When Crowd Meets Big Video Data: Cloud-Edge Collaborative Transcoding for Personal Livecast

Yifei Zhu, *Student Member, IEEE*, Qiyun He, Jiangchuan Liu, *Fellow, IEEE*, Bo Li, *Fellow, IEEE*, and Yueming Hu

Abstract—Deep penetration of personal computing devices and high-speed Internet has enabled everyone to be a broadcaster. In this crowdsourced live streaming service, numerous amateur broadcasters lively stream their video contents to viewers around the world. Consequently, these broadcasters generate a massive amount of video data. The set of video sources and recipients are big as well, so are demand for the storage and computational resources. Transcoding becomes a must to better service these viewers with different network and device configurations. However, the massive amount of video data contributed by countless channels even makes cloud significantly expensive for providing transcoding services to the whole community. In this paper, inspired by the paradigm of Edge Computing, we propose a Cloud-edge collaborative system which combines the idle end-viewers' resources with the cloud to transcode the massive amount of videos at scale. Specifically, we put forward tailored viewer selection algorithms after empirically analyses the viewer behavior data. In the meantime, we propose auction-based payment schemes to motivate these viewers participating in the transcoding. Large-scale trace-driven simulations demonstrate the superiority of our approach in cost reduction and service stability. We further implement a prototype in PlanetLab to prove the feasibility of our design.

Index Terms—Cloud computing, edge computing, video transcoding, algorithm design

1 Introduction

The advances of personal computing devices and the prevalence of high-speed Internet access have generated unprecedented amount of video traffic. Global Internet based traffic is expected to reach 3.3 ZB by 2021. Among this, videos are the dominant data format, which stand for 82% of the all consumer Internet traffic. Specifically, live videos, contributed by the emerging crowdsourced live streaming services, is becoming the prominent video format chosen by users to share their lives to the world. Live videos will account for 13 percent of video traffic by 2021 and will have increased 15-fold from 2016 to 2021. In the crowdsourced live streaming services, numerous amateur broadcasters lively stream their video contents to viewers around the world every second, every where. Fellow viewers watching the same channel constantly interact with each other and the channel broadcaster through chatting messages. Twitch TV, one of the most successful examples, hosts over two million broadcasters per month and supports 9.7 million daily active viewers in 2016 [1].

Similar to other traditional video streaming services over the Internet, interactive live streaming service providers also transcode same video content into different quality versions and distribute the appropriate version to better match the varying network conditions of end users and provide the best possible user experience [2]. These transcoding tasks are extreme CPU-intensive and require significant hardware. Cloud thus becomes a natural choice for most providers to conduct such compute-intensive transcoding tasks due to its elasticity and the "pay-as-you-go" billing model. For example, Netflix builds its whole video transcoding and delivering infrastructure in the cloud [3]. After acquired by Amazon, Twitch has also finished its migration of online chatting servers to the AWS in 2016 and keep expanding its server capacity to meet the transcoding demand.

The massive number of concurrent live channels, heterogeneous source contents and device configurations of end users in the interactive live streaming services generate a substantial amount of transcoding tasks. For example, overall Twitch hosted over two million unique monthly broadcasters, and 355 billion minutes of livecast has been watched in 2017. These video content all has to be transcoded first before they are consumed by the viewers. As a result, even a cloud-based approach becomes significantly expensive. Real-world service providers such as Twitch TV only provide transcoding services to a small portion of premium broadcasters, and only extend this service to normal broadcasters when there is extra capacity. Besides that, the requirement on streaming latency in such interactive

- *Y. Zhu and J. Liu are with the School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada. E-mail: yza323@sfu.ca, jcliu@cs.sfu.ca.*
- *Q. He is with Tableau Software, Vancouver, BC V6C 3E8, Canada. E-mail: qiyunh@cs.sfu.ca.*
- *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.*
- *Y. Hu is with the College of Natural Resources and Environment, South China Agricultural University, Guangzhou 510640, China. E-mail: ymhu@scau.edu.cn.*

Manuscript received 31 Jan. 2018; revised 13 Aug. 2018; accepted 25 Sept. 2018. Date of publication 1 Oct. 2018; date of current version 5 Mar. 2020. (Corresponding author: Jiangchuan Liu.) Recommended for acceptance by J. Li. Digital Object Identifier no. 10.1109/TNSE.2018.2873311

environment is even more stringent, as a high latency severely affects the viewer-broadcaster, viewer-viewer interactive experience [4]. While the cloud server may be far away from the live source, which inevitably causes higher streaming latency. We thus want to seek for more cost-efficient, low-latency solutions to transcode this massive amount of video data and cover more broadcasters.

