Dynamic Resource Provisioning with QoS Guarantee for Video Transcoding in Online Video Sharing Service

Guanyu Gao
School of Computer Science and Engineering
Nanyang Technological University
ggao001@e.ntu.edu.sg

Yonggang Wen
School of Computer Science and Engineering
Nanyang Technological University
ygwen@ntu.edu.sg

Cedric Westphal
Huawei Innovation Center & University of California, Santa Cruz
cedric.westphal@huawei.com
cedric@soe.ucsc.edu

ABSTRACT

Video transcoding is widely adopted in online video sharing services to encode video content into multiple representations. This solution, however, could consume huge amount of computing resource and incur excessive processing delays. Moreover, content has heterogeneous QoS requirements for transcoding. Some content must be transcoded in real time, while some are deferrable for transcoding. It needs to determine the strategy for intelligently provisioning the right amount of resource under dynamic workload to meet the heterogeneous QoS requirements. To this end, this paper develops a robust dynamic resource provisioning scheme for transcoding with heterogeneous QoS criteria. We adopt the *Preemptive Resume Priority* discipline for scheduling, so that the transcoding-deferrable content can utilize idle resources for transcoding to maximize resource utilization while remain transparent to delay-sensitive content. We leverage *Model Predictive Control* to design the online algorithm for dynamic resource provisioning using predictions to accommodate time-varying workload. To seek robustness of system performance against prediction noises, we improve our online algorithm through *Robust Design*. The experiment results in a real environment demonstrate that our proposed framework can achieve the QoS requirements while reducing 50% of resource consumption on average.

Keywords

Video transcoding, cloud computing, resource provisioning, quality of service, model predictive control, online algorithm

1. INTRODUCTION

Nowadays online video sharing service has become enormously popular, it has become an important medium for people to have entertainment and obtain information [15]. The online video sharing and social network websites (e.g., Youtube, Facebook, and Twitter) allow users and professional video content producers to upload and publish video content online. It has shown an explosive growing trend of the online video content [24]. For instances, as of 2015, the estimated amount of new videos uploaded to Youtube during one minute is 300 hours [2]. In the live streaming platform Twitch, the peak number of concurrent live streams for broadcasting the live events is above 12000 [23]. These newly uploaded video content, however, cannot be published and delivered to the viewers directly due to the heterogeneous devices and diverse network conditions at the viewer side. To cope with such problems, adaptive bitrate (ABR) streaming [19] has been widely adopted.

With ABR, each video file (or stream) needs to be encoded into multiple representations. When the viewers request the online video content, the streaming server can always deliver the best possible quality representation to the viewers according to their device capacities and network conditions. Nevertheless, video transcoding [21] is very computing intensive and time consuming, it would consume a huge amount of computing resource for transcoding the user uploaded video content into the target representations [16]. The online video sharing service providers need to deploy a large number of servers for transcoding the video content. This greatly increases the service operating cost [17].

The dynamic of transcoding workload and the heterogeneous QoS criteria in online video sharing service make it a great challenge for provisioning the computing resource. First, the arrival rate of the user uploaded content is time-varying [14]. And thus, provisioning a fixed number of dedicated transcoding servers would incur resource wastage when the arrival rate is low, and would incur large latency when the arrival rate is high. Second, video content has heterogeneous QoS requirements for transcoding. For instances, the live content is very sensitive to delay, and must be transcoded in real time. The video-on-demand (VoD) content does not need to be transcoded and delivered in real time, and the transcoding can be finished within a specified deadline. For the seldom requested content, the transcoding is deferrable and can be performed when resources are sufficient or using the idle resources. The QoS requirements for transcoding can also be categorized according to the service level agreement (SLA). For instance, the transcoding for premium service is real-time or guaranteed to finish within a deadline, while the transcoding time for free service may not be guaranteed. Thus, it is necessary to consider the dynamic workload and the heterogeneous QoS requirements of content for provisioning resources for transcoding.

Many recent research works have studied the resource pro-

MM '16, October 15-19, 2016, Amsterdam, Netherlands
© 2016 ACM. ISBN 978-1-4503-3603-1/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2964284.2964296

visioning and QoS management for video transcoding. Most of the previous works, however, considered one class of QoS criteria [25, 13, 7] or the system with known workload [18, 10]. There still lacks of some efforts on the problem of resource provisioning for transcoding with heterogeneous QoS criteria under dynamic workload. The challenges in this scenario are twofold: 1) The deterministic QoS requirements of delay-sensitive content may lead to limitations on system processing capacity, resulting in low resource utilization. The transcoding-deferrable content can utilize these idle resources for transcoding. Thus, it needs to investigate how to schedule the transcoding with heterogeneous QoS criteria to minimize overall resource consumption. 2) The performance of the transcoding system is affected by the uncertainty of time-varying workload, it needs to assess how to provision resource under the uncertainty of dynamic workload.

To address the challenges, we design a robust dynamic resource provisioning scheme for transcoding with heterogeneous QoS requirements in online video sharing services. We aim to achieve the heterogeneous QoS requirements while reducing the resource consumption. We first consider three fundamental QoS criteria for transcoding, namely:

1) *Real-time transcoding*, the QoS cost for transcoding increases with any possible processing delays.

2) *Deadline-sensitive transcoding*, the QoS cost increases if the processing time exceeds the specified deadline.

3) *Deferrable transcoding*, the QoS cost is not directly impacted by the processing time.

We adopt the *preemptive-resume priority* discipline for scheduling the content that has heterogeneous QoS requirements in each transcoding server. The transcoding for the delay-sensitive content (namely, the real-time and deadline-sensitive content) is performed in high priority to guarantee the QoS requirements, while the transcoding for the deferrable content is performed in low priority using idle resources and remain transparent to delay-sensitive content.

We formulate the system management as a problem of minimizing the overall system cost over time, including the computing cost, QoS cost, and switching cost. We design the online algorithm by leveraging *Model Predictive Control* (MPC) for dynamic resource provisioning. We improve the performance of our proposed online algorithm through *Robust Design* against the prediction noises. The experiments in a real environment demonstrate that our proposed framework can improve resource utilization for transcoding while maintaining the required QoS with reduced resource consumption. Our contributions in this paper are as follows:

- We design a preemptive-resume priority scheduling for transcoding the content with heterogeneous QoS criteria. It can greatly improve the resource utilization. To the best of our knowledge, our work is the first to utilize the preemptive-resume priority to implement the multiple-priority transcoding mechanism.

- We propose a robust dynamic resource provisioning scheme for transcoding. It can intelligently provision the right amount of resources using prediction to guarantee QoS, while keeping robust to prediction noise.

- We implement and evaluate the system performance in a real environment. The system can achieve the QoS requirements while reducing 50% of resource consumption on average compared with the *AlwaysOn* policy.

