{\mathrm{set}}$ can, thus, be calculated as follows:

$$G_{i,m}^{\mathrm{set}} = \sum_{k=1}^{N^{\mathrm{br}}} G_{i,V_{i,m,k}^{\mathrm{high}}}^{\mathrm{ver}}. \tag{8}$$

We now formulate a combination of two problems: version-set determination (VSD) and TA. We introduce a variable Y_i to indicate the Y_ith element in powerset S_i^{set}, which is selected to determine the set of versions to be transcoded for channel i. For example, if Y_i is set to N^{set}, then all the versions are transcoded. Finding the value of Y_i must satisfy the total processing capacity constraint as follows:

$$\sum_{i=1}^{N^{\mathrm{ch}}} C_{i,Y_i}^{\mathrm{set}} \leq \sum_{j=1}^{N^{\mathrm{BN}}} X_j. \tag{9}$$

Then, the VSD problem determines the values of Y_i for each channel i with the aim of maximizing the aggregate PWQ while satisfying Inequality (9) as follows:

$$\text{Maximize} \sum_{i=1}^{N^{\mathrm{ch}}} G_{i,Y_i}^{\mathrm{set}}$$

$$\text{subject to} \sum_{i=1}^{N^{\mathrm{ch}}} C_{i,Y_i}^{\mathrm{set}} \leq \sum_{j=1}^{N^{\mathrm{BN}}} X_j.$$

After determining the values of Y_i, we can establish a set S^{ver} of bitrate versions to be transcoded as follows:

$$S^{\mathrm{ver}} = \{V_{i,k} | \forall i, V_{i,k} \in S_{i,Y_i}^{\mathrm{set}}\}. \tag{10}$$

Based on this, we can formulate the TA problem, in which a task for every version $V_{i,k} \in S^{\mathrm{ver}}$ must be allocated to one of the BNs. However, to give the best chance of being able to assign all tasks to BNs, workloads must be balanced across the BNs.

To allow for this, we introduce logical variables $Z_{i,k}$, which indicate whether $V_{i,k}$, $(V_{i,k} \in S^{\mathrm{ver}})$ can be transcoded at the $Z_{i,k}$th BN $(Z_{i,k} = 0, \ldots, N^{\mathrm{BN}})$. If the task of transcoding version $V_{i,k}$ cannot be assigned to any BN, then $Z_{i,k} = 0$. The total amount of processing rates allocated for BN j must not X_j as follows:

$$\sum_{\{V_{i,k} | V_{i,k} \in S^{\mathrm{ver}},\ Z_{i,k} = j\}} C_{i,k}^{\mathrm{ch}} \leq X_j. \tag{11}$$

The goal of the TA problem then becomes that of determining the values of $Z_{i,k}$, with the aim of minimizing the loss of PWQ, which occurs because of the versions that are not transcoded because $Z_{i,k} = 0$. This goal can be expressed as follows:

$$\text{Minimize} \sum_{\{V_{i,k} | V_{i,k} \in S^{\mathrm{ver}}, Z_{i,k} = 0\}} G_{i,k}^{\mathrm{ver}}$$

$$\text{subject to} \sum_{\{V_{i,k} | V_{i,k} \in S^{\mathrm{ver}}, Z_{i,k} = j\}} C_{i,k}^{\mathrm{ch}} \leq X_j (j = 1, \ldots, N^{\mathrm{BN}}).$$

B. Algorithm Description

The combination of VSD and TA, which we call VSD-TA, can be considered as a variant of a multiple-choice multiple-knapsack problem, which is to choose one item from each of several classes and to allocate these items to multiple knapsacks with the aim of maximizing overall profit. This problem is shown to be NP-hard [36]. The VSD-TA problem requires us to select one of the elements from S_i^{set} for allocation to each channel i so as to maximize the total PWQ while allocating the transcoding task for each $V_{i,k}$, $(V_{i,k} \in S_{i,Y_i}^{\mathrm{set}})$ to one of the BNs, while further satisfying the total processing capacity constraint at each BN. It would be impractical to examine all the combinations of Y_i and $Z_{i,k}$, so we develop a heuristic for VSD-TA.

This algorithm has two phases: in the VSD phase, it finds the value of Y_i for each channel i; in the TA phase, it finds

Authorized licensed use limited to: Inha University. Downloaded on September 21,2022 at 01:41:37 UTC from IEEE Xplore. Restrictions apply.

$Z_{i,k}$ for each version $V_{i,k}$, where $V_{i,k} \in S^{\mathrm{ver}}$. We use a greedy approach to find the value of Y_i for each channel i, but to determine $Z_{i,k}$, we use a worst-fit method, in which successive tasks are allocated to the BN with the lowest existing load, in order to balance workloads across the BNs. The VSD phase has initialization and update steps as follows:

1) *Initialization Step:* The last element in S_i^{set} is selected for each channel so that a version can be transcoded at every bitrate. These elements are stored in temporary variables Y_i^{tmp}, which are initialized as follows: $\forall i, Y_i^{\mathrm{tmp}} = N^{\mathrm{set}}$.
2) *Update Step:* The values of Y_i^{tmp} are updated to form a set of versions to be transcoded for each channel. We, therefore, introduce index parameters $I_{i,m}$, which identify each selection of i and m, where $i = 1, \ldots, N^{\mathrm{ch}}$ and $m = 1, \ldots, N^{\mathrm{set}} - 1$. These parameters are determined as follows:

$$I_{i,m} = \frac{G_{i,N^{\mathrm{set}}}^{\mathrm{set}} - G_{i,m}^{\mathrm{set}}}{C_{i,N^{\mathrm{set}}}^{\mathrm{set}} - C_{i,m}^{\mathrm{set}}} \tag{12}$$

in which the numerator and the denominator, respectively, represent the difference in PWQ and processing rate, which results from selecting the $S_{i,m}^{\mathrm{set}}$th element rather than the last element in set S_i^{set}.

