FastTTPS: fast approach for video transcoding time prediction and scheduling for HTTP adaptive streaming videos

Prateek Agrawal^{1,2} · Anatoliy Zabrovskiy^{1,3} · Adithyan Ilangovan⁴ · Christian Timmerer^{1,4} · Radu Prodan¹

Received: 21 April 2020 / Revised: 28 October 2020 / Accepted: 3 November 2020 / Published online: 20 November 2020
© The Author(s) 2020

Abstract

HTTP adaptive streaming of video content becomes an integrated part of the Internet and dominates other streaming protocols and solutions. The duration of creating video content for adaptive streaming ranges from seconds or up to several hours or days, due to the plethora of video transcoding parameters and video source types. Although, the computing resources of different transcoding platforms and services constantly increase, accurate and fast transcoding time prediction and scheduling is still crucial. We propose in this paper a novel method called fast video transcoding time prediction and scheduling (FastTTPS) of x264 encoded videos based on three phases: (i) transcoding data engineering, (ii) transcoding time prediction, and (iii) transcoding scheduling. The first phase is responsible for video sequence selection, segmentation and feature data collection required for predicting the transcoding time. The second phase develops an artificial neural network (ANN) model for segment transcoding time prediction based on transcoding parameters and derived video complexity features. The third phase compares a number of parallel schedulers to map the predicted transcoding segments on the underlying high-performance computing resources. Experimental results show that our predictive ANN model minimizes the transcoding mean absolute error (MAE) and mean square error (MSE) by up to 1.7 and 26.8, respectively. In terms of scheduling, our method reduces the transcoding time by up to 38% using a Max–Min algorithm compared to the actual transcoding time without prediction information.

Keywords Transcoding time prediction · Video transcoding · Scheduling · Artificial neural networks · MPEG-DASH · Adaptive streaming

1 Introduction

According to Cisco's annual Internet report (2019–2023) [1], video applications and services are in high demand with ever increasing requirements in video quality and network bandwidth. Therefore, processing and transmitting video content over the Internet as quickly as possible and in all qualities is crucial [2]. Nowadays, adaptive streaming formats such as MPEG-DASH [3] and HTTP live streaming (HLS) [4] significantly improve the video quality of Internet streaming services [5]. They divide and transcode the video sequences in segments of the same content but with different resolutions and bitrates (or qualities) before transmitting them over the network [3]. Depending on the client network characteristics (e.g. bandwidth, latency), the rate adaptation algorithm of a media player requests segments with an appropriate bitrate [6, 7] and aims to

---

✉ Prateek Agrawal
prateek061186@gmail.com

Anatoliy Zabrovskiy
anatoliy.zabrovskiy@aau.at

Adithyan Ilangovan
adithyan.ilangovan@bitmovin.com

Christian Timmerer
christian.timmerer@aau.at

Radu Prodan
radu.prodan@aau.at

¹ University of Klagenfurt, Klagenfurt, Austria

² Lovely Professional University, Phagwara, India

³ Petrozavodsk State University, Petrozavodsk, Russia

⁴ Bitmovin Inc., San Francisco, CA, USA

maintain a high quality of experience [8, 9] by switching between segments with different qualities. Adaptive 360° video streaming applications [10, 11] widely adjust the video quality for the field of view and the rest of the image. Unfortunately, creating segments of a single video for adaptive streaming can take seconds or even hours [12], depending on many technical aspects and features, such as video codec, video file characteristics, transcoding features, and processing capabilities [2].

1.1 Motivation

To enable fast video transcoding, companies currently operate large data centers and deploy the transcoding tasks using opportunistic load balancing (OLB) algorithms to keep the processing core utilized at all times [9]. The available transcoding services and platforms process a huge amount of video files using modern transcoding architectures and state-of-the-art video codecs. Typically, such systems use different scheduling algorithms that maximize the use of processing units and minimize the transcoding time of video segments. For example, Amazon provides computing instances to various companies for video transcoding operations. As the cost of such computing units depends on the time of use (per hour or per second), the customers strive to keep the highest possible utilization for all computing cores. This is relatively easy to achieve if all the transcoding tasks have similar complexity and require similar execution times on the underlying computing units. However, a problem arises when a simple scheduling algorithm randomly assigns several and very time-consuming tasks to a single computing unit, leading to a load imbalance and a degradation of video quality on the viewer side or to an increase in expenses. The motivation for our work is to avoid such situations and provide the transcoding infrastructure with advance information on the various video transcoding tasks to ensure their fast completion with a good load balance.

1.2 Approach

To decrease the transcoding time and improve hardware resource utilisation, we propose a new method called **Fast video Transcoding Time Prediction and Scheduling** (FastTTPS) for adaptive streaming based on three phases: (i) transcoding data engineering, (ii) transcoding time prediction, and (iii) transcoding scheduling. The first phase prepares the transcoding data using a video selection and segmentation step, followed by feature and data creation. The second phase predicts the transcoding time using an

intelligent ANN model trained for accurate prediction using a novel 144-pixel transcoding method. Finally, the third step integrates higher-complexity scheduling algorithms that exploit the predicted transcoding time to achieve high quality mappings of the transcoding tasks on a parallel machine with multiple cores. We evaluated our method on a set of ten heterogeneous video sequences of different type with different duration and frame rates. Experimental results show that our ANN model was able to predict the video transcoding with low MAE and MSE of 1.7 and 26.8, respectively. On top of it, a Max–Min scheduler improved the transcoding time by 38% compared to a simple OLB method practiced in industry without prediction information [9].

1.3 Contributions

The main contributions of the FastTTPS method are:

- We analyzed several video complexity features and transcoding parameters and proposed two new features for fast transcoding time prediction using ANN: (i) *144-pixel transcoding time* and (ii) *144-pixel file size*.
- We proposed and implemented the *sample-144 pixel* method for fast features extraction for the entire video sequence.
- We compared the selective *sample-144 pixel* method with the complete *full-144 pixel* method that uses all segments of the video sequence and observed similar transcoding time patterns.
- We developed a sequential ANN model for fast transcoding time prediction for heterogeneous video segments.
- We analysed four scheduling methods and found out that the predictive Max–Min scheduling algorithm reduces the transcoding time by up to 38% over a commonly used OLB method that uses no prediction information.

1.4 Outline

The paper has six sections. Section 2 highlights the related work. Section 3 describes the proposed FastTTPS methodology comprising the three transcoding phases, with an emphasis on the ANN model and scheduling algorithms. Section 4 provides an implementation case study of our general method, followed by evaluation results in Sect. 5. Section 6 concludes the paper and highlights future work opportunities.