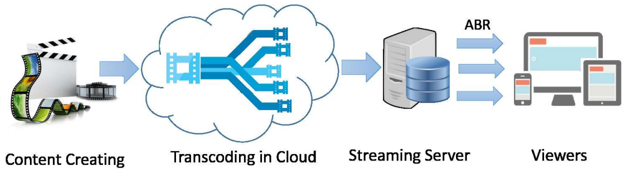


Figure 1: Video delivery flow for online sharing.

The rest of this paper is organized as follows. Section 2 introduces the background and related work. Section 3 presents the system design. Section 4 describes our system models and problem formulation. Section 5 designs the online algorithm. Section 6 introduces our system implementation. Section 7 presents the performance evaluations. Section 8 concludes the paper and presents the future work.

2. BACKGROUND AND RELATED WORK

We first start with a high-level overview of the video delivery flow in an online video sharing service, and then we introduce the related work on the cloud video transcoding.

2.1 Video Delivery Flow

We illustrate the video delivery flow in an online video sharing service in Fig. 1. It consists of the following steps for delivering the video content to the online viewers.

Content Creating: The users and the professional video content producers create the video files or streams by encoding the raw video data into compressed high-definition video using video and audio codecs (e.g., H.264 and AAC). The high-definition video files or streams will be uploaded onto the online video sharing website for publishing.

Video Transcoding: The user uploaded video content will be transcoded into multiple representations in different bitrates, resolutions, and formats. Transcoding is computing intensive. It consumes huge amounts of computing resource for transcoding the large volume of user uploaded video content. A new approach for transcoding is to use the scalable cloud infrastructure for dynamic resource provisioning.

Adaptive Bitrate Streaming: To deliver the video content in high availability and quality, the multiple representations of the video content will be cached in the streaming servers of the content delivery network (CDN). The viewers have varying network connection speeds with the streaming servers. With adaptive bitrate streaming (ABR), the viewers can dynamically adjust the requested quality of the video content by detecting the current bandwidth.

2.2 Cloud Video Transcoding

Numerous algorithms have been designed for reducing the resource consumption or achieving the target QoS. We investigate some related research, and discuss the differences between our method and the previous works.

The current research in this field mainly focuses on reducing the resource consumption and achieving the specified QoS. For reducing the resource consumption, [25] proposed a Lyapunov-based energy-efficient transcoding task dispatching algorithm to manage the tradeoff between energy consumption and processing delay. [18] proposed a dynamic voltage and frequency scaling (DVFS) scheme for adjusting the frequency and workload of each server to minimize power consumption and meet transcoding task deadlines. [7] used

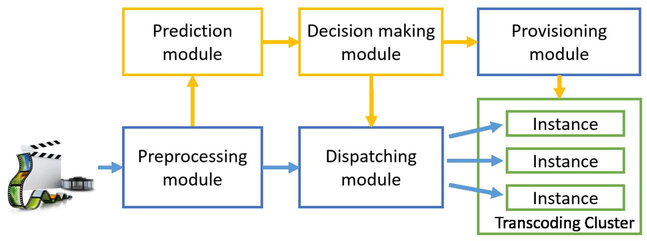


Figure 2: System design of transcoding system.

the prediction-based method for dynamic resource allocation to scale video transcoding service. [9, 12] proposed resource provisioning and performance management frameworks in cloud and data center using the predicted workload. [6] implemented a scalable open source cloud transcoding system that can dynamically provision resource. For achieving the target QoS or reducing the processing delay, [13] presented a dynamic task scheduling method to minimize the transcoding delay. [10] used the parallel video transcoding framework for reducing the transcoding time. [11] leveraged the video popularity information for prediction to pre-transcode video content to reduce the transcoding latency. [20] presented Bitcodin, which enabled transcoding and streaming as a service for live video streaming. [4] studied the transcoding settings for live streaming to maximize the QoE. [8] studied the impact of computing resource and cache size on the performance of the transcoding system.

The main difference between our work and the previous research is that we consider the heterogeneous QoS requirements of different types of content. The first-order goal of this work is to design the resource provisioning scheme under dynamic workload for transcoding with heterogeneous QoS requirements. The aim is to achieve the different QoS targets while reducing the resource consumption. The addressed challenges are twofold: 1) provisioning resource under the uncertainty of dynamic workload and 2) scheduling the transcoding with heterogeneous QoS criteria.

3. SYSTEM DESIGN AND WORKFLOW

3.1 Design

Our system leverage the elasticity of cloud infrastructure for video transcoding in the online video sharing service. We present the system design in Fig. 2. The functionalities of each module are detailed as follows.

Preprocessing module: The users and professional video content producers upload their content for online video sharing. The preprocessing module uses the media stream segmenter to segment content into a series of equal-duration (e.g., 10s) chunks, which are suitable for HTTP streaming.

Prediction module: It monitors the arrival rates of the different types of video chunks, and makes the prediction for the future arrival rate of each type of video chunk.

Decision making module: It uses the system information (e.g., available transcoding instances, QoS requirements, and future video chunk arrival rate) to make control decisions for resource provisioning and content dispatching.

Provisioning module: It dynamically adjusts the number of active transcoding instances to accommodate the time-varying workload and to satisfy the QoS requirements for transcoding. To elastically allocate the computing resource, our system can adopt Virtual Machines (VM) or Containers as transcoding servers. We refer to a VM instance or a Container provisioned for transcoding as a transcoding instance. The performance of the instances is homogeneous.

Dispatching module: The video chunks are dispatched to the active transcoding instances to be encoded into a predefined set of target representations. The dispatching module determines how to dispatch the video chunks to meet the QoS requirements for transcoding while not exceeding the processing capacity of each transcoding instance.

Table 1: Key Parameters

Parameter	Meaning
t	discrete time slot, $t = 0, 1, 2, \ldots$
r, l, d	**r**eal-time, dead**l**ine-sensitive, and **d**eferrable
N	maximum number of available instances
$b_i(t)$	state of the i-th transcoding instance at t
$b_i^m(t)$	type of content transcoded by instance i
$C(t)$, $E(t)$	computing cost, QoS cost
$q(t)$	queue length for deferrable content
$\lambda^m(t)$	arrival rate of type m at t, $m \in \{r, l, d\}$
$\lambda_i^m(t)$	dispatched rate of type m to instance i at t
V_1, V_2	weights of QoS cost and switching cost
μ,γ	expectation and second moment
$w_i(t)$	processing time of high priority video chunks

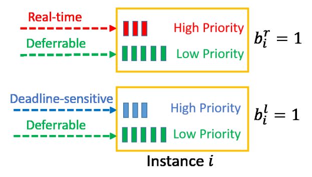
3.2 Workflow

The real-time and deadline-sensitive content has deterministic QoS requirements and are sensitive to the processing time, they will be directly dispatched to the active instances for transcoding after arriving. In contrast, the deferrable content has no strict processing time requirements, and can be dispatched for transcoding when idle resource is available. In our system, the dispatching module maintains a FIFO queue for the deferrable content, and the deferrable content can be cached in the queue if the capacity allows. For making the system modeling trackable and deriving the closed-form solution, one active instance will only transcode one type of the delay-sensitive content (i.e., either real-time or deadline-sensitive content) during each time slot. However, the transcoding for deferrable content can be performed on any active instance.