The set of versions to be transcoded for some channels must be modified in order to satisfy the processing rate requirements of all the BNs. We, therefore, favor selections with lower value of $I_{i,m}$ because they decrease the reduction in PWQ, which is associated with the largest possible decrease in the CPU processing rate. This is achieved by constructing a list A of $I_{i,m}$ and sorting it into nondescending order by the values of $I_{i,m}$. The value of Y_M^{tmp} is then updated to L, where $I_{M,L}$ is the index with the minimum value. This step is repeated until the processing capacity constraint is satisfied (line 11).

The VSD phase finally creates a set of versions S^{ver} that can be transcoded for all the channels, where $V_{i,k}$ in S^{ver} is sorted into a nonincreasing order of the corresponding values of $\frac{G_{i,k}^{\mathrm{ver}}}{C_{i,k}^{\mathrm{ch}}}$. The VSD phase repeatedly finds the lowest values of $I_{i,m}$ in set A, which requires a sorting operation. The time complexity for the VSD phase is, thus, calculated as $O(\log(n(A))n(A))$ [37].

Next, the TA phase determines the index of the BN, which receives each transcoding task, starting with the task that transcodes version $V_{i,k}$ with the highest value of $\frac{G_{i,k}^{\mathrm{ver}}}{C_{i,k}^{\mathrm{ch}}}$. This meets our objective of giving higher priority to tasks that produce version with higher PWQ at a low processing rate. The TA phase aims at minimizing the unused processing rate over all the BNs. Therefore, the transcoding task with higher values of $\frac{G_{i,k}^{\mathrm{ver}}}{C_{i,k}^{\mathrm{ch}}}$ is allocated to the BN with the largest remaining processing rate. This phase continues allocating tasks to BNs until the next allocation meets inequality (11). Again, this TA phase requires sorting of elements in set S^{ver}, where the time complexity is $O(\log(n(S^{\mathrm{ver}}))n(S^{\mathrm{ver}}))$ [37]. Algorithm 3 shows the details of the VSD-TA algorithm. The VSD phase occupies lines 7–19, and the TA phase occupies lines 20–29.

Algorithm 3: Version-set Determination and Task Allocation (VSD-TA) Algorithm.

```
Input: $C_{i,m}^{set}$, $X_j$ and $G_{i,m}^{set}$, $(i = 1, ..., N^{ch}$, $j = 1, ...N^{BN}$ and $m = 1, ..., N^{set})$;
Output: $Y_i$ and $Z_{i,k}$, $(i = 1, ..., N^{ch}$ and $k = 1, ..., N^{br})$ ;
1  Temporary variables for each channel $i$: $Y_i^{tmp} \leftarrow N^{set}$;
2  Temporary variables for each BN $j$: $C_j^{tmp} \leftarrow 0$;
3  List of $I_{i,m}$ parameters: $A$;
4  Values of $Z_{i,k}$, $(V_{i,k} \in S^{ver})$ are all initialized to 0;
5  /* VSD phase */;
6  while $A \neq \phi$ do
7      Find the lowest value of $I_{i,m} \in A$ for which $i = M$ and $m = L$ and remove $I_{M,L}$ from $A$;
8      if $C_{M,L}^{set} < C_{M,Y_M^{tmp}}^{set}$ then
9          $Y_M^{tmp} \leftarrow L$;
10         if $\sum_{i=1}^{N^{ch}} C_{i,Y_i^{tmp}}^{set} \leq \sum_{j=1}^{N^{BN}} X_j$ then
11             Break the while loop;
12         end
13     end
14 end
15 for $i = 1$ to $N^{ch}$ do
16     $Y_i \leftarrow Y_i^{tmp}$;
17 end
18 /* TA phase */;
19 while $S^{ver} \neq \phi$ do
20     Find the version $V_{i,k}$ with the highest value of $\frac{G_{i,k}^{ver}}{C_{i,k}^{ch}}$ from $S^{ver}$ for which $i = M$ and $k = H$;
21     Find the BN index $j$ corresponding to the lowest value of $C_j^{tmp}$ for which $j = L$;
22     if $C_L^{tmp} + C_{M,H}^{ch} \leq X_L$ then
23         $Z_{M,H} \leftarrow L$;
24         $C_L^{tmp} \leftarrow C_L^{tmp} + C_{M,H}^{ch}$;
25     end
26     $S^{ver} \leftarrow S^{ver} - \{V_{M,H}\}$;
27 end
```

The proposed scheme requires three steps: CPU processing-rate determination, VSD, and transcoding TA. Except for these processes, the other steps are the same as those of typical transcoding systems for live streaming. Since the CPU processing-rate determination algorithm runs offline to meet the power limit constraints, only the VSD-TA algorithm is executed during live-streaming sessions. In addition, this algorithm only runs when the workload changes as channels are added or removed in practice, suggesting that its overhead is low.

