TOWARDS QOS-AWARE CLOUD LIVE TRANSCODING: A DEEP REINFORCEMENT LEARNING APPROACH

Zhengyuan Pang[1], *Lifeng Sun*[1], *Tianchi Huang*[1], *Zhi Wang*[2], *Shiqiang Yang*[1]

[1] Department of Computer Science and Technology, Tsinghua University
[2]Graduate School at Shenzhen, Tsinghua University
{pangzy12@mails., sunlf@mail., htc17@mails., wangzhi@sz., yangshq@mail.}tsinghua.edu.cn

ABSTRACT

Video transcoding is widely adopted in live streaming services to bridge the format and resolution gap between content producers and consumers (i.e., broadcasters and viewers). Meanwhile, the cloud has been recognized as one of the most reliable and cost-effective ways for video transcoding. However, due to the dynamic and uncertainty of the transcoding workloads in live streaming, it is very challenging for cloud service providers to provision computing resources and schedule transcoding tasks while guaranteeing the Service Level Agreement (SLA). To this end, we propose a joint resource provisioning and task scheduling approach for transcoding live streams in the cloud. We adopt Deep Reinforcement Learning (DRL) to train a neural network model for resource provisioning under dynamic workloads. Moreover, we design a QoS-aware task scheduling algorithm that maps transcoding tasks to Virtual Machines (VMs) by considering the real-time QoS requirement. We evaluate our approach with trace-driven experiments and the results demonstrate that our approach outperforms heuristic baselines by up to 89% improvements on average QoS with 4% extra resource overhead at most.

***Index Terms*—** Cloud transcoding, live streaming, deep reinforcement learning, resource provisioning

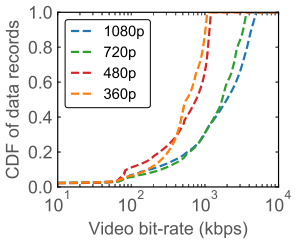
1. INTRODUCTION

Nowadays online live streaming service has become enormously popular, such as Twitch, YouTube Live, and so on. On these platforms, numerous viewers watch online videos produced by large amounts of amateur broadcasters (e.g., gamers, dancers, etc.). As the devices used by broadcasters and viewers get increasingly more diverse, it becomes necessary to transcode video contents into different quality versions, and deliver appropriate versions to viewers to match their device and network characteristics. Since video transcoding is extremely computation-intensive and requires a huge amount of hardware, cloud computing offers a natural way to solve this problem due to its elasticity and the "pay-as-you-go" billing model. With the rapid growth of this market, many cloud service providers such as Amazon, Azure, etc, have delivered cloud transcoding services to assist live streaming platforms handling their transcoding tasks.

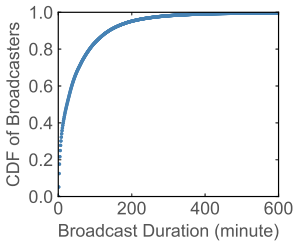
Cloud service providers offer transcoding services to their users with a specific guarantee of the Quality of Service (QoS), which also appears as the Service Level Agreement (SLA) [1]. For instance, live content must be transcoded in real-time. However, the transcoding workloads of live streams are dynamic and uncertain, which makes it a great challenge for cloud service providers to provision computing resources and schedule transcoding tasks while meeting the SLA requirement.

First, broadcasters could start or end uploading streams at any time and the uploaded video streams have various quality levels, with a variety of resolutions, bit-rates, etc. Therefore, the cloud service providers are not aware of how many resources will be needed by transcoding tasks at the next moment. In Fig. 1 we illustrate the uncertainty of broadcasters and the diversity of the uploaded videos using a dataset collected on Kuaishou, one of the most popular live streaming platforms in China. We observe that the bit-rates of uploaded streams cover a wide range from 10 to more than 6000 kbps and the broadcasting durations vary from minutes to hours. Second, in live streaming, the execution time of each transcoding task is unknown until the task is completed [2], which makes it different from the case of Video on Demand (VoD). Therefore, it is challenging for cloud service providers to schedule transcoding tasks with unknown execution time while meeting the real-time requirement.

