Quality-Oriented Task Allocation and Scheduling in Transcoding Servers With Heterogeneous Processors

Dayoung Lee and Minseok Song, *Member, IEEE*

***Abstract*— Dynamically adaptive streaming over HTTP requires a large-scale server to transcode various bitrate versions in which different preset parameters can be used to provide different video qualities at each resolution. When transcoding servers contain a heterogeneous mix of CPUs and GPUs, the task scheduler must choose a processor and preset parameter for each transcoding task to meet the transcoding deadlines while achieving the best possible video quality. We apply regression analysis to sample variable-bit-rate videos to provide accurate (mean absolute percentage error values from 1.3% to 13.9%) model for predicting bitrate, transcoding time and video quality at each resolution on different processors. We build this into a greedy allocation and scheduling algorithm which first satisfies deadlines with low video quality, and then redistributes the workload to improve that quality while continuing to meet the deadlines. This scheme was both simulated and implemented on a testbed server. It satisfies all deadlines while outperforming standard algorithms by between 3.12% and 15.59% in terms of popularity-weighted video quality divided by bitrate.**

***Index Terms*— Multimedia systems, transcoding, Scheduling algorithms.**

I. Introduction

OVER-THE-TOP (OTT) streaming, which provides video services over the Internet, is being increasingly used to support various video-based applications, including Twitch, TV Everywhere, N-Screen, and those with user-created contents. OTT streaming naturally increases the amount of Internet video traffic, requiring a significant server infrastructure [1]. For example, YouTube serves an estimated 40 million videos every day, amounting to 200 TB of data [2].

The dynamically adaptive streaming over HTTP (DASH) technique has now become the defacto standard for OTT streaming as it has been adopted by numerous service providers, including Hulu, YouTube, and Netflix [1], [3]–[5]. In DASH, each video is divided into segments that are further encoded into versions at a number of bitrates; this allows media players on the client side to choose a bitrate to match the current device and its network conditions to improve quality-of-experience (QoE).

Transcoding is used to produce versions with different bitrates and resolutions as required by DASH [1], [3], [4]. It has been reported [1] that a single video clip could undergo 120 transcoding operations for Netflix streaming. Transcoding is computationally intensive, and server scalability is negatively affected by its complexity [6]. This problem is compounded if processors are added incrementally, resulting in a server with heterogeneous central processing units (CPUs) and graphics processing units (GPUs). Various processing units typically have different characteristics in terms of the speed and cost of transcoding videos with varying bitrates; this must be reflected in the assignment of the transcoding tasks.

Recently, the variable-bit-rate (VBR) technique has been adopted by streaming companies to reduce bandwidth requirements while providing a similar user experience to constant-bit-rate (CBR) video [7]–[10]. For example, streaming network traffic analysis has demonstrated that Netflix uses VBR streaming significantly [8]. Because VBR has a higher compression rate than CBR, allowing higher video quality per bitrate, it is expected to be commonly used in DASH streaming [7]. Several methods for selecting the bitrate of each video segment in DASH have been proposed [7]; however, none of these techniques can manage VBR video clips in transcoding servers.

Typically, the tradeoff between the transcoding time, video quality, and compression ratio is determined by preset parameters [11]–[13]. To provide superior video quality using the same bitrate, preset parameters with high quality must be selected; however, this requires more transcoding time. In addition, a deadline constraint is often associated with a transcoding task [14]. For example, tasks could require virtually immediate transcoding to provide rapid streaming after a video is uploaded or to achieve a service level agreement. Thus, transcoding servers require a scheduling algorithm.

We present a new scheme to address VBR processing, task allocation on heterogeneous processors, preset parameter selection, and deadline constraints. We first examine the characteristics of clips from VBR-encoded segments and construct multiple regression models to predict the bitrates, transcoding times, and video qualities. Based on this, we formulate an optimization problem that selects a preset parameter and processor node for each task to maximize the overall video

Manuscript received December 1, 2020; revised February 25, 2021; accepted April 9, 2021. Date of publication April 20, 2021; date of current version March 9, 2022. 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 and ICT under Grant 2017M3C4A7080248 and in part by the Basic Science Research Program through the NRF funded by the Ministry of Education under Grant NRF-2020R1F1A1068984. This article was recommended by Associate Editor C. Wu. *(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).

Color versions of one or more figures in this article are available at https:doi.org/10.1109/TCSVT.2021.3074158.

Digital Object Identifier 10.1109/TCSVT.2021.3074158

quality divided by the bitrate in streaming while achieving all task deadlines. We propose an algorithm that gradually redistributes workloads by prioritizing selections that produce higher video quality for segments with close deadlines while maximizing the overall video quality divided by bitrate and completing tasks with close deadlines first. This scheme has been implemented and evaluated on a testbed server.

The main contributions of this paper are summarized as follows:

1) We propose a new regression method to predict video quality index, bitrate and transcoding times specifically for VBR-encoded video, which allows an optimization model to be formulated by characterizing transcoding workloads in terms of deadlines, computation times and video quality.
2) We propose a new task scheduling and allocation algorithm that chooses processor type and preset parameter for each task with the aim of maximizing overall popularity-weighted video quality divided by bitrate while meeting the deadlines of all the tasks.
3) We have implemented the proposed scheme in a prototype server. For this purpose, we developed a communication protocol between processor nodes, a scheduler which ensures the deadlines and a queue management scheme which handles multiple transcoding tasks effectively.
4) We evaluate our scheme in terms of popularity-weighted video quality, deadline miss ratio, regression accuracy and algorithm complexity through simulations as well as real measurements on a testbed.

The remainder of this paper is organized as follows. We review related work in Section II and introduce a system model in Section III. We then present regression results in Section IV, a transcoding task allocation algorithm in Section V and implementation details in Section VI. Finally, we assess the effectiveness of these schemes in Section VII and draw conclusions in Section VIII. The list of abbreviations used in this paper are summarized in Table I.

TABLE I
ABBREVIATIONS USED IN THIS PAPER

Abbreviation	Meaning
OTT	over-the-top
DASH	dynamically adaptive streaming over HTTP
QoE	quality-of-experience
QoS	quality-of-service
VBR	variable-bit-rate
CBR	constant-bit-rate
CDN	content distribution network
bpp	bit-per-pixel
HQ	high-quality (preset parameter)
HP	high-performance (preset parameter)
LLHQ	low-latency high-quality (preset parameter)
LLHP	low-latency high-performance (preset parameter)
VQ/B	video quality divided by bitrate
EDF	earliest-deadline-first
SSIM	similarity index measure
VMAF	video multi-method assessment fusion
MAPE	mean absolute percentage error
TSA	transcoding scheduling and allocation
HRP	high-resolution popular
MRP	medium-resolution popular
LRP	low-resolution popular
RR	round-robin
WF	worst-fit
BQ	best quality
LT	lowest transcoding time
DP	default preset
MCMC	Markov chain Monte Carlo
GA	genetic algorithm

II. RELATED WORK

Workload allocation and scheduling are important in servers that run computationally intensive video transcoding tasks. For example, Gao *et al.* [15] introduced a cloud transcoding system to minimize cost using a stochastic approach where cloud resources are scheduled dynamically under time-varying workloads. Song *et al.* [14] proposed an algorithm that allocates a CPU frequency and a processor to each transcoding task to minimize the power consumption while meeting all transcoding deadlines. Huang *et al.* [16] presented a technique that assigns transcoding task priorities by considering the transcoding complexity. Based on this, a task allocation scheme was developed to balance the workload among the servers. Ma *et al.* [17] proposed a task allocation scheme for multiple servers in which the transcoding servers are selected to minimize response times using transcoding times that are estimated based on moving averaging techniques. However, all of these schemes operate on a homogeneous CPU system basis.

Several schemes have been developed to address quality-of-service (QoS) issues in transcoding servers. Gao *et al.* [18] presented a priority-based provisioning scheme for computing resources to satisfy the heterogeneous QoS requirement of each transcoding task. Li *et al.* [19] presented a self-configurable transcoder for heterogeneous cloud platforms in which transcoding tasks are dynamically mapped to virtual machines (VMs) by considering their affinity and QoS requirements. Ma *et al.* [20] presented a priority-based scheduling algorithm that assigns transcoding jobs to processors based on observed workloads to maximize transcoding throughput. Lee *et al.* [21] proposed two algorithms to maximize QoE subject to a power limit. However, none of these authors considered the selection of transcoding preset parameters and VBR issues.

Several transcoding systems have been proposed for cloud environments to reduce cloud cost while improving QoS for DASH-based video streaming. For example, Panarello *et al.* [22] presented a new cloud federation system that allows cooperation among multiple clouds to accelerate the entire video processing service. Wang *et al.* [23] considered transcoding task placement across servers in a content distribution network (CDN) by considering user preferences for the CDN regions and the regional preferences of the video versions. Zheng *et al.* [4] proposed a transcoding scheme specifically for a live streaming system to minimize the operating costs of service providers by allocating a data center to each viewer while determining the transcodeable versions for each channel. Zhao *et al.* [24] proposed an algorithm to schedule transcoding tasks in a cloud environment composed of heterogeneous clusters to balance the workload of each

cluster by utilizing data locality. Wei *et al.* [25] proposed a cloud-based video transcoding system to minimize the number of CPU cores used and latency by adjusting the transcoding time and video chunk size based on preset selection. However, none of these works were based on a heterogeneous mix of CPUs and GPUs.

Online transcoding delays the execution of the transcoding operations until a video is requested. Krishnappa *et al.* [1] introduced a workload prediction model for online transcoding that preserves QoE by transcoding the prefixes of the video in advance. Zhao *et al.* [26] addressed the tradeoff between storage and computation to minimize the cost of obtaining these resources from the cloud. Jokhio *et al.* [27] also addressed this tradeoff, considering the video length and frequency of transcoding operations. Baccoura *et al.* [28] proposed a scheme that combines caching of highly popular video chunks with the online transcoding of uncached low-bitrate versions at mobile edge servers to improve edge storage utilization. Lee *et al.* [29] utilized online transcoding techniques by delaying unpopular transcoding tasks to limit the amount of power consumed for offline transcoding. Zhang *et al.* [30] combined online with offline transcoding to reduce the storage overhead required to store transcodeable versions on SSD. This scheme stores footprint information used as a hint to reduce online transcoding delays. All of these studies focused primarily on the tradeoff between storage and computation costs in online transcoding schemes.