VI. Experimental Results

We perform simulations to evaluate our scheme in terms of power consumption and video quality. We model the power and processing-rate characteristics of ten commercial servers released between 2020 and 2021, as tabulated in Table III [35].

Our model accounts for four resolutions and seven bitrates, as recommended by public transcoding cloud provider Zencoder [3], as shown in Table IV. We use the measured values

Authorized licensed use limited to: Inha University. Downloaded on September 21,2022 at 01:41:37 UTC from IEEE Xplore. Restrictions apply.

TABLE III
BACKEND NODE SPECIFICATION [35]

Server name	Processing	Active power	Idle power
ASUS RS700A-E9-RS4V2	2.25GHz, 128cores	199W–425W	92.9W
ASUS RS620SA-E10-RS12	2.25GHz, 384cores	492W–1,140W	228W
Dell PowerEdge R7525	2.00GHz, 128cores	181W–393W	72.3W
Dell PowerEdge R6525	2.00GHz, 128cores	188W–404W	81.6W
HPE ProLiant DL580 Gen10	2.70GHz, 112cores	243W–894W	123W
HPE ProLiant DL325 Gen10	2.45GHz, 64cores	100W–268W	56.8W
HPE Apollo XL225n Gen10 Plus	2.45GHz, 512cores	812W–2,260W	523W
FUJITSU Primergy RX2530 M6	2.30GHz, 80cores	194W–580W	131W
FUJITSU Primergy RX4770 M6	2.9GHz, 112cores	257W–834W	129W
Lenovo ThinkSystem SR665	2.45GHz, 128cores	171W–455W	108W

TABLE IV
RESOLUTION AND BITRATE INFORMATION [3], [40]

Resolution	Bitrate (kbps)	Mean VMAF
400×224	200, 400, 600	40, 60, 72
640×360	1000, 1500	90, 92.5
1280×720	2000	95
1920×1080 (Original)	2750	100

given in Table II for the parameters a and b, which are used to determine the processor rate (CPU frequency in gigahertz) required to transcode a version at each bitrate. The processing rate for each channel is subject to a random adjustment of between -5% and +5%, to reflect the effect of different types of content.

VMAF values for the version at each bitrate with a normal distribution are generated from the measured values [40]. The mean values of the distribution at each bitrate are given in Table IV [40]. The standard deviations of the distributions are varied between 2 and 14 [40]. The access probability of each channel is modeled with a gamma distribution using data from Twitch.tv [41], so the parameter values of the distribution, k and θ are set to 0.399 and 14260, respectively [41]. The number of concurrent channels on Twitch.tv has been reported to be between 4000 and 8000 [3], so N^{ch} was set to 6000.

To express the relative popularity of different versions, we use a normal distribution with a mean value of μ and a variance of σ [42]. We consider four models of version popularity: if the high-bitrate versions are popular (HVP), then μ is set to N^{set}; if the medium-bitrate versions are popular (MVP), then μ is set to $\frac{1+N^{\mathrm{set}}}{2}$; if the low-bitrate versions are popular (LVP), then μ is set to 1; and if random versions are popular (RVP), then a value of μ between 1 and N^{set} is chosen at random. The value of σ is set independently and determines the dominance of the most popular versions in each case [42]. As a default, we use MVP with σ set to 1 because medium-bitrate versions are typically found [42] to be the most popular in practice.

Three methods of allocating transcoding tasks are considered: a round-robin (RR) method, in which each task is assigned to each BN in sequence, a random assignment (RA) method that selects BN randomly for each task, and a power-aware (PA) method that selects the BN, which currently has the lowest ratio of active power to CPU utilization. Both RR and RA methods can effectively achieve load balancing [43], [44], whereas the PA method attempts to reduce power consumption per CPU usage.

Two methods of determining the version set are considered, both of which take channel popularity into account: the All

TABLE V
PERCENTAGE DIFFERENCE BETWEEN THE TOTAL PROCESSING RATES ACHIEVED BY CPD-H AND CPD-DP

P^{ratio}	40%	50%	60%	70%	80%	90%
Difference	1.07%	1.31%	0.64%	0.66%	0.29%	0%

TABLE VI
EXECUTION TIMES FOR THE MUPD PROBLEM (S) FOR DIFFERENT VALUES OF P^{ratio}

Scheme	P^{ratio}					
	40%	50%	60%	70%	80%	90%
CPD-DP	603.71	936.14	1212.65	1401.7	1486.06	1547.84
CPD-H	0.001	0.001	0.001	0.001	0.001	0.001

versions transcoded in Popular channels (AP) method divides the channels into popular and unpopular groups and then transcodes all versions $S^{\mathrm{set}}_{i,N^{\mathrm{set}}}$ for the popular channels but only $S^{\mathrm{set}}_{i,1}$ for the unpopular channels. The highest popularity first (HPF) method first transcodes the lowest bitrate versions for all channels and then transcodes the versions with the highest values of access probability ($p_{i,k}$) successively, until the power limitation is encountered. To examine the effect of power limit values, we select the values of P^{ratio}, which is the percentage of the power consumption compared to when all BNs are fully utilized.