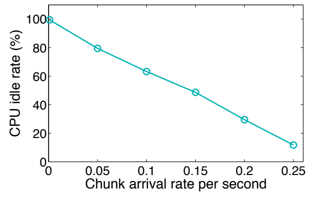
We adopt the *Preemptive Resume Priority* discipline in each active instance for transcoding different types of content. Specifically, the transcoding for real-time and deadline-sensitive content is performed in high priority, while the transcoding for deferrable content is performed in low priority. The new arrival of content in high priority can immediately preempt the low priority transcoding operation which is currently being performed, and start the high priority transcoding operation. When computing resource is available, the previously preempted low priority transcoding operations can be resumed from the point where it was interrupted earlier. We adopt *POSIX Signal* [1] for the preemptive resume priority implementation.

4. SYSTEM MODELS

We adopt a discrete time model, where the time slot is denoted as $t = 0, 1, 2, \ldots$. The duration of one time slot could be from 10 minutes to an hour.

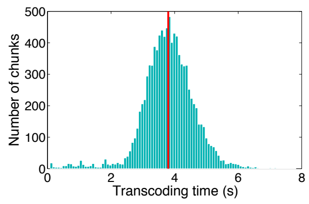


(a) Instance provisioning for different types of content.

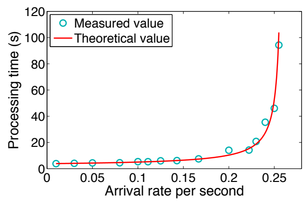


(b) CPU idle rates under varying chunk arrival rates.

Figure 3: Instance provisioning model.



(a) Video chunk transcoding time distribution



(b) Processing time under varying arrival rates

Figure 4: Video chunk processing time model.

4.1 Video Chunk Arrival Model

We consider three fundamental QoS criteria for transcoding, and correspondingly categorize the content into three types, namely, **r**eal-time, dead**l**ine-sensitive, and **d**eferrable content, represented as r, l, d, respectively. We assume that the video chunks of each type of content arrive according to the Poisson process during each time slot. The arrival rate for the video chunks of type m during the time slot t is denoted as $\lambda^m(t)$, where $m \in \{r, l, d\}$.

4.2 Computing Cost Model

The computing cost is incurred by provisioning VM instances or Containers. The system has N transcoding instances, and performance is homogeneous. At the beginning of each time slot, the system determines whether an instance should be activated according to the workload and QoS requirement. We denote the computing cost for an active instance during one time slot as B^c. The overall computing cost incurred at time t can be calculated as:

$$C(t) = \sum_{i=1}^{N} b_i(t) B^c, \tag{1}$$

where $b_i(t)$ is a binary variable to indicate that whether the transcoding instance i is active at time t.

4.3 Instance Provisioning Model

In our design, each active instance only transcodes one type of delay-sensitive content during each time slot, either real-time or deadline-sensitive content. A more sophisticated design would be mingling the three types of content in an active instance using three priority classes. This, however, would bring great difficulties for both practical implementation and deriving the solutions. To make the modeling trackable, we simplify it in this work: one active instance can transcode deferrable content in low priority, and transcode one type of delay-sensitive content in high priority.

As shown in Fig. 3(a), we adopt binary variables $b_i^m(t)$, $m \in \{r, l\}$, to denote which type of delay-sensitive content will be transcoded by instance i at time t; if $b_i^r(t)$ equals one, it transcodes real-time content; if $b_i^l(t)$ equals one, it transcodes deadline-sensitive content. If an active instance only transcodes delay-sensitive content, the deterministic QoS requirements would limit the video chunk dispatched rate to the instance. As shown in Fig. 3(b), the low chunk dispatched rate to an instance would lead to high CPU idle rate. Thus, deferrable content can be transcoded in low priority in an instance to improve resource utilization when no delay-sensitive content is being transcoded.

4.4 Dispatching Model

The arriving video chunks are dispatched to the active instances for transcoding. Each of the video chunks will be transcoded into a set of pre-defined representations, e.g., 240p, 360p, and 720p. We denote the dispatched rate of the chunks of type m to instance i at time t as $\lambda_i^m(t)$, where $m \in \{r, l, d\}$. The deferrable content can be temporally cached in the queue, and dispatched for transcoding at some later time. The queue dynamic for deferrable content is

$$q(t+1) = q(t) + \lambda^d(t) - \sum_{i=1}^{N} \lambda_i^d(t), \tag{2}$$

where $q(t)$ is the queue length for deferrable content at the beginning of each time t.

4.5 Video Chunk Processing Time Model

Each video chunk has the same duration, however, the transcoding time for video chunks is varying based on many factors, e.g., bitrate, resolution, and server performance. We adopt the empirical method to measure the distribution of the transcoding time for video chunks. We denote the transcoding time for a video chunk as x, and the expectation and second moment of x are represented as

$$E(x) = \frac{1}{\mu}, E(x^2) = \gamma, \tag{3}$$

where μ and γ can be measured in the system offline. In practice, we measure the time for transcoding 9800 video chunks into three target resolutions[1] for determining μ and γ. The distribution of the transcoding time for the video chunks is illustrated in Fig. 4(a). Further, we determine the values of μ and γ as 0.2632 and 15.1, respectively.

4.6 Video Chunk Queueing Model

The queueing for video chunks in each instance can be modelled as a *M/G/1* system with *preemptive resume priority*. The delay-sensitive (real-time or deadline-sensitive) content is transcoded in high priority to guarantee the QoS requirements. The deferrable content is transcoded in low priority when no delay-sensitive content is being transcoded. The transcoding of deferrable content can be interrupted on arrival of delay-sensitive content, and will be resumed from the point of interruption when idle resource is available again. Thus, deferrable content can use the idle resource in an active instance for transcoding, while remain transparent to delay-sensitive content.

[1]The target resolutions are 240p, 360p, and 480p.

Given the chunk dispatched rate $\lambda_i^m(t)$, $m \in \{r,l\}$, of delay-sensitive content to instance i, the average processing time for video chunks of delay-sensitive content is

$$w_i(t) = \frac{1}{\mu} + \frac{\mu\gamma\lambda_i^m(t)}{2(\mu - \lambda_i^m(t))}, m \in \{r,l\}. \tag{4}$$

Using μ and γ derived from Fig. 4(a), we measure the average chunk processing time of delay-sensitive content under varying dispatched rate to an instance, with deferrable content is transcoded in low priority when CPU is idle. As shown in Fig. 4(b), the average chunk processing time of delay-sensitive content fits with Eq. (4).