Many recent research works have focused on the problem of resource provisioning and QoS management for cloud transcoding. However, most of the existing works tried to solve the problem in VoD streaming, in which the transcoding workloads are known in prior and the exact transcoding time of a source video is estimated by empirical study [3] or from the historical execution information of the video [4]. Moreover, most of the previous works did not consider system uncertainties such as the variation of transcoding speed or the Virtual Machine (VM) startup time, which could make significant impact on the real-time requirement.



(a) Video bit-rates distributions of different resolutions.



(b) Distribution of the broadcasting durations.

Fig. 1: Statistics of live broadcasting dataset.

In this paper, we attempt to solve the problem of provisioning computing resources under dynamic and uncertain transcoding workloads as well as scheduling transcoding tasks with unknown execution time and real-time QoS requirement. Recent breakthroughs of Deep Reinforcement Learning (DRL) in complicated control problems [5] [6] prove that DRL is well suited for decision making in a complex, uncertain environment without any explicit assumptions about the environment. Inspired by these results, we are motivated to enable DRL for dynamic resource provisioning in the cloud.

To this end, we propose a joint resource provisioning and task scheduling approach for transcoding live streams in the cloud. We use DRL to train a neural network (NN) model that periodically makes resource provisioning decisions. The system uncertainties are taken into consideration in the design of the simulation environment for training the model. Moreover, we propose a method to estimate the bounds of the transcoding time. Based on the bounded estimations, we design a transcoding task scheduling policy which is aware of the real-time QoS requirement.

In summary, this paper makes the following contributions:

- We propose a joint resource provisioning and task scheduling approach for cloud live transcoding with strict SLA requirement.
- We design a DRL-based method for resource provisioning under dynamic transcoding workloads.
- We present a QoS-aware task scheduling algorithm to map transcoding tasks to VM instances by considering the real-time QoS requirement.
- The results of trace-driven experiments show that our approach achieves up to 89% improvements on average QoS with only 4% extra resource overhead.

2. SYSTEM MODEL AND WORKFLOW

In this section, we introduce the system model and describe the system workflow.

Time model: Similar to [7], we consider a two-level time model which is adapted to our system. At the first level, we adopt a discrete time model, where the time slot is denoted as $t = 0, 1, 2....$ Each time slot is T seconds long. Unless otherwise noted, we use the term "time slot t" and "time step t" interchangeably in the rest of our paper. At the second level within each time slot, we use a continuous time model, where the system time is denoted as $\widehat{t}$.

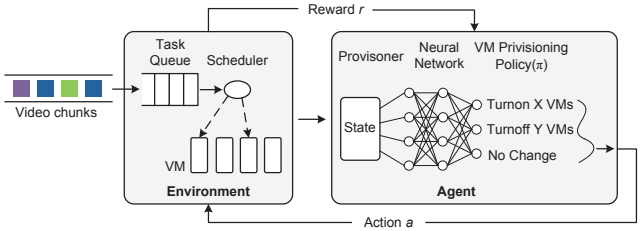


Fig. 2: Overview of the system workflow.

VM model: We consider a cloud system with M VM instances with homogeneous computing performance. Let $V = \{vm_1, vm_2, ..., vm_M\}$ be the whole VM instance set. At any time $\widehat{t}$, a VM instance vm_i can be active or turned off. When an active VM instance is idle, it can be assigned a new task by the task scheduler.

Video model: A live video stream v_i is segmented into a set of consecutive video chunks v_{ij} with equal duration τ. Each video chunk will be transcoded into a set of predefined formats. We define a *transcoding pattern* as a tuple of source chunk format and target chunk format, denoted by $p := \{$source resolution, source bit-rate, target resolution, target bit-rate$\}$. We denote the transcoding task of chunk v_{ij} in pattern p as x_{ij}^p , or simply x for short.

Cost model: The system cost is comprised of the computing cost of active VM instances and the QoS cost caused by violating the real-time constraint. The overall computing cost in time slot t can be calculated as:

$$c(t) = c_v \cdot n_t, \tag{1}$$

where c_v is the computing cost incurred by an active VM instance in one time slot and n_t is the number of active VM instances in time slot t.

