Coping With Heterogeneous Video Contributors and Viewers in Crowdsourced Live Streaming: A Cloud-Based Approach

Qiyun He, *Student Member, IEEE*, Jiangchuan Liu, *Senior Member, IEEE*, Chonggang Wang, *Senior Member, IEEE*, and Bo Li, *Fellow, IEEE*

***Abstract*—With the advances in personal computing devices and the prevalence of broadband network and wireless mobile network accesses, end-users are no longer pure content consumers, but contributors, too. In today's crowdsourced streaming systems, numerous broadcasters lively stream their video content, e.g., live events or online game scenes, to fellow viewers. Compared to professional video producers and broadcasters, these new generation broadcasters are geo-distributed globally and highly heterogeneous in terms of the generated video quality and the network/system configurations. The scalability and heterogeneity challenges therefore lie on both broadcasters and the viewers, which call for massive transcoding, and two critical issues: 1) choosing video representation set that maximizes viewer satisfaction and 2) allocating computational resources that minimize operational costs, must be systematically optimized in the global scale. In this paper, we present a generic framework utilizing the powerful and elastic cloud computing services for crowdsourced live streaming with heterogeneous broadcasters and viewers. We jointly consider the viewer satisfaction and the service availability/pricing of geo-distributed cloud resources for transcoding. We develop an optimal scheduler for allocating cloud instances with no regional constraints. We then extend the solution to accommodate regional constraints, and discuss a series of practical enhancements, including popularity forecasting, initialization latency, and viewer feedbacks. Our solutions have been evaluated under diverse networks and cloud system configurations as well as parameter settings. The trace-driven simulation confirms the superiority of our design, while our Planetlab-based experiment offers further practical hints toward real-world migration.**

***Index Terms*—Cloud computing, crowdsourced live streaming, video transcoding.**

I. Introduction

With the advances in personal computing devices, and the prevalence of broadband and mobile network accesses, end-users are no longer pure content consumers, but contributors, too. Crowdsourcing was first introduced in 2005 by Merriam-Webster as a way to obtain resources by collecting contributions from crowds of people, instead of employees or suppliers, particularly in the online community. Since then, crowdsourcing has been attempted as the method for various tasks and applications. Over the past three years, crowdsourced live streaming have emerged too, with a series of real-world platforms being available in the market, such as Youtube Live, Azubu.tv, Hitbox.tv, Dailymotion Games, and Ustream, to name but a few. In these platforms, numerous broadcasters lively stream their video content, e.g., live events or online game scenes, to massive fellow viewers. One of the most successful representatives, Twitch.tv (also referred to as Twitch TV), allows any users to broadcast their live gaming videos from a wide range of platforms. It attracted more than 100 million unique viewers per month in 2014 with the peak online viewer number exceeding 1 million,[1] which further reached 1.5 million in March 2015 as measured by us. It is now ranked fourth among all the Internet traffic contributors in the US, only below Netflix, Google and Apple [6]. In September 2014, Twitch TV was acquired by Amazon for US$970 million, and more recently, in August 2015, Google has also officially launched YouTube Gaming to directly compete with Twitch, bolstering YouTube's streaming capabilities and offering the ability to upload videos running at 60 frames/s.

Compared to professional video producers and distributors (e.g., TV channels or Netflix), the new generation broadcasters are geo-distributed globally and highly heterogeneous in terms of the generated video quality and the network/system configurations. The system is also extremely dynamic over time, for both the overall population and the distribution of viewers in each region. For example, based on our measurement through Twitch API and data from [1], Twitch TV now has live video content coming from contributing sources in more than 100 different countries with over 150 different resolutions, being shared by viewers all over the world. Although the total number of online viewers peaks at 1.5 million, it can be less than 200 thousand in the same day.

Manuscript received August 30, 2015; revised January 14, 2016 and March 3, 2016; accepted March 11, 2016. Date of publication March 21, 2016; date of current version April 15, 2016. This work was supported by an NSERC Discovery Grant and an NSERC Strategic Project Grant. The work of B. Li was supported in part by the RGC under Contract 615613, Contract 16211715, and Contract C7036-15G (CRF), and in part by the NSF (China) under Contract U1301253. The guest editor coordinating the review of this manuscript and approving it for publication was Prof. Dapeng Oliver Wu.

Q. He and J. Liu are with the School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada (e-mail: qiyunh@sfu.ca; jcliu@cs.sfu.ca).

C. Wang is with InterDigital Communications, King of Prussia, PA 19404 USA (e-mail: chonggang.wang@interdigital.com).

B. Li is with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong, China (e-mail: bli@cse.ust.hk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2016.2544698

[1][Online]. Available: http://www.twitch.tv/year/2014

To cope with the heterogeneity and dynamics of the networks and end viewers, such major video content providers as YouTube and Netflix have deployed adaptive bitrate (ABR) streaming and more recently dynamic adaptive streaming over HTTP (DASH) [2]. The source videos with diverse qualities and formats are encoded into a uniformed format and further transcoded into several quality representations. Each representation (i.e., a quality version) of a video corresponds to a resolution at certain bitrate [3]. Crowdsourced live streaming platforms have also started deploying ABR and DASH with adaptive switching, but so far for a small number of broadcasters only. e.g., Twitch TV only offers transcoding service to its premium broadcasters, who only make up only 1% to 1.5% of all broadcasters. In the rest of the paper, we will interchangeably refer to terms $channel$ and $broadcaster$ to indicate the video broadcasting unit in crowdsourced live streaming systems.

Meanwhile, cloud computing has been widely deployed, and we have witnessed many cloud-assisted/-based multimedia services emerging in both academia and industry [2], [4], [5]. As video encoding and transcoding are both computation-intensive, and the demands from broadcasters and viewers vary dramatically over time, cloud with elastic resource provisioning is an accelerator of great potentials for the new generation of crowdsourced live streaming systems. The scalability and heterogeneity challenges from both broadcasters and viewers however are unprecedent, calling for advanced solutions beyond those for conventional video services [6].

In this paper, from realworld data, we identify the benefits and challenges when deploying cloud-based systems for crowdsourced live streaming. We examine the resource allocation problem of assigning geo-distributed cloud service to broadcasters for video transcoding and delivery, as well as the decision problem of choosing video representation set for every individual broadcaster to enable ABR/DASH. These two problems interact with each other: different video representation sets result in different computation workload for the cloud assignment, while the pricing and performance of the cloud services in different regions also affect the decision of video representation sets. We present a generic framework with an optimal scheduler for allocating cloud instances with no regional constraints. We then extend the solution to accommodate regional constraints, and discuss a series of practical enhancements. Our solutions have been evaluated with large-scale trace-driven simulations and Planetlab/Amazon EC2 based experiments.

II. Background and Related Work

Video streaming over Internet has become a mainstream "killer" application over the past two decades [2]. Free video sharing site Youtube now has over 1 billion viewers and around 300 h of video being uploaded every minute;[2] Fee-based commercial video provider Netflix has 65.55 million subscribers worldwide, doubled its subscription number in just two and half years.[3] The content generation and delivery architectures have also evolved, from a single server framework, to content distribution networks (CDN) [7] and peer-to-peer [8] for large scales. More recently, cloud-assisted/-based systems have emerged for video streaming services [4], [5], [9], [10]. To accommodate network and device heterogeneity and dynamics, ABR streaming [2] and the standardized DASH have been widely used [11]. They allow heterogeneous viewers to select among multiple quality versions of the same video, but require intensive computational resources for transcoding these video quality versions.

There have been many earlier works addressing resource allocation and optimization in streaming systems. Wu *et al.* [12] considered coping with geo-distributed and time-varying video demand worldwide; Wang *et al.* [4] introduces the strategy of renting cloud servers with consideration of viewer locality and viewer assistance. For transcoding, Huang *et al.* [5] designed CloudStream, which schedules the video transcoding tasks inside a cluster based on video properties. Chen *et al.* [9] made an initial study on the emerging crowdsourced live streaming systems, and proposed a cloud renting strategy to optimize the cloud site allocation for transcoding video contents provided by geo-distributed live broadcasters. They have generally assumed that the workload is fixed, e.g., the video representation set for transcoding are pre-determined.

The overall workload for transcoding indeed depends on both the set of channels to be transcoded and the set of targeted video representations. Pires and Simon [13] studied the tradeoff between the cost for transcoding and video delivery. Toni *et al.* [14] introduced an integer linear programming model to find the optimal set of video representation from the users' *Quality of Experience* (QoE) perspective. Aparicio-Pardo *et al.* [15] further examined live video stream processing in the cloud, with adaptive selection for the video representation set. Nevertheless, these works have not addressed the geo-distributed workload assignment and video delivery in the crowdsourced context, particularly with cloud.