Edge Computing is an architecture that uses a collaborative multitude of end-user clients or near-user edge devices to carry out a substantial amount of storage or computational workload [5]. It is a complementary component of cloud computing by extending the cloud computing paradigm to the network edges. While cloud is still the mainstream choice for deploying large-scale virtual infrastructure, centralized datacenter-based cloud is now migrating toward a hybrid mode with edge assistance [6]. Looking deeper into our services, we also observe the abundant computational resources residing in edge viewers, especially those high-end game devices. Therefore, inspired by the paradigm of Edge Computing and this observation, it would be great if we could involve these viewers into taking transcoding tasks to support cloud.

However, taking transcoding tasks inevitably costs extra power and bandwidth consumption; Viewers' willingness to take such tasks also varies from person to person. Though we notice the formation of channel oriented online communities and the support from users in terms of donation and subscription [7], [8], involving these viewers into computational intensive transcoding tasks still needs much stronger incentives to guarantee transcoding performance. Besides that, not all viewers are appropriate for taking these tasks because they have different network situations and device configurations. Therefore, an efficient mechanism that can select the qualified viewers to transcode the massive video data, and determine the appropriate rewards to motivate such contributions is needed.

To this end, we propose an auction-based selection approach combining cloud and edge computing together to offer transcoding services at low costs and with smaller delays. We first identify the key components for selecting a viewer to take transcoding tasks. We then study the offline situation with/without strict task number constraint separately, and extend our proposed algorithms to handle the online situation. To be specific, we use the proportional share mechanism as the basis for filtering qualified viewers in all offline situations. We improve the straightforward dynamic programming algorithm by proposing a more computational efficient heuristic algorithm supported by the priority queue. In the online situation, we propose a greedy algorithm to select viewers and determine payment guaranteeing computational efficiency, individual rationality, budget feasibility and truthfulness simultaneously. Large scale simulations driven by real traces demonstrate that our solution can provide reliable transcoding services at low cost. A prototype of our system is further implemented on the PlanetLab to demonstrate the feasibility of our system.

The remainder of this paper proceeds as follows: Section 2 examines the underlying challenges in conducting transcoding for crowdsourced live streaming services, presents the system overview of our cloud-edge collaborative system, and formally formulate the problem. Section 3 presents the algorithms to efficiently select reviewers and make payments for transcoding tasks in both offline and online situation. In Section 4, we evaluate our design through trace-driven simulations. We further build a prototype and implement it on PlanetLab in Section 5. Section 6 introduces the related work and discuss the difference between our work and the previous ones. Finally, Section 7 concludes the paper and discusses potential future directions.

2 System Model and Problem Formulation

In this section, we first examine the importance of latency in processing our big live video data. We then discuss the possibility of providing low-cost, small latency transcoding services through involving the computational power from end viewers. We further illustrate the overall architecture of our cloud-edge collaborative system and formally formulate the studied problem in our systems.

2.1 Why Delay Matters for Big Video?

Traditional video streaming services, like Youtube, strive to make their videos startup quickly and play with less rebuffering. Researchers have shown that viewers start to abandon videos after 2 seconds startup time in these traditional video streaming services [9]. High latency in playout delay greatly increases the abandon rate of viewers, making users less engaged in videos, and eventually harm the profit of content providers. For interactive live streaming services, it is the interaction feature that makes delay play an even more critical role. In twitch-like interactive streaming services, the interaction among viewers and between viewers and broadcasters also require bandwidth-hungry video streams to match its pace with other communication channels like audio and messages, otherwise users may become frustrated by other early spoilers or fail to fully engage in this participatory community. Undesirable interaction with peer viewers and the broadcasters can even drive viewers to abandon current channels [10]. In addition, applications like Twitch and Periscope allow viewers to express their enjoyment to certain content (using like or heart button). These positive feedbacks can help broadcasters to modify their content to better match viewers' tastes and attract more viewers. While mismatching between likes and streaming scenes could generate false positive feedbacks to the broadcasters, which greatly affects the information value hidden in these messages [11]. Latency plays a more significant role in this interactive live streaming services. It is, however, not easy to reduce it considering the scale and heterogeneity of this system. With naive deployment on cloud, 90% users have an interaction latency over 200 ms [12]. Several other research works have also indicated that with current cloud infrastructure, cloud is unable to satisfy the strict latency requirements in interactive live streaming services [13]. Researchers therefore are actively seeking for new approaches or architectures to reduce the interaction delay for viewers.