A transcoding server can consist of heterogeneous nodes with different processor types. Based on this, Liu *et al.* [12] examined the tradeoff between the transcoding speed and video quality for different types of processors and preset parameters. Li *et al.* [31] presented a task scheduling scheme that maps transcoding tasks to heterogeneous VMs by considering the affinity of the transcoding tasks. Chang *et al.* [32] developed a heterogeneous transcoding platform; here, CPUs and GPUs are mixed and distributed by dividing the real-time captured video into multiple chunks and assigning them to each processor to reduce the CPU load on the server. Katsak *et al.* [33] developed a framework for scheduling on a single server consisting of one CPU and multiple GPUs, each of which uses a different encoder. They considered interference from simultaneous execution and predicted the CBR encoding time of each encoder using Gaussian process regression technology. However, task assignment issues to improve video quality were not considered in these studies.

Several authors have addressed VBR processing in DASH streaming. For example, Chang *et al.* [34] proposed a playback scheduling algorithm that allows escalation of the selected resolution level to improve QoE at the client side while effectively using the available network bandwidth. Yu *et al.* [9] proposed a probabilistic bandwidth prediction model in which accurate bitrate information for each segment can be delivered to the client device such that a higher bitrate can be selected under the current network conditions. Qin *et al.* [7] proposed a bitrate adaptation scheme in which a control-theoretic feedback mechanism is used to predict the VBR bitrate accurately. They also designed several principles to improve QoE based on the analysis of VBR-encoded video. Zhang *et al.* [35] quantitatively analyzed the effect of chunk size changes on QoE when each chuck is encoded using VBR techniques. However, all of these schemes address bitrate adaptation issues on the client perspective rather than on the transcoding server perspective.

To the best of our knowledge, this paper is the first attempt to address task assignments on transcoding servers with heterogeneous processor types by taking account of deadlines while choosing transcoding preset parameters for VBR-encoded videos.

III. System Architecture and Basic Idea

A. System Architecture

The two-tier transcoding server presented in Fig. 1 handles transcoding task requests, each of which has its own deadline. It has frontend and backend nodes, and a single frontend node assigns each transcoding task to the most appropriate backend node. The backend nodes, which can be GPUs or CPUs, perform the actual transcoding operations.

The majority of transcoders have preset parameters that determine the transcoding speed, video quality, and compression ratio. For example, an H.264 codec has nine preset parameters, namely, ultrafast, superfast, veryfast, faster, fast, medium, slow, slower, and veryslow [12]. The h264_cuvid and h264_nvenc codecs used for GPU transcoding have four parameters: high-quality (HQ), high-performance (HP), low-latency HQ (LLHQ), and low-latency HP (LLHP) [36].

A video is divided into segments, each of which is further transcoded into multiple versions with different resolution, video quality, and target bitrate [7]. Using the VBR technique, the actual bitrate can be slightly different for each segment. Each preset parameter also has a different compression ratio, which corresponds to its target bitrate. Therefore, an accurate VBR model is important for the efficient distribution of transcoding tasks across the backend nodes.

B. Basic Idea

We use the popularity-weighted VQ/B as the quantity to be maximized in VBR streaming [7]. Thus, we aim to select the preset parameter and processor node for each transcoding task that maximizes this metric, averaged over all segments, while achieving all transcoding deadlines. This raises the following four issues.

1) Modeling bitrate, video quality, and transcoding time: Optimization requires these data for each video segment; however, these data are not known prior to segment transcoding. In addition, the effects of a codec's preset parameters are not constant across the range of CPUs and GPUs that can be found on a server. Therefore, we measured video quality indices, transcoding times, and bitrates for 3000 segments from 162 different video clips [37], and then we constructed multiple regression models to predict the effect of the preset parameter.
2) Satisfying transcoding deadlines: Each backend node has a queue that contains a list of the tasks allocated to that processor. Earliest-deadline-first (EDF) scheduling is applied to each queue to satisfy the deadlines of the

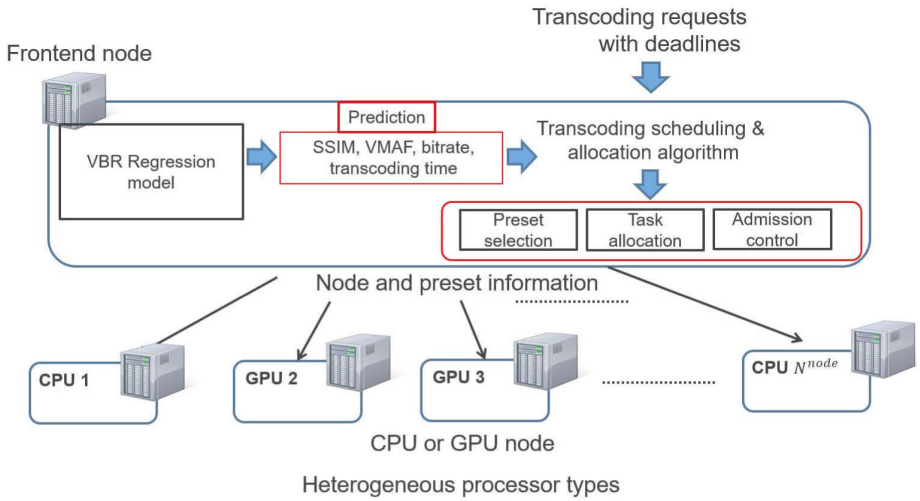

Fig. 1. System architecture.

maximum number of tasks. Thus, each queue is sorted in order of deadline, and the task with the earliest deadline is executed first. This technique provides a schedulability condition to achieve the transcoding deadlines.

3) Algorithm development: We demonstrate that the optimization problem is NP-hard and thus propose a heuristic algorithm. The algorithm assigns a utility to each selection of the backend node and preset parameter and then gradually redistributes workloads to maximize the popularity-weighted VQ/B.
4) Implementation: In the two-tier transcoding server shown in Fig. 1, issues such as task assignment and scheduling, communication between frontend and backend nodes, task queue management, and time complexity processing must be addressed to implement the proposed technique. All these issues are addressed in our implementation.

IV. Problem Formulation

A. Prediction of Bitrate, Transcoding Time, and Video Quality

For the efficient distribution of transcoding tasks across the backend nodes, we need all the values of the video quality index, the bitrate and the transcoding time for task τ_i executed by node j using the preset parameter k, but only those values for the original 1080p versions are available before transcoding. Thus those values for other resolutions need to be predicted. Compared to the prediction for CBR videos, VBR requires bitrate prediction to allow for bitrate variability. Previous studies on prediction methods have suggested a simple relationship between the number of frames and transcoding time [19] or between segment length and transcoding time [20]. However, such approaches cannot be used for the prediction of VBR videos owing to the dependency between bitrate and video quality or transcoding times.

A server processes N^{task} tasks, each of which involves the transcoding of a single video segment. Table II summarizes important symbols used in this paper. When new tasks arrive, the value of N^{task} is updated by adding the number of new requests to the number of existing requests in the queue. The ith task, τ_i, has an absolute deadline where the relative deadline for task τ_i, D_i, can be calculated by subtracting the current time from the absolute deadline.

A server comprises N^{node} heterogeneous processor nodes, each of which is either a CPU or GPU. N^{CPU} and N^{GPU} are the number of CPUs and GPUs, respectively, and $N^{\text{node}} = N^{\text{CPU}} + N^{\text{GPU}}$. N_j^{para} is the number of preset parameters for node j, $(j = 1, \ldots, N^{\text{node}})$.

Let $Q_{i,j,k}$ be the video quality index for the segment transcoded by task τ_i at node j when the preset parameter k is selected. We assume that the preset parameters are sorted in ascending order of the video quality produced; thus, $Q_{i,j,N_j^{\text{para}}}$ is the best obtainable video quality. $B_{i,j,k}$ is the bitrate of the transcoded version of a video segment when task τ_i is executed at node j using the preset parameter k $(i = 1, \ldots, N^{\text{task}})$, $(j = 1, \ldots, N^{\text{node}})$, and $(k = 1, \ldots, N_j^{\text{para}})$. $T_{i,j,k}$ is the transcoding time for task τ_i executed by node j using the preset parameter k.