In live video streaming, due to the real-time constraint, the transcoding task for each video chunk has an individual hard deadline related to the presentation time of the chunk. We denote the number of tasks missing the deadlines in time slot t as y_t, and the total number of tasks in t as z_t. The QoS cost in one time slot can be represented by the deadline violation percentage (DVP), which is calculated as:

$$d(t) = y_t/z_t \cdot 100\%. \tag{2}$$

It is worth noting that since cloud service providers make the SLA with their users, there is a threshold η of the deadline violation percentage. Specifically, if the SLA of transcoding service is ϕ (in percentage), $\eta = (100 - \phi)\%$ and $d(t)$ should not exceed η.

System workflow: As shown in Fig. 2, in DRL, the system consists of two components: the *environment* and the *agent*.

The environment includes a task queue Q, a task scheduler and M VM instances, while the agent refers to the resource provisioner. The agent trains the NN model in an offline fashion by experiencing historical transcoding task traces and use the trained model to make online resource provisioning decisions. At each time step t, the agent observes a set of metrics as the state s_t of the environment. The neural network then output action a_t based on the state, which determines the number of active VM instances in the next time slot. Within the time slot, transcoding tasks of arrived video chunks are put into the task queue and dispatched to idle VM instances by the scheduler. After that, the state of the environment transits to s_{t+1} and the overall system cost is passed to the agent as the reward r_t. The objective of the learning process is to maximize the expected cumulative discounted reward $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$, where $\gamma \in (0, 1]$ is the discounting factor.

3. DRL-BASED RESOURCE PROVISIONING

In this paper, instead of using preset rules in the form of fine-tuned heuristics, we attempt to let the neural network learn a resource provisioning policy purely through experience. In this section, we describe the design and implementation of our DRL based resource provisioning algorithm.

Simulation environment: To accelerate the training process, we need to design a simulation environment that faithfully models the real world workflow and system behaviors. We build up the simulation environment based on the system model introduced in Sec. 2. The details of the environment are present below.

Table 1: Transcoding Templates

Source Resolutions	Source Bit-rates (kbps)	Target Formats
1080p	3000 $\sim$ 6000	720p, 2500 kbps
1080p	3000 $\sim$ 6000	480p, 1500 kbps
720p	2000 $\sim$ 4000	480p, 1500 kbps

To simulate the transcoding time, we measure the execution time of converting a set of videos into different formats. Following the recommendations of live encoder settings on YouTube [8], we define three transcoding templates in our system. The characteristics of the templates are given in Table. 1. As an example, the first row in Table. 1 means that source chunks with 1080p resolution and bit-rates ranging from 3000 to 6000 kbps will be transcoded into chunks with 720p resolution and 2500 kbps bit-rate. We use the video dataset provided by [9], which contains twenty high-quality 1080p videos of different content types and all the videos have an equal length of 10 seconds. We convert the videos in the dataset to the source formats defined in Table. 1, which generates 1040 CBR videos. Then we measure the transcoding time of these videos using the templates in Table. 1 with FFmpeg. Each experiment is repeated 30 times and the mean value is taken. With the measurements, the execution time $g(x)$ of the transcoding task x is set up as follows:

First, we select the measured results under the same transcoding pattern of x, denoted by m_p. Then we calculate the mean and standard deviation of m_p, denoted by m_p^{μ} and m_p^{σ} respectively. Finally, we randomly sample a bias e from [-m_p^{σ}, m_p^{σ}] and set $g(x) = m_p^{\mu} + e$.

To simulate the VM startup time, we adopt the results introduced in [10], where the boot time of a VM instance is between 2.5 and 5.5 seconds in general. In our environment, when a VM instance is turned on, we sample a random value from [2.5, 5.5] under uniform distribution as the startup time.

State: We define the workload w_t of time slot t as $\sum g(x)/T$, where x is the task arriving within t. At each time step, the agent feeds input state $s_t = \{\vec{w}_t, d_t, n_t\}$ to the neural network, where $\vec{w}_t$ is the workloads for the past k time slots; d_t is the deadline violation percentage in the last time slot; n_t is the number of active VM instances in the last time slot.