We then compare our scheme (CPD-H+VSA-TA) with six benchmark combinations of the schemes described above: RR+AP, RR+HPF, RA+AP, RA+HPF, PA+AP, and PA+HPF.

A. Algorithm Efficiency

In practice, CPD-DP cannot produce an optimal solution for a large number of BNs, so we compare CPD-H with CPD-DP, setting N^{BN} to 10. Table V gives the percentage difference in total processing rate between CPD-H and CPD-DP for different values of P^{ratio}. CPD-H provides a near-optimal solution with results that are only 0–1.31% worse than those from CPD-DP. Table VI gives the execution times of CPD-DP and CPD-H for different values of P^{ratio} on an Intel i7-10700 processor. The CPD-H scheme is vastly superior, and we use it as the default scheme for allocating CPU processing rates.

To assess the efficiency of the VSD-TA algorithm, we find an upper bound on the PWQ achieved by the VSD component. Like CPD-DP, the problem of determining version-sets has an affinity with the MCKP, and we use the DP algorithm to approach it in a similar way. However, this DP algorithm assumes that each processor is fully used, which is why its results form an upper bound on results from the actual VSD-TA algorithm.

Since this DP algorithm can only produce results in a reasonable time for low values of N^{BN}, we restricted our comparison to $N^{\mathrm{BN}} = 10$. The unsuitability of this algorithm for larger values of N^{BN} is illustrated by the fact that it took nearly 40 min to obtain a result for $P^{\mathrm{ratio}} = 90\%$ on an Intel i7-10700.

Table VII gives the percentage of total PWQ by which the results from VSD differed from the upper bound, for different values of P^{ratio}. These results suggest that the VSD-TA algorithm gets close to the upper bound for small workloads.

Authorized licensed use limited to: Inha University. Downloaded on September 21,2022 at 01:41:37 UTC from IEEE Xplore. Restrictions apply.

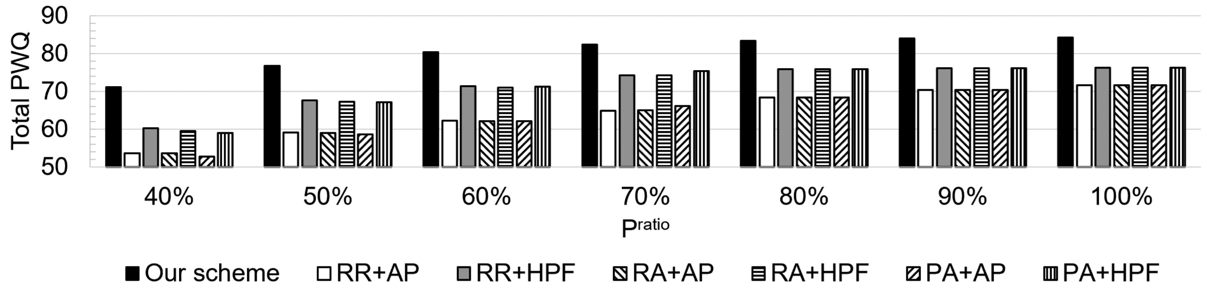

Fig. 2. Total PWQ against P^{ratio}, when $N^{\text{BN}} = 120$ and $N^{\text{ch}} = 6000$.

TABLE VII
PERCENTAGE DIFFERENCE BETWEEN THE TOTAL PWQ VALUES OBTAINED BY VSD-TA AND THE UPPER BOUND

P^{ratio}	40%	50%	60%	70%	80%	90%
Difference	2.27%	2.31%	0.86%	0.47%	0.51%	0.39%

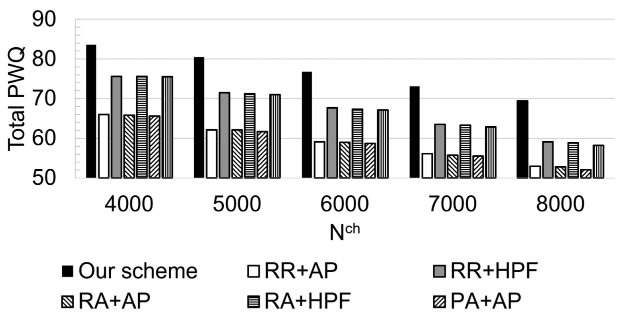

Fig. 3. Total PWQ against N^{ch}, when $P^{\text{ratio}} = 50\%$ and $N^{\text{BN}} = 120$.

B. Effect of the Power Limit

We examine the effect of the power limit on the total PWQ when $N^{\text{BN}} = 120$ and $N^{\text{ch}} = 6000$. Fig. 2 shows how total PWQ values vary against P^{ratio} for all the different schemes. As expected, the total PWQ increases with the power limit. Our scheme produces the highest PWQ under all power limits, achieving 9.3–34.83% more PWQ than the other schemes, for all power limit values. RA+HPF is the best of the other schemes, but our scheme produces PWQ values that are even better, by 9.92–19.38%. This difference is greater at lower power limits.