III. Motivations and System Structure

A. Observations and Insights

From February 2015 to June 2015, we captured the data of the broadcasters from Twitch TV every five minutes, using Twitch's public application programming Interface (API).[4] This public API provides the game name, viewer number, stream resolution, broadcaster language, premium partner status (Yes/No), and some other related information of each channel. It however does not provide certain detailed information about the viewers, such as their network conditions, choices of video resolution, or physical distributions. The geographical and bandwidth information of the broadcasters is not provided, either. Therefore, we rely on datasets from [16] and [1] to estimate network and demographic statistics, respectively.

Many of the observations from the Twitch TV data are consistent with those from conventional video streaming systems, e.g., heterogeneous and dynamic viewers [2], [17]. Yet the heterogeneity becomes much stronger, not only on the viewers' side, but also on the broadcasters' side. There is a variety of source

[2][Online]. Available: https://www.youtube.com/yt/press/statistics.html

[3][Online]. Available: http://www.statista.com/statistics/250934/quarterly-number-of-netflix-streaming-subscribers-worldwide

[4][Online]. Available: http://dev.twitch.tv

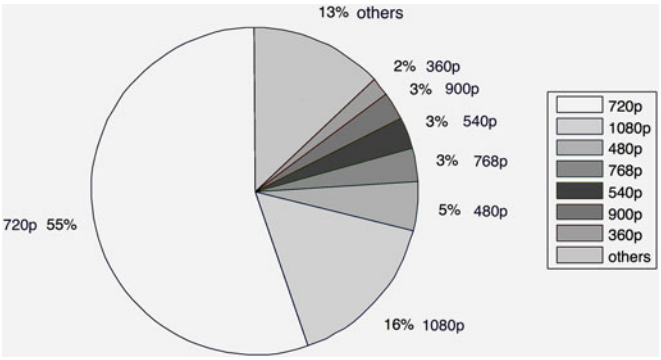


Fig. 1. Distribution of source stream resolutions.

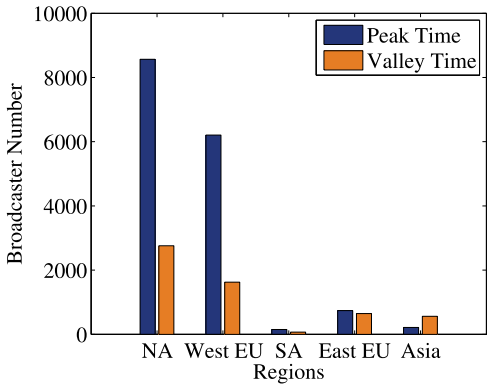


Fig. 2. Geographical distribution of Twitch crowdsourcers (broadcasters) worldwide.

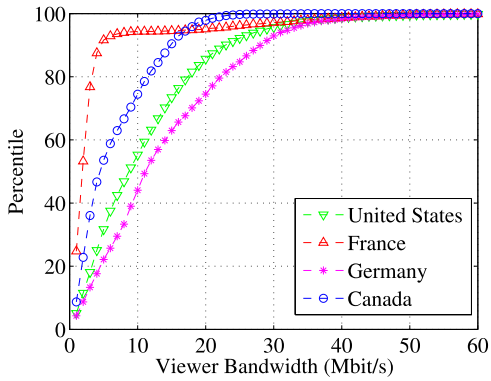


Fig. 3. Viewer bandwidth distribution (CDF) in different regions.

stream resolutions, as shown in Fig. 1, where 177 different resolutions ranging from 116 to 1600 p were detected from the data captured on March 7, 2015 at 15:00 PST. There is clearly a need to unify the video representations of the highly heterogeneous source videos.

The viewers' and the broadcasters' distributions are also heterogeneous across regions. Fig. 2 shows the distribution of the broadcasters in different continents. On the other hand, although it is hard to identify the actual stream resolution choice for each viewer, the preferred video representations of the viewers are likely vary from region to region, according to the general bandwidth conditions shown in Fig. 3. e.g., viewers in France may generally prefer higher quality video than those in Germany. As such, when unifying the representations, the regional differences should be taken care as well, and the target quality

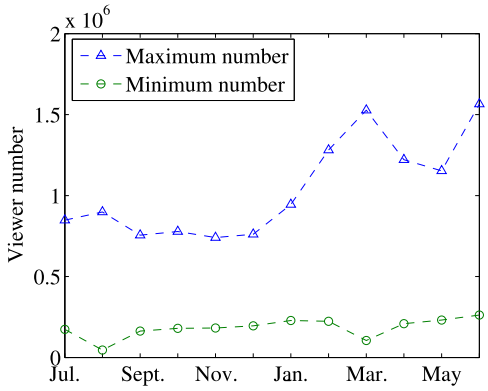


Fig. 4. Maximum and minimum online viewer number of every month from July 2014 to June 2015.

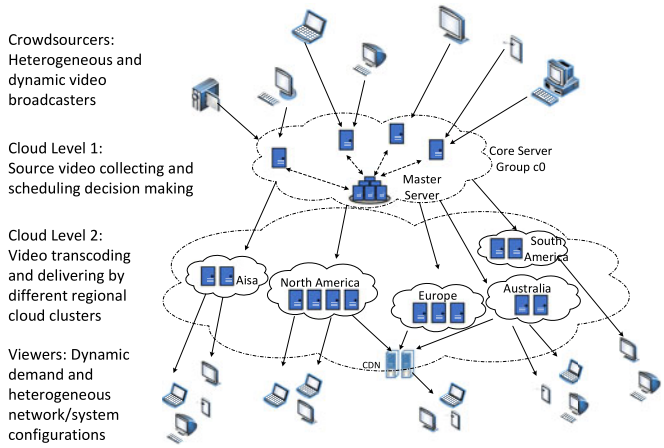


Fig. 5. Overall system architecture of crowdsourced live streaming with cloud.

versions therefore should carefully chosen. Given the massive broadcasters, the system scale is also becoming more dynamic. Fig. 4 illustrates the maximum and the minimum viewer numbers within each month from July 2014 to June 2015 (data from Stats TwitchApps[5]). The area between two lines in the figure can be regarded as the scale dynamics of the system, which is noticeably growing over time.

Above observations motivate us to use elastic cloud to transcode crowdsourced video content, with careful management of resource allocation and target video representation set.

B. System Architecture Overview

We now illustrate the overall system architecture in Fig. 5, which seeks to address the unique challenges of the crowdsourced live streaming. At the top level, the many crowdsourcers (i.e., broadcasters) upload video streams from different platforms to the cloud in real time through such protocols as real time messaging protocol [6]. The video broadcasters form a set $B = \{b_1, b_2, \ldots, b_n\}$. Once entering the cloud, a source stream will first reach one *core server* from a core server group c_0, which is regarded as cloud level 1. These core servers are distributed in different regions, $\mathbb{A} = \{A_1, A_2, \ldots, A_l\}$, and keep

[5][Online]. Available: http://stats.twitchapps.com/

TABLE I
NOTATIONS IN THE FORMALIZED PROBLEM

Name	Description
$\mathbb{C}$	set of available cloud instances for renting
$\mathbb{R}$	global video representation set
w_i	target video representation set for channel i
B	set of broadcasters
$S(w_i)$	satisfaction level with representation set w_i
$P(i)$	popularity of channel / broadcaster i
$\mathbb{S}$	set of renting schedule
$l(j)$	cost for renting a cloud server j per unit time
$v(j)$	cost for unit outbound data of cloud server j
$C_v(\cdot)$	cost for outbound data of amount $(\cdot)$
$\mathbb{A}$	set of regions
$D(A,t,r)$	demand of video at bitrate r at time t in region A

reporting the broadcasters' information to a *master server*. The master server will further assign server instances in the cloud for transcoding tasks. The core server then forward the streams to the allocated cloud instances defined as $\mathbb{C} = \{c_1, c_2, \ldots, c_m\}$, in cloud level 2. Each cloud instance will transcode the original source stream into a target quality version, and then broadcast to viewers. For each channel, we try to assign cloud servers from the same region, to avoid duplicated cross-region transfer of source streams with extra cost, and more importantly, to mitigate the delay variance of different quality versions. Given that video transcoding, especially realtime video transcoding, is computationally intense, we need one dedicated CPU (also referred to as *core*) for each output video representation to guarantee the transcoding quality. Though multi-core cloud instances can also be deployed to generate multiple video representations simultaneously, the number of output video representations shall not exceed the number of cores, and from [18] we know the rental price of a cloud instance is proportional to the number of cores of the instance. Therefore, we regard one cloud instance per core as the basic computational unit. In cloud level 1, we propose to use *General Purpose* cloud instances (e.g., Amazon `m3` and `m4`) to collect source stream, and use *Compute Optimized* instances (e.g., Amazon `c3`) as master server for decision making. In cloud level 2, we choose *Compute Optimized* instances for video transcoding and delivering. Table I summarizes the important notations.

IV. CLOUD RESOURCE SCHEDULING: PROBLEM AND OPTIMAL SOLUTION

We first formulate the problem of cloud resource scheduling, and then put forward a greedy approach for optimal scheduling when there are sufficient cloud instances.