Action: After receiving s_t, the agent takes an action a_t under policy π, which is the resource provisioning decision of the next time slot. Intuitively, the simplest way to set the action space is directly letting $a_t = n_{t+1}$. However, it will significantly increase the size of the action space, which could make the training process difficult to converge. To this end, we set the action to be the number of VMs which are turned on or shutdown in the next time slot. Specifically, $a_t \in [-N, N]$ is an integer and defined as:

$$a_t := \begin{cases} \text{turn } a_t \text{ VMs on}, & \text{if } a_t > 0, \\ \text{do nothing}, & \text{if } a_t = 0, \\ \text{turn } a_t \text{ VMs off}, & \text{if } a_t < 0, \end{cases} \tag{3}$$

where N is a hyper-parameter that controls the maximum variation of active VMs.

Reward: Rewards are fed back to the agent with a delay as signals reflecting the inherent impacts of different actions on the overall system cost. We define the reward at each time step as:

$$r_t = -c(t)/w_t - \lambda \cdot f(d(t)), \tag{4}$$

where λ is the tuning parameter, and $f(d(t))$ is defined as:

$$f(d_t) = \begin{cases} \epsilon \cdot d(t) & d(t) < \eta, \\ d(t) & \text{otherwise}, \end{cases} \tag{5}$$

where $\epsilon < 1$ is the discounting factor. Note that by dividing w_t, the first term in Eq. 4 represents the VM provisioning efficiency under different workloads, and the second term takes into account the SLA requirement by downscaling the QoS cost when the deadline violation percentage is below the threshold.

Training: After applying each action, the agent receives reward r_t from the environment. The training process of the agent is to continuously modify the parameters of the neural network so that the neuron-represented policy π_θ is revised on the direction that maximizes the accumulated reward. In our

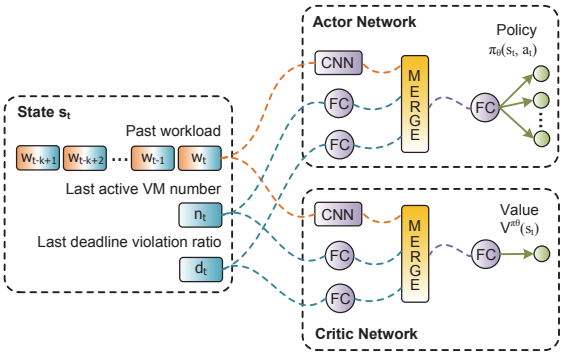


Fig. 3: The Actor-Critic algorithm of resource provisioning.

paper, we use A3C[11], a state of the art actor-critic learning method, as our training algorithm, where the policy training is done by performing policy gradient method. Specifically, as shown in Fig. 3, the neural network is comprised of an *actor-network* and a *critic-network*, parameterized by θ and θ_v respectively. The output of the actor-network is the policy $\pi_\theta(s_t, a_t)$ for selecting the action a_t in state s_t, which is represented by the adjustable parameters θ of the actor-network. The output of the critic-network is a value function $V^{\pi_\theta}(s;\theta_v)$, which is the estimation of the expected total reward starting at state s and following the policy π_θ. In A3C, there are multiple agents training the neural network in parallel, each of them owns an independent copy of the neural network and the simulation environment but experiences a different set of input traces. Inspired by [5], we use a central agent to update the parameters of the neural network periodically. At every time step during training, each agent collects a tuple of $\{s_t, a_t, r_t\}$. After every n steps, the agents send the collected tuples to a central agent, which aggregates them to perform a batch update to θ and θ_v. Each update of the actor-network parameter θ follows the policy gradient, which is the direction of increasing the accumulated reward. According to [11], the update of θ can be calculated as:

$$\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta log\pi_\theta(s_t, a_t)A(s_t, a_t) + \beta\nabla_\theta H(\pi_\theta(\cdot|s_t)), \tag{6}$$

where α is the learning rate of the actor network, β is a hyper-parameter, $A(s_t, a_t)$ is the advantage function defined as:

$$A(s_t, a_t) = r_t + \gamma V^{\pi_\theta}(s_{t+1};\theta_v) - V^{\pi_\theta}(s_t;\theta_v), \tag{7}$$

and $H(\cdot)$ is the entropy of the policy. The update of θ_v is calculated by Temporal Difference method[12] as follow:

$$\theta_v \leftarrow \theta_v - \alpha' \sum_t \nabla_{\theta_v}\big(A(s_t, a_t)\big)^2, \tag{8}$$

where α' is the learning rate of the critic-network. More details of the A3C algorithm can be found in [11].