To predict the values of $Q_{i,j,k}$, $B_{i,j,k}$, and $T_{i,j,k}$ based on multiple regression models, we downloaded 162 video trailers of different genres [37] and encoded 1080p video clips using a VBR technique by setting the frame rate to 30 fps and the bit-per-pixel (bpp) to 0.15. To establish regression, we obtained bitrates, transcoding times, and video quality indices for 3000 6 s video segments with typical transcoded resolutions of 240p, 360p, 480p, and 720p, using an FFmpeg tool [36]. We performed measurements with seven types of processors: an Intel i7-9700 CPU, NVIDIA RTX 2070 GPU, Intel i7-8700 CPU, NVIDIA GTX 1060 GPU, Intel i5-7500 CPU, NVIDIA GT 720 GPU, and Intel i7-4790 CPU. Because nine preset parameters are available for the CPUs and four for the GPUs, 192 different transcoding operations were used to produce four transcoded versions, with a total of 576,000 transcoding operations. We applied multiple regression techniques [45] to determine the bitrates, video quality indices, and transcoding

TABLE II
IMPORTANT SYMBOLS USED IN THIS PAPER

Symbol	Meaning
N^{task}	Number of total tasks
N^{node}	Number of processor nodes
N_j^{para}	Number of preset parameters for node j
τ_i	ith task
N_t	Number of total transcoded versions
$Q_{i,j,k}^{\text{up}}$	Video quality index at resolution $l+1$ for segment transcoded by τ_i at node j with kth preset parameter
$B_{i,j,k}^{\text{up}}$	Bitrate at resolution $l+1$ for segment transcoded by τ_i at node j with kth preset parameter
H_i	Bitrate at the highest resolution for segment transcoded by τ_i
$\alpha_{j,k}^{1,\ldots,N_t+2}$	Regression parameters for bitrate at node j and kth preset parameter
$\beta_{j,k}^{1,\ldots,N_t+4}$	Regression parameters for video quality index at node j and kth preset parameter
$\gamma_{j,k}^{1,\ldots,N_t+4}$	Regression parameters for transcoding time at node j and kth preset parameter
p_i	Access probability for video segment transcoded by τ_i
$Q_{i,j,k}$	Video quality index for segment transcoded by τ_i at node j and kth preset parameter
$B_{i,j,k}$	Bitrate for segment transcoded by τ_i at node j and kth preset parameter
$G_{i,j,k}$	Popularity-weighted VQ/B for segment transcoded by τ_i at node j and kth preset parameter
$T_{i,j,k}$	Transcoding time for segment transcoded by τ_i at node j and kth preset parameter
D_i	Relative deadline for task τ_i
X_i	Processor node index allocated for task τ_i
Y_i	Preset parameter index selected for task τ_i
A_j	Array for tasks allocated to processor node j
$L_{i,j}$	Tasks in subset of A_j which have deadlines earlier than that of task τ_i
$T_{i,j,k}^{\text{slack}}$	$D_i - T_{i,j,k}$
$U_{i,j,k}$	Utility value for segment transcoded by τ_i at node j and kth preset parameter
X_i^{first}, Y_i^{first}	Initialization values for X_i and Y_i
S	List of $\frac{G_{i,j,1}}{T_{i,j,1}^{\text{slack}}}$'s
X_i^{temp},Y_i^{temp}	Temporary variables used for task τ_i
A	List of $U_{i,j,k}$'s

times for each preset parameter and each type of processor using the following steps.

1) Predicting bitrates and video quality indices: The bitrates and video quality indices of the four transcoded versions of each video segment were obtained, starting from the original segment and proceeding incrementally to lower resolutions. We used log-log regression to determine the bitrates and linear-log regression to determine the video quality index [45].
2) Predicting CPU transcoding times: Using the bitrate and video quality indices for each resolution obtained in the previous step, we used linear-log regression to determine the CPU transcoding times.
3) Predicting GPU transcoding times: Regression analysis suggested that the relationship between resolution, bitrate, and GPU transcoding time was weakly correlated indicating the unpredictability of the GPU transcoding time. For example, the adjusted R^2 for the GPU transcoding time was found to be weak (an average value of 0.229, which ranged between 0.037 and 0.61). Thus, we used the average transcoding time on each GPU for the optimization parameter;.

Let N_t be the number of transcoded versions. To simplify our presentation, we introduced a dummy variable V_l, ($l = 1, \ldots, N_t$), which selects one of the N_t transcoded resolutions, as follows:

$$V_l = \begin{cases} 1 & \text{When resolution } l \text{ is selected} \\ 0 & \text{Otherwise.} \end{cases} \quad (1)$$

If the resolution of the original video was 1080p and the transcoded resolutions were 240p, 360p, 480p, and 720p, then, N_t was four, and (V_1, V_2, V_3, V_4) = (0, 0, 0, 1) represented 720p.

Different bitrates were produced by different processors and preset parameters. We observed that the bitrate of a video of resolution l can be estimated from the bitrate and quality index of that video at resolution $l+1$. Based on this, using the bitrate and the video quality index for the original resolution (i.e., 1080p), which were available, those for the lower resolution (720p) were derived using regression equations. Then, these predicted values were used to predict values for the next lower resolution (480p). These steps were repeated until the values for the lowest resolution were predicted.

If l is the resolution of a video segment transcoded by τ_i at node j with preset parameter k, then $B_{i,j,k}^{\text{up}}$ with $Q_{i,j,k}^{\text{up}}$ are the bitrate and video quality index of that video segment at resolution $l+1$, respectively.

The bitrate of the segment transcoded by task τ_i at node j with preset parameters k and $B_{i,j,k}$ can be estimated as follows:

$$B_{i,j,k} = \exp\left(\alpha_{j,k}^1 + \alpha_{j,k}^2 \ln B_{i,j,k}^{\text{up}} + \alpha_{j,k}^3 V_1 + \ldots + \alpha_{j,k}^{N_t+2} V_{N_t}\right), \quad (2)$$

where $\alpha_{j,k}^1,\ldots,\alpha_{j,k}^{N_t+2}$ are regression parameters for node j and preset parameter k. For example, when the "medium" preset parameter is selected for the Intel i7-9700 CPU: $B_{i,j,k} = \exp(-0.281 + 0.92 \ln B_{i,j,k}^{\text{up}} + 0\ V_1 + 0.282\ V_2 + 0.11\ V_3 + 0.185\ V_4)$; if the transcoded resolution is 240p, then $B_{i,j,k}$ is $\exp(-0.281 + 0.92 \ln B_{i,j,k}^{\text{up}})$.

The video quality index for the version transcoded by task τ_i can be predicted as follows:

$$Q_{i,j,k} = \beta_{j,k}^1 + \beta_{j,k}^2 Q_{i,j,k}^{\text{up}} + \beta_{j,k}^3 \ln B_{i,j,k}^{\text{up}} + \beta_{j,k}^4 \ln B_{i,j,k} + \beta_{j,k}^5 V_1 + \ldots + \beta_{j,k}^{N_t+4} V_{N_t}, \quad (3)$$

where $\beta_{j,k}^1,\ldots,\beta_{j,k}^{N_t+4}$ are the regression parameters for the video quality index values for node j and preset parameter k.

H_i is the value of the bitrate (kbps) for the highest resolution used for the transcoding of the video segment requested by task τ_i. The transcoding time on a CPU can be predicted by logarithmic multiple regression as follows:

$$T_{i,j,k} = \gamma_{j,k}^1 + \gamma_{j,k}^2 \ln H_i + \gamma_{j,k}^3 \ln B_{i,j,k} + \gamma_{j,k}^4 Q_{i,j,k} + \gamma_{j,k}^5 V_1 + \ldots + \gamma_{j,k}^{N_t+4} V_{N_t}, \quad (4)$$

TABLE III
VALUES OF ADJUSTED R^2 VALUES AT EACH PRESET PARAMETER

Processor	Parameter	Preset No.								
		1	2	3	4	5	6	7	8	9
i7-9700	Bitrate	0.992	0.994	0.993	0.993	0.994	0.993	0.993	0.993	0.993
	SSIM	0.934	0.941	0.94	0.942	0.942	0.94	0.939	0.935	0.938
	VMAF	0.932	0.906	0.896	0.894	0.896	0.895	0.893	0.888	0.892
	Transcoding time	0.932	0.952	0.953	0.954	0.957	0.953	0.959	0.943	0.923
RTX 2070	Bitrate	0.991	0.991	0.991	0.992	-	-	-	-	-
	SSIM	0.931	0.931	0.931	0.931	-	-	-	-	-
	VMAF	0.845	0.84	0.834	0.836	-	-	-	-	-
i7-8700	Bitrate	0.996	0.998	0.998	0.998	0.998	0.998	0.998	0.998	0.998
	SSIM	0.933	0.942	0.941	0.943	0.942	0.94	0.94	0.939	0.938
	VMAF	0.932	0.907	0.896	0.894	0.896	0.894	0.893	0.891	0.892
	Transcoding time	0.933	0.932	0.936	0.941	0.946	0.944	0.951	0.936	0.912
GTX 1060	Bitrate	0.99	0.991	0.99	0.99	-	-	-	-	-
	SSIM	0.93	0.929	0.93	0.93	-	-	-	-	-
	VMAF	0.847	0.857	0.843	0.841	-	-	-	-	-
i5-7500	Bitrate	0.992	0.994	0.993	0.993	0.994	0.994	0.993	0.993	0.993
	SSIM	0.935	0.941	0.941	0.942	0.942	0.94	0.94	0.939	0.938
	VMAF	0.934	0.907	0.896	0.894	0.896	0.895	0.893	0.892	0.892
	Transcoding time	0.8	0.918	0.931	0.942	0.949	0.945	0.902	0.937	0.925
GT 720	Bitrate	0.98	0.985	0.983	0.982	-	-	-	-	-
	SSIM	0.931	0.93	0.932	0.932	-	-	-	-	-
	VMAF	0.846	0.858	0.842	0.846	-	-	-	-	-
i7-4790	Bitrate	0.992	0.993	0.993	0.993	0.994	0.993	0.993	0.993	0.993
	SSIM	0.934	0.941	0.94	0.942	0.942	0.94	0.939	0.938	0.938
	VMAF	0.933	0.907	0.895	0.894	0.896	0.894	0.893	0.891	0.892
	Transcoding time	0.673	0.817	0.865	0.899	0.888	0.859	0.854	0.874	0.862

where $\gamma_{j,k}^{1},\ldots,\gamma_{j,k}^{N_t+4}$ are the regression parameters for node j and preset parameter k. Table III presents the adjusted coefficients of determination for these regression models to validate the efficacy of the regression models. This indicates that the relationship between the input and output of the regression model is strong.

B. Problem Formulation

The transcoding time, video quality and compression ratio can vary greatly depending on the preset parameters and processor type selected. Due to the computational limitations provided by the transcoding server, it is not possible to select high-quality preset parameters for all videos. Thus, we use the video popularity information to select high quality preset parameters for popular videos, but low quality parameters for unpopular ones, maximizing the overall viewers' QoE while making effective use of limited processing capacity.