A. The Cloud Resource Scheduling Problem

Our task is to assign cloud server instances[6] in different regions to live channels for video transcoding. In particular, we need to decide the target *video representation set* for each broadcaster, according to its popularity and viewer distribution, as well as the cloud service pricing policies in each region.

Let $\mathbb{R} = \{r_1, r_2, \ldots, r_m\}$ be the global video representation set; e.g., if $\mathbb{R}$ includes 360, 480, 720, 1080, 4 K, then each channel, after transcoding, can have at most these five video quality levels, but can be less with resource constraints. Let w_i be the target video representation set of broadcasting channel b_i, which starts from including the lowest level only, and if resources are available, expands in the ascending order of video quality levels, i.e., $\{360p\} \rightarrow \{360p, 480p\} \rightarrow \cdots \rightarrow \{360p, 480p, 720p, 1080p, 4K\}$. This way, each channel will offer at least the lowest quality level that most of the viewers can watch and the user experience can be gradually enhanced by adding more levels; yet other orders of expansion would be used as well depending on the specific application. We further define the average *satisfaction level* for a channel with representation set w_i as $S(w_i)$, and $P(i)$ as the popularity level for broadcaster b_i, which is proportional to the number of its viewers. The overall satisfaction level for a live channel is $P(i) \cdot S(w_i)$. Specifically, we base our satisfaction function on the experimental results in [14], and we further modify the parameter settings of this logarithmic shaped function to adjust to our scenario, so that the maximum value is 1 when w_i contains all the available video representations. In our evaluation, the parameters a and b in the below satisfaction function are set to 1 and $1 - log(5)$, respectively, so that $S(w_i) = 1$ when $|w_i| = 5$. The overall satisfaction below then measures the viewers' QoE for the whole system

$$S(w_i) = a \cdot log(|w_i|) + b; \;\; Q = \sum_{i \in B} P(i) \cdot S(w_i).$$

In terms of the cloud service, c_0 is the pre-allocated core server group, and $\mathbb{C} = \{c_1, c_2, \ldots, c_m\}$ is the set of cloud servers we can rent from. $l(j)$ and $v(j)$ are the rental cost per unit time and unit outbound data charge of cloud instance j, respectively. $C_v(\cdot)$ presents the total cost of outbound data of amount $(\cdot)$. We assume that a cloud instance, once being rented, can start service instantly. We will later address the impact of the initialization latency. We also define the cloud service rental schedule as $\mathbb{S} = \{(x_1, t_1, d_1), (x_2, t_2, d_2), \ldots, (x_k, t_k, d_k)\}$, in which one unit (x_j, t_j, d_j) means we rent cloud instance x_j from time t_j for duration d_j. Note that the scheduling policy must be *conflict free*, that is, one cloud instance cannot to be rented twice at the same time.

Given the physical distribution of viewers, the locations of the rented cloud instances should be optimized as well, which reduces the delay between viewers and cloud servers and minimizes the long-haul traffic, particularly the expensive cross-ISP traffic. Given regions set $\mathbb{A}$, for a cloud server c_j, we use $c_j \in A$ to denote that it is in region A. Additionally, we use $D(A,t,r)$ to represent the online population of viewers (or demand) watching channel at r bitrate in region A at time t. The overall cost for renting the cloud service and bandwidth usage for data traffic at

[6]Unless otherwise specified, we use *cloud server* and *cloud instance* interchangeably in the paper.

time t is therefore

$$l(c_0) + \sum_{(x_j,t_j,d_j)\in\mathbb{S}} l(x_j)\cdot I_{[t\in[t_j,t_j+d_j)]} + C_v\Big(\sum_{A\in\mathbb{A}}\sum_{r\in\mathbb{R}} D(A,t,r)\cdot r\Big)$$

where the second and third parts are for rental costs and outbound data charges, respectively. As the core server group c_0 is predetermined and fixed, we focus on minimizing the left parts, which is denoted as C

$$C = \sum_{(x_j,t_j,d_j)\in\mathbb{S}} l(x_j)\cdot I_{[t\in[t_j,t_j+d_j)]} + C_v\Big(\sum_{A\in\mathbb{A}}\sum_{r\in\mathbb{R}} D(A,t,r)\cdot r\Big).$$

The cost for the cross-region traffic F is

$$F = \sum_{A\in\mathbb{A}}\sum_{r\in\mathbb{R}}\Big(\sum_{(x_j,t_j,d_j)\in\mathbb{S}} D(A,t,r)\cdot r\cdot I_{[t\in[t_j,t_j+d_j),x_j\notin A]}\Big)$$

where the indicator function $I_{[\cdot]}$ is valued 1 if $[\cdot]$ is true, or 0 otherwise. $D(A,t,r)\cdot r$ is the total amount instant data traffic consumed at representation r by viewers in region A. And $D(A,t,r)$ of each region is determined by the video representation set w_i of each channel, and the distribution of viewers consuming each representation of w_i in that region.

Thus, our goal is to match one or more cloud servers to one broadcaster to maximize the reward and to minimize the overall cost. In particular, the reward is the total viewer satisfaction, or the overall QoE, while the cost comes from renting servers, outbound data charges and cross-boundary network traffic; The number of cloud instances matched to a broadcaster is the number of video representations that broadcaster has. Unfortunately, these objectives come with inherent contradictions, e.g., providing more video representations leads to higher viewer satisfaction level but increases the rental cost. We therefore resort to a comprehensive cost, a sum of them with different weights that reflect the tradeoff

$$Cost_{\text{comprehensive}} = \alpha\cdot(Q_{\max}-Q) + \beta\cdot C + \gamma\cdot F$$

where α, β, γ are the weights depending on specific application and the network, cloud, and end-user configurations. We will examine their respective impact later. $Q_{\max}$ is the maximum viewer QoE, which is achieved when all viewers are watching the video with maximum satisfaction, i.e., when every channel is provided with all available video representations. The second part C is only about the actual financial cost we need to minimize, while the third part F relates to the overall system performance, especially the streaming delay and playback fluency.

B. Greedy Rental Scheduler (GRS)

We now consider the solution for renting cloud instances at a given time t. Algorithm 1 shows a GRS. Given the detailed viewer information of each channel, the channels are first ranked in decreasing order of popularity (line 1). The cloud instances in each region are sorted in ascending order of $l(\cdot)$, as the rental cost is the major part of overall cost, and indeed the data charge rate $v(\cdot)$ is always proportional to $l(\cdot)$. Such pre-sorting allows more popular broadcasters to be considered first, and cloud servers with cheaper unit cost to be selected first in each region. Then, for each channel, a queue is constructed containing all possible rental schemes (line 3–9). The scheduler uses the comprehensive cost as key to sort this queue and chooses the optimal scheme with the least comprehensive cost of that channel (line 10–12). If the least-cost scheme is not feasible (i.e., not enough cloud instance for the remaining channels), the next scheme along the queue is selected (line 13–15). The GRS records the rental scheme in every step and returns the overall schedule before its termination. Note in the algorithm, the function $cost(i,j,s)$ calculates the comprehensive cost to serve the ith channel with j cheapest cloud instances in region s, using the last formula provided in the previous Section IV-A (the linear combination of $Cost_{\text{comprehensive}}$). The time complexity of GRS is $O(|B|\,|\mathbb{A}|)$, as every broadcaster (line 3) will be tried with cloud instances in every region (line 5), assuming $|\mathbb{R}|$ is a small and fixed number (e.g., $|\mathbb{R}|$ is 5 for Twitch TV).

Algorithm 1: Greedy Rental Scheduler

1: Sort B in decreasing order of their Popularity $P(\cdot)$
2: Sort $\mathbb{C}$ in each region in ascendant order of $l(\cdot)$
3: **for** i from 1 to $|B|$ **do**
4: initialize an empty queue $queue$
5: **for** region s in $\mathbb{A}$ **do**
6: **for** j from 1 to $|\mathbb{R}|$ **do**
7: push $cost(i,j,s)$ together with its corresponding scheme into $queue$
8: **end for**
9: **end for**
10: sort $queue$ in ascending order of cost
11: **if** the first item x in queue is feasible **then**
12: add x to schedule and decrease corresponding cloud instance number
13: **else**
14: check the next item in $queue$ until one is feasible
15: **end if**
16: **end for**
17: return the schedule

Theorem 1. The GRS is optimal when there is sufficient cloud instances in each region.

Proof: We prove by induction that after $|B|$ iterations of the main loop, the GRS has the minimum comprehensive cost.

Base case: if we only have one channel, the algorithm returns the schedule with minimum comprehensive cost, which is the optimal schedule for this channel. The result is optimal.

Induction step: assume the claim is true after the optimal schedule for k channels have been calculated. For the $(k+1)$th channel, the algorithm again tries to use the cheapest idle cloud instances from each region to calculate possible rental schemes and sorts them by comprehensive cost in a queue. Because we have unlimited number of cloud instances in each region, the first

scheme in the sorted queue with the least comprehensive cost can always be achieved, and therefore it again provides the optimal schedule for the $(k+1)$th channel. The new schedule for all first $(k+1)$ channels is still optimal. Therefore the greedy algorithm is optimal when there are always sufficient cloud instances in each region.