2 Related work

2.1 Transcoding time prediction

Current research focuses on analysing the transcoding and video complexity features based on a set of video file characteristics. Tewodros et. al. [13, 14] proposed a method based on machine learning algorithms such as linear regression, support vector machines (SVM), and ANN for several codecs using bitrate, frame rate, resolution, codec (i.e. x264, MPEG4, VP8 and H.263), number and size of intra, predictive and bidirectional frames inputs. They used bitrates of up to 5 Mbit/s and 1080 px resolutions, which we extended to 20 Mbit/s and 2160 px. They achieved a MAE in the range between 1.757 and 4.591, while we achieved a MAE of 1.7 for heterogeneous random video sequences. FastTTPS also achieved a better coefficient of determination (i.e. 0.988) for transcoding time prediction compared to [14] (i.e. 0.958). The relatively large time required to extract the video file characteristics is another drawback of [14]. Li et al. [15, 16] analyzed the most common video transcoding operations on different computing units. They investigated the effect of video frame rate on the transcoding time and proposed a simple prediction approach based on a previous run-time history. Zakerinasab et al. [17] investigated the effect of video chunk size on transcoding speed and video quality without any specific prediction for any configurations. Andujar et al. [18] implemented a SVM to verify video transcoding computations for the x264 codec as part of the Livepeer open source project. Zabrovskiy et al. [19] implemented a video complexity classification method using ANN that reduces the MAE by up to 1.37 at the expense of an approximately ten times increase in transcoding time prediction. Paakkonen et al. [20] presented an online architecture for predicting video transcoding resources on a Kubernetes platform.

Table 1 summarises the available video transcoding prediction work on different codecs. However, none of these studies address transcoding time prediction for x264 codec and various encoding presets using a fast approach based on the analysis of small video sequence samples.

2.2 Cluster computing scheduling

Kirubha and Ramar [21] implemented a modified controlled channel access scheduling method to improve the quality of service-based video streaming. Munagala and Prasad [22] proposed a cluster entropy-based transcoding method for efficient video encoding and compression, experimented on eight video sequences. Carballeira et al. [23] combined randomized techniques with static local

balancing in a round-robin manner for tasks scheduling. Chhabra et al. [24] combined opposition-based learning, cuckoo search, and differential evolution algorithms to schedule high performance computing tasks in the cloud. Ebadifard and Babamir [25] proposed a dynamic load balancing task scheduling algorithm for a cloud environment that minimizes the communication overhead. Milan et al. [26] implemented a swarm intelligence algorithm for priority based task scheduling using a bacterial foraging optimization that reduces the tasks idle time and run-time with effective load balancing.

2.3 Transcoding task scheduling

Recent works made important contributions on scheduling the video transcoding tasks [27–29]. Jokhio et al. [30] presented a distributed video transcoding method with focus on reducing video bitrates. Li et al. [31] presented a QoS-aware scheduling approach for mapping transcoding jobs to the heterogeneous virtual machines. Zhao et al. [32] proposed a scheduling method that uses video complexity metrics to parallelize the transcoding over a heterogeneous MapReduce system. Mostafa et al. [33] presented an autonomous resource provisioning framework to control and manage computational resources using a fuzzy logic auto-scaling algorithm in a cloud environment. Recently, Sameti et al. [34] proposed a container-based transcoding method for interactive video streaming that automatically calculates the number of processing cores that maintain a certain frame rate for any given video segment and transcoding resolution. The authors use benchmarking to find the optimal parallelism for interactive streaming video. Li et al. [35] proposed a HAS delivery scheme that combines caching, transcoding for energy and resource efficient scheduling. Mostafa et al. [36] presented a moth-flame optimization algorithm that defines and assigns the appropriate jobs to fog computing units to reduce the total task execution time, evaluated using the iFogSim toolkit [37]. Linh et al. [9] proposed a scheduling method for transcoding MPEG-DASH video segments using a node that manages all other servers in the system (rather than predicting the transcoding times), and reported a saving time of up to 30%.

Table 2 summarises the available scheduling work on video transcoding on different codecs. However, not many publications [14, 15] studied task scheduling based on complexity analysis of the video content. Previous studies also do not take increasingly popular high bitrates and resolutions into account.

Table 1 Summary of the video transcoding prediction methods

Authors	Techniques	Codecs / encoding standards	Video types	Remarks
Tewodros et al. [13, 14]	Linear regression SVM ANN	x264 MPEG-4 VP8 H.263	Different length	MAE: 1.757 ± 2.834
Li et al. [15, 16]	Regression analysis	H.264	Group of Pictures (GoPs)	Optimal computing unit identification
Zakerinasab et al. [17]	Benchmarking analysis	H.264	GoPs	Video chunk size vs. transcoding speed
Andujar et al. [18]	SVM	x264	Same length	Video transcoding verification
Zabrovskiy et al. [19]	ANN	x264 x265	Different length	MAE: 1.37 10× slower than FastTTPS
Paakkonen et al. [20]	Random forest regressor Reinforcement learning Stochastic gradient descent	H.264	Different Length	Average accuracy: 3–6
FastTTPS	ANN	x264	Different length	MAE: 1.7 MSE: 26.8

Table 2 Summary of video transcoding scheduling methods

Authors	Techniques	Video type	Remark
Tewodros et al. [38]	Proactive load balancing	x264 MPEG-4 VP8 H.263	18% transcoding time reduction
Linh et al. [9]	Fuzzy representation inference	MPEG-DASH	30% transcoding time reduction
Ma et al. [39]	Dynamic scheduling	MPEG-DASH	Video transcoding mode switching
FastTTPS	Max–Min algorithm	x264	38% transcoding time reduction

3 Methodology

This section outlines the FastTTPS methodology composed of three phases: (i) transcoding data engineering, (ii) transcoding time prediction, and (iii) transcoding time scheduling. Each phase has further steps as shown in Fig. 1 and explained in the following subsections:

Fig. 1 FastTTPS methodology

3.1 Transcoding data engineering

This phase is responsible for video sequence selection and feature data collection to predict the transcoding time. This phase consists of two steps.

3.1.1 Video sequence selection and segmentation

This step selects videos of different type for proper training and validation. Some sequences have minor movements such as moving head on a static black background, while others have significant movements such as changing street view or riding jockeys [2]. After the video sequence selection, the next step splits the video files in different segments, for example 2 s, 4 s or 10 s long, as commonly used in industry.

3.1.2 Feature selection and data creation

After creating the video segments, the next step is to select and extract different features from each video segment for data collection. We identify the following relevant features for our method: *transcoding bitrate*, *segment duration*, *width*, *height*, *encoding preset*, *frame rate*, and *transcoding time*. We calculate the *number of pixels* of a video file by multiplying its width and height. After defining and extracting the features from each video segment, we prepare the dataset for the second transcoding time prediction phase.

To improve the training and the accuracy of the ANN model, we augmented the primary features (having predefined value ranges) with two additional features that purely depend on the nature of the video content:

- *144-pixel transcoding time* is the transcoding time of any video segment to low resolution (i.e. 256×144 px) and low bitrate (100 kbit/s);
- *144-pixel file size* is the video segment size (in bytes) transcoded at low bitrate and low resolution.

We choose 144-pixel features because transcoding to low resolution and bitrate is approximately ten times faster than to a high bitrate and high resolution.

3.2 Transcoding time prediction

This phase builds the ANN model for predicting the transcoding time of video sequences in two steps.

3.2.1 Training and testing data preparation

The first step analyzes and preprocesses the data collected in the previous phase to predict the transcoding time of video segments. After preparing the feature data of

segmented videos, we analyze and classify its input and output features. We consider *transcoding bitrate*, *segment duration*, *number of pixels*, *height*, *encoding preset*, *frame rate*, *144-pixel transcoding time* and *144-pixel file size* as input features, and *transcoding time* of each segment as output feature. Afterwards, the next task distributes the entire dataset between the training and testing datasets.