The popularity of a video typically goes through three stages: a hot stage when it is first released, an intermediate warm stage, and a cold stage when the video is rarely viewed. Hence, we define a variable, p_i, as the access probability for a video segment transcoded by task τ_i, where $\sum_{i=1}^{N^{\text{task}}} p_i = 1$. Some works have been developed to predict video popularity, in which future popularity can be obtained through machine learning models based on existing access patterns [11], [38], [39]. For example, Facebook developed a scalable and accurate popularity prediction system based on neural network model trained with an existing access pattern [11]. We assume that the popularity of each video can be obtained in advance using these methods.

To determine the quality of a video, we use both structural similarity index measure (SSIM) and video multi-method assessment fusion (VMAF) indices, each of which has its advantages. The SSIM metric is known to provide an estimate of the exact perceptual quality of a video by effectively reproducing the human visual system [40], [41]; the VMAF metric developed by Netflix is known to outperform other popular video quality indexes in terms of prediction accuracy [42], [43]. We introduce a variable, $G_{i,j,k}$, to represent the popularity-weighted VQ/B when τ_i is executed at node j using a transcoding preset parameter k as follows:

$$G_{i,j,k} = \frac{Q_{i,j,k}\, p_i}{B_{i,j,k}}. \tag{5}$$

For each task τ_i, we must select a computing processor node, X_i, $(X_i = 1, \ldots, N^{\text{node}})$, and transcoding preset parameter Y_i, $(Y_i = 1, \ldots, N_{X_i}^{\text{para}})$. When new tasks arrive so that the EDF order of the task set containing both new and existing requests is changed, the algorithm needs to be run again to reflect the changes in the task set. EDF scheduling is then applied to the tasks at each node such that all the tasks in set A_j, which are allocated to processor node j, are sorted by deadline in ascending order. The tasks in set $L_{i,j}$, which is a subset of A_j, have deadlines earlier than that of task τ_i. Then, by Jackson's theorem [44], the tasks at node j satisfy their deadlines if

$$\forall\ \tau_i \in A_j, \sum_{\forall \tau_m\ \in\ L_{i,j}} T_{m,X_m,Y_m} + \epsilon \le D_m, \tag{6}$$

where ϵ is additional time for when the actual transcoding time is greater than the predicted time.

We define the transcoding task allocation problem ($\mathcal{TTAP}$) as that of selecting values of X_i and Y_i that maximize the popularity-weighted VQ/B while satisfying all the task

deadlines:

$$\text{Maximize} \sum_{i=1}^{N^{\text{task}}} G_{i,X_i,Y_i}$$

Subject tosatisfying Inequality (6) for all j $(j=1,\ldots,N^{\text{node}})$.

V. Algorithm

A. Algorithm Design

Suppose that a number of groups of items are available, each with a benefit and weight, and we must select one item from each group to maximize the total benefit, subject to a limit on the weight in each knapsack. This is a multi-dimensional multiple-choice knapsack (MMKP) problem, which has been proved to be NP-hard [46].

A transcoding server is faced with a number of tasks, each of which can be run on one of a finite number of combinations of processor and preset parameter, where each of these combinations is associated with a value of $G_{i,j,k}$ and a predicted transcoding time. Solving $\mathcal{TTAP}$ allocates tasks to the processors to maximize the sum of $G_{i,j,k}$ while satisfying transcoding time constraint (Inequality (6)) on each processor. In this case, the number of possibilities makes optimization by enumeration infeasible. We have therefore designed a heuristic.

The proposed heuristic algorithm has two phases: initialization and workload redistribution. In the initialization phase, all the values of X_i and Y_i are initialized to the processor node and preset parameter that produce the shortest transcoding time (X_i^{first} and Y_i^{first}). In the redistribution phase, these values are updated to maximize $\sum_{i=1}^{N^{\text{task}}} G_{i,X_i,Y_i}$ while satisfying Inequality (6).

We introduce the variable $T_{i,j,k}^{\text{slack}}$ to represent the slack time that remains before the deadline of task τ_i if it is executed at node j using the preset parameter k. $T_{i,j,k}^{\text{slack}}$ can be expressed as $D_i - T_{i,j,k}$. We then define utility $U_{i,j,k}$ for task τ_i, node j, and preset parameter k as follows:

$$\forall (j,k) \neq (X_i^{\text{first}}, Y_i^{\text{first}}), U_{i,j,k} = \frac{G_{i,j,k} - G_{i,X_i^{\text{first}},Y_i^{\text{first}}}}{T_{i,X_i^{\text{first}},Y_i^{\text{first}}}^{\text{slack}} - T_{i,j,k}^{\text{slack}}}, \quad (7)$$

where the numerator is the increase in the popularity-weighted VQ/B and the denominator is the reduction in slack time compared to the shortest transcoding time. The algorithm greedily redistributes the workload by selecting combinations of (j,k), which produces higher values of $U_{i,j,k}$ to increase the popularity-weighted VQ/B to a maximum (numerator) while decreasing the slack time to a minimum (denominator). This step is then repeated while Inequality (6) continues to be satisfied.

B. Algorithm Description

1) Initialization Phase: The values of Y_i^{first} are initialized to "1" (the first preset option). Then, the node values of X_i^{first} are initialized as follows:

1) The node indices of existing tasks are initialized to their previous X_i value. For every new task τ_i and each node j, $(j = 1,\ldots,N^{\text{node}})$, the value of $\frac{G_{i,j,1}}{T_{i,j,1}^{\text{slack}}}$ is calculated and placed into an ordered list S. Processor nodes with high values of $\frac{G_{i,j,1}}{T_{i,j,1}^{\text{slack}}}$ are examined first.
2) The indices h and n are introduced to refer to the task and node corresponding to the highest value of $\frac{G_{i,j,1}}{T_{i,j,1}^{\text{slack}}}$ in S if task h is inserted into the task queue of node n. The algorithm first checks whether Inequality (6) will remain satisfied. If yes, then X_h^{first} is initialized to n, and all the values $\frac{G_{h,j,1}}{T_{h,j,1}^{\text{slack}}}$, $(j = 1,\ldots,N^{\text{node}})$ for task h are removed from S; otherwise, only the value $\frac{G_{h,n,1}}{T_{h,n,1}^{\text{slack}}}$ is removed from S.
3) Step 2 is repeated until S is empty and all values of X_i^{first} have been determined. If the values of X_i^{first} have not been determined, then the deadline cannot be met, regardless whether the preset parameter with the lowest transcoding time is selected.

2) Workload Redistribution Phase: In this phase, tasks are redistributed, and the preset parameter of each task is changed to improve the popularity-weighted VQ/B while satisfying Inequality (6) at each node. This requires two temporary variables, X_i^{temp} and Y_i^{temp}, which are initialized to X_i^{first} and Y_i^{first}, respectively. A is a list of $U_{i,j,k}$ values, which are sorted in non-decreasing order. The highest value of $U_{i,j,k}$ in A guides updating the values of X_i^{temp} and Y_i^{temp}. Thus, we introduce I^{task}, I^{para}, and I^{node} to represent the task, transcoding parameter, and node indices of the highest value of $U_{i,j,k}$ in A. Then, the workload redistribution occurs as described in the following.

1) Elimination step: To ensure that the workload redistribution phase improves the popularity-weighted VQ/B, the following inequality must hold:

$$G_{I^{\text{task}},X_{I^{\text{task}}}^{\text{temp}},Y_{I^{\text{task}}}^{\text{temp}}} < G_{I^{\text{task}},I^{\text{node}},I^{\text{para}}}. \quad (8)$$

If Inequality (8) is not satisfied, then the value of $U_{I^{\text{task}},I^{\text{node}},I^{\text{para}}}$ is removed from A because it corresponds to a reduction in the popularity-weighted VQ/B and not the increase that we are seeking. In this case, the elimination step is repeated with the next value from A until values of I^{task}, I^{node}, and I^{para} that satisfy Inequality (8) can be determined.
2) Update step: The algorithm first checks whether changing the values from $X_{I^{\text{task}}}^{\text{temp}}$ to I^{node} and from $Y_{I^{\text{task}}}^{\text{temp}}$ to I^{para} will violate Inequality (6). If not, then $X_{I^{\text{task}}}^{\text{temp}}$ is replaced by I^{node} and $Y_{I^{\text{task}}}^{\text{temp}}$ by I^{para}. Next, $U_{I^{\text{task}},I^{\text{node}},I^{\text{para}}}$ is removed from A.
3) The above two steps are repeated until A is empty. Finally, the values of X_i and Y_i are replaced by those of X_i^{temp} and Y_i^{temp}, respectively.

The details of this transcoding scheduling and allocation (TSA) algorithm are presented in Fig. 2: initialization occupies lines 5–15, workload redistribution occupies lines 16–25, and finalization occupies lines 26–29.

Temporary variables: $\forall\ i,\ (i = 1, ..., N^{\text{task}})$, X_i^{temp} and Y_i^{temp};
List of all the values of $\frac{G_{i,j,1}}{T_{i,j,1}^{\text{slack}}}$ for new tasks: S;
List of all the values of $U_{i,j,k}$: A;
Variables for initialization: $\forall\ i,\ (i = 1, ..., N^{\text{task}})$, X_i^{first} and Y_i^{first};
$\forall\ i,\ (i = 1, ..., N^{\text{task}})$, $Y_i^{\text{first}} \leftarrow 1$, $Y_i^{\text{temp}} \leftarrow 1$;
For all existing tasks τ_i, X_i^{first} is initialized to its previous X_i value;
while $S \neq \phi$ **do**
    Find the highest value in S, where h and n are the task and node indices for this value;
    if Inequality (6) is satisfied **then**
        $X_h^{\text{first}} \leftarrow n$, $X_i^{\text{temp}} \leftarrow n$;
        All the values of $\frac{G_{h,j,1}}{T_{h,j,1}^{\text{slack}}}$, $(j = 1, ..., N^{\text{node}})$ for task h are removed from set S;
    else
        The value of $\frac{G_{h,n,1}}{T_{h,n,1}^{\text{slack}}}$ is removed from S;
    end if
end while
while $A \neq \phi$ **do**
    Choose the highest value of $U_{I^{\text{task}},I^{\text{node}},I^{\text{para}}}$ from A;
    if Inequality (8) is satisfied for $I^{\text{task}}, I^{\text{node}}, I^{\text{para}}$ **then**
        if Inequality (6) is satisfied for node I^{node} **then**
            $X_{I^{\text{task}}}^{\text{temp}} \leftarrow I^{\text{node}}$;
            $Y_{I^{\text{task}}}^{\text{temp}} \leftarrow I^{\text{para}}$;
        end if
    end if
    Removes $U_{I^{\text{task}},I^{\text{node}},I^{\text{para}}}$ from A
end while
for $i = 1$ to N^{task} **do**
    $X_i \leftarrow X_i^{\text{temp}}$;
    $Y_i \leftarrow Y_i^{\text{temp}}$;
end for

Fig. 2. Transcoding task scheduling and allocation (TSA) algorithm.