V. Accommodating Regional Limit on Service Supply

The GRS works optimally with sufficient cloud service provided in every region. Although ideally the cloud should provide virtually infinite amount of computational resources, most existing cloud service providers indeed have imposed strict usage constraints. For example, the on-demand and reserved instance limit for Amazon `c4.4xlarge` are 10 and 20 per account, respectively [19]; in Microsoft Azure, the limit for virtual machines per availability set is 100 [20]. Any demand beyond these limits requires special approval, and will not always be granted. Given the limited service supply, the greedy GRS keeps choosing optimal sub-schedules until the point where only enough cloud instances are to serve the same number of channels, resulting in poor schedule overall.

A. Scheduling With Limited Cloud Service Supply (SLCS)

We now extend the greedy solution to a dynamic programming based algorithm (SLCS). As shown in Algorithm 2, we define a $table$ (line 3) of size $|B| \cdot |\mathbb{C}|$ where $table(i,j)$ stores the best state found for first i channels with j cloud instances rented. A state is a data structure that stores the comprehensive cost ($state.cost$) and numbers of cloud instances left ($state.cloudOfRegion[regionNumber]$) in each region. $tmpState$ and $finalState$ are two instances of state representing the temporal intermediate state and final result state, respectively. We also define a $backtrackTable$ (line 3) to store the state transition information in $table$. For example, if the state of $table(i,j)$ is transferred from the state of $table(i-1,j-1)$, then the value of $backtrackTable(i,j)$ is $(i-1,j-1)$. Before scheduling, all broadcasters are ranked in descending order of popularity, and the cloud instances in each region are sorted in ascending order of $l(\cdot)\cdot v(\cdot)$. The table is initialized to be empty. For each broadcaster i, a transition is made by adding a new rental scheme for this broadcaster to the previous state which is for the $(i-1)$th broadcaster (line 4–15). The new rental scheme is optimally calculated in the greedy manner introduced in the GRS. Meanwhile, $backtrackTable$ records the transition information of each update. After the iteration finishes, the last row of $table$ contains the final states of all possible schedules, from which the state with the least comprehensive cost is selected. We backtrack from that state using the $backtrackTable$, and return the whole schedule.

Algorithm 3 shows the $calculateNewState(i,m,s,state)$ in Algorithm 2, which returns a state of serving the ith channel with m cloud instances in region s based on previous state $state$. The function $cost(i,m,s)$ is the same as has been used and explained in Section IV-B. The time complexity of SLCS is $O(|B|\,|\mathbb{C}|\,|\mathbb{A}|)$, as it iterates through all broadcasters and all cloud instances (line 4 and 5) to update the $table$, and for each update it iterates through all regions (line 8).

Algorithm 2: Scheduling With Limited Cloud Service Supply (SLCS)

1: Sort B in decreasing order of $P(\cdot)$
2: Sort $\mathbb{C}$ in each region in ascendant order of $l(\cdot)$
3: Initialize $table$, $tmpState$, $finalState$, $backtrackTable$
4: **for** i from 1 to $|B|$ **do**
5: **for** j from i to $\min\left(|\mathbb{R}|\cdot i, |\mathbb{C}|-(|B|-i)\right)$ **do**
6: **for** m from 1 to $\min(|\mathbb{R}|,\ j-i+1)$ **do**
7: **for** region s in $\mathbb{A}$ **do**
8: $newState \leftarrow calculateNewState(i,m,s,table(i-1,j-m))$
9: **if** $newState.cost < tmpState.cost$ and $newState$ is feasible **then**
10: $tmpState \leftarrow newState$
11: **end if**
12: **end for**
13: **end for**
14: **if** $table(i,j).cost > tmpState.cost$ **then** $table(i,j) \leftarrow tmpState$
15: **end if**
16: record the transition of $table(i,j)$ into $backtrackTable$ for backtracking
17: **end for**
18: **end for**
19: $finalState \leftarrow \min_{|B|\le c\le|\mathbb{C}|}(table(|B|,c))$
20: Backtrack from $finalState$ and return the whole schedule

Algorithm 3: $calculateNewState(i,m,s,previousState)$

1: Input: i is the ordered number of current channel
m is the number of representations
s is the region of cloud servers chosen from
$previousState$ is the old state to be based on
2: initialize $newState \leftarrow previousState$
3: initialize $newCost \leftarrow cost(i,m,s)$
4: $newState.cost \leftarrow newState.cost + newCost$
5: $newState.cloudOfRegion[s] \leftarrow newState.cloudOfRegion[s] - m$
6: return $newState$

B. Faster Implementation for Large Scale

The SLCS algorithm runs reasonably fast when the system scale is small to medium. For larger scale, we suggest a faster implementation, SLCS*, which simplifies SLCS by using a heuristic to rank cloud instances ahead of time, and then chooses the generally most preferred instance first when assigning the rental schedule. Algorithm 4 shows the details of SLCS*. We use the heuristic to rank all cloud instances based on their unit rental price $l(\cdot)$. We leave it for further study to find more

Algorithm 4: SLCS* - Fast Heuristic for SLCS

1: Sort B in decreasing order of $P(\cdot)$
2: Sort all $\mathbb{C}$ in ascendant order of $l(\cdot)$
3: Initialize $table$, $finalState$, $backtrackTable$
4: **for** i from 1 to $|B|$ **do**
5: **for** j from i to $\min\left(|\mathbb{R}|\cdot i, |\mathbb{C}| - (|B|-i)\right)$ **do**
6: $table(i,j) \leftarrow \min_{1\le m\le|\mathbb{R}|}(table(i-1, j-m) + cost_2(i, j-m, m))$
7: record the transition of $table(i,j)$ into $backtrackTable$ for backtracking
8: **end for**
9: **end for**
10: $finalState \leftarrow \min_{|B|\le c\le|\mathbb{C}|}(table(|B|, c))$
11: backtrack and return the whole schedule

comprehensive heuristics. In the algorithm, the $table$ is of size $|B|\cdot|\mathbb{C}|$ where $table(i,j)$ stores the cost of the best sub-schedule for serving the first i most popular channels with first j most preferred cloud instances. The $backtrackTable$ is the same to that in SLCS. The function $cost_2(i,j,m)$ returns the comprehensive cost for the ith broadcaster using m cloud instances, selected from the jth preferred instance on the list. More specifically, $cost_2(i,j,m)$ calculates the $Cost_{\text{comprehensive}}$ for the ith broadcaster using the jth, $(j+1)$th, ..., and $(j+m-1)$th cloud servers on the ranked list.

The heuristic works best when each cost component has similar preference on cloud instances. Since the faster implementation does not keep the information of cloud supply in each region explicitly, the time complexity of SLCS* algorithm is reduced to $O(|B|\,|\mathbb{C}|)$, as it only iterates through broadcasters (line 4) and cloud instances (line 5).

VI. Performance Evaluation

A. Data Sets

1) Twitch Trace Data: We use the trace data captured by the Twitch's API for our large scale simulation. We refer to [1] to assign the global viewer distribution for each channel. For a given channel, we assume there is an equal number of viewers watching each video quality version. This assumption does not need to be very strict; As long as there are some viewers consuming each target video representation, our system works, since the overall trend of viewer satisfaction and outbound bandwidth cost is consistent. The only scenario affecting the scheduling result is the extreme case where certain target representations are not consumed by viewers at all, in which case we can save the rental cost for cloud instances transcoding unwatched video representations. However, with the massive viewer base, the possibility of such extreme cases is little, and for simplicity of calculation, we assume the number of viewers for each video representation to be identical. On the other hand, video representations are provided in a fixed order to ensure the majority of viewers can watch the stream (i.e., lower quality version is provided first), though in fact certain quality versions are more preferred [21]. By default, $|\mathbb{R}|$ is set to 5 as in Twitch TV.

2) Amazon EC2 Pricing: Given that video transcoding is computation-intensive, we use the *Compute Optimized* Amazon `c3` server for transcoding. Amazon `c3` server is designed for heavy computation works, and also has enhanced networking feature for significantly high packet per second performance, lower network jitter and latency. In most regions, Amazon provides `c3` service at five power levels from `large` to `8×large`, with their prices mostly linear to their theoretical computation power/CPU number. In terms of data traffic, Amazon only charges outbound data, not the in-cloud data within the same region. Noticeably, cross-region in-cloud data transfer will be charged, and the pricing policies vary from region to region. Since the majority of the in-cloud data traffic happen in the same region, we do not consider the cost of cross-region in-cloud data traffic. Table II lists the pricing policies of Amazon EC2's `c3` servers in five typical regions [18], which are used in our simulations.