3.2.2 ANN model creation and tuning

After preparing the training and testing data, the next step builds the ANN model [40]. We select a sequential model for training because only the first layer needs to receive the information about the input shape (like batch size and unit), while the rest infers the input shapes automatically. After deciding the model based on the data, the next step selects an appropriate activation function for ANN training. We use both linear and non-linear types of activation functions to train the model.

We further define two important transcoding metrics:

- *Predicted transcoding time (PTT)* of a video segment resulting from an ANN prediction;
- *Actual transcoding time (ATT)* of a video segment, measured after effectively running a transcoding task.

We use two metrics to evaluate the ANN model accuracy:

- *Mean absolute error (MAE)* that represents the average model prediction error (in seconds):

$$MAE = \frac{1}{n} \sum_{j=1}^n |PTT_j - ATT_j|;$$

- *Mean square error (MSE)* defined as the squared average difference between the ATT and PTT:

$$MSE = \frac{1}{n} \sum_{j=1}^n (PTT_j - ATT_j)^2,$$

where PTT_j is the predicted and ATT_j is the actual output value for each training observation j . Based on MAE and MSE, we tune and update the training features of the ANN model until we obtain a consistent accuracy.

3.3 Transcoding time scheduling

After predicting the transcoding time of each video segment, the next phase maps the video segments onto the underlying parallel machines. There are many scheduling methods [41, 42] with different complexity for allocating a heterogeneous set of tasks m to a set of computing cores n . To evaluate the performance of a scheduling algorithm, we use the total transcoding time [43] of all segments with a good load balance on the underlying parallel machines.

3.3.1 Round robin (RR)

RR method uses an ordered list of resources and allocates each task to the next processing core in a cyclic fashion, starting from the beginning once it reaches the last one [44]. This simple $O(n)$ -complexity method statically assigns all tasks at once without considering the transcoding time or the resource load.

3.3.2 Opportunistic load balancing (OLB)

OLB is an $O(n \cdot m)$ -complexity method [45] that assigns each task to the resource that becomes available next, again with no prior information about the task execution times. This method is currently employed by the state-of-the-art industrial streaming platforms.

3.3.3 Min–Min algorithm

Min–Min algorithm is an $O(n^2 \cdot m)$ -complexity algorithm that iterates twice over the set of tasks in a nested loop. In each iteration, it schedules the task with the lowest completion time across all cores and reconsiders the others using the same policy [45, 46]. Min–Min executes smallest tasks first and may suffer from load imbalance if a comparatively lower number of large tasks remain to be executed in the later part.

3.3.4 Max–Min algorithm

Max–Min is another $O(n^2 \cdot m)$ -complexity algorithm that, in contrast to Min–Min, schedules first the task with the highest minimum completion time across all cores and reconsiders the rest [47, 48]. Max–Min typically performs better than Min–Min as the workflow contains a relatively small number of large tasks compared to the majority of the tasks.

4 Case study evaluation

We used for transcoding a dedicated server with 80 Intel Xeon Gold 6148 processing cores with a frequency of 2.4 GHz. We divided the entire evaluation workflow in three phases, as discussed in Sect. 3: transcoding data engineering, transcoding time prediction and transcoding time scheduling. We performed various operations within these phases as shown in Fig. 2:

- We split the selected video sequences in 2 s and 4 s segments.
- We identified different features values for 19 bitrates and nine presets.

Fig. 2 FastTTPS sequence diagram interaction roadmap

- We transcoded the middle segment of each video file at a low resolution and low bitrate, and calculated their transcoding time and file size.
- We combined these features with other features to prepare the training and testing data for the ANN model.
- We trained and tuned the proposed ANN model for transcoding time prediction and calculated the PTT.
- We applied four scheduling methods and compared their saving time.

We explain these phases and their interaction in the remaining of this section.

4.1 Transcoding data engineering

4.1.1 Video sequence selection and segmentation

In this phase, we selected ten video sequences from our previous work [2] comprising different types of video content in YUV format with the characteristics shown in Table 3. The video sequences *BBB*, *Sintel* and *TOS* contain *animation* content (fully or mixed), *ReadySetGo* and *Jocky* are *sports*-related, *HoneyBee* and *WindAndNature* are of *nature* category, while *YachtRide*, *DrivingPOV* and *Beauty* classified as *general* do not match any category.

We split all video sequences into raw YUV segments using the FFmpeg 4.1.3 software and prepared a total of 240 segments with a duration of 2 s and 4 s. The segment length is a crucial feature in adaptive streaming, as each segment usually starts with a random access point that enables dynamic switching to other representations at segment boundaries [49]. The 4 s length adopted in commercial deployments shows a good tradeoff with respect to streaming and coding performance [50]. The 2 s length, also used in commercial deployments, confirms the trend towards low-latency requirements [51].

4.1.2 Feature selection and data creation

After creating the video segments, we selected the features required for the data creation based on existing datasets proposed in the literature [50, 52–54] and industry best-practices or guidelines [55–58]. We performed the transcoding using FFmpeg and Python scripts and measured the ATT. Since the focus of this paper is on adaptive streaming, we adopted the *bitrate ladder* shown in Table 4 consisting of a wide range of bitrate/resolutions including ultra high-definition (4K). We focus in our work on x264 codec (H.264/AVC compression format) which is predominately used in industry [59]. We executed a total of 41,040 transcoding tasks, as follows: 27,360 tasks for 2 s segments ($19 \text{ bitrates} \times 9 \text{ encoding presets} \times 160$

segments) and 13,680 tasks for 4 s segments ($19 \text{ bitrates} \times 9 \text{ encoding presets} \times 80 \text{ segments}$). The total execution time for all transcoding tasks was of approximately 209 hours. We generated for all transcodings a raw dataset with 41,040 records containing the following features and output metrics: *encoding bitrate*, *segment duration*, *video width*, *video height*, *encoding preset*, *frame rate* (per second), and *ATT*.

4.1.3 144-pixel transcoding

After identifying the input features (i.e. encoding bitrate, encoding preset, segment duration, frame rate, video height, number of pixels) from video segments for training data records, we derived two additional features (i.e. 144-pixel transcoding time, 144-pixel file size) to improve the ANN training process, as presented in Sect. 4.1.3. We transcoded each segment to a low 144-pixel resolution at a 100 kbit/s bitrate using the *ultrafast* x264 encoding preset. Afterwards, we calculated the *file size* in bytes of each transcoded video segment. We investigated two approaches to reduce the time for creating the derived 144-pixel transcoding features:

- *Full-144-pixel* uses all segments from all input video sequences;
- *Sample-144-pixel* splits each video in 60 s long sequences (up to 30 segments) and considers the ATT of middle segment for the entire sequence.