VI. Implementation

We implemented the proposed scheduling algorithm on a two-tier transcoding server, as displayed in Fig. 1. The program running in the frontend node, including the scheduling algorithm, is approximately 4300 lines of code; the backend node code is 500 lines, excluding FFmpeg [36].

For new tasks, the frontend node executes the TSA algorithm to select a backend node and preset parameter for each task. The results are communicated to the backend nodes using sockets and a MySQL database. The MySQL database contains N^{node} task queues for each backend node together with task information, including video, segment and version indices, preset parameters, arrival times, completion times, and deadlines. This database is shared by all backend nodes.

Each time the TSA algorithm is executed, the task queue is updated based on the EDF, and the backend nodes are notified through socket communication. The current time is periodically communicated between the frontend and backend nodes for synchronization. Admission control is also performed for new tasks to ensure that that all existing tasks completed transcoding prior to their deadlines. The initialization phase assigns the tasks using the preset parameter with the shortest transcoding time. Hence, if a task cannot satisfy Inequality (6) in the initialization phase, meaning the deadline cannot be satisfied under any condition, the new task is rejected.

The backend nodes perform the transcoding operations using FFmpeg. When a task is completed, it updates its queue with the completion time and executes the next task in the queue with the preset parameter produced by the TSA algorithm. Fig. 3 illustrates how our implementation works on the two-tier transcoding server in Fig. 1.

VII. Experimental Results

We used video trailers of different genres [37] to analyze the accuracy of the regression method. Subsequently, we simulated our scheduler to assess the effect of the rate at which transcoding tasks arrived and the number of nodes. Finally, we measured the popularity-weighted VQ/B and deadline miss ratios on a testbed.

A. Accuracy of Regression

We assessed the accuracy of the regression equations in subsection IV-A by measuring the SSIM and VMAF indices, transcoding times, and bitrates for 1000 video segments [37]; these were not used in the regression analysis. They were transcoded from 1080p to different resolution versions: 240p, 360p, 480p, and 720p. We then calculated the mean absolute percentage error (MAPE) between the actual bitrates, SSIM and VMAF indices, and transcoding times and predicted values. The results are presented in Table IV. MAPE is widely used to measure prediction accuracy and can be easily interpreted [47], [48]. A MAPE of less than 10% is considered highly accurate, whereas a MAPE of less than 20% is considered an acceptable prediction [47], [48]. Table IV suggests that our regression model is accurate.

B. Simulation

1) Simulation Setup: We simulated a transcoding server to examine the average values of the popularity-weighted VQ/B and deadline miss ratio for transcoding tasks created for 24 h as described in the following.

1) Modeling characteristics of the videos: Because the bitrate of VBR video follows the normal distribution [35], the bitrate of the 3,000 video segments that we used for regression analysis follows a normal distribution, with an average of 9,115 kbps and a standard deviation of 1,331 kbps. The bitrates of the simulated videos were selected from this distribution, and their lengths were evenly distributed between a minute and an hour. Each simulated video was divided into 6 s segments, which were required to be transcoded from 1080p to 720p, 480p, 360p, and 240p.
2) Popularity modeling: The Zipf distribution is typically used to model video [49]. We set the θ parameter to 0.271 to model the popularity of each video clip [49]. The popularity of the video segments also follows the Zipf distribution with θ = 0.2 because later segments

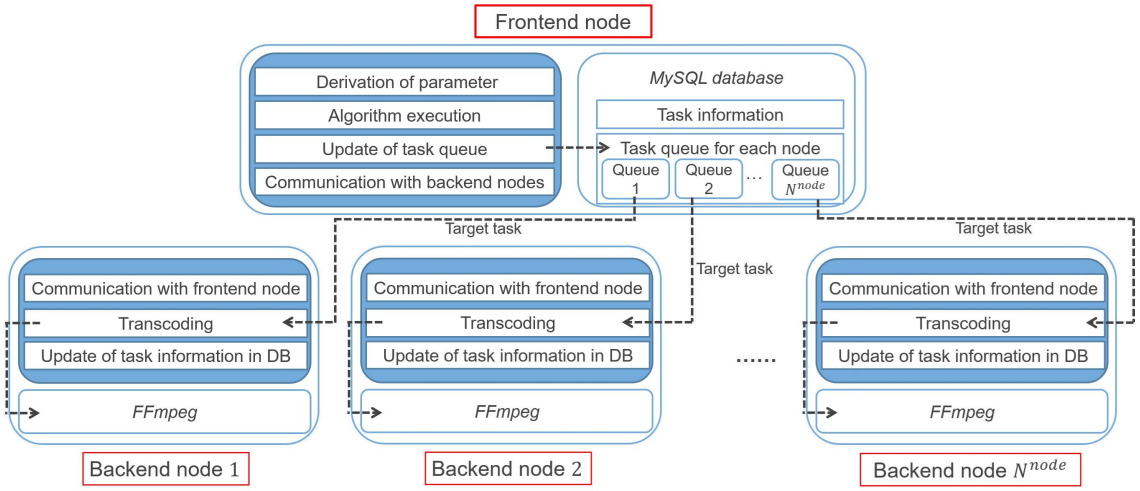

Fig. 3. Implementation flow on frontend and backend nodes.

TABLE IV
MAPE VALUES (%) FOR EACH PRESET PARAMETER

Processor	Parameter	Preset No.								
		1	2	3	4	5	6	7	8	9
i7-9700	Bitrate	6.594	4.246	5.447	5.427	6.735	5.316	5.45	5.393	5.601
	SSIM	2.016	1.475	1.399	1.366	1.347	1.342	1.336	1.316	1.312
	VMAF	8.455	4.435	4.416	3.822	3.913	3.958	4.019	3.956	4.03
	Transcoding time	3.549	3.819	4.463	5.392	6.267	6.140	6.500	9.628	12.163
RTX 2070	Bitrate	10.882	10.617	11.226	10.237	-	-	-	-	-
	SSIM	1.715	1.628	1.627	1.607	-	-	-	-	-
	VMAF	4.762	4.666	4.515	4.562	-	-	-	-	-
i7-8700	Bitrate	6.654	4.362	5.374	5.36	5.133	5.238	5.372	5.34	5.611
	SSIM	2.018	1.477	1.399	1.366	1.349	1.343	1.336	1.324	1.312
	VMAF	8.475	4.384	4.032	3.818	3.909	3.95	4.023	3.94	4.031
	Transcoding time	3.240	3.568	4.475	4.655	5.251	5.924	6.701	10.211	11.903
GTX 1060	Bitrate	10.899	6.905	7.715	7.038	-	-	-	-	-
	SSIM	1.692	1.708	1.64	1.626	-	-	-	-	-
	VMAF	4.766	5.183	4.685	4.667	-	-	-	-	-
i5-7500	Bitrate	6.507	4.259	7.229	5.205	6.979	5.107	5.222	5.178	5.372
	SSIM	2.013	1.475	1.396	1.362	1.35	1.339	1.333	1.311	1.307
	VMAF	8.369	4.431	4.009	3.802	3.891	3.993	4.023	3.901	3.991
	Transcoding time	5.309	5.330	6.450	7.254	7.648	7.816	12.031	11.715	13.906
GT 720	Bitrate	11.144	9.898	11.136	11.226	-	-	-	-	-
	SSIM	1.644	1.711	1.644	1.626	-	-	-	-	-
	VMAF	4.7	5.179	4.654	4.67	-	-	-	-	-
i7-4790	Bitrate	6.596	4.248	5.447	5.431	5.213	5.315	5.457	5.402	5.609
	SSIM	2.015	1.475	1.398	1.366	1.349	1.342	1.336	1.315	1.312
	VMAF	8.477	4.446	4.036	3.83	3.92	3.963	4.026	5.626	4.034
	Transcoding time	4.891	5.368	5.879	6.262	7.551	8.633	9.317	13.324	13.891

could possibly not be watched [50]. We considered three models for the relative popularity of the versions at each of the five resolutions. In the "low-resolution popular" (LRP) model, the 240p, 360p, 480p, 720p, and 1080p versions had relative popularities of 0.3, 0.3, 0.2, 0.1, and 0.1, respectively. When medium or high resolutions were popular (MRP or HRP models), the equivalent popularities were set to 0.1, 0.25, 0.3, 0.25, and 0.1, and 0.1, 0.1, 0.2, 0.3, and 0.3, respectively. Based on this popularity model, we could obtain the values of p_i. To evaluate the actual transcoding server performance, all values of the popularity-weighted VQ/B were calculated for only the transcoded versions (not the original version).

3) Transcoding server model: The scheduling algorithm executed once per second. The number of transcoding requests received during this period was modeled by a Poisson distribution with an average value of λ^{trans}. Unless otherwise stated, task execution deadlines were assumed to range between 30 and 60 s.

We modeled two methods of task allocation. First, task allocation can be easily implemented with round-robin (RR) allocation methods by allocating each task on a cyclic basis. We also simulated worst-fit (WF) allocation, where the sum of the deadlines of all the outstanding segments allocated to each processor was maintained; the next segment to arrive was allocated to the processor with the lowest sum.