B. Methodology

For comparison, we implemented four approaches to schedule the cloud service: $Top\text{-}N$, GRS, $SLCS^*$, $SLCS$, and the special scenario with no supply limit. Top-N, the current strategy of Twitch TV, provides full video representation set (i.e., all quality levels from Low to $Source$) to the top N most popular broadcasters, but only the original video quality for the rest of the broadcasters. We set $N = 300$ according to the Twitch partner program requirements and the trace data we collected. As Twitch's partner program requires each broadcaster to have at least 500 viewers regularly in order to become the premium partner,[7] only around 300 broadcasters are qualified. We ignore the channels with zero viewers, assuming there is no video transcoding and delivering need for them. The default parameters α, β and γ are set to 0.33, 0.34 and 0.33, respectively, and whenever changed later, their sum remains to be 1. For simplicity, we also assume there are equal number of cloud instances in each region, and the default limit is set to 2000. The situation where there are sufficient cloud instances in each region serves as an optimal baseline for comparison.

C. Simulation Results With Default Settings

We now present the results at the peak time (around 7:30 pm PST) and valley time (around 2:30 am PST) of a day, which correspond to the maximum and the minimum workloads of the system, respectively. Fig. 6 shows the overall comprehensive costs of schedules incurred by different approaches at peak time hours, of ten days in March 2015. To examine the impact of system dynamics, we use the data with significant oscillations. The total viewer number of the data over time is shown in Fig. 7. For ease of comparison, we normalize the comprehensive costs into the range from 0 to 1. We see that the greedy approach with no limit on server number (referred to as "No Limit" in figures) always has the lowest cost, which is regarded as the optimal

[7][Online]. Available: http://www.twitch.tv/p/partners

TABLE II
PRICING POLICIES OF AMAZON EC2 c3 SERVERS IN DIFFERENT REGIONS

	Instance price (per hour)			Outbound Data Price (per GB)			
Different Region	large	2xlarge	8xlarge	Up to 10 TB	Next 40 TB	Next 100 TB	Next 350 TB
US East (N. Virginia)	$0.105	$0.420	$1.680	$0.090	$0.085	$0.070	$0.050
US West (Northern California)	$0.120	$0.478	$1.912	$0.090	$0.085	$0.070	$0.050
EU (Frankfurt)	$0.129	$0.516	$2.064	$0.090	$0.085	$0.070	$0.050
Asia Pacific (Sydney)	$0.132	$0.529	$2.117	$0.140	$0.135	$0.130	$0.120
South America (São Paulo)	$0.163	$0.650	N/A	$0.250	$0.230	$0.210	$0.190

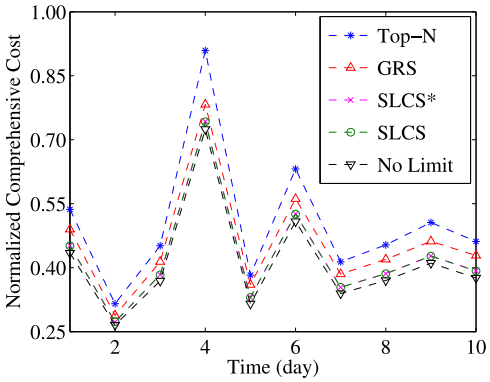


Fig. 6. Comprehensive costs at peak time in 10 days.

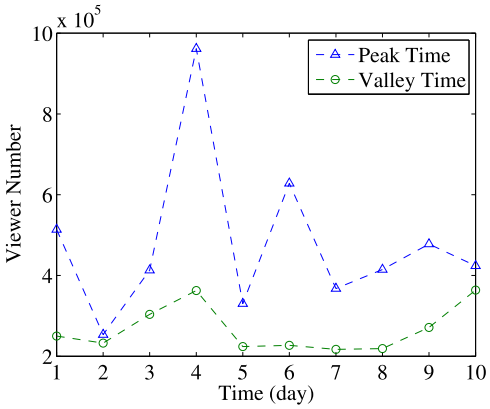


Fig. 7. Viewer number during peak/valley time of 10 days.

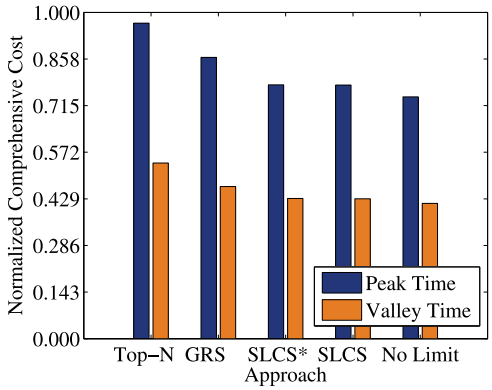


Fig. 8. Comprehensive cost at peak/valley time.

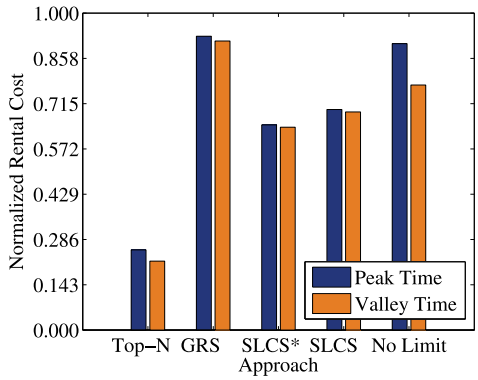


Fig. 9. Rental cost of different approaches.

baseline. Considering the constraint on cloud service supply, Twitch's default Top-N approach in general has the highest cost, followed by the GRS. The SLCS and SLCS* have lower total costs, and their results are very close. The SLCS* only has slightly higher costs than SLCS, suggesting that the heuristics closely match for the reality (e.g., in our trace setup, many viewers are in North America while the cloud services are the cheapest in that region as well). Over the time, the results of all approaches changes closely following the pattern of time-varying demand.

Fig. 8 shows the average comprehensive costs of the four approaches and the optimal baseline during peak hours and valley hours. In either case, the baseline solution with no limit gives the best average result. The second best results are generated by SLCS, which are 5% and 3.4% more than those of the "no limit" baseline during the peak and valley time, respectively. The cost difference between SLCS* and SLCS is again below 1%. Compared with SLCS, GRS incurs 10.9% and 8.5% more costs during peak time and valley time, respectively, while Top-N incurs 24.4% and 25.6% more.

We also compare two important components of the comprehensive cost: the rental cost for cloud service and the total viewer satisfaction, under the same supply constraint. Fig. 9 shows the normalized rental cost of the different approaches. We see Top-N has the least rental cost charged, which is reasonable given that it only provides transcoding to a very small portion of broadcasters. GRS however has the highest rental cost with around 39% more cost than SLCS. This is highly related to the shortage of cloud service, as the GRS tries to use more cloud service for the popular channels until there are

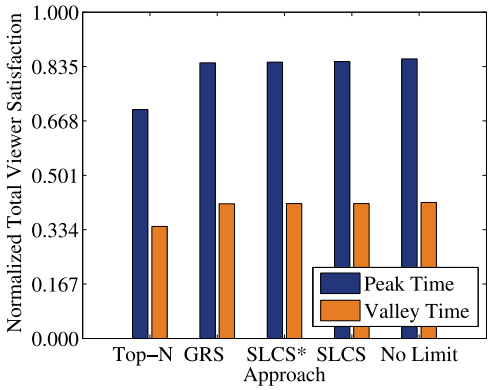


Fig. 10. Total viewer satisfaction.

TABLE III
TIME CONSUMPTION OF DIFFERENT APPROACHES

Approach	Top-N	GRS	SLCS	SLCS*
Time (sec)	5.7×10^{-5}	7.7×10^{-3}	7.3	1.9

only enough instances left, resulting in renting almost all cloud service in the end. The SLCS and SLCS* have lower rental cost compared to the greedy approach, and the advantage of SLCS* shown here is mostly related to its pricing based heuristics. The "no limit" scenario incurs excessive cost compared to SLCS, implying that more suitable cloud instances can be used to improve the overall performance if there is no regional constraint. Fig. 10 shows the total satisfaction achieved by each approach. We can see that, except for the Top-N approach, the total satisfactions are kept at high levels by other approaches as well as the "no limit" scenario. Again the Top-N approach suffers from its strategy of providing only a small portion of channels full transcoding service, resulting in relatively lower satisfaction level. However, the gap between Top-N and others is not large, as the top 300 channels in fact covers around 80% of the total viewers. The "no limit" scenario is only slightly better, with around 1% to 2% higher satisfaction than SLCS. As the satisfaction is proportional to the total popularity, the peak time results are always higher than the valley time results.

From the perspective of scheduling results, our approaches GRS, SLCS and SLCS* suggest different video representation sets for different channels, while existing strategy Top-N has a single threshold that provides either a full representation set or a single quality version. In terms of time consumption, we list the average scheduling computation time in Table III, from which we see SLCS* reduced the calculation time by 74% compared to SLCS. This difference could be more notable when the system evolves into larger scale, and more region choices are available. The running time measurement was conducted using a desktop with Intel Core i7-3770 CPU @ 3.4 GHz × 8, which emulates the *master server* in our model.

So far we have only discussed the situation under the same supply constraint. In the following parts, we further explore the performance of these approaches under different regional constraints and parameter settings.