If we look at the methods of version-set selection used by six other schemes, we see that HPF shows consistently better performance than AP. In terms of TA, RR and RA yield similar results since these have the same probability of each BN being selected, which is $\frac{1}{N^{\text{BN}}}$. Therefore, RR and RA are better than those from PA when P^{ratio} is between 40% and 60%, but when P^{ratio} increases above 70%, PA performs best.

C. Effect of the Number of Channels

To examine the effect of channel workloads, we determine the total PWQ for a range of values of N^{ch} values when $P^{\text{ratio}} = 50\%$ and $N^{\text{BN}} = 120$, with the results shown in Fig. 3. Again, our scheme outperforms the others, producing 10.54%–33.08% more PWQ than the other schemes. This difference tends to be higher when more channels are available, giving more scope for effective selection of transcoded versions for each channel.

D. Effect of the Number of Backend Nodes

We also examine how the number of BNs N^{BN} affects the total PWQ when $P^{\text{ratio}} = 50\%$ and $N^{\text{ch}} = 6000$, with the results shown in Fig. 4. Our scheme produces PWQ values that are 10.79% to 38.37% higher than those produced by the other schemes. This percentage difference gets smaller as the number of BNs increases, which shows that our scheme is effective under heavy loads. Even when there are many BNs, our scheme performs relatively better, suggesting that it is more scalable.

E. Effect of Version Popularity

We examine the effect of the distribution of version popularity on the total PWQ when $P^{\text{ratio}} = 50\%$, $N^{\text{BN}} = 120$, and $N^{\text{ch}} = 6000$, with the results shown in Fig. 5. Our scheme produces PWQ values, which are 3.82–30.83% higher than those produced by other schemes.

This gap is highest for the MVP distribution and lowest for the LVP distribution. This can be explained as follows: the version with the lowest bitrate is transcoded for all the channels by all the schemes. With the LVP distribution, the lower versions with low bitrate but slightly higher in VMAF indices than the version with the lowest bitrate are often requested. This produces marginal PWQ difference among schemes. However, for the MVP distribution, our scheme transcodes medium bitrate versions with the aim of maximizing total PWQ, making better use of the available CPU processing rate.

F. Effect of Different Sets of Bitrates

Different video streaming service providers recommend different sets of bitrates: these recommendations are given in Table VIII. To examine the effect of these sets, we determine total PWQ values for the five different bitrate sets in the table, when $P^{\text{ratio}} = 50\%$, $N^{\text{BN}} = 120$, and $N^{\text{ch}} = 6000$. The results, shown in Fig. 6, indicates that the choice of bitrate set has a great effect on the relative performance of our scheme. With the Watson's bitrates, the gap ranges between 14.46% and

Authorized licensed use limited to: Inha University. Downloaded on September 21,2022 at 01:41:37 UTC from IEEE Xplore. Restrictions apply.

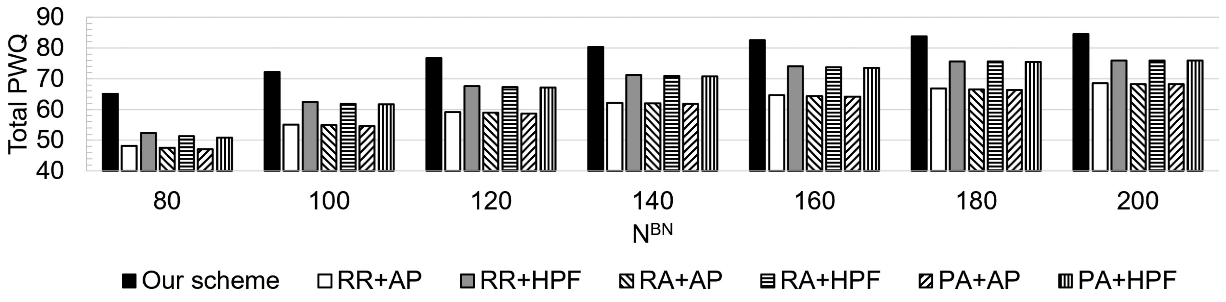

Fig. 4. Total PWQ against N^{BN}, when $P^{\mathrm{ratio}} = 50\%$ and $N^{\mathrm{ch}} = 6000$.

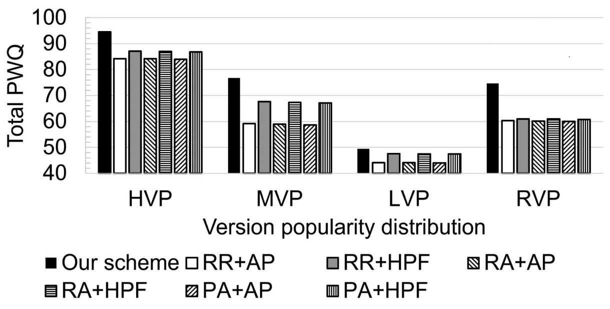

Fig. 5. Total PWQ against version popularity distributions, when $P^{\mathrm{ratio}} = 50\%$, $N^{\mathrm{BN}} = 120$, and $N^{\mathrm{ch}} = 6000$.