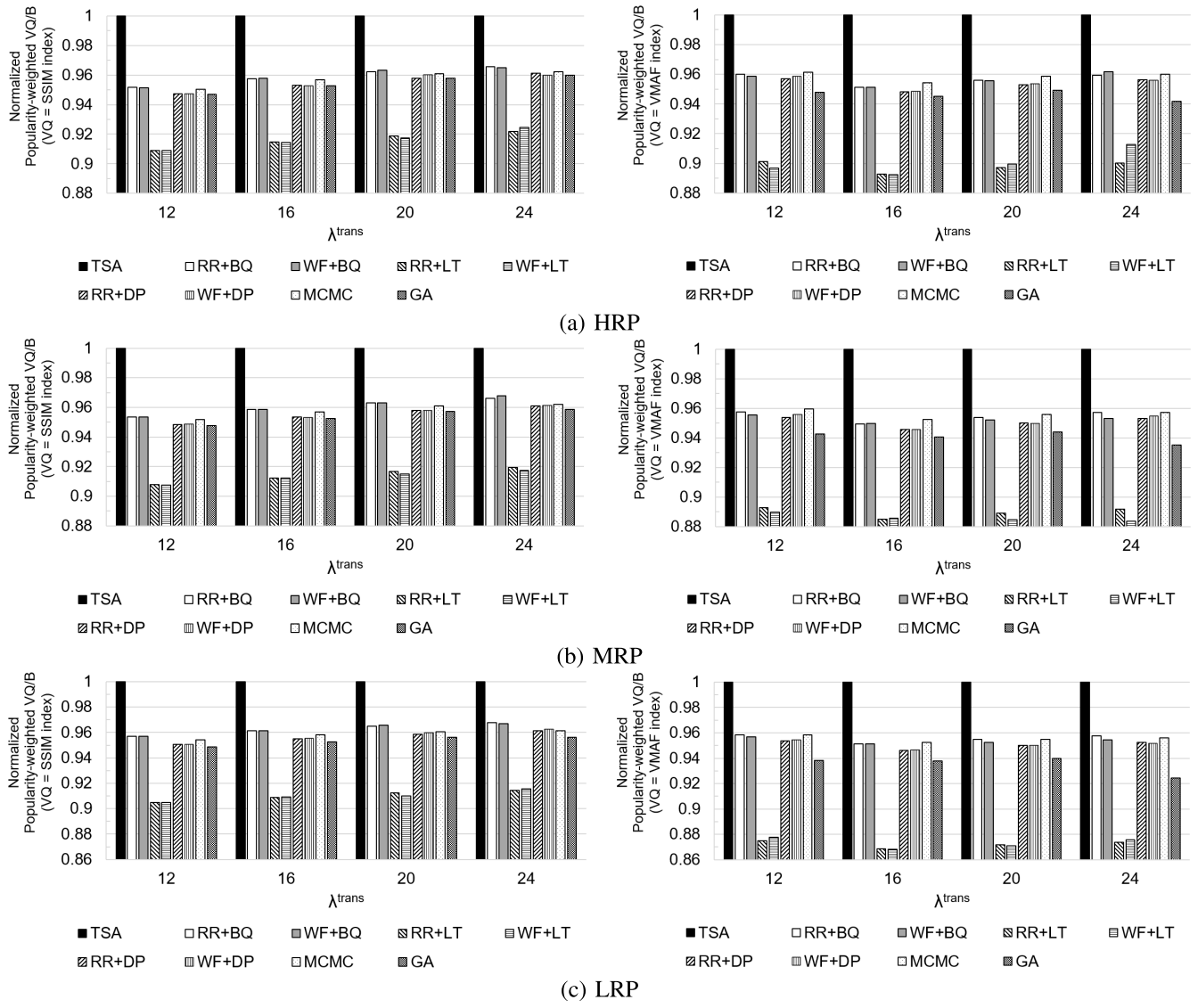

Fig. 4. Popularity-weighted VQ/B for seven task allocation and preset parameter selection schemes, normalized against results for TSA against average number of requests λ^{trans}.

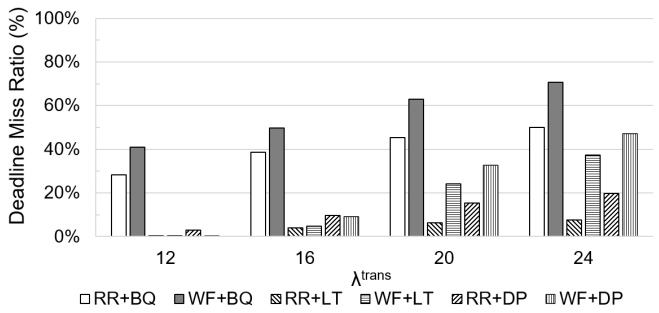

Fig. 5. Percentage (%) of missed deadlines against λ^{trans}.

We considered three schemes for initializing the preset parameters: best quality (BQ) used the "veryslow" parameter on the CPUs and HQ parameter on the GPUs; the lowest transcoding time (LT) uses the ultrafast parameter on the CPUs and HP parameter on the GPUs; the default preset (DP) uses the "medium" parameter on the CPUs and "HQ" parameter on the GPUs.

We implemented two additional methods for comparison: Markov chain Monte Carlo (MCMC) and genetic algorithm (GA), each of which can be used for knapsack problems effectively [51], [52]. These schemes are based on the proposed problem formulation model in that the deadline constraint (Inequality (1)) is checked when the values of X_i and Y_i are determined. They also require appropriate selection of parameters that greatly affect the execution time of the algorithm. For fair comparison, the parameters of these schemes were chosen to take a time comparable to the time it would take to run the TSA algorithm. In MCMC, the sampling count was set to 100,000; in GA, the population size and the number of generations were set to 300 and 20, respectively. Then, on average, it takes 26ms to run MCMC and 32ms to run GA in simulations. To implement GA, 10% of elite solutions are passed to the next generation, mutation probability was set

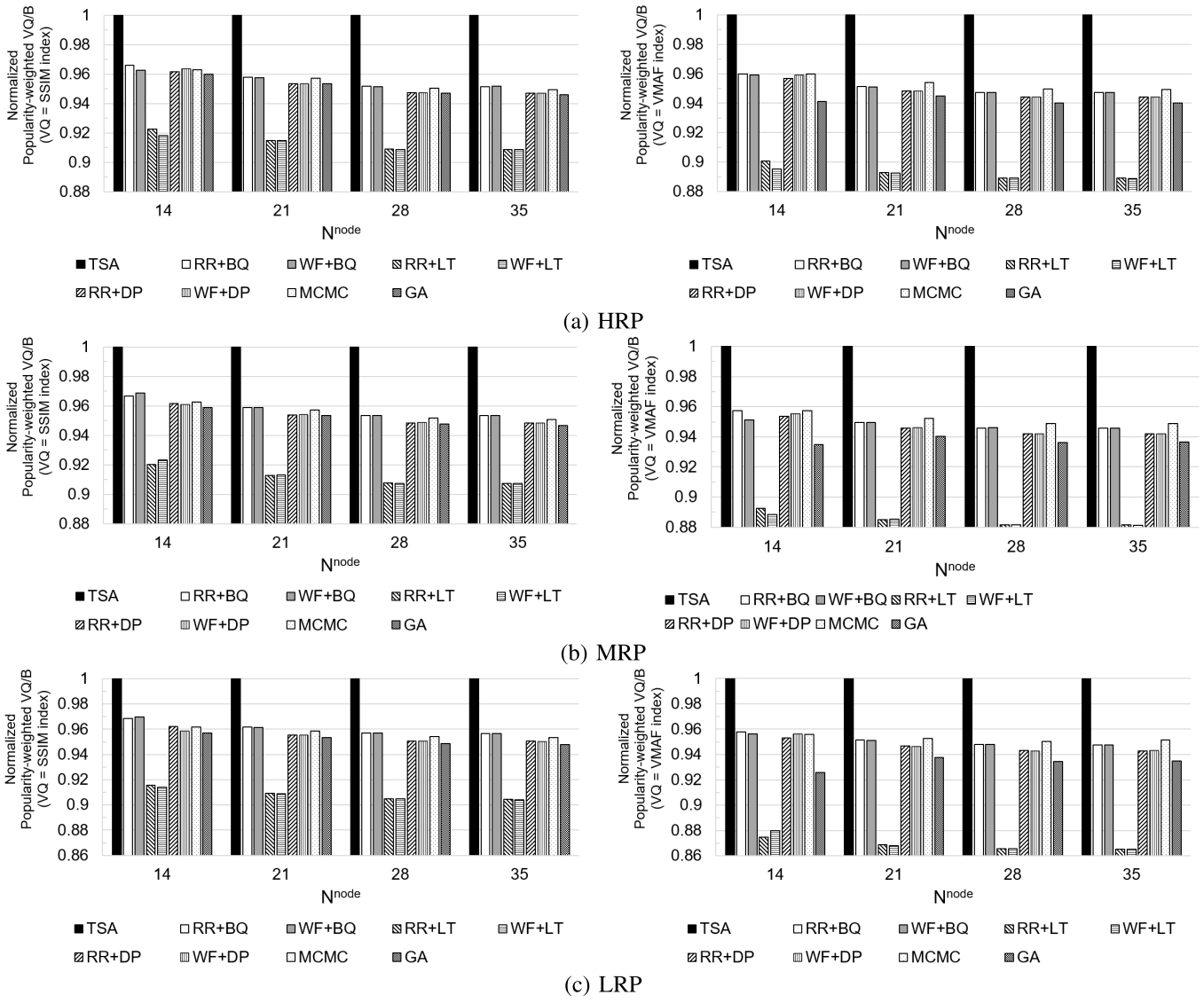

Fig. 6. Popularity-weighted VQ/B for seven task allocation and preset parameter selections, normalized against results for TSA against number of nodes N^{node}.

to 1%, two-point crossover and roulette wheel selection were used.

We then compare the TSA algorithm with the following schemes: RR + BQ, WF + BQ, RR + LT, WF + LT, RR + DP, WF + DP, MCMC and GA.