2.2 Why a Cloud-Edge Hybrid Approach?

Cloud is a natural choice for conducting video transcoding services. The elasticity of cloud allows service providers to scale up and down to match the dynamics of computational demand when hosting channels. Strict service level

agreements offered by cloud providers also guarantee crowdsourced streaming service providers stable performances in conducting computational-intensive tasks. Nonetheless, user experience in these delay-sensitive streaming services still cannot be easily guaranteed by total reliance on the cloud. The primary challenge is that new interactive live streaming services pose more strict requirement on latency as we have stated previously. Possible long distances between the centralized cloud datacenters and the ingesting server can incur large network delay. Similar observations in CDN literature prove that regional wide network latency can be 10 times more than that of local communication latency [14]. Besides that, the massive number of concurrent channels and heterogeneous configurations of viewers in this novel service generate a large combination of transcoding tasks which need to be fulfilled. These CPU-intensive tasks require significant hardware and thus are very costly, as confirmed by Twitch [15]. Constrained by the budget, the freemium-based service providers usually just offer transcoding services to premium channels now.

Behind the huge transcoding tasks are viewers who are actively involved in this interactive live streaming services. Each viewer spends twice as much as time watching interactive streaming events than on traditional streaming services. Among these massive viewer base, some of these viewers may also close to the ingress server. Furthermore, the increased computing power of personal computer also makes these viewers, especially game players, capable of handling computational tasks at ease. These factors all make edge viewers promising candidates to deliver low latency transcoding services, enabling more viewers watching their desired channels without interruptions. However, end viewers may come and leave the interactive video streaming service whenever they want. This flexibility on viewer side becomes the instability and uncertainty for conducting transcoding services, which makes total reliance on edge viewers not practical. A more promising solution naturally is to combine the power of cloud and edge viewers together to accomplish our transcoding services. Benefited by the computation potential lies in viewer side and their natural distance advantage to the servers, this hybrid approach can offer service provider high-quality transcoding services at low costs. The most critical problem thus is whom we should allocate these transcoding tasks to and at what price.

2.3 System Model

Globally, our system is divided into multiple regions, where each region has its own regional datacenter (also referred to as "regional server") for ingesting source videos, assigning transcoding tasks to viewers or cloud, recollecting transcoded video and forwarding the processed streams for further delivery. Fig. 1 shows the overall design. Specifically, source live streaming contents are first collected by the regional server through protocols such as Real Time Messaging Protocol (RTMP). Several viewers will then be selected for taking video transcoding tasks according to certain criteria. Unmatched tasks after this selection or during the live streaming process will be sent to dedicated cloud transcoding servers if no further satisfiable transcoding viewers can be found locally (results in cross-region assignment). The selected viewers and cloud servers transcode

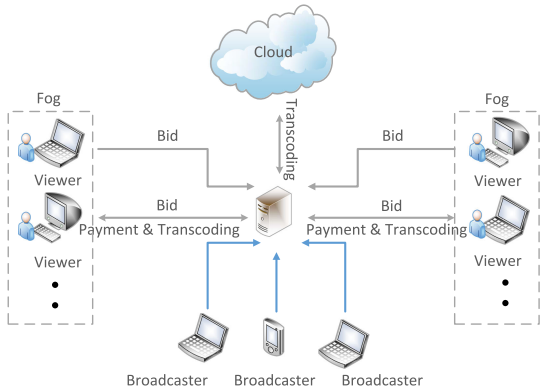


Fig. 1. System overview.

video contents into different quality versions, and send them back to the regional datacenter. Finally, the transcoded video is forwarded to other regional datacenters to serve all viewers. As can be seen, in our system, the computation processes for transcoding are distributed among the selected edge viewers and cloud; The decision process for selecting valid users is finished in each regional server.