4.7 QoS Cost Model

The QoS cost for delay-sensitive content is incurred by the video chunk processing delay. Specifically, the QoS cost for real-time content incurred on instance i at time t is

$$F_i^r(t) = \lambda_i^r(t)\mathcal{G}(w_i(t)), \tag{5}$$

where $\mathcal{G}(\cdot)$ is QoS function for real-time content. We let the deadline for average chunk processing time be u. The deadline-sensitive content will incur QoS cost if the average processing time exceeds the deadline. The QoS cost for deadline-sensitive content incurred on instance i at t is

$$F_i^l(t) = \lambda_i^l(t)\mathcal{H}(\max(w_i(t) - u, 0)), \tag{6}$$

where $\mathcal{H}(\cdot)$ is QoS function for deadline-sensitive content. The overall QoS cost for delay-sensitive content incurred on all active instances at time t is

$$E(t) = \sum_{i=1}^{N}(F_i^r(t) + F_i^l(t)). \tag{7}$$

Our formulation requires $E(t)$ to be convex for λ_i^r and λ_i^l for deriving the closed-form solution. Examples for $\mathcal{G}(\cdot)$ and $\mathcal{H}(\cdot)$ are the increasing linear function or the logarithmic function. We only consider the queue capacity constraint for deferrable content in our system. Our framework, however, can also add the QoS cost for deferrable content into consideration. For instance, the QoS cost incurred for deferrable content in each time slot is an non-decreasing convex function with related to the queue length; more QoS cost would be incurred with more content is deferred.

4.8 Switching Cost Model

Frequently toggling the instances back and forth between the active and sleep mode will increase the risk of system failure. We adopt the switching cost to represent the operation cost and risk associated with activating an instance from the sleep mode. The overall switching cost for activating the instances from the sleep mode at time t is

$$S(t) = \sum_{i=1}^{N} B^s \max(b_i(t) - b_i(t-1), 0), \tag{8}$$

where B^s is the marginal switching cost for activating an instance from sleep mode.

4.9 System Cost Minimization Problem

We aim to minimize the overall system cost over time under dynamic workload. We consider three costs, namely, the computing cost, QoS cost, and switching cost. If more instances are provisioned, the QoS cost will be low, but it would incur more computing cost. On the contrary, if fewer instances are provisioned, it can save the computing cost, but the QoS cost may be high. Meanwhile, frequently switching the instances between the sleep mode and active mode would increase the switching cost. The control decisions are the provisioning of the instances and the scheduling of dispatching the content for transcoding. The system cost minimization problem is presented as:

$$\begin{aligned}
\mathcal{P}1: \min \quad & \sum_{t=1}^{T}\{C(t) + V_1E(t) + V_2S(t)\}, && (9)\\
\text{s.t.} \quad & \lambda_i^m(t) \geq 0, \forall i,t,m, && (10)\\
& \sum_{i=1}^{N}\lambda_i^m(t) = \lambda^m(t), \forall t, m \in \{r,l\}, && (11)\\
& b_i(t) = \{0,1\}, \forall i,t, && (12)\\
& b_i^m(t) = \{0,1\}, \forall i,t,m \in \{r,l\}, && (13)\\
& b_i(t) = b_i^r(t) + b_i^l(t), \forall i,t, && (14)\\
& \lambda_i^m(t) \leq b_i^m(t)\mu, \forall i,t,m \in \{r,l\}, && (15)\\
& \lambda_i^r(t) + \lambda_i^l(t) + \lambda_i^d(t) \leq b_i(t)\mu, && (16)\\
& 0 \leq q(t) \leq L, \ \forall t, && (17)
\end{aligned}$$

where V_1 and V_2 are the tunable parameters to adjust the weight of each cost, and L is the maximum capacity of the queue for the deferrable content.

Constraint (10) and (11) ensure that the delay-sensitive content can be dispatched directly. Constraint (12) - (14) ensure that one active instance can only transcode one type of delay-sensitive content during each time slot. Constraint (15) ensures that the delay-sensitive content is only dispatched to the active instances provisioned for this type of content. Constraint (16) ensures that the overall amount of video chunks dispatched to an active instance will not exceed its processing capacity. Constraint (17) validates the queue capacity constraint for deferrable content .

5. ALGORITHM DESIGN

This section presents methods for deriving the offline solution of the cost minimization problem, and introduces the online algorithm design with the MPC and robust design.

5.1 Offline Solution

The chunk arrival rates are random processes and cannot be known in advance. Given the access to the precise chunk arrival information for the next T time slots, we can derive the offline solution by solving $\mathcal{P}1$. While $\mathcal{P}1$ is a *mixed boolean-convex problem* [5], it is NP-hard and cannot be solved in polynomial time. However, we can solve it optimally with *Branch-and-Bound* via convex relaxation when the number of binary variables is small [5]. In practice, we can also obtain the optimal solution with the CVX solver (e.g., Gurobi, MOSEK, and GLPK). In a real production environment, the number of provisioned instances could be several thousands. It may take many hours to derive the optimal solution with Branch-and-Bound or CVX solvers. Next, we discuss how to obtain the approximate solution when the number of provisioned instances is large.

We let the number of instances provisioned for the real-time and deadline-sensitive content at time t be $n^r(t)$, $n^l(t)$, respectively, while deferrable content can be transcoded on these instances in low priority. Given the numbers of active

Table 2: Multiple-Step Prediction

Step	Inputs of Neural Network	Output
1	$\lambda^m(t-h)$, ... , $\lambda^m(t-1)$	$\hat{\lambda}^m(t)$
2	$\lambda^m(t-h+1)$, ... , $\hat{\lambda}^m(t)$	$\hat{\lambda}^m(t+1)$
3	$\lambda^m(t-h+2)$, ... , $\hat{\lambda}^m(t+1)$	$\hat{\lambda}^m(t+2)$
4	$\lambda^m(t-h+3)$, ... , $\hat{\lambda}^m(t+2)$	$\hat{\lambda}^m(t+3)$

instances $n^m(t)$, $m \in \{r,l\}$, we can use the KKT condition to prove that the optimal dispatching pattern for each type of content is to equally dispatch the video chunks among the $n^m(t)$ active instances. Let $\lambda^m_*(t)$, $m \in \{r,l\}$, be the dispatched rates of real-time or deadline-sensitive content to each of the $n^m(t)$ active instances, and we have $\lambda^m_*(t) = \lambda^m(t)/n^m(t)$ for delay-sensitive content. Let $\lambda^d_*(t)$ be the amount of deferrable content dispatched for transcoding at time t. We can rewrite $\mathcal{P}1$ as follow for deriving $\lambda^d_*(t)$, $n^m(t)$,

$$\begin{aligned} \min \quad & \sum_{t=1}^{T}\{C(t)+V_1E(t)+V_2S(t)\}, \qquad (18)\\ \text{s.t.} \quad & \lambda^m(t) \le n^m(t)\mu, \forall t, m \in \{r,l\},\\ & \lambda^r(t)+\lambda^l(t)+\lambda^d_*(t) \le n(t)\mu, \forall t,\\ & q(t+1) = q(t)+\lambda^d(t)-\lambda^d_*(t),\\ & 0 \le q(t) \le L,\ \forall t,\\ & n(t) \le N, \end{aligned}$$

where $n(t)$ is the total number of active instances at time t, and $n(t) = n^r(t) + n^l(t)$. The computing cost $C(t)$ can be rewritten as $n(t)B^c$, and the switching cost $S(t)$ can be rewritten as $\max(n(t)-n(t-1),0)B^s$. When the number of provisioned instances is large, it is acceptable to allow $n^m(t)$ to be fractional. We can relax the integral variables $n^m(t)$ to be $n^m(t) \ge 0$, allowing fractional values for $n^m(t)$. The problem after relaxing is convex, and we can derive the solution by Lagrange method, and it can also be solved efficiently by CVX solvers.