2) Effect of λ^{trans}: We examined the effect of the number of average transcoding requests per second, λ^{trans}, on the popularity-weighted VQ/B and deadline miss ratio. The server that we modeled included a frontend node and 28 backend nodes: four i7-9700 CPUs, four RTX 2070 GPUs, four i7-8700 CPUs, four GTX 1060 GPUs, four i5-7500 CPUs, four GT 720 GPUs, and four i7-4790 CPUs.

Fig. 4 shows the comparison results for different workloads. TSA always shows the best performance in all cases; for example, the SSIM differences are between 3.34% and 10.54%, and the VMAF differences are between 3.98% and 15.19%. These differences tend to be marginally greater when the load is light, and HRP workloads and VMAF indices.

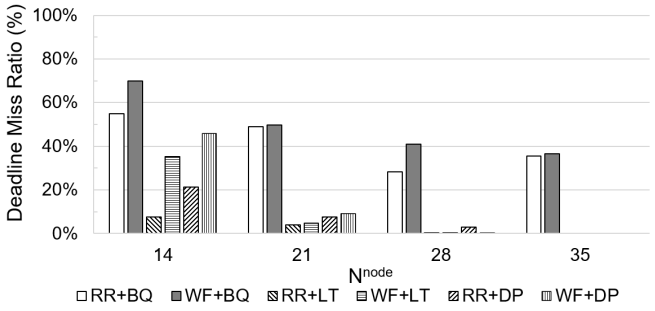

Fig. 7. Percentage (%) of missed deadlines against N^{node}.

Fig. 5 shows the number of missed deadlines and their ratio against λ^{trans}. TSA, MCMC and GA check the deadline constraint in Inequality (1) to meet all deadlines, so they are not shown in Fig. 5. WF + HF selects the highest video quality, resulting in the average of deadline miss ratio of 56.14% when the loads are heavy ($\lambda^{\text{trans}} = 24$). Even

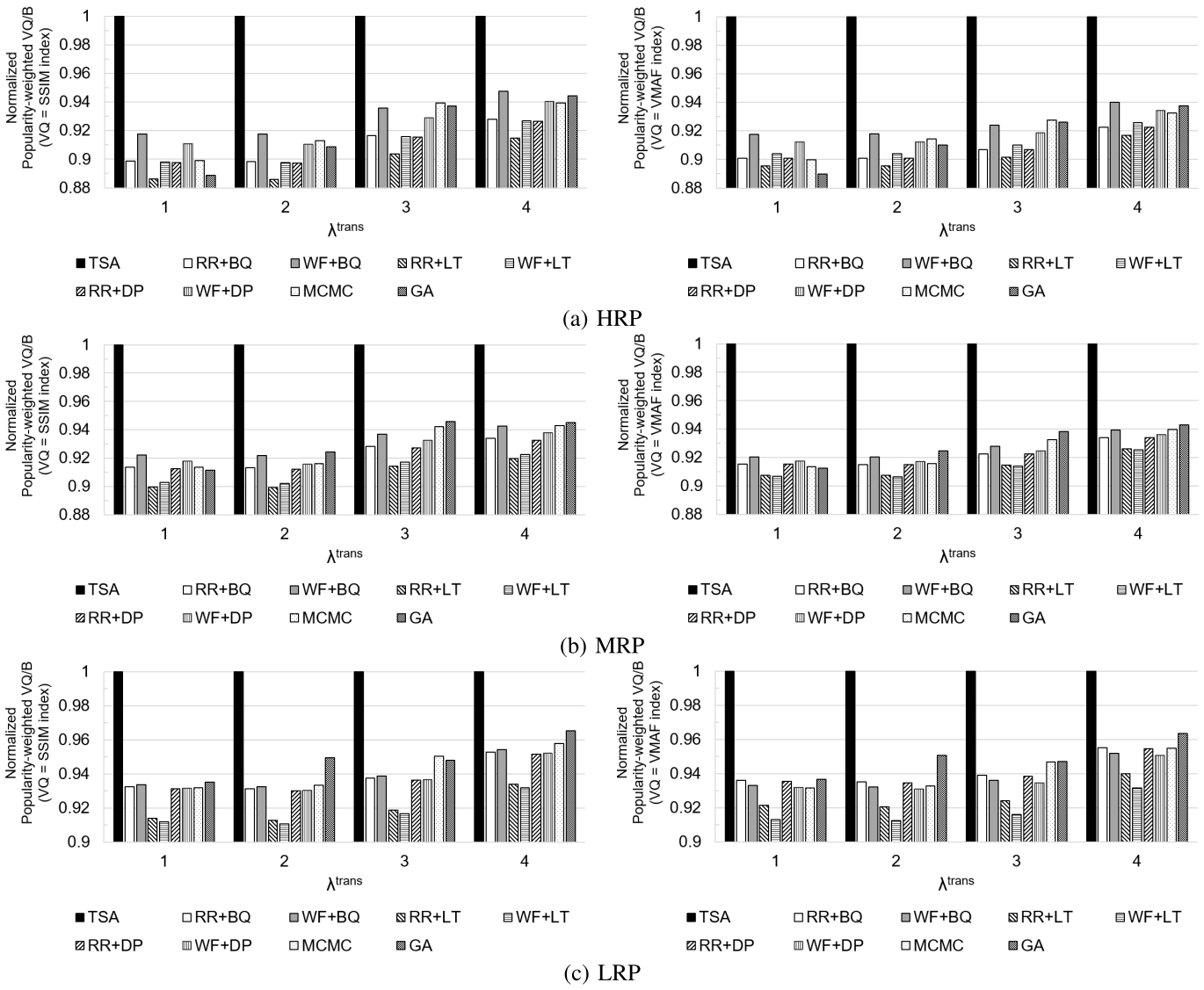

Fig. 8. Popularity-weighted VQ/B normalized against results for TSA on a testbed.

when the lowest transcoding time is chosen with round-robin allocation (RR + LT), the average of deadline miss ratio is 4.47% with $\lambda^{\text{trans}} = 24$, which highlights the importance of backend node selection. For all the other schemes, the number of missed deadlines increases with λ^{trans}, thereby increasing their percentage.

3) Effect of N^{node}: Further, we examined the effect of the number of nodes, N^{node}, on the popularity-weighted VQ/B and deadline miss ratio when λ^{trans} was 12. The backend nodes comprise seven groups with different processor types, each of which includes the same number of processors. Fig. 6 displays the comparison results. Again, TSA produces the highest popularity-weighted VQ/B in all cases. For example, the SSIM index is 3.12%–10.61%, and the VMAP index is 4.19%–15.59% greater than that of the other schemes. The gap between TSA and the other schemes increases marginally with the number of nodes. TSA can improve the popularity-weighted VQ/B for a large number of nodes with a variety of preset parameters from which to choose.

Fig. 7 shows the number and percentage of missed deadlines against N^{node}. Again, TSA, MCMC and GA meet all deadlines, whereas the other schemes do not. For example, WF + HF selects the highest video quality, and, hence, the average of deadline miss ratio is 49.28% when the load is heavy ($N^{\text{node}} = 14$). Even when the lowest transcoding time is chosen with RR allocation (RR + LT), the average of deadline miss ratio approaches 7.98% with $N^{\text{node}} = 14$. As N^{node} increases, the workload become lighter, reducing the percentage of missed deadlines in all the other schemes.

4) Algorithm Execution Time: To evaluate the overhead of the TSA algorithm, we measured the average execution time on the i7-10700 CPU, as tabulated in Table V. The execution times are between 9 ms and 44.33 ms, which clearly indicates that it executes with a small overhead. The execution time increases with the values of λ^{trans} and N^{node} because more computation is required to support the additional tasks and nodes.

TABLE V
AVERAGE EXECUTION TIME (MS) OF TSA ALGORITHM AGAINST VALUES OF λ^{trans} AND N^{node}

	Effect of λ^{trans}				Effect of N^{node}			
λ^{trans}	12	16	20	24	12	12	12	12
N^{node}	28	28	28	28	14	21	28	35
HRP	16.5	24.5	34	44.5	9	13	16.5	20
MRP	16	24.5	34	44.5	9	13	16	20
LRP	16	24.5	33	44	9	13	16	20
Average	16.17	24.5	33.67	44.33	9	13	16.17	20

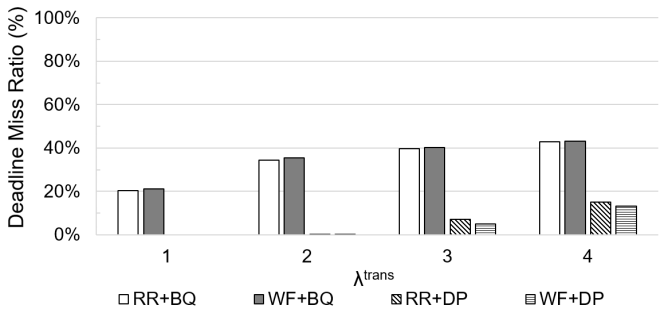

Fig. 9. Percentage of average deadline misses against λ^{trans} on a testbed.

C. Experiments on a Testbed

We evaluated the proposed scheme on an actual transcoding server comprising a frontend node and four backend nodes, one each of an i7-9700 CPU, RTX 2070 GPU, i7-8700 CPU, and GTX 1060 GPU. We randomly selected 250 original 1080p video segments from 18 video clips [37] that were not used for the regression establishment. A thousand tasks were executed on the server to transcode four different versions: 720p, 480p, 360p, and 240p. The value of ϵ was set to 3.5 s. We used the Zipf model in Section VII-B to determine the popularity-weighted VQ/B.

The task requests arrived at the server at the rate of λ^{trans}, and the frontend node executed the TSA once per second to determine the values of X_i and Y_i for each task τ_i. Fig. 8 shows the values of the popularity-weighted VQ/B for 1,000 tasks for three different workloads: HRP, MRP, and LRP. Compared with other approaches, TSA improves the popularity-weighted VQ/B values between 3.58% and 12.88% for the SSIM index and 3.78% and 12.37% for the VMAF index. The difference tends to be greater at lighter loads.