2.4 Problem Formulation

We now consider specifically the incentive issue, which includes viewer selection and payment determination, in the above scenario. Intuitively, transcoding tasks can always be satisfied by viewers in our overwhelming viewer pool. However, in reality, the viewer and broadcaster numbers are highly dynamic over time. In both a single region and the whole global system, independent users may come, stay in the system for a distinct amount of time, and leave by their will. Causal selection cannot guarantee the involvement of highly qualified viewers and may cause a large number of reassignments, leading to high system overhead for recalculation and short absence of the target quality version during the reassigning period.

On the other hand, it is difficult to determine the right price to motivate these viewers since viewers have their own private cost functions. Casually setting a fixed price could lead to the overpricing or underpricing problem which either incurs a huge cost for the service provider or fails to provide enough incentives to motivate viewers. What is more, the sum of payments for all selected viewers is constrained by the limited budget for each channel since we are seeking a cost-effective design for video transcoding services. Therefore, efficiently utilizing this budget to fully motivate viewers and recruit qualified viewers is what we are aiming at.

We thus propose an auction-based approach to facilitate the transcoding task assignments from channels to the crowd of viewers and determine the payment for these transcoding viewers at the same time. Dynamic prices generated by auctions can help us fully motivate users while at a low cost in the competitive environment thanks to our large viewer pool. Carefully designed selection algorithms can also ensure us to select the appropriate viewers to take the transcoding tasks. For each region, we denote the live video channels as a set $C = \{c_1, c_2, \ldots, c_m\}$, and viewers as $V = \{v_1, v_2, \ldots, v_n\}$. Each channel $c_j \in C$ has a budget B_j which is part of the revenue from advertisements, and R_j is

the total number of transcoded video representations required for a channel c_j. Independent viewers have their private cost functions for taking the tasks, and they make strategic decisions to maximize their individual utility u_i, which follows the classical quasi-linear form. The utility of viewer i at time t is

$$u_i = \begin{cases} p_i - m_i & if\ v_i\, is\, selected \\ 0 & otherwise. \end{cases}$$

p_i is the payment to viewer v_i and m_i is the cost for users to transcode the corresponding channel. Under truthful bidding, m_i equals to b_i.

Each viewer v_i submits its bid b_i based on its own valuation function for taking such a task. After receiving the bid, the regional server acting as an auctioneer makes the assignment and payment decisions. Notably, the arrival/departure time and the cost function of each viewer are private, and may only be known to itself. Similarly, the arrival/departure time of each channel is also private. In other words, at time t, the scheduler has no knowledge of any incoming channels or viewers, neither does it know if any channel/viewer is going to terminate/leave soon. Therefore, the auctioneer must have an online mechanism to determine the task assignment and payment as each bidder emerges to play.

As we have argued previously, stability is a critical factor for choosing viewers to take our transcoding tasks. Viewers should be able to continue offering transcoding services during the channel streaming session without leaving. We first need to extract stable users from the viewer pool. Based on our previous study, we set a waiting threshold only after passing which can the viewer be qualified as stable, and be selected into our candidate pool. This threshold is determined by maximizing the mathematical expectation of the non-stop transcoding time of all viewers [16]. Further, to differentiate the stability level among these candidates, we need to have a more fine-grained metric of stability. We use a simple yet effective method which jointly considers the average online duration ($\bar{d}$) and standard deviation (σ) of a viewer's online record. We use a linear combination of them to represent the stability D. In our simulation the default λ is set to 0.8 as it gives the best result.

$$D(v_i) = \lambda \cdot \bar{d}_i - (1-\lambda) \cdot \sigma_i; \lambda \in (0,1);$$

Besides stability, latency obviously should also be optimized explicitly in this interactive streaming services. Therefore, we propose our quality of viewer metric as follows:

$$S(v_i) = \frac{D(v_i)^{\gamma}}{\ln(1+l_i)^{\beta}},$$

where γ and β is the weight for each component in our metric, lying between 0 and 1. When $\beta = 0$, $\gamma = 1$, we only maximize the stability of all selected viewers, and vice versa. The intuition for defining the stability of users in this form is inspired by the fact that, a longer average online duration indicates the viewer tends to stay longer, and a smaller standard deviation means such behavior is more consistent [16]. The effect of latency on quality of viewers uses the widely adopted logarithmic function to reflect the decrease of marginal quality degradation due to the increase of latency.