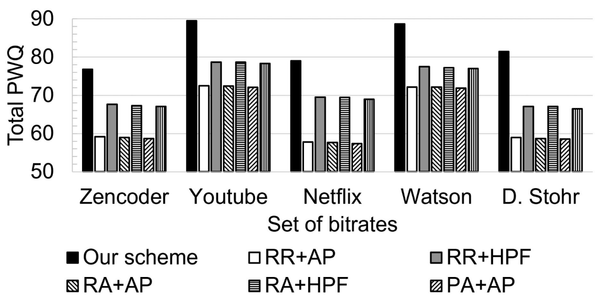

Fig. 6. Total PWQ against bitrate recommendations.

TABLE VIII
SETS OF BITRATES RECOMMENDED BY FIVE VIDEO STREAMING SERVICE PROVIDERS [45]–[48]

Service provider	Resolution	Bitrate (in kbps)	Mean VMAF
Youtube [45]	426×240	500	65
	640×360	700	77
	854×480	1250	91.3
	1280×720	2750, 4125	96, 98
	1920×1080	4500, 6750	98.2, 98.4
	2560×1440	9500, 13500	98.6, 99.5
	3840×2160	23500, 35000	99.5, 100
Netflix [46]	320×240	235	43
	384×288	375	57.5
	512×384	560, 750	67.5, 80
	640×480	1050	90.2
	720×480	1750	93.8
	1280×720	2350, 3000	95.5, 97
	1920×1080	4300, 5800	98.1, 100
IBM Watson Media [47]	480×270	400	60
	640×480	1000	90
	854×480	1350	91.5
	960×560	1350	91.5
	1280×720	2750	96
	1920×1080	6000	98.3
	3840×2160	11000	100
D. Stohr [48]	480×270	253	45
	640×360	505, 807	65, 82
	1280×720	1500, 2400	92.5, 95.5
	1920×1080	3000, 4000, 6000, 10000	97, 97.8, 98.3, 100

TABLE IX
MEAN PSNR VALUES FOR BITRATE VERSIONS [3], [40]

Resolution	Bitrate (kbps)	Mean PSNR
400×224	200, 400, 600	32, 33.5, 35
640×360	1000, 1500	38.5, 40
1280×720	2000	41.5
1920×1080 (Original)	2750	43

TABLE X
CONVERSION FROM PSNR TO MOS [49]

PSNR	MOS	Mearning
>37	5	Excellent
31–37	4	Good
25–31	3	Fair
20–25	2	Poor
<20	1	Bad

23.45%, against RR+HPF and PA+AP, respectively; with the D. Stohr bitrates, it ranges between 21.31% and 39.12%, against RR+HPF and PA+AP, respectively. We can attribute this to the larger difference between the medium- and low-bitrate versions, for which the effect of version selection is more pronounced.

G. Effect of Different Video Quality Metrics

To examine the effect of various video quality metrics, we determine total PWQ values for the three different metrics, including VMAF, peak signal-to-noise ratio (PSNR) and mean opinion score (MOS), when $P^{\mathrm{ratio}} = 50\%$, $N^{\mathrm{BN}} = 120$, and $N^{\mathrm{ch}} = 6000$. A PSNR value is generated for each bitrate version using a normal distribution from the measured PSNR values, where the mean values of the PSNR distribution at each bitrate version are given in Table IX [40], [49]. The standard deviations of the distributions for PSNR values are varied between 4.1 and 5.4 [40]. Then, the MOS values can be derived from the PSNR values, as described in Table X [49].

Fig. 7 shows the average PWQ normalized against results for our scheme, which suggests that the similar PWQ results can be obtained regardless of the video quality metrics. For example, for the PSNR metric, our scheme has a value of 41.14, while other schemes have values from 36.56 to 38.38, yielding the average difference between 7.18% and 12.53%. On the other

Authorized licensed use limited to: Inha University. Downloaded on September 21,2022 at 01:41:37 UTC from IEEE Xplore. Restrictions apply.

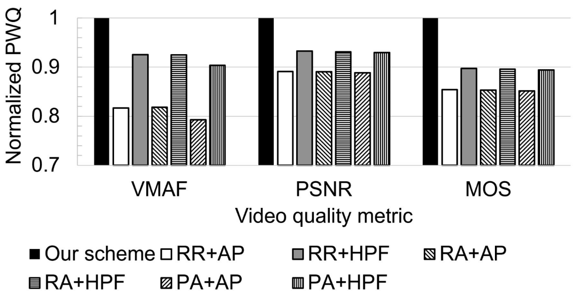

Fig. 7. Normalized PWQ against three video quality metrics.

TABLE XI
AVERAGE EXECUTION TIME FOR OUR SCHEME (S) AGAINST P^{ratio}

	Average execution time against P^{ratio}						
P^{ratio}	40%	50%	60%	70%	80%	90%	100%
HVP	0.061	0.062	0.062	0.064	0.064	0.065	0.067
MVP	0.065	0.065	0.066	0.066	0.063	0.066	0.067
LVP	0.061	0.061	0.061	0.062	0.062	0.065	0.071
RVP	0.062	0.064	0.064	0.066	0.066	0.066	0.067
Average	0.062	0.063	0.063	0.064	0.064	0.065	0.068

hand, for the MOS metric, our scheme has a value of 4.88, while other schemes have values between 4.16 and 4.38, yielding the gap between 11.46% and 17.45%.