Figure 3 shows that the ATT variation of different segments within the same video is relatively low. Moreover, it also shows that the ATT follows the same pattern for the same video types (see Table 3), such as animation (with 30 segments of 2 s each) and nature. For example, if the length of a video sequence is 300 s, the Sample-144-pixel method considers five middle segment ATTs (i.e. one for each 60 s interval) and assigns these values to the rest of the segments in the fixed time length sequences. We managed with the Sample-144-pixel method to reduce the total

Table 3 Original video file characteristics

Video description	Video category	Frames per second	Duration (s)
BBB	Animation	30	60
Beauty	General	30	20
DrivingPOV	General	60	20
HoneyBee	Nature	30	20
Jockey	Sports	30	20
Sintel	Animation	24	60
TOS	Animation&real	24	60
WindAndNature	Nature	60	20
ReadySetGo	Sports	30	20
YachtRide	General	30	20

Table 4 Videos bitrate ladder (bitrate/resolution)

No	Bitrate	Resolution	No	Bitrate	Resolution
1	100	256 × 144	11	4300	1920 × 1080
2	200	320 × 180	12	5800	1920 × 1080
3	240	384 × 216	13	6500	2560 × 1440
4	375	384 × 216	14	7000	2560 × 1440
5	550	512 × 288	15	7500	2560 × 1440
6	750	640 × 360	16	8000	3840 × 2160
7	1000	768 × 432	17	12,000	3840 × 2160
8	1500	1024 × 576	18	17,000	3840 × 2160
9	2300	1280 × 720	19	20,000	3840 × 2160
10	3000	1280 × 720			

Fig. 3 ATT to 144-pixel at 100 kbit/s for 2 s segment of video sequences

sequential transcoding time of the 2 s video segments for all ten video sequences by a factor of 13 to 18 s only (from 248 s) on an Intel Core 2 Quad processor Q9650 at 3 GHz.

As the Sample 144-pixel method generates redundancy in the created dataset by reusing the same 144-pixel transcoding values for 30 video segments, we reduced the number of records by only considering the conservative maximum PTT and eliminating the rest. This resulted in a dataset with 3422 records for training and testing the ANN.

4.2 Transcoding time prediction

To compute the PTT of video segments (of 2 s and 4 s), we developed an ANN model using datasets collected with both Full-144-pixel and Sample-144-pixel methods

presented in Sect. 4.1.3. We therefore report an ANN model consisting of eight input neurons, three hidden layers and one output neuron, further detailed in Table 5. Each input neuron represents one input feature and output neuron executes the final PTT output, as shown in Fig. 4. We used a linear activation function at the output layer because it does not confine within any range. We used a half rectified linear unit (ReLU) as the hidden layer non-linear activation function, which immediately turns all negative input values to zero: $f(x) = \max(0, x)$. We used an 80% random sample of the created dataset for training and the remaining 20% for testing, as mentioned in Table 5.

The ANN model performance depends on the selection of training parameters, like inputs, output, activation function, batch size, and optimizer. Table 5 shows the optimized parameters for efficient ANN model training. We used the Adadelta method after evaluating the performance of other loss function optimizers, such as Adagrad and RMSprop [60]. Adadelta is an advanced optimizer of Adagrad that adapts the learning rates based on a moving window of gradient updates, rather than collecting the information of all past gradients. This special feature gives an advantage to Adadelta compared to other methods in adjusting the default learning parameters, even for multiple updates.

Figure 4 schematically shows all ANN input and output features, further detailed in Table 6. We separately trained the ANN model using data for the two Full-144-pixel and Sample-144-pixel methods and calculated the PTT used for

Table 5 ANN model parameters

Parameter	Values
Input layer neurons	8
Hidden layers	3
Hidden layer neurons	64/32/64
Output layer neurons	1
Learning rate	Default value
Passed epochs	600
Training dataset	2736
Testing dataset	684
Training and testing data ratio	80%/20%
Hidden layer activation function	ReLU
Output layer activation function	Linear
Optimizer	Adadelta
Loss function	MAE
Metric	MAE, MSE
Batch size	64
Initializing weights	Truncated normal

Fig. 4 ANN structure

scheduling the video transcoding tasks. We observe in Table 7 that the MAE of both ANN training methods are equal, while the MSE of the Sample-144-pixel is better than for the Full-144-pixel by considering a more representative sample of the training data with less variation. We therefore evaluate the scheduling algorithms on the Sample-144-pixel method only in the following section.

4.3 Transcoding task scheduling

After calculating the PTT of all video segments for a given sequence, a scheduling algorithm maps the transcoding tasks on an underlying parallel computer.

We performed a preliminary experiment that compares the performance of different scheduling algorithms using the tasks' ATT (representing the ideal 100% accurate PTT) on 190 *Beauty* video segments and using an increasing number of cores from 5 to 50. Figure 5 shows that Max–Min algorithm reduces the ATT on five cores by 30%, 6% and 6% compared to RR, OLB, and Min–Min methods respectively. As we gradually increase the number of cores to fifty, Max–Min increases its advantage by 65%, 38%

Table 7 Full- versus Sample-144-pixel transcoding

Metric	Full-144-pixel	Sample-144-pixel
MAE	1.7	1.7
MSE	30.9	26.8

and 33% due to the higher video content complexity generated by the different amount of video movements that bring more variety and an irregular distribution in the task ATTs.

5 Results and Analysis

This section presents and analyses the advantages of FastTTPS for transcoding x264 encoded video segments with improved prediction and scheduling. We analysed the PTT accuracy and performed a statistical analysis [61] to evaluate the quality, transcoding time, throughput, and saving time.

5.1 PTT accuracy analysis

We trained our ANN model on 2736 dataset records and tested it on 684 testing records representing 80%, respectively 20% of the overall data (see Table 5). We observe in Table 7 that the trained ANN model approximates the PTT with MAE and MSE up to 1.7 and 26.8, respectively. Figure 6 confirms a linear function $f(x) = x$ between PTT and ATT on the testing dataset. To further analyse the relationship between the PTT and the ATT, we calculated the Pearson's [62] coefficient for all 41 040 video transcodings, which confirms a good correlation value of 0.84. Figure 7 shows the PTT and ATT of 855 segment transcoding tasks with a 4 s duration each for the *Beauty* video sequence on all nine presets and 19 bitrates. The results show that, although there is a steady difference

Table 6 ANN input parameters

Input parameter	Description
Encoding bitrate (kbit/s)	See Table 4
Encoding preset	Ultrafast, superfast, veryfast, faster, fast, Medium, slower, slow, veryslow
Input file frame rate (per second)	24, 30, 60
Video height (pixels)	144p, 180p, 216p, 288p, 360p, 432p, 576p, 720p, 1080p, 1440p, 2160p
Number of pixels	Video height $\times$ width
Segment duration	2 s, 4 s
144-pixel transcoding time	1 segment per video sequence
144-pixel file size	1 segment per video sequence

Fig. 5 Scheduled ATT comparison for 190 *Beauty* video segments

Fig. 6 PTT versus ATT for the ANN model in Table 5

between ATT and PTT, the Pearson's correlation has a high value of 0.97, which helps in scheduling the transcoding tasks with a low ATT.

Fig. 7 PTT versus ATT for 855 *Beauty* video transcodings using nine encoding presets, 19 bitrates and five 4 s video segments

5.2 Quality, transcoding time, and throughput analysis

Table 8 analyses the video quality, processing time and throughput on a total of 41,040 transcoding operations with different presets. For all transcoding tasks with the same preset, we measured the average of two quality metrics: (i) weighted Peak Signal-to-Noise Ratio (wPSNR) for the luminance (Y) and chrominance (UV) components according to [63], and (ii) average YPSNR [64]. We also calculated the average ATT and bitrate for all presets and transcodings. The average quality for both metrics increases from ultrafast (includes less encoding features) to a veryslow preset (includes more encoding features). Interestingly, the average bitrate of the transcoded files slightly drops for more complex presets. The ATT significantly grows from ultrafast to veryslow presets (i.e. from 3.2 to 65.4 s), which shows the importance of choosing the correct preset that provides balanced quality, transcoding speed, and file size (or bitrate). While the default FFmpeg preset is medium, Table 8 shows that the faster and fast presets have the same YPSNR quality, but a slight difference in wPSNR and average bitrate compared to the medium preset. Important is that the veryfast, faster and fast presets have lower average ATT than the medium preset (i.e. 6 s, 9.1 s and 11.1 s versus 13.5 s). The veryfast, faster and fast presets have less than 1% variation in YPSNR and wPSNR compared to medium preset, however, their average ATT decreases by more than 55%, 32% and 17%, respectively. Thus, we recommend using one of the three veryfast, faster or fast presets based on this analysis.