Fig. 9 shows the percentage of average missed deadlines to the total number of tasks against λ^{trans}. TSA, RR + LT, WF + LT, MCMF and GA satisfy all the deadlines, whereas the other schemes do not. If the LT preset parameter is selected such that the transcoding time is the lowest, then all deadlines can be met; however, this shows the worst popularity-weighted VQ/B values, as shown in Fig. 8.

VIII. CONCLUSION

We proposed a new task scheduling and allocation scheme for a transcoding server with heterogeneous processor types. We performed a multiple regression analysis on sample videos so that the effect of video resolution on bitrate, transcoding time, and video quality can be predicted. We then described an algorithm that selects a preset parameter and processor node for each task with the aim of maximizing the popularity-weighted VQ/B while satisfying all the task deadlines. The scheme uses a utility model that expresses the ratio of video quality to slack time and gradually distributes workloads to improve the popularity-weighted VQ/B while completing the tasks with close deadlines first. We implemented this scheme on a two-tier server.

The experimental results demonstrated that the proposed scheme effectively addressed four issues: VBR processing, task allocation on heterogeneous processor types, preset parameter selection, and meeting deadline constraints. In particular, the MAPE values suggest that our regression can predict bitrates, transcoding times, and video quality accurately. The proposed algorithm improves the popularity-weighted VQ/B between 3.12% and 15.59% compared with alternative methods of task allocation and preset parameter selection while meeting all deadlines.

To support the growing demand for transcoding, processors must be added incrementally, making servers with heterogeneous processor types essential. The results presented in this paper confirmed that the proposed scheme can contribute to the effective management of such servers to support different transcoding tasks with different deadlines, popularities, and VBR characteristics. In future work, we plan to extend the proposed scheme to reduce energy consumption by considering the power characteristics of each processor type.

REFERENCES

[1] D. K. Krishnappa, M. Zink, and R. K. Sitaraman, "Optimizing the video transcoding workflow in content delivery networks," in *Proc. 6th ACM Multimedia Syst. Conf.*, Mar. 2015, pp. 37–48.
[2] *YouTube for Press*. Accessed: May 8, 2020. [Online]. Available: https://www.youtube.com/intl/en-gb/yt/about/press/
[3] L. Toni, R. Aparicio-Pardo, G. Simon, A. Blanc, and P. Frossard, "Optimal set of video representations in adaptive streaming," in *Proc. 5th ACM Multimedia Syst. Conf. (MMSys)*, 2014, pp. 271–282.
[4] 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.
[5] C.-L. Fan, S.-C. Yen, C.-Y. Huang, and C.-H. Hsu, "On the optimal encoding ladder of tiled 360° videos for head-mounted virtual reality," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 4, pp. 1632–1647, Apr. 2021.
[6] Q. Huang *et al.*, "SVE: Distributed video processing at Facebook scale," in *Proc. 26th Symp. Operating Syst. Princ.*, Oct. 2017, pp. 407–419.
[7] Y. Qin *et al.*, "ABR streaming of VBR-encoded videos: Characterization, challenges, and solutions," in *Proc. 14th Int. Conf. Emerg. Netw. Exp. Technol.*, Dec. 2018, pp. 1–6.
[8] A. Reed and B. Klimkowski, "Leaky streams: Identifying variable bitrate DASH videos streamed over encrypted 802.11n connections," in *Proc. 13th IEEE Annu. Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2016, pp. 1–6.
[9] L. Yu, T. Tillo, and J. Xiao, "QoE-driven dynamic adaptive video streaming strategy with future information," *IEEE Trans. Broadcast.*, vol. 63, no. 3, pp. 523–534, Sep. 2017.
[10] J. Summers, T. Brecht, D. Eager, and A. Gutarin, "Characterizing the workload of a Netflix streaming video server," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Sep. 2016, pp. 43–54.
[11] L. Tang, Q. Huang, A. Puntambekar, Y. Vigfusson, W. Lloyd, and K. Li, "Popularity prediction of Facebook videos for higher quality streaming," in *Proc. USENIX Annu. Tech. Conf.*, Santa Clara, CA, USA, Jul. 2017, pp. 111–123.
[12] 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.

[13] Y. Chen, J. Zhu, T. A. Khan, and B. Kasikci, "CPU microarchitectural performance characterization of cloud video transcoding," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Oct. 2020, pp. 72–82.

[14] 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, pp. 1–23, Feb. 2015.

[15] 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.

[16] C.-C. Huang, J.-J. Chen, and Y.-H. Tsai, "A dynamic and complexity aware cloud scheduling algorithm for video transcoding," in *Proc. IEEE Int. Conf. Multimedia Expo Workshops (ICMEW)*, Jul. 2016, pp. 1–6.

[17] L. Van Ma, J. Park, J. Nam, J. Jang, and J. Kim, "An efficient scheduling multimedia transcoding method for DASH streaming in cloud environment," *Cluster Comput.*, vol. 22, no. S1, pp. 1043–1053, Jan. 2019.

[18] G. Gao, Y. Wen, and C. Westphal, "Dynamic priority-based resource provisioning for video transcoding with heterogeneous QoS," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 29, no. 5, pp. 1515–1529, May 2019.

[19] X. Li, M. A. Salehi, Y. Joshi, M. K. 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, Apr. 2019.

[20] H. Ma, B. Seo, and R. Zimmermann, "Dynamic scheduling on video transcoding for MPEG DASH in the cloud environment," in *Proc. 5th ACM Multimedia Syst. Conf. (MMSys)*, 2014, pp. 283–294.

[21] 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 Int. Symp. Multimedia (ISM)*, Dec. 2017, pp. 334–337.

[22] 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, nos. 13–14, pp. 9037–9061, May 2019.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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. 39th Euromicro Conf. Softw. Eng. Adv. Appl.*, Sep. 2013, pp. 1–4.

[28] E. Baccour, A. Erbad, K. Bilal, A. Mohamed, and M. Guizani, "PCCP: Proactive video chunks caching and processing in edge networks," *Future Gener. Comput. Syst.*, vol. 105, pp. 44–60, Apr. 2020.

[29] 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.

[30] X. Zhang, D. Xiong, K. Zhao, C. W. 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.

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

[32] Z. H. Chang, B. F. Jong, W. J. Wong, and M. L. Dennis Wong, "Distributed video transcoding on a heterogeneous computing platform," in *Proc. IEEE Asia Pacific Conf. Circuits Syst. (APCCAS)*, Oct. 2016, pp. 1–4.

[33] W. Katsak, H. Nguyen, K. Nagaraja, J. Halen, N. Radia, and T. D. Nguyen, "Catalyst: A cloud-based media processing framework," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2019, pp. 1–11.

[34] S.-H. Chang, K.-J. Wang, and J.-M. Ho, "Optimal DASH video scheduling over variable-bit-rate networks," in *Proc. 9th Int. Symp. Parallel Archit., Algorithms Program. (PAAP)*, Dec. 2018, pp. 1–8.

[35] 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 INFOCOM Conf. Comput. Commun.*, May 2017, pp. 1–9.

[36] FFmpeg. (2019). *H.264 Video Encoding Guide*. [Online]. Available: https://trac.ffmpeg.org/wiki/encode/h.264

[37] *HD-Trailers.net*. Accessed: Apr. 21, 2021. [Online]. Available: http://www.hd-trailers.net/

[38] S. Mekouar, N. Zrira, and E.-H. Bouyakhf, "Popularity prediction of videos in YouTube as case study," in *Proc. 2nd Int. Conf. Big Data, Cloud Appl.*, Mar. 2017, p. 51.

[39] T. Trzcinski and P. Rokita, "Predicting popularity of online videos using support vector regression," *IEEE Trans. Multimedia*, vol. 19, no. 11, pp. 2561–2570, Apr. 2017.

[40] T. Zinner, O. Hohlfeld, O. Abboud, and T. Hossfeld, "Impact of frame rate and resolution on objective QoE metrics," in *Proc. 2nd Int. Workshop Qual. Multimedia Exper. (QoMEX)*, Jun. 2010, pp. 29–34.

[41] 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.

[42] *VMAF: The Journey Continues*. Accessed: Apr. 21, 2021. [Online]. Available: https://netflixtechblog.com/vmaf-the-journey-continues-44b51ee9ed12/

[43] R. Rassool, "VMAF reproducibility: Validating a perceptual practical video quality metric," in *Proc. IEEE Int. Symp. Broadband Multimedia Syst. Broadcast. (BMSB)*, Jun. 2017, pp. 1–2.

[44] G. Buttazzo, *Hard Real-Tme Computing Systems: Predictable Scheduling Algorithms and Applications*. New York, NY, USA: Springer, 1997.

[45] T. Andren, *Econometrics*. Stockholm, Sweden: Thomas Andren & Ventus Publishing ApS, 2007.

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

[47] C. D. Lewis, *Industrial and Business Forecasting Methods: A Practical Guide to Exponential Smoothing and Curve Fitting Butterworth Scientific*. New York, NY, USA: Butterworths Scientific, 1982.

[48] H. Elmousalami, "Comparison of artificial intelligence techniques for project conceptual cost prediction: A case study and comparative analysis," *IEEE Trans. Eng. Manage.*, vol. 68, no. 1, pp. 183–196, Feb. 2020.

[49] 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, Jun. 1996.

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

[51] P. Diaconis, "The Markov chain Monte Carlo revolution," *Bull. Amer. Math. Soc.*, vol. 46, no. 2, pp. 179–205, Apr. 2009.

[52] S. K .Shil, *An Approach to Solve MMKP: Using Genetic Algorithm: Basic, Solving Procedure and Discussion*. New York, NY, USA: Academic, 2010.

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.

Minseok Song (Member, IEEE) received the B.S., M.S., and Ph.D. degrees 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, multimedia, and the IoT systems.