5.2 Online Algorithm

The future video chunk arrival information cannot be known in advance, and thus it is infeasible to obtain the offline optimal decisions. We design the online algorithm with MPC, which makes decisions taking into account of predictions.

5.2.1 Arrival Rate Prediction

We adopt *neural network* method for predicting the future video chunk arrival rates based on historical arrival information. We assume that the video chunk arrival rates of different types of content are independent. The neural network method for prediction is to train a nonlinear function $f : \mathcal{R}^h \to \mathcal{R}$, which can estimate $\lambda^m(t)$ using the last h observations, i.e., $\lambda^m(t-1)$, ..., $\lambda^m(t-h)$. The nonlinear relation among the past observations and the prediction is

$$\hat{\lambda}^m(t) = f(\lambda^m(t-1), ..., \lambda^m(t-h)),$$

where $f(\cdot)$ is determined by the parameters of the neural network and can be learnt offline using the training data. Using the last h observations and the trained neural network, we can predict the arrival rate for one time slot ahead.

The cost minimization problem takes into account of the video chunk arrival information for the next T time slots. To make the prediction for the next T time slots, we can iteratively use the predicted arrival rates as the inputs of the neural network, and get the prediction for the next time slot. We illustrate the case of making four-step prediction at time $t-1$ in Table 2. At $t-1$, we first use the last h observations, $\lambda^m(t-1)$, ..., $\lambda^m(t-h)$, as the inputs of the neural network to get the prediction for $\hat{\lambda}^m(t)$. Next, we can use $\hat{\lambda}^m(t)$, ..., $\lambda^m(t-h+1)$ as the inputs for predicting $\hat{\lambda}^m(t+1)$. This can be iteratively done until we get the predictions for the next T time slots.

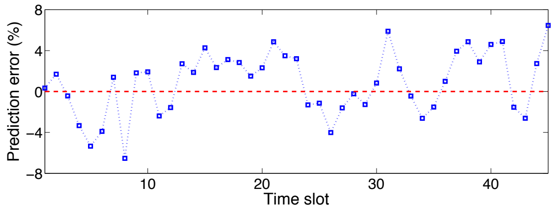


Figure 5: Fluctuation of prediction error over time.

5.2.2 Online Algorithm Design

We design the online algorithm with MPC using predictions for making control decisions. The control decisions are affected by the video chunk arrival rates, and thus the prediction accuracy affects the performance of the online algorithm. The predictions always have noise, and the noise may increase as one looks more steps ahead. If the noise for multiple-step ahead prediction is too large, it would be better to ignore the predicted information. The MPC is an online method by looking ahead for T time slots, and it makes decisions in each time slot in the following steps.

Predict: the arrival rate predictor predicts the video chunk arrival rates for the next T time slots.

Optimize: use the prediction to derive the solution of $\mathcal{P}1$ for minimizing the cost over the next T time slots.

Apply: apply the derived control decisions for the next time slot in the system.

The predictions always have noise. We illustrate the prediction error distribution for neural network method in Fig. 5, we can observe that the prediction error fluctuates in an range over time. The prediction error may lead to bad decisions and deteriorate the system performance, especially when the workload is underestimated and the provisioned resource cannot satisfy the QoS requirements. To seek the robustness of the performance against the uncertainty of the prediction error, we adopt *Robust Design* to minimize the worst-case of the system cost for deriving control decisions.

The predicted arrival rates lie around the actual values within an error bound. Instead of predicting a specified value for $\lambda^m(t)$, we estimate the possible range of $\lambda^m(t)$. We let $\underline{\lambda}^m(t)$ be the lower bound of $\lambda^m(t)$, and $\overline{\lambda}^m(t)$ be the upper bound, i.e., $\lambda^m(t) \in [\underline{\lambda}^m(t), \overline{\lambda}^m(t)]$. Given the possible range of the arrival rates, we first need to determine the arrival rates which would incur the largest system cost, and then derive control decisions for the worst-case system cost. We can use the contradiction method to prove that the worst-case system cost takes place when the video chunk arrival rates are at the upper bound of $[\underline{\lambda}^m(t), \overline{\lambda}^m(t)]$, i.e., at $\overline{\lambda}^m(t)$. As such, the minimization of the worst-case system cost is equivalent to take upper bounds of the predicted chunk arrival rates as inputs of $\mathcal{P}1$ to derive the solution.

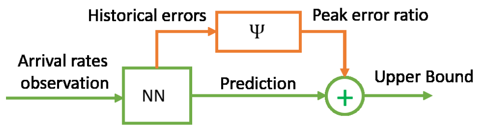


Figure 6: Robust prediction framework.

Instead of directly using the predicted arrival rates as inputs of $\mathcal{P}1$ in MPC method, our online algorithm takes upper bounds as inputs for deriving solutions. The prediction for the arrival rates are optimal in the sense of mean-square-error, and provides the best average performances. The underlying noise, however, often conflicts with the prediction models, making system unstable and leading to poor performance. The aim of obtaining the upper bound for the prediction (also known as robust prediction) is to accommodate uncertainties against prediction noise. We illustrate the procedure for making the robust prediction for the arrival rates in Fig. 6. The neural network predictor estimates the arrival rates, and the noise analyzer determines the peak error ratio using observed historical prediction errors. The upper bound for the predicted arrival rate is calculated as $\overline{\lambda}^m(t) = \hat{\lambda}^m(t)/(1+\Psi)$, where$\Psi \leq 0$ is the peak error ratio [22]. In our system, we set the peak error ratio as the minimum of the last 10 prediction errors and the training errors. The prediction error and training error are calculated as $e_t = (\hat{\lambda}^m(t) - \lambda^m(t))/\lambda^m(t)$, and $\Psi = \min(e_{t-10}, .., e_{t-1}, e_T)$, where e_T is the minimum in the training set. We illustrate the details of the online algorithm in Algorithm 1.