5.3 Saving time analysis

We introduce two additional metrics to evaluate the effective gain achieved by FastTTPS compared to the OLB method employed in industry:

- *PTT saving time* is real total transcoding time saved by the Max–Min scheduling algorithm applied on the PTT of the video segments;
- *ATT saving time* is the ideal (i.e. 100% accurate PTT) transcoding time saved by the Max–Min algorithm applied on the ATT of the video segments.

Figures 8a and b show the PTT and ATT saving times of all ten video sequences with nine presets and 19 bitrates for 2 s and 4 s video segments on increasing number of cores. Each graph title represents the video sequence name along with the number of transcodings. FastTTPS attains the highest saving time on more than 70 cores for the 2 s animation video segments (i.e. BBB, Sintel, TOS), and on

Table 8 Performance analysis of video transcodings on different presets

#	Encoding preset	Avg YPSNR (dB)	Avg wPSNR (dB)	Avg ATT (s)	Avg bitrate (kbit/s)
1	Ultrafast	37.3	39.1	3.2	5065
2	Superfast	38.6	40.3	4.0	5091
3	Veryfast	39.2	40.9	6.0	4966
4	Faster	39.5	41.1	9.1	4920
5	Fast	39.5	41.3	11.1	4925
6	Medium	39.5	41.3	13.5	4889
7	Slow	39.6	41.4	18.7	4859
8	Slower	39.7	41.5	34.1	4824
9	Veryslow	39.7	41.5	65.4	4738

50 cores while for the rest. Similarly, it achieves the maximum saving time on 70 cores for the 4 s animation video segments, and between 20 and 30 cores for the rest. The maximum saving time is 36% for the 2 s video segments and 38% for the 4 s video segments. FastTTPS shows a PTT saving time with a maximum loss of 4% (at the spike of the curve) compared to the ATT saving time for 2 s animation video segments, and between 0 and 2% for the rest. Similarly, the PTT saving time loss varies between 0 and 6% for 4 s animation video segments, and between 0 and 3% for the rest.

Figures 9a and b depict the difference between the PTT and ATT saving times for the 2 s and 4 s segments of all video sequences on an increasing number of cores. The comparison shows negligible difference up to 15 cores and a negative difference between 15 and 35 cores for general-type videos. FastTTPS works better than ideal scenario for 2 s segments of general-type videos between 15 and 35 cores. For 4 s segments, FastTTPS works well for non-animated videos regardless the number of cores and increases the gap by 10% over the ATT saving time for animated videos using more than 40 cores. Between 40 and 60 cores, FastTTPS under-performs by up to 4% for 2 s segments of all videos types. Beyond 60 cores, FastTTPS achieves again similar PTT and ATT saving times for all non-animated videos, with a difference between 0 and 0.3%. We conclude that FastTTPS works better in both segment scenarios and for all videos types for less than 40 cores, beyond which it loses its advantage especially for animation videos.

Similarly, Fig. 10 analyses the PTT saving time for the nine x264 encoding presets by applying the Max–Min algorithm on 1710 2 s segments of the *Beauty* video on an increasing number of cores. The figure shows that FastTTPS gains up to 40% saving time, with the best results between 40 and 60 cores.

5.4 Statistical preset analysis

To find the performance difference among the presets, we statistically compared saving time on 1710 *Beauty* video transcoding tasks of the medium (default) preset with the rest eight presets using the paired sample t-test [33, 65] with two competing hypotheses:

- *Null hypothesis* H_0 assumes that the true mean difference between saving time samples for two presets is zero: $\mu_1 = \mu_2$. Acceptance of H_0 means that there is no statistically significant difference between the two samples.
- *Alternative hypothesis* H_1 assumes that the true mean difference between the paired samples is not zero: $\mu_1 \neq \mu_2$.

Table 9 shows the analysis of the results using the following metrics:

- *Number of values* represents the twelve different sets of servers (i.e. 2, 4, 6, 8, 10, 20, 30, 40, 50, 60, 70 and 90), as shown in Fig. 10.
- *Mean value* is the average saving time percentage of all transcoding tasks for each preset on different number of servers. The superfast, veryfast, faster, fast and slow presets have less than 10% difference from the mean value of medium preset, while the ultrafast, slower and veryslow presets have more than 10% difference.
- *Standard deviation* measures the amount of dispersion of saving times for each preset. The superfast, veryfast, fast and slow presets have very similar standard deviation (with a difference of less than 5%) compared to medium preset, while the other ultrafast, slower and veryslow presets have considerably larger differences.
- *Degree of freedom (df)* is the number of independent values used for statistical analysis. Ideally, it is one less than the number of values, i.e. eleven.
- *t-value* evaluates a t-test, which is an inferential statistic used to determine significant differences between the means of two groups related to certain features. A

Fig. 8 PTT versus ATT saving time for ten video sequences with nine presets and 19 bitrates

t-value of 0 indicates that the sample is exactly equal the null hypothesis. As the difference between the sample data and the null hypothesis increases, the absolute t-value increases too. The superfast, fast and slow presets have less difference in t-values with respect to the medium preset, while ultrafast, veryfast and faster presets are considerably different. On the other hand, the slower and veryslow presets have significantly larger t-value differences compared to the medium preset.

- *p-value* is a quantitative measure of the statistical probability of the observed result (as extreme as it can) by assuming that H_0 is true. A small p-value (e.g. ≤ 0.5) means that the extreme observed result is very unlikely under that null hypothesis. The slower and veryslow are the only presets with the p-value less than 0.05, while rest have larger p-values. This means that we reject the slow and veryslow presets for the given H_0 .

Fig. 9 ATT versus PTT saving time difference for all ten video sequences on increasing number of cores

Fig. 10 PTT saving time on 1710 *Beauty* video transcoding tasks with nine presets and increasing number of cores

Overall, the results show that the medium preset has a statistically significant difference (and accordingly rejects H_0) of saving time compared to the slower and veryslow preset and these two presets have a relatively lower mean saving time than the others. The remaining six presets (i.e. ultrafast, superfast, veryfast, faster, fast, slow) do not present significant difference and accept H_0 with a significance of 0.05.

5.5 Summary

Table 10 summarises best transcoding preset for different computing cores. The fast preset performs best for 40 or

less number of cores, and worsens its performance between 65 and 75 cores. The veryfast and medium presets perform best for 50 cores, while below 50 cores the veryfast preset performs better. The superfast preset gives best results between 60 and 68 cores and the faster preset between 70 and 75 cores. Interestingly, ultrafast delivers the lowest saving time until 75 cores and the highest above. The slow, veryslow and slower presets give only average results above 75 cores, the latter ones being the slowest. The statistical analysis in Table 9 and the results in Table 10 confirm the difference and the reduced saving time for the slower and veryslow encoding presets. Overall, the fast preset achieved best saving time up to 40%, followed by veryfast, medium, superfast, slow, slower, veryslow, faster and ultrafast.