Our objective thus is to find desirable and affordable viewers to take transcoding tasks. Let $S(v_i)$ be the estimated quality of viewer v_i in terms of taking transcoding tasks. We introduce a set of 0-1 variable $x_{i,j}$ for each pair of viewer and channel. Variable $x_{i,j}$ equals one if viewer v_i is assigned for channel c_j. Let X denote the total selected viewer set; t_i and p_i are the total transcoding time and payment per unit time of viewer v_i. $l_{\min}$ is the minimum delay requirement for a satisfying transcoding process. The formal formalization is as follows:

$$\max \quad \sum_i S(v_i) x_{i,j} \tag{1}$$

$$\text{s.t.} \quad \sum_i p_i t_i x_{i,j} \le B_j, \forall B_j \tag{2}$$

$$\sum_j x_{i,j} \le 1, \forall i \in V \tag{3}$$

$$l_i x_{i,j} < l_{\min}, \forall x_{i,j} \in X \tag{4}$$

$$x_{i,j} \in \{0,1\}. \tag{5}$$

In constraint (2), the sum of all payments of a channel in the streaming period should be smaller than its planned budget. In constraint (3), each viewer can take at most one transcoding task to mitigate the risk of unreliable transcoding brought by viewers and guarantee transcoding performance. In constraint (4), the transcoding time of selected viewers should be less than the minimum requirement to guarantee high quality interactive live streaming.

3 Cloud-Edge Collaborations for Big Video Transcoding

In this section, we first study two offline situations and then extend our proposed algorithms to a more generic online scenario. In the first offline case, there is no strict requirement on the number of transcoding viewers for each channel, which means, given the budget limit, a channel c_j could have less than R_j transcoding viewers. The scheduler makes best-effort decisions to assign most stable candidates while providing reasonable payments. In the second offline case, the requirement on the number of transcoding viewers for each channel is strict, which means that each channel should have R transcoding viewers unless there is no such possibility. In both offline cases, we assume the scheduler has the bid information from all candidates before the assigning process. Studying the offline scenario gives us the understanding to this problem, especially in complexity, optimality, and provides the baseline situations for us to compare with the online case. Furthermore, since more and more personal livecast applications start to allow broadcasters to upload pre-recored content and publish later, like "Uploads" function in Twitch released in late 2016. Our solution in offline scenarios also work for these type of transcoding tasks that are more tolerant to latency brought by decision making, and can wait until all bids information are collected. Finally, we further consider the online situation where the candidates comes to the system sequentially, and the scheduler has to make a decision on-the-fly.

3.1 Baseline Scheduler with Flexible Transcoding Viewers

In this subsection, we consider the first case of the proposed problem where there is no strict requirement on the number of transcoding viewers for each channel. With the budget constraint, the crux of designing a good algorithm lies in how to efficiently use the limited budget, and filtering out the valuable viewers. We adopt a variant of the proportional share mechanism introduced in [17], which serves as the basis for budget-constrained viewer recruitment. To be specific, when a new transcoding request from a starting channel c_j arrives, the scheduler first generates a threshold according to the bid, estimated quality information of all candidates, and the given budget as summarized in Algorithm 2. The resulted threshold p represents the reasonable price per unit quality value. The main scheduling algorithm (shown in Algorithm 1) first orders all candidates in decreasing order of estimated quality (line 2), and attempts to choose them in a greedy manner: the most desirable candidate is examined first. For each candidate, the scheduler first checks whether its bid is worthwhile for its estimated quality and budget allow such reasonable payment $S(v_i) * p$. If it is desirable and affordable, then select this viewer, pay the corresponding payment, and skip the current one if not worthwhile (lines 8-9). Note that in the algorithm $S(X)$ represents the total quality of candidate set X. Here, we simply sum up the quality of every candidate in X to get $S(X)$ as we formulated before, While other methods, e.g., monotone submodular function, can also be applied. Budget and payment here are all represented as price per unit time. Eventually, we pay viewers according to its transcoding service time as well as its price per unit time.