H. Algorithm Execution Time

To evaluate the overhead of our scheme, we measured the average execution time on the i7-10700 CPU when $N^{\text{BN}} = 120$ and $N^{\text{ch}} = 6000$. Table XI tabulates how average execution times vary against P^{ratio}. The execution times are between 61 and 71 ms, which clearly indicates that it runs with reasonable overhead. The execution time increases with the values of P^{ratio} because more computation is required to support the additional power limit.

VII. CONCLUSION

In this article, we proposed a new scheme to improve aggregate PWQ in a live-streaming server with heterogeneous BNs, under a peak power constraint. We developed a DP algorithm, together with a heuristic that determines the processing rate of the CPU in each BN, while meeting the power constraint. This heuristic forms the basis of an algorithm that determines what content is transcoded to what bitrates for each channel, and assigns each transcoding task to a processor node. It first assumes that transcoding produces versions of all contents at all bitrates for every channel and then greedily reduces the number of versions to be produced for certain channels, until all the associated transcoding tasks can be performed with the processor power available in the BNs.

We conducted simulations to assess our scheme against six benchmark schemes in terms of total PWQ, while varying the power limit, the number of channels, the number of BNs, the set of available bitrates, the distribution of content popularity, and the metrics of video quality. The results show that, with the same power constraint, our scheme improves PWQ by 3.82%–39.12%, compared to the benchmark schemes, working under the same power constraint. Our scheme is particularly effective when power is tight, when there is a large workload because more channels have to be serviced by fewer BNs, when there are large differences in the bitrates of transcoded versions, and when versions with medium bitrates are the most popular.

The demand for live-streaming applications is steadily increasing, and with it the power consumption of the associated transcoding server infrastructure. The results show that our scheme can contribute to the effective power management of live-streaming systems. In future work, we plan to extend our scheme to fog-based live-streaming architectures.

REFERENCES

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

[2] D. Vergados, A. Michalas, A. Sgora, D. Vergados, and P. Chatzimisios, "FDASH: A fuzzy-based MPEG/DASH adaptation algorithm," *IEEE Syst. J.*, vol. 10, no. 2, pp. 859–868, Jun. 2016.

[3] 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.*, Mar. 2015, pp. 49–60.

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

[5] K. Pires and G. Simon, "YouTube live and Twitch: A tour of user-generated live streaming systems," in *Proc. ACM Multimedia Syst. Conf.*, Mar. 2015, pp. 225–230.

[6] 2021. [Online]. Available: https://www.businessofapps.com/data/twitch-statistics/

[7] N. Dao *et al.*, "Hit ratio and content quality tradeoff for adaptive bitrate streaming in edge caching systems," *IEEE Syst. J.*, vol. 15, no. 4, pp. 5094–5097, Mar. 2021.

[8] T. Zhang, F. Ren, W. Cheng, X. Luo, R. Shu, and X. Liu, "Modeling and analyzing the influence of chunk size variation on bitrate adaptation in DASH," in *Proc. IEEE Int. Conf. Comput. Commun.*, May 2017, pp. 1–9.

[9] W. Lin, W. Wu, H. Wang, J. Z. Wang, and C.-H. Hsu, "Experimental and quantitative analysis of server power model for cloud data centers," *Future Gener. Comput. Syst.*, vol. 86, pp. 940–950, 2018.

[10] Y. Wang, D. Nortershauser, S. Masson, and J. Menaud, "An empirical study of power characterization approaches for servers," in *Proc. Int. Conf. Smart Grids, Green Commun. IT Energy-Aware Technol.*, Jun. 2019, pp. 1–6.

[11] 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. 13, no. 4, pp. 600–612, Apr. 2018.

[12] D. Lee, J. Lee, and M. Song, "Video quality adaptation for limiting transcoding energy consumption in video servers," *IEEE Access*, vol. 7, pp. 126253–126264, 2019.

[13] 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, Feb. 2015, Art. no. 45.

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

[15] 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.

[16] 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.

[17] X. Li, M. Salehi, Y. Joshi, M. Darwich, B. Landreneau, and M. Bayoumi, "Performance analysis and modeling of video transcoding using heterogeneous cloud services," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 4, pp. 910–922, Sep. 2018.

[18] 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.

Authorized licensed use limited to: Inha University. Downloaded on September 21,2022 at 01:41:37 UTC from IEEE Xplore. Restrictions apply.