Algorithm 1 Online algorithm for dynamic system control

```
1: Set $t$ as the current time.
2: while the system is in service do
3:     Predict video chunk arrival rates for the next $T$ time slots, namely, $\hat{\lambda}^m(t)$, $\hat{\lambda}^m(t+1)$, ..., $\hat{\lambda}^m(t+T-1)$.
4:     Calculate upper bounds of the predicted arrival rates, namely, $\overline{\lambda}^m(t)$, $\overline{\lambda}^m(t+1)$, ..., $\overline{\lambda}^m(t+T-1)$.
5:     Optimize $\mathcal{P}1$ using the upper bounds of the predicted arrival rates as inputs.
6:     Apply control decisions for time slot $t$ in the system.
7:     $t \leftarrow t+1$
8: end while
```

The algorithm running time is dominated by the time for solving $\mathcal{P}1$. We can apply the method discussed in Section 5.1 to solve the problem in an acceptable time.

6. SYSTEM IMPLEMENTATION

We build the cloud environment with *Docker* and implement our proposed system with *Python*. The main components of our system are illustrated in Fig. 7.

Portal: It segments video files and streams into a series of 10-second chunks. The chunk data is stored into the storage system, and the chunk information is stored in *MySQL*. The arrivals of the chunks will be informed to the dispatcher.

Controller: It makes predictions by neural network and control decisions by performing the online algorithm.

Dispatcher: It sends the arriving chunk information to the distributed queue implement with *RabbitMQ*. Each instance has one queue for delay-sensitive content and one queue for

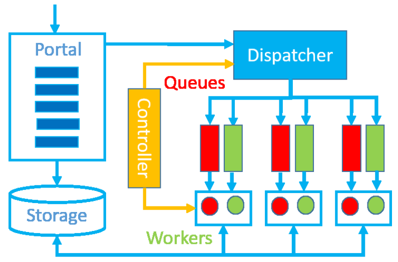


Figure 7: Implementation of the system

deferrable content. The transcoding procedure in the worker will be invoked when new chunk enters the queue. The worker will fetch the chunk information and download chunk data from storage system for transcoding.

Worker: Each worker in an instance has two processes for transcoding, namely, background process and foreground process, corresponding to the two queues. The background process transcodes deferrable content. The foreground process transcodes delay-sensitive content. The workers use *avconv* for transcoding, and the preemptive resume mechanism is implemented with *POSIX Signals*. When the transcoding procedure of foreground process is invoked by incoming delay-sensitive chunks, it first send a *SIGSTOP signal* to suspend background process and start its own transcoding. It will resume background process by sending a *SIGCONT signal* when finishing and no chunk left in its queue.

7. PERFORMANCE EVALUATION

We first introduce the dataset and experiment settings. We then evaluate the performance of our system under different settings, and compare it with the baseline schemes.

7.1 Dataset and Experiment Setting

We obtain the number of concurrent online video channels in Twitch during each minute through *Twitch* API [3]. In our experiment, we assume that the video chunk arrival rate is proportional to the number of online channels. Each time slot lasts for 10 minutes. To speed up the experiment, each time slot corresponds to one hour in the dataset. The arrival rates for each time slot are averaged over 60 minute intervals in the dataset. We use the arrival rates in different days as the arrival rates for the three types of content.

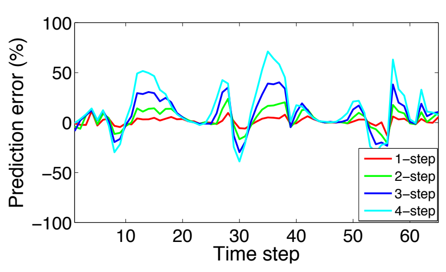
Each container in our system has 4 CPU cores. The containers are connected with Gigabit Ethernet. We adopt the Amazon EC2-based pricing mechanism to calculate the computing cost, and the price for an instance is \$0.0398 per time slot. The online algorithm looks ahead for 4 time slots. The QoS cost functions $\mathcal{G}(\cdot)$ and $\mathcal{H}(\cdot)$ are linear, and the coefficient terms are 1 and constant terms are 0. The default values for the other parameters are summarized in Table 3.

Table 3: Default values of parameters

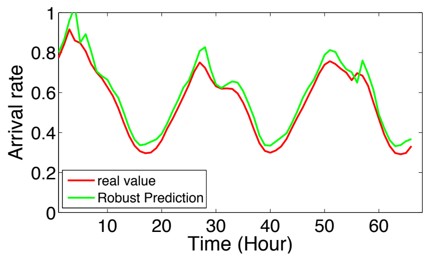
V_1	=	10^{-4}	V_2	=	1	B^s	=	0.0040
u	=	20	μ	=	0.2632	B^c	=	0.0398
L	=	1000	γ	=	15.1	N	=	20

7.2 Prediction Performance

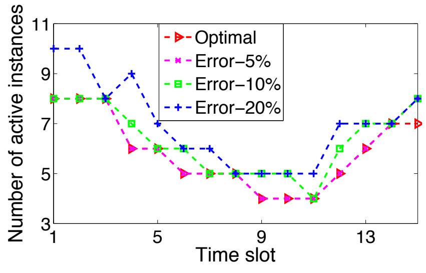
We evaluate the prediction performance of the neural network. We use a 3-layer feed-forward neural network with



(a) The prediction error for the test data using different prediction steps.

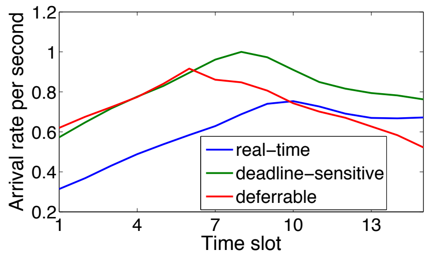


(b) The robust prediction for arrival rates.

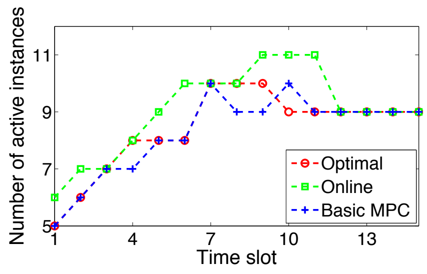


(c) The number of active instances under different prediction errors.

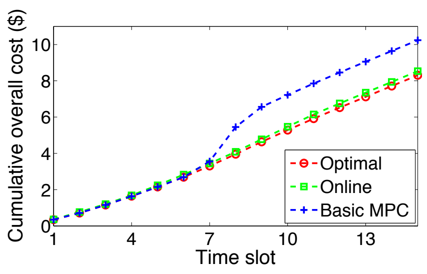
Figure 8: Prediction performance and impact of prediction error.



(a) The video chunk arrival rates of the three types of content.



(b) The number of active transcoding instances during each time slot.



(c) The cumulative overall system cost.