Neural Network: As shown in Fig. 3, in the actor-network, variables of s_t are pushed into a 1D convolution (CNN) layer and two fully-connected (FC) layers separately. The CNN layer has 128 filters, each of size 4 with stride 1. The outputs of these layers are then merged and passed to the last FC layer with 128 neurons to form the results. Note that a softmax function is applied to the output of the last FC layer to get a normalized probability distribution over the action space. The critic-network has the same network structure as the actor-network, with the final softmax function replaced by a linear neuron.

4. QOS-AWARE TRANSCODING TASK SCHEDULING

In this section, we describe the transcoding task scheduling policy in the system. Let $\widehat{t}^a_{ij}$ be the arrival time of video chunk v_{ij}. For any x generated from v_{ij}, we denote the deadline of x as $b^d(x)$, which is the presentation time of v_{ij} and can be calculated as: $b^d(x) = \widehat{t}^a_{ij} + \delta$, where δ is the broadcasting delay configured by live service providers.

In live video streaming, the execution time of a transcoding task is unknown until the task is completed. To estimate the transcoding time, we follow the conclusion in [2] that the transcoding time of a chunk has a correlation with the transcoding time of other chunks within the same stream. It is reasonable because chunks of the same live stream usually have similar contents, such as game scenes or dancing rooms. Moreover, taking into consideration of system uncertainties, we estimate the upper and lower bounds of the execution time of transcoding tasks instead of the exact values. Specifically, the estimated upper bound of $g(x)$, i.e., the execution time of task x, is calculated as :

$$g^u(x^p_{ij}) = \mathrm{P}_{95th}\big(\{g(x^p_{iq}), \forall q < j\}\big), \tag{9}$$

where $\mathrm{P}_{95th}(X)$ means the 95^{th} percentile of the set X. Similarly, the estimated lower bound of $g(x)$ is calculated as:

$$g^l(x^p_{ij}) = \mathrm{P}_{5th}\big(\{g(x^p_{iq}), \forall q < j\}\big). \tag{10}$$

It is worth noting that the estimated upper and lower bounds represent the worst and best case estimations of the transcoding time respectively, and they are known prior to the execution of x. Then, we define the priority of task x as[1]:

$$b^u(x) := b^d(x) - g^u(x^p_{ij}), \tag{11}$$

and the *bottom line* of task x as:

$$b^l(x) := b^d(x) - g^l(x^p_{ij}). \tag{12}$$

Note that the bottom line $b^l(x)$ represents the latest time point that the task could meet the deadline.

We design our QoS-aware task scheduling policy based on $b^d(x)$, $b^u(x)$ and $b^l(x)$, which is detailed in Alg. 1. Basically, tasks with smaller $b^u(x)$ should be scheduled preferentially because the time they could wait in the task queue before the

[1]We assume that $g^u(x) <= \delta$, which means that the worst case transcoding time of a chunk should not exceed the broadcasting delay.

Algorithm 1: QoS-Aware Task Scheduling Policy

input :
$\widehat{t}$: current system time
Q_t : current task queue
V_t : current VM instance set
output:
ρ : task scheduling policy

for *each* $x \in Q_t$ **do**
  if $b^l(x) \leq \widehat{t}$ **then**
    Discard x
for *each* $vm_i \in V_t$ **do**
  if *task* x *is on* vm_j *and* $b^d(x) < \widehat{t}$ **then**
    Discard x
while *there is any idle* vm_i **do**
  Find x with the minimum $b^u(x)$ in Q_t
  Dispatch x to vm_i

deadlines is shorter. When a task executing on a VM instance misses the deadline $b^d(x)$, it will be discarded immediately, because there is no value for the task to continue. Moreover, tasks missing the bottom line $b^l(x)$ while still waiting in Q will be discarded either, since it is likely that they will miss the deadlines as well.