6 Conclusions and future work

We presented a new method called FastTTPS for predicting the transcoding time of video segments and schedule them on a high-performance parallel computer based on three phases: transcoding data engineering, time prediction and scheduling. The first phase prepares the transcoding data using a video selection and segmentation step, followed by feature and data creation. The second phase predicts the transcoding time using an ANN model trained for accurate prediction using a novel 144-pixel transcoding method. Experimental results show that employing a sample-144-pixel dataset with a few video segments from each sequence as input feature is able to produce better ANN training and more accurate PTT results than the complete dataset. Finally, the third step integrates higher-complexity scheduling algorithms that exploit the predicted transcoding time to achieve high quality mappings of the transcoding tasks on a parallel machine.

We performed a detailed analysis using various performance parameters such as YPSNR, wPSNR and average bitrate of transcoded video files for different presets. We found out that the veryfast, faster and fast presets save more than 55%, 32%, and 17% average ATT compared to medium preset, with a compromise of less than 1% in YPSNR and wPSNR of transcoded segments. We further evaluated our method on a set of ten heterogeneous video sequences of different type with different duration and frame rates. Experimental results show that our ANN model was able to predict the video transcoding time by minimizing the MAE up to 1.7 for x264 encoded video sequences. On top of it, a Max–Min scheduler improved the transcoding time by 38% compared to a simple OLB method practiced in industry without prediction information.

Table 9 Statistical comparison of saving time on 1710 *Beauty* video transcoding tasks for medium (default) preset against the other eight presets

Encoding preset	No. of values	Mean value	Standard deviation	df	t value	p value	H_0 hypothesis
Medium	12	17.79	13.30	11	–	–	–
Ultrafast	12	14.07	11.32	11	1.562	0.147	Accept
Superfast	12	17.71	13.91	11	0.096	0.926	Accept
Veryfast	12	18.45	13.92	11	–1.411	0.186	Accept
Faster	12	16.57	11.92	11	1.048	0.317	Accept
Fast	12	18.40	13.81	11	–0.503	0.625	Accept
Slow	12	17.76	12.79	11	0.152	0.882	Accept
Slower	12	15.59	12.07	11	2.965	0.013	Reject
Veryslow	12	14.66	11.57	11	2.883	0.015	Reject

Table 10 Best transcoding preset for different parallel core ranges

Core range	Best preset
0–45	Fast
45–55	Veryfast
55–68	Superfast
68–75	Faster
75–90	Ultrafast

The FastTTPS approach can significantly improve the processing speed of transcoding services and infrastructures by video complexity analysis, transcoding time prediction and scheduling. As a result of our work, we consider the entire transcoding workflow and propose important insights and tools for improving transcoding infrastructures through a:

- fast method for video transcoding time prediction based on sample segments of the video file;
- prediction-based scheduling method that saves transcoding processing time;
- set of preferred x264 encoding presets in terms of quality and saving time.

We plan in the future to extend our method for predicting the transcoding time for long video sequences and using multiple codecs on heterogeneous computers, including different cloud computing instances. Furthermore, we plan to develop an intelligent autotuner to automate the process of transcoding. Finally, we will develop a reinforcement learning method to accurately learn and manage the operation of heterogeneous transcoding by assigning transcoding tasks to the most appropriate processing units.

Acknowledgements This work received support from: Austrian Research Promotion Agency (FFG) under Grant Agreement 877503 (APOLLO Project); European Union Horizon 2020 research and innovation programme under Grant Agreement 801091 (ASPIDA Project).

Funding Open access funding provided by University of Klagenfurt.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Cisco annual internet report (2018–2023). Technical report. Cisco, San Jose (2020)
2. Zbrovskiy, A., Feldmann, C., Timmerer, C.: Multi-Codec DASH Dataset. In: Proceedings of the 9th ACM Multimedia Systems Conference, MMSys '18. ACM, New York, pp. 438–443 (2018)
3. Sodagar, I.: The MPEG-DASH standard for multimedia streaming over the internet. *IEEE MultiMedia* **18**(4), 62–67 (2011)
4. HTTP Live Streaming draft-pantos-http-live-streaming-20. Internet-draft, Apple Inc. (2016)
5. Zbrovskiy, A., Petrov, E., Kuzmin, E., Timmerer, C.: Evaluation of the performance of adaptive HTTP streaming systems. CoRR. [arXiv:1710.02459](https://arxiv.org/abs/1710.02459) (2017)
6. Barman, N., Martini, M.G.: QoE modeling for HTTP adaptive video streaming—a survey and open challenges. *IEEE Access* **7**, 30831–30859 (2019)
7. Guo, Y., Yu, F.R., An, J., Yang, K., Yu, C., Leung, V.C.M.: Adaptive bitrate streaming in wireless networks with transcoding at network edge using deep reinforcement learning. *IEEE Trans. Veh. Technol.* **69**(4), 3879–3892 (2020)
8. Seufert, M., Egger, S., Slanina, M., Zinner, T., Hossfeld, T., Tran-Gia, P.: A survey on quality of experience of HTTP adaptive streaming. *IEEE Commun. Surv. Tutorials* **17**(1), 469–492 (2015)
9. Ma, L., et al.: An efficient scheduling multimedia transcoding method for DASH streaming in cloud environment. *Clust. Comput.* **22**, 1043–1053 (2017)