Figure 9: Effectiveness of the online algorithm.

sigmoid hidden neurons and linear output neurons. The neural network has two inputs (i.e., the arrival rates in the past two time slots), and the hidden layer consists of 10 neurons. The network is trained with Levenberg-Marquardt backpropagation algorithm. We select the data of 100 continuous time steps for training. The data point in each time step is the arrival rate in each hour of the real dataset. We evaluate the performance for predicting the arrival rates of another 65 time slots. The prediction errors of using different prediction steps are illustrated in Fig. 8(a). The mean prediction error is 2.5% for 1-step prediction, 6.4% for 2-step prediction, 11.0% for 3-step prediction, and 16.0% for 4-step prediction. The prediction noise increases with more time steps to look ahead. We illustrate the performance of the robust prediction in Fig. 8(b). The predicted arrival rates by the robust prediction are slightly higher than the real value to avoid resource under-provisioning.

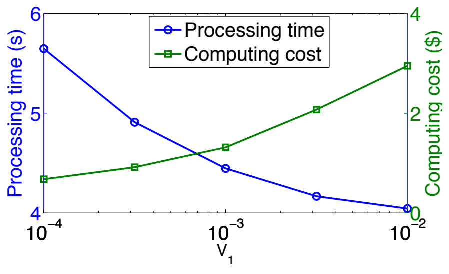
We evaluate the performance of our online algorithm under different prediction errors. To generate different prediction errors, we add uniform distributed random noise to the real arrival rates, and use the synthetic data as the prediction, and thus the prediction is independent of the specified predictor. We illustrate the number of provisioned instances over 15 time slots under different prediction errors in Fig. 8(c). The optimal policy, knowing the precise future arrival rates, provisions the minimum number of instances to meet the QoS requirements. When the prediction error is within 5%, our online algorithm can achieve the near-optimal performance. With larger prediction errors, our online algorithm provisions more instances against the larger uncertainty. Compared with the optimal policy, however, the system cost only increases 2.3% when prediction error is within 10%, and increases 5.8% when prediction error is within 20%.

7.3 Effectiveness of the online algorithm

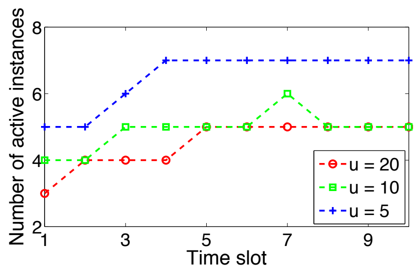
We evaluate the effectiveness of our proposed online algorithm, and compare its performance with the offline optimal solution and the *Basic MPC method*. The Basic MPC method is the same as our online algorithm, but without the robust design. The video chunk arrival rates for the three types of content over 15 time slots are illustrated in Fig. 9(a). We can observe in Fig. 9(b) that the number of instances provisioned by our online algorithm is always no less than offline optimal solution. This is due to the robust design of our online algorithm, which makes decisions according to the maximum possible arrival rates. In contrast, the Basic MPC method makes decisions directly according to the predictions of the neural network. As such, the number of provisioned transcoding instances may be lower than required, if the predictor underestimates the video chunk arrival rates. In this case, it will deteriorate the system performance and incur more QoS cost. As described in Fig. 9(c), the cumulative cost of our online algorithm is only slightly higher than the offline optimal solution, which is incurred by the resource over-provisioning. While the Basic MPC method has the near-optimal average performance in most of the time slots, but would incurs more cost when the provisioned resource cannot meet the QoS requirements.

7.4 Impact of Tunable Parameters

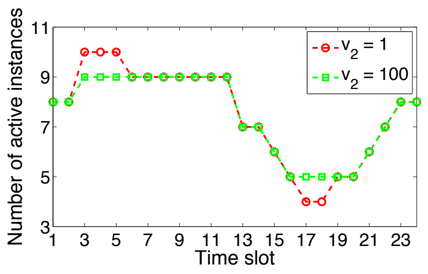
We illustrate the computing cost and the average processing time for real-time content over 5 time slots under different values of V_1 in Fig. 10(a). The computing cost for real-time content increases with V_1, while the average processing time for the video chunks decreases with it. Intuitively, the QoS cost takes more weights with the larger values of V_1, and the system will provision more computing resource to reduce the processing time. One can select



(a) The computing cost and processing time under different values of V_1.

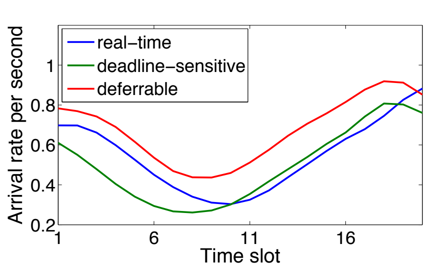


(b) The number of active instances in each time slot under different values of u.

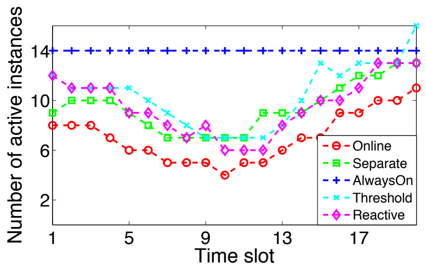


(c) The number of active instances in each time slot under different values of V_2.

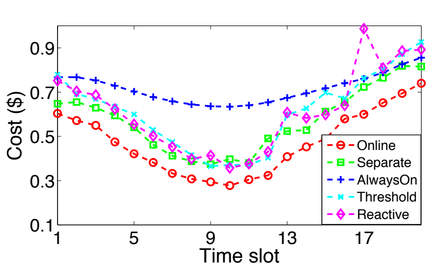
Figure 10: Impact of Tunable Parameters.



(a) The arrival rates for different types of video content.



(b) The number of active instances in each time slot under different policies.



(c) The overall cost in each time slot under different policies.

Figure 11: Comparison with baseline policies.

Table 4: Cost and resource consumption reduction percentage compared with AlwaysOn

	Online	Separate	Threshold	Reactive
Cost	34%	20%	14%	14%
Resource	50%	32%	25%	31%

the suitable value of V_1 for managing the QoS for real-time content. We illustrate the number of provisioned transcoding instances for deadline-sensitive content over time under different values of u in Fig. 10(b). With the smaller values of u, the system needs to provision more instances to reduce the dispatched rates of deadline-sensitive content to each active instance for achieving the average processing time deadline. We illustrate the impact of the switching cost in Fig. 10(c). With the larger weight of the switching cost, there will be less switching of the states of the transcoding instances. This is because switching the state of the transcoding instance frequently will incur more cost than keeping the transcoding instances active for a few more time slots.

7.5 Comparison with Baseline Methods

We compare the performance of our online algorithm with the following baselines: 1) *AlwaysOn*: It provisions a fixed number of instances for each type of content according to the peak workloads. 2) *Separate*: It uses the same dynamic resource provisioning policy with our online algorithm, however, each active instance only transcodes one type of content during each time slot. The queue in each instance is M/G/1 with FIFO discipline. 3) *Threshold*: It uses a minimum number of active instances for transcoding to guarantee QoS, keeping the over-provisioned instances idle. When an instance is idle for two time slots, it would be shut down. If the workload exceeds the current capacity, it will increase the capacity by 30% above the requested workload. 4) *Reactive*: It provisions the computing resource based on the observed arrival rates in the past time slot.