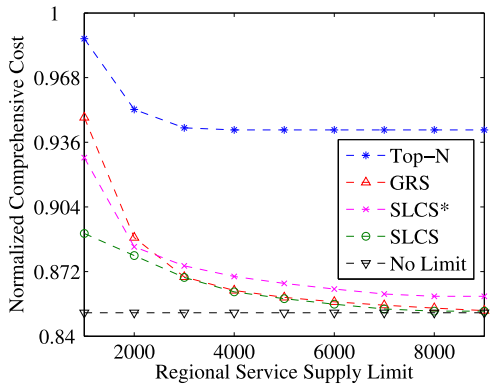


Fig. 11. Comprehensive cost with different regional limits.

D. Different Regional Constraints

To understand the influence of different service supply limits, we change the regional cloud instance number from 1000 to 9000. Fig. 11 shows the comprehensive costs of these approaches over different supply limits.

At the bottom of the figure, the "no limit" solution has the same optimal costs all the time as it is not influenced by the change of the limit. At the top, the Top-N approach always has the highest cost among all. Before the limit reaches 3000, the cost is decreasing. This is because there are not enough overall most suitable servers and when the limit increases, more best-fit servers are used. When the limit is more than 3000, the cost remains the same as the schedule converges and does not change any more. In the middle, we see SLCS performs the best overall, as compared to GRS and SLCS*, especially when the limit is below 3000. SLCS* has considerable advantage compared to GRS when the limit is lower than 2000, but becomes worse than GRS when the limit goes up. Eventually, when the limit is large enough (e.g., when it is 9000, which can be regarded as unlimited), both SLCS and GRS converge to the "no limit" result, while SLCS* always has a higher cost than the optimal baseline. This is because when there are infinitely many cloud instances, SLCS generate the schedule in exactly the same greedy manner as GRS does, resulting in the same rental schedule. However, since with current heuristic of SLCS* always choose the cheapest instance first rather than the best-fit one, it fails to generate the comprehensively optimal schedule when there are enough instance.

In short, SLCS has the overwhelming advantage when the cloud service supply is low while GRS is more suitable when there are sufficient cloud instances. SLCS* has good performance but suffers the bottleneck with the current heuristics.

E. Impact of Weight Settings

We next conduct simulations to investigate the impact of different weight settings. We change the ratio of two weights in a wide range (e.g., change α/γ from 10^{-3} to 10^{3}) while keeping the third equal to the denominator (e.g., keep γ equal to β). The two extreme cases (i.e., when the ratio is 10^{-3} and 10^{3}) respectively correspond to the situations where the scheduler ignores the corresponding component of the numerator and where

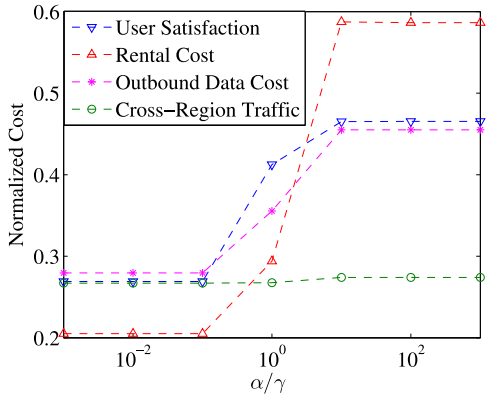


Fig. 12. Impacts of different α/γ on SLCS.

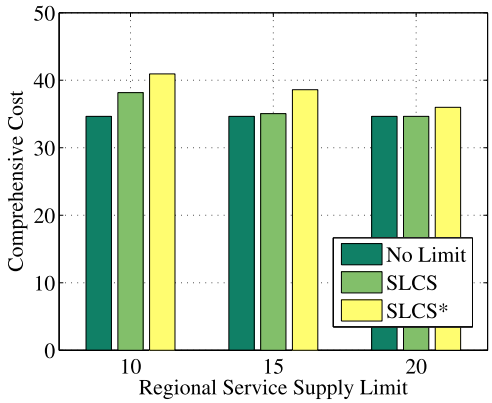


Fig. 13. Comprehensive cost with different regional limit.

the schedule is majorly decided by that component. Fig. 12 shows the impacts of different α/γ on different components of the cost, where β and γ are valued to be the same. We see when the weight α increases, more cloud servers are rented in order to improve the viewer satisfaction, with much more outbound data and slightly increased cross-region traffic. The cost for cloud service renting increased more than four times, and more expensive servers are rented. The outbound data cost is increased as more higher quality video versions are provided for a greater number of channels to ensure the overall viewer satisfaction. The cross-region traffic has been sightly increased, mainly caused by the increasing number of servers rented in regions not preferred, as the server in preferred regions have been used up. But as the change of cross-region traffic is very limited (only 4.5%), only a small portion of viewers will suffer from increased streaming discontinuity and latency.

Interestingly, we see the influences of weight settings are mostly between 10^{-1} to 10^{1}, as the result converges toward two extreme cases: schedule ignoring one factor and dominated by that factor. This indicates that each component cost is highly sensitive with different weight settings in the indicated middle range above, and also confirms the necessity to carefully consider multiple factors and the real application scenario in order to achieve the overall optimization. Specifically, our current simulation results suggest to select factor parameters with the same or similar values, or at least be within the same range to evenly reflect each component, with which the scheduler can have stable and consistent performance, though the exact parameter values shall shift dynamically to reflect the change of importance of different components (e.g., cloud rental price change, cross-region delay change, etc.).

VII. PlanetLab/Amazon EC2-Based Experiments and Practical Considerations

A. Planetlab/Amazon EC2-Based Experiments

We also have implemented a prototype of our proposed system. In our implementation, we deployed 223 PlanetLab nodes, in which ten nodes are broadcasters and 213 nodes are viewers. All viewer nodes are assigned to broadcasters according to the extremely skewed distribution similar to that in our Twitch data. Among these nodes, 88 of them are in Europe, 90 are in North America (NA), 6 are in South America (SA), 28 are in Asia and 11 are in Oceania. The virtual system infrastructure for scheduling and transcoding is built with Amazon EC2 c3 cloud servers from five different regions: Singapore (Asia), Sydney (Oceania), Oregon (NA), São Paulo (SA) and Frankfurt (Europe). All broadcasters and viewers keep updating their information to the master server located in NA. The master server uses the information collected to assign the transcoding schedule to other cloud servers. For comparison, we have implemented both SLCS and SLCS*, as well as the greedy approach with no limit in service supply. During the experiment, the broadcasters send a short source video to the assigned transcoding servers, where the video is transcoded by *ffmpeg* into different representations, and forwarded to corresponding viewers.

We first tested the scheduling result under different regional limit, as shown in Fig. 13. The result is consistent with that of our previous simulation. The "no limit" situation provides the optimal baseline, while with the increase of regional limit, SLCS and SLCS* are generating schedules with lower comprehensive costs. In the optimal case where the limit is 20 in each region, both the optimal baseline and SLCS use 10 Oregon instances and 20 Frankfurt instances, whereas SLCS* rents 20 Oregon instances and 11 Frankfurt instances. The cloud instances from these two sites (Oregon and Frankfurt) are highly favored given their relatively cheaper price and the fact that over 80% of the viewers are from these two regions. The difference in scheduling results lies in the different selecting order of cloud instances of SLCS and SLCS*. Under the current experiment setup, the scheduling result of SLCS converges when the regional limit is 20. Next, we measured the end-to-end streaming delay under the optimal schedule of SLCS with regional limit of 20. Fig. 14 shows the streaming delay of different resolutions perceived by viewers. The delay of different resolutions is similar, especially for viewers with fast network speed. The difference becomes obvious for slow network viewers (delay longer than 15 s), as a result of larger stream transfer time. Overall around 50% of the viewers have the delay within 5 s, and around 80% of the viewers have the streaming delay within 8 s. We also analyze the streaming delay by region, as shown in Fig. 15. Unlike the previous, regional delay difference is huge. In general viewers

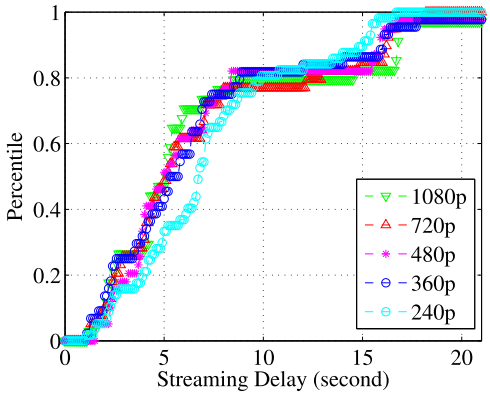


Fig. 14. Streaming delay of different resolutions (CDF).