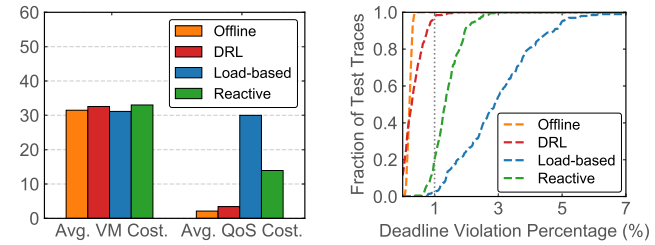
5. PERFORMANCE EVALUATION

In this section we first introduce the dataset and experiment settings, then we evaluate the performance of our approach and compare it to baseline methods.

5.1. Dataset and Experiment Settings

Dataset: To evaluate the performance of our approach on real world live streaming workloads, we use a broadcasting dataset collected on Kuaishou, as mentioned in §1. The dataset contains 1.3 million broadcasting traces in one month (March 2018 - April 2018). Each trace item records the start timestamp, the duration, the resolution and the bitrate of one live stream.

Experiment Settings: We generate transcoding workloads from the dataset above. We randomly select 10% from streams with 1080p and 720p resolutions respectively. The selected streams are then segmented into a series of 10-seconds video chunks. Each chunk generates a set of transcoding tasks based on its format and the transcoding templates introduced in Sec. 3, which end up with 21 million transcoding tasks. We randomly create 2000 task traces from this task set to form our corpus, with each trace spanning one hour. Unless otherwise noted, we randomly split the corpus into training (80%), validation (10%), and test (10%) sets. Note that the validation set is used for tuning



(a) Average cost of all test traces. (b) QoS cost distributions.

Fig. 4: Comparing the system cost under different resource provisioning policies.

Table 2: Simulation Parameters Setting.

Parameter	Value	Parameter	Value
Time slot length: T	60 seconds	VMs in the system: M	100
Video chunk duration: τ	10 seconds	Unit computing cost: c_v	1
Broadcasting delay: δ	5 seconds	Cost tuning parameter: λ	3
QoS threshold: η	1%	Action bound: N	3
DRL parameters:	k=16, n=60, ϵ=0.01, α=1e-4, α'=1e-3, γ=0.9, β=0.5		

hyper-parameters and the final performance comparison is conducted on the test set. During training and testing, we set the SLA requirement ϕ to 99% according to current industrial schemes [1]. The default values for other parameters in our experiments are summarized in Table 2. The training process takes 6 hours to converge with 10 agents working asynchronously. After the training is converged, we use the trained model to make decisions on the test set. The average execution time of one decision step is 5 milliseconds, which is acceptable in a real world system.

5.2. Comparison with Baseline Methods

Comparison of Resource Provisioning Policy: We compare the system cost under our DRL-based resource provisioning policy with the following baselines: 1) The Offline Policy (Offline) which knows the future workload information in prior and set n_t to be the minimum number that will not lead to $d(t) > \eta$. In our experiments, the offline policy provides the upper performance bound of resource provisioning. 2) The Workload based Policy (Load-based) which set n_t according to the workload of the last time slot, i.e., $n_t = \lceil w_{t-1} \rceil$. 3) The Dynamic Provisioning Policy (Reactive) proposed in [4] which allocates VM instances when $d(t)$ exceeds an upper bound threshold and deallocates one VM when $d(t)$ is less than a lower bound threshold. In our experiments, we set the upper and lower bounds in this method as $0.8 \cdot \eta$ and $0.5 \cdot \eta$ respectively.

The average system cost with different resource provisioning methods on our test set is shown in Fig. 4(a), where the average QoS cost is converted to *per-mille* value, i.e., $d(t) \cdot 10$ ‰ , for clarity. We can observe that the DRL-based policy achieves much better QoS than the two heuristic methods with close average VM cost. Specifically, compared

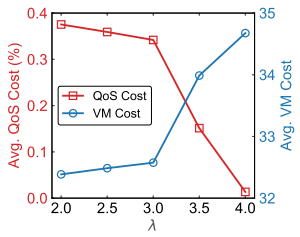


Fig. 5: The impact of the tuning parameter λ.