The arrival rates for the three types of content are shown in Fig. 11(a). We illustrate the number of provisioned instances and the incurred cost in each time slot under different policies in Fig. 11(b) and 11(c), respectively. The overall cost for the AlwaysOn policy decreases when the arrival rates are low, because less QoS cost is incurred. Our online algorithm provisions fewer instances and incurs less overall cost than the baseline policies. The performance gains are from two parts. First, our system can utilize the idle computing resource for transcoding deferrable content to improve resource utilization. This can be verified by comparison with *Separate* policy. If each type of content is transcoded using separate instances, it needs to provision more active instances. Meanwhile, the computing resource of the instances transcoding delay-sensitive content cannot be fully utilized, due to the deterministic QoS requirements. Adopting the preemptive resume priority for transcoding multiple types of content can improve the resource utilization without affecting the QoS of the delay-sensitive content.

The second part for the performance gain is that our online algorithm can intelligently provision resource under dynamic workload using prediction. We verify this by comparing with the *Reactive* policy and the *Threshold* policy. The *Reactive* policy attempts to keep the exact number of instances reacting to the past observed workload. By comparing it with the *Separate* policy, we can find that the *Reactive* policy may lag behind the changing workload, because it only uses the past workloads for decision. This, without considering changing trends, may induce over- or under-provisioning under dynamic workload. The *Threshold* policy is more conservative, keeping idle instances active for 2 more time slots and provisioning an extra capacity to avoid resource under-provisioning. However, without future workload information, this still may lead to over-provisioning if the extra capacity is large, and under-provisioning if the extra capacity is small. We summarize the overall cost and resource consumption reduction percentage of different policies compared with the *AlwaysOn* policy in Table 4. Our online policy can on average reduce the overall cost by 34%, and reduce the resource consumption by 50%, which shows a great performance gain compared with the baseline policies.

8. CONCLUSION

This paper designs and implements a video transcoding system for dynamic resource provisioning with QoS guarantee in online video sharing service. We propose a framework for transcoding with heterogeneous QoS criteria. We adopt the preemptive resume priority for processing different types of content to improve resource utilization without affecting QoS for delay-sensitive content. We design an online algorithm which can intelligently provision the right amount of resource using prediction to guarantee QoS, while keeping robust to prediction noise. The experiment results demonstrates that our online algorithm can achieve the QoS requirements while reducing 50% of resource consumption. As future work, we may consider the resource provisioning for hardware-accelerated (e.g. GPU) video transcoding.

9. REFERENCES

[1] POSIX-style signals. https://en.wikipedia.org/wiki/Unix_signal.

[2] Statistics and facts about YouTube. http://www.statista.com/topics/2019/youtube/.

[3] Twitch. https://github.com/justintv/Twitch-API.

[4] R. Aparicio-Pardo, K. Pires, A. Blanc, and G. Simon. Transcoding live adaptive video streams at a massive scale in the cloud. In *Proceedings of the 6th ACM Multimedia Systems Conference*, pages 49–60, 2015.

[5] S. Boyd and J. Mattingley. Branch and bound methods. *Notes for EE364b, Stanford University.*

[6] G. Gao and Y. Wen. Morph: A fast and scalable cloud transcoding system. In *Proceedings of the 24th ACM international conference on Multimedia.*

[7] F. Jokhio, A. Ashraf, S. Lafond, I. Porres, and J. Lilius. Prediction-based dynamic resource allocation for video transcoding in cloud computing. In *Parallel, Distributed and Network-Based Processing, 21st Euromicro International Conference on.* IEEE, 2013.

[8] S. Ko, S. Park, and H. Han. Design analysis for real-time video transcoding on cloud systems. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1610–1615. ACM, 2013.

[9] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster computing*, 12(1):1–15, 2009.

[10] F. Lao, X. Zhang, and Z. Guo. Parallelizing video transcoding using map-reduce-based cloud computing. In *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, pages 2905–2908, 2012.

[11] Z. Li, Y. Huang, G. Liu, F. Wang, Z.-L. Zhang, and Y. Dai. Cloud transcoder: Bridging the format and resolution gap between internet videos and mobile devices. In *NOSSDAV '12.*

[12] M. Lin, A. Wierman, L. L. Andrew, and E. Thereska. Dynamic right-sizing for power-proportional data centers. *IEEE/ACM Transactions on Networking (TON)*, 21(5):1378–1391, 2013.

[13] H. Ma, B. Seo, and R. Zimmermann. Dynamic scheduling on video transcoding for mpeg dash in the cloud environment. In *Proceedings of the 5th ACM Multimedia Systems Conference.* ACM, 2014.

[14] R. Mehrotra and P. Bhattacharya. Modeling the evolution of user-generated content on a large video sharing platform. In *Proceedings of the 24th International Conference on World Wide Web Companion*, pages 365–366. International World Wide Web Conferences Steering Committee, 2015.

[15] S. Mitra, M. Agrawal, A. Yadav, N. Carlsson, D. Eager, and A. Mahanti. Characterizing web-based video sharing workloads. *ACM Transactions on the Web (TWEB)*, 5(2):8, 2011.

[16] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi. Video coding with h. 264/avc: tools, performance, and complexity. *Circuits and Systems magazine, IEEE*, 4(1):7–28, 2004.

[17] K. Pires and G. Simon. Dash in twitch: Adaptive bitrate streaming in live game streaming platforms. In *Workshop on Design, Quality and Deployment of Adaptive Video Streaming*, pages 13–18. ACM, 2014.

[18] M. Song, Y. Lee, and J. Park. Scheduling a video transcoding server to save energy. *ACM Trans. Multimedia Comput. Commun. Appl.*, Feb. 2015.

[19] T. Stockhammer. Dynamic adaptive streaming over http: standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 133–144. ACM, 2011.

[20] C. Timmerer, D. Weinberger, M. Smole, R. Grandl, C. Müller, and S. Lederer. Cloud-based transcoding and adaptive video streaming-as-a-service. *E-Letter.*

[21] A. Vetro, C. Christopoulos, and H. Sun. Video transcoding architectures and techniques: an overview. *Signal Processing Magazine, IEEE*, 20(2):18–29, 2003.

[22] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. *ACM SIGCOMM Computer Communication Review*, 45(4):325–338, 2015.

[23] C. Zhang and J. Liu. On crowdsourced interactive live streaming: a twitch. tv-based measurement study. In *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 55–60. ACM, 2015.

[24] L. Zhang, F. Wang, and J. Liu. Understand instant video clip sharing on mobile platforms: Twitter's vine as a case study. In *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*, page 85. ACM, 2014.

[25] W. Zhang, Y. Wen, J. Cai, and D. O. Wu. Toward transcoding as a service in a multimedia cloud: energy-efficient job-dispatching algorithm. *Vehicular Technology, IEEE Transactions on*, 2014.