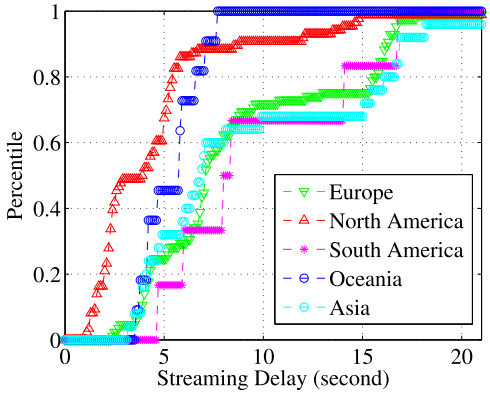


Fig. 15. Streaming delay in different regions (CDF).

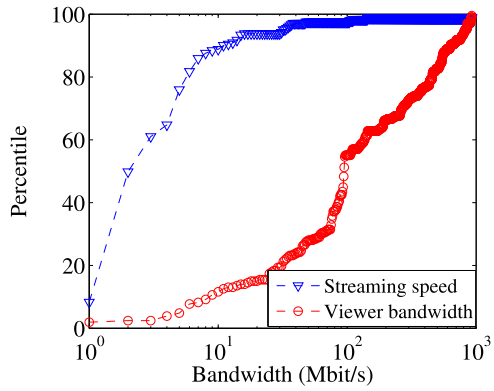


Fig. 16. Streaming speed and viewers' bandwidth (CDF).

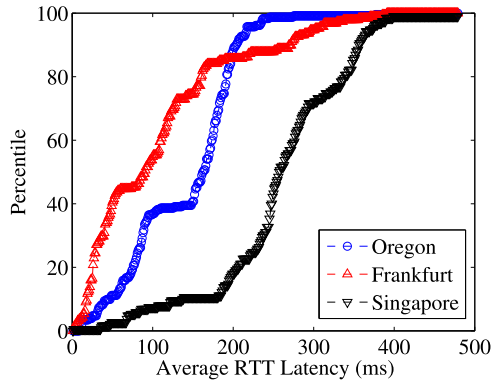


Fig. 17. Viewer-to-cloud average RTT latency (CDF).

in NA have the shortest delay, as around 50% of them have delay within 3 s and around 90% of them have delay within 7 s. Viewers in Europe and viewers in Asia have similar delay, despite the fact some cloud servers are deployed at the European site. Results from Oceania and SA are not very representative due to the small sample size.

B. Complimentary Experiments and Enhancements on Streaming Performance

We also conducted three complimentary experiments to explore the difference of theoretical and actual streaming performance, as well as the general performance of cloud servers in different regions.

In the first experiment, we assume one cloud instance is assigned to provide certain video quality to all viewers. We measured the HTTP streaming speed from that cloud instance to all its viewers (Planetlab nodes) and compare the result with the general bandwidth measured on those viewers. Fig. 16 shows the results measured with the cloud instance located in Oregon. We see huge gap between the general bandwidth and the actual cloud-viewer streaming speed, the latter of which is much lower. This indicates a portion of viewers will be forced to choose lower video quality though they indeed have enough bandwidth. Those viewers can greatly benefit from the assistance of a proxy local CDN. In the second experiment, we measured the average RTT latency between viewers and cloud instances in different regions. Fig. 17 shows the results with cloud instances located in Oregon, Frankfurt and Singapore, respectively. We see the cloud service in Oregon is highly preferred with 90% of nodes having RTT in 200 ms, while the streaming quality will be heavily affected if the instances locates in Singapore, given its overall high RTT. In the third experiment, we conducted an in-cloud data transfer experiment and compared the result with the user-cloud network performance. We found that though the network connection is enhanced within the same Amazon region, cross-region in-cloud data transfer has not been well optimized. Table IV shows the network performances of user-cloud and in-cloud connections within the same region A (Oregon) and between two typical regions A and B (Singapore). We see in both in-region and cross-region cases the in-cloud connection outperforms the user-cloud connection, but in-cloud data transfer suffers more from the cross-region bottleneck. Similar

TABLE IV
NETWORK PERFORMANCES OF USER-CLOUD AND IN-CLOUD CONNECTION

	RTT (ms)		Data Transfer Speed (MB/s)	
	Cloud Instance in A	Cloud Instance in B	Cloud Instance in A	Cloud Instance in B
User in A	134	191	3.88	0.302
Cloud Instance in A	0.733	160	93.2	5.52

phenomenon was also observed during the experiments conducted in other regions.

As suggested by our experiment results, CDN can be beneficial in some specific regions, and careful selection on cloud instance is important to ensure streaming performance. We can also differentiate the impacts of cross-region traffic between different regions. and quantize such difference in the modeling, and quantize such difference in the modeling. Once we have the statistics for the actual influence of cross-region traffic between every pair of regions, we can add an $InfluenceTable$, in which $InfluenceTable(i, j)$ stores the impact weight of the cross-region traffic from region i to region j. And thus in the comprehensive cost, we can have the below formula where F_{ij} is the data traffic from region i to region j

$$F = \sum_{i \in A} \sum_{j \in A, j \neq i} \cdot InfluenceTable(i, j) \cdot F_{\mathrm{ij}}.$$

C. Initialization Latency and Popularity Forecasting

So far we have focused on cloud service scheduling at a given time t. As shown earlier, the continuously running system has time-varying scale in both broadcasters and viewers; our algorithms therefore should be periodically executed with updated data, adding or removing cloud servers as needed. A newly activated cloud instance however needs time for the configuration and boot up [4]. Based on our measurement, it typically takes 2 min for an Amazon c3 server to boot up and function, and 30–40 s for an instance to be activated from stopped status. There is also a minimum rental duration, e.g., once an Amazon cloud instance is started, it immediately charges one hour's rental fee, even if this transition is made multiple times within a single hour.

Such a coarse granularity implies that periodical execution of the scheduling algorithm is practical. On the other hand, to minimize the influence caused by the initialization latency, we should pre-rent certain amount of cloud instances that is suitable for the coming time frame. The highly popular broadcasters are building up their own communities of viewers and tend to set up fixed schedules for broadcasting, so are popular events. For example, the live event *LCK Summer* (a tournament of League of Legends) on channel *Riot Games* alone attracted more than 340 000 viewers during its peak time on 9 August, 2015, and in fact the viewer number reached the same level when the same event happened before.[8] These make the demands from viewers and broadcasters more regular and predictable. As such, machine learning based models, e.g., regression models such as ARIMA [22] can be adapted to forecast the demand of the system, for renting the corresponding amount of cloud instances in advance. In the case that the excessive system demand is to be triggered by an special event, the system can reserve extra cloud instances well in advance given the event schedule.

D. Community Interaction and Synchronization

As mentioned, a new trend in the crowdsourced live streaming systems is that the channels-based communities of fellow viewers have emerged as a platform for communicating with each other, as well as interacting with broadcasters. The communication in these communities are mostly done through text messages, which can be easily transferred, processed, and posted in real time. The latency of the live streaming itself however may vary from viewer to viewer. Often the case, a much more delayed viewer comments on a scene watched by others a while ago. This issue becomes even more dramatic when it comes to some channels in which the game is collaboratively played by all the viewers, such as Twitch Plays Pokemon,[9] as viewers having unsynchronized game information may give misleading commands.

The existing Twitch implementation does not address the out-of-sync issue, which substantially affect the viewers' iterative experience [6]. We observe that the video rate plays a key role in controlling the live streaming latency, given that the initial latency is mainly decided by the buffering time, and after every video continuity, the re-buffering time adds to the latency. To this end, the DASH mechanism can be revised to control the video rate at the server/cloud side. The revision jointly considers the cloud-to-viewer data transmission delay (mostly related to the bandwidth), the network delays (e.g., processing delay, queuing delay and propagation delay), and the viewer player buffer load when deciding the video rate. As such, despite of the difference in network configurations, heterogeneous end-viewers can better synchronize their live streaming with other peer-viewers.

VIII. Conclusion

In this paper, we examined the emerging crowdsourced live streaming systems, in which both the broadcasters and the viewers can be highly heterogeneous. Our observation motivated the design of cloud rental strategy for an important job—transcoding, to optimize the resource allocation for geo-distributed live broadcasters. We formulated the problem, and provided a GRS which provides optimal solutions when there are sufficient supply of cloud service in each region. We then considered the real world case with limitation on cloud service supply, and put forward a dynamic programming based renting algorithm (SLCS). We further provided a heuristic-assisted SLCS* which ranks the cloud instances in advance to achieve faster running time. Our trace-driven simulation proved the superiority the scheduling results of SLCS and SLCS*. We further discussed a series of critical issues and enhancements toward practical deployment.

References

[1] "Twitch traffic and demographic statistics," Quantcast, San Francisco, CA, USA. [Online]. Available: https://www.quantcast.com/twitch.tv?country=MY#!countries.