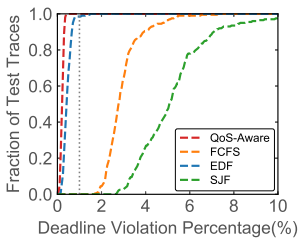


Fig. 6: QoS cost under different task scheduling schemes.

to the Load-based policy and the Reactive policy, the average deadline violation percentage under the DRL-based policy decreases up to 89% while the average VM cost increases 4 % at most. Furthermore, we present the distributions of the deadline violation percentage with these policies on our test set in Fig. 4(b). We can see that only the DRL-based policy and the Offline policy meet the SLA requirement effectively, while the other two heuristic methods fail to control the deadline violation percentage under 1%. It is obvious that simply provisioning the computing resources based on current information without looking ahead does not work well under dynamic workloads.

We evaluate the impact of the tuning parameter λ in Fig. 5. We observe that with the increase of λ, the DRL agent tends to allocate more VM instances to prevent the increase of the deadline violation percentage. This is reasonable because the larger λ will amplify the impact of $d(t)$ in the reward function (Eq. 4). Finally, the choice of λ is an overall consideration of the QoS threshold and the system budget.

Comparison of Task Scheduling Policy: We compare our task scheduling scheme with the following baselines: 1) First Come First Served (FCFS): tasks are scheduled by their arriving orders. 2) Earliest Deadline First (EDF): the task with the smallest $b^l(x)$ is dispatched first. 3) Soonest Job First (SJF): the task with the soonest estimated finishing time, i.e., $\widehat{t}^a_{ij} + g^l(x)$, is selected first. For fairness consideration, we deploy the resource provisioning plan generated by the Offline policy in Fig. 4 when testing the scheduling policies. The distributions of the deadline violation percentage on our test set under different task scheduling methods are illustrated in Fig. 6. We can observe that our QoS-Aware scheduling policy outperforms the baselines. Moreover, only our approach and the EDF policy succeed to meet the SLA requirement.

6. CONCLUSION

In this paper, we study how to provision resources and schedule tasks in cloud transcoding system for live streaming. We proposed a joint approach to solving the problem of resource provisioning and task scheduling under the dynamic workloads and strict SLA requirement. We design a DRL-based method for resource provisioning and a QoS-aware heuristic algorithm for task scheduling. The results of trace-driven experiments confirm the superiority of the proposed approach.

Acknowledgments. This work is supported in part by the National Key R&D Program of China (2018YFB1003703), Beijing Key Lab of Networked Multimedia, NSFC under Grant No.61521002, ∼61872215 and 61531006, Alibaba-Tsinghua Joint Project (20172001689), and SZSTI JCYJ20180306174057899.

7. REFERENCES

[1] "SLA - encoding.com," https://www.encoding.com/sla/, accessed: 2018-12-10.

[2] Xiangbo Li, Mohsen Amini Salehi, and Magdy Bayoumi, "Vlsc: Video live streaming using cloud services," in *IEEE BDCloud*, 2016.

[3] Guanyu Gao, Yonggang Wen, and Cedric Westphal, "Dynamic resource provisioning with qos guarantee for video transcoding in online video sharing service," in *ACM Multimedia*, 2016.

[4] Xiangbo Li, Mohsen Amini Salehi, Magdy Bayoumi, Nian-Feng Tzeng, and Rajkumar Buyya, "Cost-efficient and robust on-demand video transcoding using heterogeneous cloud services," *IEEE TPDS*, 2018.

[5] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh, "Neural adaptive video streaming with pensieve," in *ACM SIGCOMM*, 2017.

[6] Li Chen, Justinas Lingys, Kai Chen, and Feng Liu, "Auto: scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *ACM SIGCOMM*, 2018.

[7] Guanyu Gao, Han Hu, Yonggang Wen, and Cedric Westphal, "Resource provisioning and profit maximization for transcoding in clouds: A two-timescale approach," *IEEE TMM*, 2017.

[8] "Live encoder settings, bitrates, and resolutions," https://support.google.com/youtube/answer/2853702?hl=en, accessed: 2018-12-10.

[9] Zhengfang Duanmu, Abdul Rehman, and Zhou Wang, "A quality-of-experience database for adaptive video streaming," *IEEE TBC*, 2018.

[10] Adrien Lebre Thuy Nguyen, "Virtual machine boot time model," in *IEEE PDP*, 2017.

[11] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *ICML*, 2016.

[12] Richard S Sutton and Andrew G Barto, *Reinforcement learning: An introduction*, MIT press, 2018.