10. Hou, X., Dey, S., Zhang, J., Budagavi, M.: Predictive Adaptive streaming to enable mobile 360-degree and VR experiences. *IEEE Trans. Multimedia* **22**, 1–16 (2020)
11. Schatz, R., Zabrovskiy, A., Timmerer, C.: Tile-based Streaming of 8K Omnidirectional Video: subjective and objective QoE evaluation. In: *Proceedings of the 11th International Conference on Quality of Multimedia Experience (QoMEX)*, pp. 1–6 (2019)
12. Ozer, J.: AV1 encoding times drop to near-reasonable levels. <https://www.streamingmedia.com/Articles/ReadArticle.aspx?ArticleID=130284> (2019)
13. Deneke, T., Lafond, S., Lilius, J.: Analysis and transcoding time prediction of online videos. In: *proceedings of the IEEE International Symposium on Multimedia (ISM)*, pp. 319–322 (2015)
14. Tewodros, D.: *Proactive Management of Video Transcoding Services*. Ph.D. Dissertation, Abo Akademi University (2017)
15. Li, X., Salehi, M.A., Joshi, Y., Darwich, M.K., Landreneau, B., Bayoumi, M.: Performance analysis and modeling of video transcoding using heterogeneous cloud services. *IEEE Trans. Parallel Distrib. Syst.* **30**(4), 910–922 (2019)
16. Li, X., Salehi, M.A., Bayoumi, M.: VLSC: Video live streaming using cloud services. In: *Proceedings of the IEEE International Conferences on Big Data and Cloud Computing (BDCloud)*, pp. 595–600 (2016)
17. Zakerinasab, M.R., Wang, M.: Does chunk size matter in distributed video transcoding? In: *Proceedings of the IEEE 23rd International Symposium on Quality of Service (IWQoS)*, pp. 69–70 (2015)
18. Andujar, R., Peletier, I., Oliva, J., Cymontkowski, M., Fu, Y., Tang, E., Allman, J.: Video tampering detection for decentralized video transcoding networks. In: *Campilho, A., Karray, F., Wang, Z. (eds.) Image Analysis and Recognition*, pp. 16–327. Springer, Cham (2020)
19. Zabrovskiy, A., Agrawal, P., Mathá, R., Timmerer, C., Prodan, R.: ComplexCTTP: complexity class based transcoding time prediction for video sequences using artificial neural network. In: *Proceedings of the 2020 IEEE Sixth International Conference on Multimedia Big Data (BigMM)*, pp. 316–325 (2020)
20. Pääkkönen, P., Heikkinen, A., Aihkissalo, T.: Online architecture for predicting live video transcoding resources. *J. Cloud Comput.* **8**(1), 1321–13224 (2019)
21. Kirubha, D., Ramar, K.: MCCA scheduling for enhancing QoS based video streaming for video surveillance applications. *Cluster Comput.* **22**(6), 13945–13955 (2019)
22. Munagala, V., Prasad, K.S.: Clustered entropy computing: a holentropy based encoding scheme for high efficient computing systems. *Cluster Comput.* **22**(1), 1429–1441 (2019)
23. Garcia-Carballeira, F., Calderon, A., Carretero, J.: Enhancing the power of two choices load balancing algorithm using round robin policy. *Clust. Comput.* (2020). <https://doi.org/10.1007/s10586-020-03139-6>
24. Chhabra, A., Singh, G., Kahlon, K.S.: Multi-criteria HPC task scheduling on IaaS cloud infrastructures using meta-heuristics. *Cluster Comput.* (2020). <https://doi.org/10.1007/s10586-020-03168-1>
25. Ebadifard, F., Babamir, S.M.: Autonomic task scheduling algorithm for dynamic workloads through a load balancing technique for the cloud-computing environment. *Cluster Comput.* (2020). <https://doi.org/10.1007/s10586-020-03177-0>
26. Milan, S.T., Rajabion, L., Darwesh, A., Hosseinzadeh, M., Navimipour, N.J.: Priority-based task scheduling method over cloudlet using a swarm intelligence algorithm. *Clust. Comput.* **23**(2), 663–671 (2020)
27. Aparicio-Pardo, R., Pires, K., Blanc, A., Simon, G.: Transcoding live adaptive video streams at a massive scale in the cloud. In: *Proceedings of the 6th ACM Multimedia Systems Conference MMSys '15*. ACM, New York, pp. 49–60 (2015)
28. Oikonomou, P., Koziri, M.G., Tziritas, N., Dadaliaris, A.N., Loukopoulou, T., Stamoulis, G.I., Khan, S.U.: Scheduling video transcoding jobs in the cloud. In: *Proceedings of the IEEE International Conference on Internet of Things (iThings)*, pp. 442–449 (2018)
29. Wei, L., Cai, J., Foh, C.H., He, B.: QoS-aware resource allocation for video transcoding in clouds. *IEEE Trans. Circ. Syst. Video Technol.* **27**(1), 49–61 (2017)
30. Jokhio, F., Deneke, T., Lafond, S., Lilius, J.: Bit rate reduction video transcoding with distributed computing. In: *Proceedings of the 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pp. 206–212 (2012)
31. Li, X., Salehi, M.A., Bayoumi, M., Tzeng, N., Buyya, R.: Cost-efficient and robust on-demand video transcoding using heterogeneous cloud services. *IEEE Trans. Parallel Distrib. Syst.* **29**(3), 556–571 (2018)
32. Zhao, H., Zheng, Q., Zhang, W., Wang, J.: Prediction-based and locality-aware task scheduling for parallelizing video transcoding over heterogeneous mapreduce cluster. *IEEE Trans. Circ. Syst. Video Technol.* **28**(4), 1009–1020 (2018)
33. Ghobaei-Arani, M., Rahmanian, A., Souri, A., Rahmani, A.M.: A moth-flame optimization algorithm for web service composition in cloud computing: simulation and verification. *Softw. Pract. Exp.* **48**(10), 1865–1892 (2018)
34. Sameti, S., Wang, M., Krishnamurthy, D.: CONTRAST: container-based transcoding for interactive video streaming. In: *Proceedings of the NOMS 2020—2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9 (2020)
35. Li, L., Shi, D., Hou, R., Chen, R., Lin, B., Pan, M.: Energy-efficient proactive caching for adaptive video streaming via data-driven optimization. *IEEE Internet Things J.* **7**(6), 5549–5561 (2020)
36. Ghobaei-Arani, M., Khorsand, R., Ramezanpour, M.: An autonomous resource provisioning framework for massively multi-player online games in cloud environment. *J. Netw. Comput. Appl.* **142**, 76–97 (2019)
37. Mahmud, R.: fogsim. <https://github.com/Cloudslab/iFogSim> (2020)
38. Deneke, T., Haile, H., Lafond, S., Lilius, J.: Video transcoding time prediction for proactive load balancing. In: *proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6 (2014)
39. Ma, H., Seo, B., Zimmermann, R.: Dynamic scheduling on video transcoding for MPEG DASH in the Cloud environment. In: *Proceedings of the 5th ACM Multimedia Systems Conference, MMSys '14*. ACM, New York, pp. 283–294 (2014)
40. Isaac, A.O., Jantan, A., Omolara, A.E., Dada, K.V., Mohamed, A.E.N., Arshad, H.: State-of-the-art in artificial neural network applications: a survey. *Heliyon* **4**(11), e00938 (2018)
41. Mathew, T., Sekaran, K.C., Jose, J.: Study and analysis of various task scheduling algorithms in the cloud computing environment. In: *proceedings of the 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 658–664 (2014)
42. Malik, B.H., Amir, M., Mazhar, B., Ali, S., Jalil, R., Khalid, J.: Comparison of task scheduling algorithms in cloud environment. *Int. J. Adv. Comput. Sci. Appl.* (2018). <https://doi.org/10.14569/IJACSA.2018.090550>
43. Ali, S.A., Alam, M.: A relative study of task scheduling algorithms in cloud computing environment. In: *Proceedings of the 2nd International Conference on Contemporary Computing and Informatics (IC3I)*, pp. 105–111 (2016)
44. Balharith, T., Alhaidari, F.: Round robin scheduling algorithm in CPU and cloud computing: a review. In: *Proceedings of the 2nd International Conference on Computer Applications Information Security (ICCAIS)*, pp. 1–7 (2019)