[8] "Stream view numbers, February 2014 (Twitch only)," [Online]. Available: https://www.reddit.com/r/leagueoflegends/comments/1zcuid/stream_viewer_numbers_febuary_2014_twitch_only/

[9] [Online]. Available: http://www.twitch.tv/twitchplayspokemon/profile

[2] B. Li *et al.*, "Two decades of internet video streaming: A retrospective view," *ACM Trans. Multimedia Comput., Commun., Appl.*, vol. 9, no. 1s, 2013, Art. ID. 33.
[3] S. Akhshabi *et al.*, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http," in *Proc. 2nd Annu. ACM Conf. Multimedia Syst.*, 2011, pp. 157–168.
[4] F. Wang, J. Liu, and M. Chen, "CALMS: Cloud-assisted live media streaming for globalized demands with time/region diversities," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 199–207.
[5] Z. Huang, C. Mei, L. E. Li, and T. Woo, "CloudStream: Delivering high-quality streaming videos through a cloud-based SVC proxy," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 201–205.
[6] C. Zhang and J. Liu, "On crowdsourced interactive live streaming: a twitch. tv-based measurement study," in *Proc. 25th ACM Workshop Netw. Oper. Syst. Support Digit. Audio Video*, 2015, pp. 55–60.
[7] V. K. Adhikari *et al.*, "Unreeling Netflix: Understanding and improving multi-cdn movie delivery," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 1620–1628.
[8] J. Liu, S. G. Rao, B. Li, and H. Zhang, "Opportunities and challenges of peer-to-peer internet video broadcast," *Proc. IEEE*, vol. 96, no. 1, pp. 11–24, Jan. 2008.
[9] F. Chen, C. Zhang, F. Wang, and J. Liu, "Crowdsourced live streaming over the cloud," in *Proc. IEEE INFOCOM*, Apr.-May 2015, pp. 2524–2532.
[10] V. Aggarwal, V. Gopalakrishnan, R. Jana, K. K. Ramakrishnan, and V. A. Vaishampayan, "Optimizing cloud resources for delivering IPTV services through virtualization," in *Proc. 4th Int. Conf. IEEE Commun. Syst. Jan. Netw.*, Jan. 2012, pp. 1–10.
[11] C. Liu, I. Bouazizi, and M. Gabbouj, "Rate adaptation for adaptive http streaming," in *Proc. 2nd ACM Conf. Multimedia Syst.*, 2011, pp. 169–174.
[12] Y. Wu *et al.*, "Scaling social media applications into geo-distributed clouds," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 684–692.
[13] K. Pires and G. Simon, "DASH in twitch: Adaptive bitrate streaming in live game streaming platforms," in *Proc. ACM VideoNext CoNEXT Workshop*, 2014, pp. 13–18.
[14] L. Toni, R. Aparicio-Pardo, G. Simon, A. Blanc, and P. Frossard, "Optimal set of video representations in adaptive streaming," in *Proc. 5th ACM Conf. Multimedia Syst.*, 2014, pp. 271–282.
[15] R. Aparicio-Pardo, K. Pires, A. Blanc, and G. Simon, "Transcoding live adaptive video streams at a massive scale in the cloud," in *Proc. 6th ACM Conf. Multimedia Syst.*, 2015, pp. 49–60.
[16] S. Basso, A. Servetti, E. Masala, and J. C. De Martin, "Measuring DASH streaming performance from the end users perspective using neubot," in *Proc. 5th ACM Conf. Multimedia Syst.*, 2014, pp. 1–6.
[17] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "YouTube traffic characterization: A view from the edge," in *Proc. ACM Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2007, pp. 15–28.
[18] "Amazon EC2 pricing," Amazon Web Services, Inc., Seattle, WA, USA. [Online]. Available: http://aws.amazon.com/ec2/pricing.
[19] "Amazon EC2 FAQ," Amazon Web Services, Inc., Seattle, WA, USA. [Online]. Available: http://aws.amazon.com/ec2/faqs/#How_many_instances_can_I_run_in_Amazon_EC2
[20] "Azure subscription and service limits, quotas, and constraints," Microsoft Corp., Redmond, WA, USA. [Online]. Available: https://azure.microsoft.com/en-us/documentation/articles/azure-subscription-service-limits/#virtual-machines-limits.
[21] Z. Wang, L. Sun, C. Wu, W. Zhu, and S. Yang, "Joint online transcoding and geo-distributed delivery for dynamic adaptive streaming," in *Proc. IEEE INFOCOM*, Apr.-May 2014, pp. 91–99.
[22] G. P. Zhang, "Time series forecasting using a hybrid arima and neural network model," *Neurocomputing*, vol. 50, pp. 159–175, 2003.

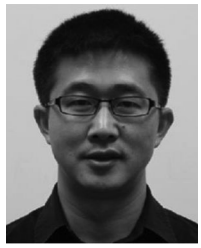
Qiyun He (S'16) received the B.Eng. degree from Zhejiang University, Zhejiang, China, in 2015, the B.Sc. degree from Simon Fraser University, Burnaby, BC, Canada, in 2015, and is currently working toward the M.Sc. degree in computing science at Simon Fraser University.

His research interests include cloud computing, social media, multimedia systems, and networks.

Jiangchuan Liu (S'01–M'03–SM'08) received the B.Eng. degree (cum laude) in computer science from Tsinghua University, Beijing, China, in 1999, and the Ph.D. degree in computer science from The Hong Kong University of Science and Technology, Hong Kong, China, in 2003.

He is a University Professor with the School of Computing Science, Simon Fraser University, Burnaby, BC, Canada. He is an EMC-Endowed Visiting Chair Professor of Tsinghua University, Beijing, China (2013–2016). From 2003 to 2004, he was an Assistant Professor with The Chinese University of Hong Kong, Hong Kong, China. His research interests include multimedia systems and networks, cloud computing, social networking, online gaming, big data computing, wireless sensor networks, and peer-to-peer networks.

Prof. Liu is an NSERC E.W.R. Steacie Memorial Fellow. He has served on the Editorial Board of the IEEE TRANSACTIONS ON BIG DATA, the IEEE TRANSACTIONS ON MULTIMEDIA, the IEEE COMMUNICATIONS SURVEYS AND TUTORIALS, IEEE ACCESS, the IEEE INTERNET OF THINGS JOURNAL, *Computer Communications*, and *Wiley Wireless Communications and Mobile Computing*. He is the Steering Committee Chair of IEEE/ACM IWQoS from 2015 to 2017, and Area Chair of IEEE INFOCOM (2017), ICME (2016), and ACM Multimedia (2014, 2016). He was a corecipient of the inaugural Test of Time Paper Award of IEEE INFOCOM (2015), the ACM TOMCCAP Nicolas D. Georganas Best Paper Award (2013), the ACM Multimedia Best Paper Award (2012), the IEEE Globecom Best Paper Award (2011), and the IEEE Communications Society Best Paper Award on Multimedia Communications (2009).



Chonggang Wang (S'99–A'02–M'04–SM'09) received the Ph.D. degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 2002.

He is currently a Member of Technical Staff/Senior Manager with InterDigital Communications, King of Prussia, PA, USA. His current research interests includes IoT, machine-to-machine communications, mobile and cloud computing, and big data management and analytics.

Dr. Wang served as a Panelist/Reviewer for the US National Science Foundation and Canada NSERC. He was the Vice-Chair of the IEEE ComSoc Multimedia Technical Committee (2012–2014). He is the founding Editor-in-Chief of the IEEE INTERNET OF THINGS JOURNAL (2014–2016), an Advisory Board Member of The Institute-IEEE (2015–2017), and is an Editorial Board Member for several journals, including the IEEE TRANSACTIONS ON BIG DATA and IEEE ACCESS. He has been selected as an IEEE ComSoc Distinguished Lecturer (2015–2016). He was a corecipient of the National Award for Science and Technology Achievement in Telecommunications in 2004 on IP Quality of Service from the China Institute of Communications. He was the recipient of the InterDigital Innovation Award in 2012 and 2013.

Bo Li (S'90–M'92–SM'99–F'11) received the B.Eng. (summa cum laude) degree in computer science from Tsinghua University, Beijing, China, and the Ph.D. degree in the electrical and computer engineering from the University of Massachusetts at Amherst, Amherst, MA, USA.

He is currently a Professor with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China. He was the Chief Technical Advisor for ChinaCache Corporation, Beijing, China. He was a Cheung Kong Visiting Chair Professor with Shanghai Jiao Tong University, Shanghai, China (2010–2013), and an Adjunct Researcher with Microsoft Research Asia, Beijing, China (1999–2007), and with Microsoft Advance Technology Center, Shanghai, China (2007–2009). He made pioneering contributions in the Internet video broadcast with a system called Coolstreaming, which was credited as first large-scale peer-to-peer live video streaming system in the world. This work appeared received the inaugural The Test-of-Time Award from IEEE INFOCOM (2015). His current research interests include datacenter networking, cloud computing, content distribution in the Internet, and mobile networking.

Prof. Li has been an Editor or a Guest Editor for over a dozen IEEE journals and magazines. He was the Co-TPC Chair for IEEE INFOCOM 2004. He was the recipient of six Best Paper Awards from the IEEE. He was the recipient of the Young Investigator Award from the Natural Science Foundation of China in 2005, and the State Natural Science Award (Second Class) from China in 2011.