[19] F. Jokhio, A. Ashraf, S. Lafond, and J. Lilius, “A computation and storage trade-off strategy for cost-efficient video transcoding in the cloud,” in *Proc. Euromicro Conf. Softw. Eng. Adv. Appl.*, Sep. 2013, pp. 1–4.
[20] X. Zhang, D. Xiong, K. Zhao, C. Chen, and T. Zhang, “Realizing low-cost flash memory based video caching in content delivery systems,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 4, pp. 984–996, Apr. 2018.
[21] A. Panarello, A. Celesti, M. Fazio, A. Puliafito, and M. Villari, “A big video data transcoding service for social media over federated clouds,” *Multimedia Tools Appl.*, vol. 79, pp. 9037–9061, May 2019.
[22] Y. Zheng, D. Wu, Y. Ke, C. Yang, M. Chen, and G. Zhang, “Online cloud transcoding and distribution for crowdsourced live game video streaming,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 27, no. 8, pp. 1777–1789, Aug. 2017.
[23] B. Mada, M. Bagaa, and T. Taleb, “Efficient transcoding and streaming mechanism in multiple cloud domains,” in *Proc. IEEE Global Commun. Conf.*, Dec. 2017, pp. 1–6.
[24] Q. He, C. Zhang, and J. Liu, “Utilizing massive viewers for video transcoding in crowdsourced live streaming,” in *Proc. IEEE Int. Conf. Cloud Comput.*, Jun. 2016, pp. 116–123.
[25] Q. He, C. Zhang, X. Ma, and J. Liu, “Fog-based transcoding for crowdsourced video livecast,” *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 28–33, Apr. 2017.
[26] Z. Zhang, R. Wang, F. Yu, F. Fu, and Q. Yan, “QoS aware transcoding for live streaming in edge-clouds aided HetNets: An enhanced actor-critic approach,” *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 11295–11308, Nov. 2019.
[27] D. Wang *et al.*, “Adaptive wireless video streaming based on edge computing: Opportunities and approaches,” *IEEE Trans. Serv. Comput.*, vol. 12, no. 5, pp. 685–697, Sep. 2018.
[28] Y. Zhu, Q. He, J. Liu, B. Li, and Y. Hu, “When crowd meets big video data: Cloud-edge collaborative transcoding for personal livecast,” *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 1, pp. 42–53, Oct. 2018.
[29] J. Kim, Y. Jung, H. Yeo, J. Ye, and D. Han, “Neural-enhanced live streaming: Improving live video ingest via online learning,” in *Proc. ACM Spec. Int. Group Data Commun.*, Jan. 2020, pp. 107–125.
[30] J. Liu, W. Zhang, S. Huang, H. Du, and Q. Zheng, “QoE-driven HAS live video channel placement in the media cloud integration,” *IEEE Trans. Multimedia*, vol. 23, pp. 1530–1541, Jun. 2020.
[31] D. Lee and M. Song, “Popularity-based transcoding workload allocation for improving video quality in live streaming systems,” in *Proc. ACM Int. Conf. Emerg. Netw. Exp. Technol.*, Nov. 2020, pp. 540–541.
[32] L. Costero, A. Iranfar, M. Zapater, F. Igual, K. Olcoz, and D. Atienzaz, “Resource management for power-constrained HEVC transcoding using reinforcement learning,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 12, pp. 2834–2850, Dec. 2020.
[33] 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*, May 2018, pp. 1–4.
[34] 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.
[35] 2021. [Online]. Available: http://www.spec.org/power_ssj2008/results
[36] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. Hoboken, NJ, USA: Wiley, 1990.
[37] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 1989.
[38] 2018. [Online]. Available: https://netflixtechblog.com/vmaf-the-journey-continues-44b51ee9ed12/
[39] R. Rassool, “VMAF reproducibility: Validating a perceptual practical video quality metric,” in *Proc. IEEE Int. Symp. Broadband Multimedia Syst. Broadcast.*, Jun. 2017, pp. 1–2.
[40] Y. Qin *et al.*, “Quality-aware strategies for optimizing ABR video streaming QoE and reducing data usage,” in *Proc. ACM Multimedia Syst. Conf.*, Jun. 2019, 189–200.
[41] C. Zhang and J. Liu, “On crowdsourced interactive live streaming: A Twitch.tv-based measurement study,” in *Proc. ACM Workshop Netw. Oper. Syst. Support Digit. Audio Video*, Mar. 2015, pp. 55–60.
[42] 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.
[43] B. Shirazi, A. Hurson, and K. Kavi, *Scheduling and Load Balancing in Parallel and Distributed Systems*. Washington, DC, USA: IEEE CS Press, 1995.
[44] J. Guo, L. Bhuyan, R. Kumar, and S. Basu, “QoS aware job scheduling in a cluster-based web server for multimedia applications,” in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, Apr. 2005, pp. 1–10.
[45] 2016. [Online]. Available: https://support.google.com/youtube/answer/2853702
[46] 2015. [Online]. Available: https://netflixtechblog.com/per-title-encode-optimization-7e99442b62a2
[47] 2018. [Online]. Available: https://support.video.ibm.com/hc/en-us/articles/207852117-internet-connection-and-recommended-encoding-settings
[48] D. Stohr, A. Frömmgen, A. Rizk, M. Zink, and R. Steinmetz, “Where are the sweet spots? A systematic approach to reproducible DASH player comparisons,” in *Proc. ACM Int. Conf. Multimedia*, Oct. 2017, pp. 1113–1121.
[49] A. Costa *et al.*, “QoE-based packet dropper controllers for multimedia streaming in WiMAX networks,” in *Proc. IFIP/ACM Latin Amer. Netw. Conf.*, Oct. 2011, pp. 12–19.

Dayoung Lee (Member, IEEE) received the B.S. and M.S. degrees in computer engineering in 2016 and 2018, respectively, from Inha University, Incheon, South Korea, where she is currently working toward the Ph.D. degree in computer science with the Department of Computer Engineering.

Her current research interests include embedded software and multimedia systems.

Minseok Song (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer engineering from Seoul National University, Seoul, 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, multimedia, and Internet of Things systems.

Authorized licensed use limited to: Inha University. Downloaded on September 21,2022 at 01:41:37 UTC from IEEE Xplore. Restrictions apply.