45. Maheswaran, M., Ali, S., Siegel, H. J., Hensgen, D., Freundm, R. F.: Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In: Proceedings of the 8th Heterogeneous Computing Workshop (HCW'99), pp. 30–44 (1999)
46. Chen, N., Helian Huankai, Wang, F., Akanmu, G.: User-priority guided Min-Min scheduling algorithm for load balancing in cloud computing. In: Proceedings of the National Conference on Parallel Computing Technologies (PARCOMPTECH), pp. 1–8 (2013)
47. Konjaang, J. K., Maipanuku, J.Y., Kubuga, K. K.: An efficient Max-Min resource allocator and task scheduling algorithm in cloud computing environment. CoRR. [arXiv:1611.08864](https://arxiv.org/abs/1611.08864) (2016)
48. Mao, Y., Chen, Xi C., Li, Xiao fang: Max-Min task scheduling algorithm for load balance in cloud computing. In: Proceedings of the International Conference on Computer Science and Information Technology, pp. 457–465 (2014)
49. Zabrovskiy, A., Kuzmin, E., Petrov, E., Timmerer, C., Mueller, C.: AdViSE: adaptive video streaming evaluation framework for the automated testing of media players. In: Proceedings of the 8th ACM on Multimedia Systems Conference, MMSys'17, ACM, New York, pp. 217–220 (2017)
50. Lederer, S., Müller, C., Timmerer, C.: Dynamic adaptive streaming over HTTP dataset. In: Proceedings of the 3rd Multimedia Systems Conference, MMSys '12. ACM, New York, pp. 89–94 (2012)
51. Bouzakaria, N., Concolato, C., Feuvre, J.L.: Overhead and performance of low latency live streaming using MPEG-DASH. In: Proceedings of the 5th International Conference on Information, Intelligence, Systems and Applications, pp. 92–97 (2014)
52. Le Feuvre, J., Thiesse, J.-M., Parmentier, M., Raulet, M., Daguet, C.: Ultra high definition HEVC DASH data set. In: Proceedings of the 5th ACM Multimedia Systems Conference, MMSys'14, ACM, New York, pp. 7–12 (2014)
53. Lederer, S. et al.: Distributed DASH dataset. In: Proceedings of the 4th ACM Multimedia Systems Conference, MMSys'13, ACM, New York, pp. 131–135 (2013)
54. Quinlan, Jason J. et al.: Datasets for AVC (H.264) and HEVC (H.265) evaluation of dynamic adaptive streaming over HTTP (DASH). In: Proceedings of the 7th International Conference on Multimedia Systems, MMSys'16, ACM, New York, pp. 51:1–51:6 (2016)
55. Ozer, J.: How Netflix pioneered per-title video encoding optimization. [http://bit.ly/2H7fzAD](https://bit.ly/2H7fzAD)(2018)
56. Twitch. Broadcasting guidelines. <https://stream.twitch.tv/encoding> (2019)
57. IBM Watson Media Support Center. Internet connection and recommended encoding settings. <https://support.ustream.tv/hc/en-us/articles/207852117-Internet-connection-and-recommended-encoding-settings> (2019)
58. YouTube. Choose live encoder settings, bitrates, and resolutions. <https://support.google.com/youtube/answer/2853702> (2018)
59. Bitmovin. Video Developer Report 2019. <https://go.bitmovin.com/video-developer-report-2019> (2019)
60. Zeiler, M.D.: ADADELTA: an adaptive learning rate method. CoRR. [arXiv:1212.5701](https://arxiv.org/abs/1212.5701) (2012)
61. Pham, H.T., Nguyen, T.D., Pham, N.N., Nguyen, H.T., Nguyen, H.M., Truong, T.H.: An efficient QoE-Aware HTTP adaptive streaming over software defined networking. *Mobile Netw. Appl.* **80**, 1–13 (2020)
62. Sheugh, L., Alizadeh, S.H.: A note on Pearson correlation coefficient as a metric of similarity in recommender system. In: Proceedings of the AI Robotics (IRANOPEN), pp. 1–6 (2015)
63. Ohm, J., Sullivan, G.J., Schwarz, H., Tan, T.K., Wiegand, T.: Comparison of the coding efficiency of video coding standards-

including high efficiency video coding (HEVC). *IEEE Trans. Circ. Syst. Video Technol.* **22**(12), 1669–1684 (2012)

64. Zabrovskiy, A., Feldmann, C., Timmerer, C.: A practical evaluation of video codecs for large-scale http adaptive streaming services. In: Proceedings of the 25th IEEE International Conference on Image Processing (ICIP), pp. 998–1002 (2018)
65. Ghobaei-Arani, M., Rahmanian, A.A., Aslanpour, M.S., Dashti, S.E.: CSA-WSC: cuckoo search algorithm for web service composition in cloud environments. *Soft Comput.* **22**, 8353–8378 (2018)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Prateek Agrawal completed his B.Tech and M.Tech. in computer science engineering from Uttar Pradesh Technical University, India and ABV-IIITM, India respectively. He received his Ph.D. from IKG-Punjab Technical University, India. He is having more than 10 years of research and teaching experience. He is currently working as post-doctoral researcher at department of ITEC, University of Klagenfurt, Austria and also serves as

associate professor in school of computer science engineering, Lovely Professional University, India. He authored more than 50 research papers in various peer reviewed journals and conferences. He is life member of IET. He is reviewer and editorial member of many journals of high repute. He also served as technical program committee member of many international conferences. His major research areas are Natural Language Processing, Machine Learning, Image Processing, Scheduling, Parallel Processing and Big data etc.

Anatoliy Zabrovskiy received his B.S. and M.S. degrees in information and computer technology from Petrozavodsk State University, Russia in 2006 and 2008 respectively, and a Ph.D. degree in engineering from the same university in 2013. He has been working in the field of network and multimedia communication technologies for over ten years. He was a Cisco certified academy instructor for CCNA. He was award winner of two international programs:

Scholarships of the Scholarship Foundation of the Republic of Austria for Postdocs and Erasmus Mundus External Cooperation Window program for doctorate students. He was a prize winner of Sun Microsystems contest “Idea2Project”. He is currently a post doctoral researcher at the Department of Information Technology (ITEC), University of Klagenfurt, Austria. He is a member of the Technical Program Committee of several international conferences. His research interests include Video streaming, Network technologies, Quality of Experience, and Machine Learning.

multimedia delivery formats.

SOLE, and the COST Action IC1003 QUALINET. He also

Adithyan Ilango received the B.Tech degree from IIT Hyderabad, Telangana, India, and the M.Sc. degree in communications engineering from the Technische Universität München (TUM), Munich, Germany, in 2015. Since then, he has been working in the field of multimedia technologies for over six years. He is currently a Video Encoding Engineer at Bitmovin. His research interests include video streaming, cloud based video encoding, and

Christian Timmerer is an associate professor with University of Klagenfurt, Austria. He is a Co-Founder of Bitmovin Inc., San Francisco, USA, as well as the CIO and the Head of Research and Standardization. He has co-authored seven patents and over 200 publications in workshops, conferences, journals, and book chapters. He participated in several EC-funded projects, notably DANAE, ENTHRONE, P2P-Next, ALLCANTO, SocialSensor, ICo-

participated in ISO/MPEG work for several years, notably in the areas of MPEG-21, MPEG-M, MPEG-V, and MPEGDASH. His research interests include immersive multimedia communications, streaming, adaptation, and quality of experience. He was the General Chair of WIAMIS 2008, QoMEX 2013, ACM MMSys 2016, and Packet Video 2018. Further information can be found at <http://blog.timmerer.com>.

than 100 publications and received two IEEE best paper awards.

Radu Prodan (Member, IEEE) received the Ph.D. degree from the Vienna University of Technology, in 2004. He was an associate professor until 2018 at the University of Innsbruck, Austria. He is currently professor in distributed systems at the Institute of Software Technology, University of Klagenfurt, Austria. His research interests include performance, optimisation, and resource management tools for parallel and distributed applications. He authored more