Algorithm 1. Baseline Scheduler

1: Input: Budget B_j for channel c_j and number R of transcoding viewers required
2: Sort V in decreasing order of its estimated quality $S(v_i)$
3: $count \leftarrow 0$, $index \leftarrow 0$, $X_j \leftarrow \emptyset$
4: $p \leftarrow GetThreshold(B_j)$
5: **while** $count \leq R$ **do**
6: $index \leftarrow index + 1$
7: $v_i \leftarrow V[index]$ and b_i is the bid of v_i
8: **if** $b_i \leq S(v_i) * p \leq B_j - \sum_{x \in X_j} p_x$ **then**
9: $p_i \leftarrow S(v_i) * p$
10: $X_j \leftarrow X_j \cup \{v_i\}$
11: **else**
12: $p_i \leftarrow 0$
13: **end if**
14: **end while**
15: return X_j

Algorithm 2. GetThreshold

1: Input: Budget B_j
2: Sort V in decreasing order of $S(v_i)/b_i$
3: $X \leftarrow \emptyset$, $i \leftarrow 0$
4: **while** $b_i \leq S(v_i)B/S(X \cup v_i)$ **do**
5: $X \leftarrow X \cup v_i$
6: $i \leftarrow i + 1$
7: **end while**
8: $p \leftarrow B/S(X)$
9: return p

Since the baseline scheduler chooses the most stable candidates until either R assignments are made or it is running out of budget, it may lead to a great number of cross-region assignment (turning to cloud) especially when the budget is low. On the other hand, since there is a portion of candidates not worthwhile judged by the threshold price, some stable candidates are not selected, which is undesirable when only few candidates are available. This is the performance compromise we have to make given the limited budget constraint. The baseline scheduler runs in linear time complexity of N, namely $O(N)$, where N is the total number of qualified candidates.

3.2 Comprehensive Scheduler

Now we consider the second case where the requirement on the number of transcoding viewers for each channel has to be fulfilled unless impossible. The problem thus becomes to choose R viewers such that their quality is maximized while the sum of their bids does not exceed the given budget. We can view this problem as a variant of classical Knapsack problem. If the price is pre-determined by the auctioneer, given the budget constraint, we are choosing viewers from the candidate pool to maximize the total value of the selected viewers (sum of quality metrics). We can easily extend the baseline scheduling algorithm into a dynamic programming algorithm (shown in Algorithm 3) to solve it optimally with the same payment rule. The key part of this algorithm relies on the three-dimensional table $table$, where $table[i, B, r]$ represents the maximum total estimated quality we could have considering first i candidates with budget B for r assignments. For every transition, there are two cases (line 8 and line 11), representing (1) current candidate is not worthwhile, or not affordable, or not needed as all assignments are fulfilled, and (2) current candidate can be attempted for the assignment, respectively. More specifically, the time complexity of this approach is $O(NBR)$, where N is the total number of candidates, B is the budget and R is the number of transcoding viewers needed.

Algorithm 3. Psudo-Polynomial Optimal Approach

1: $p \leftarrow GetThreshold(B_j)$
2: **for** i from 1 to N **do**
3: $p_i = p * S(v_i)$
4: **end for**
5: initialize $table$ as a three-dimensional table
6: **for** B from 1 to B_j **do**
7: **for** i from 1 to N **do**
8: **for** r from 1 to R **do**
9: **if** $b_i > p_i$ or $p_i > B$ or $r == 0$ **then**
10: $table[i, B, r] \leftarrow table[i-1, B, r]$
11: **else**
12: $v[i, B, r] \leftarrow max(table[i-1, B, r], S(v_i) +$
13: $table[i-1, B-p_i, r-1])$
14: **end if**
15: record such transition in a backtrack table
16: **end for**
17: **end for**
18: **end for**
19: backtrack and return the whole schedule

Computational efficiency is extremely important for a delay-sensitive system like our studied one, while our pseudo-polynomial algorithm may take a long time if the given B is large and the minimum budget metric is small. We therefore improve it with an efficient comprehensive algorithm using the specifically selected data structures, as shown in Algorithm 4. Similar to the baseline scheduler, the comprehensive scheduler calculates the price threshold and orders all candidates (lines 2-3). Then, in each round, it pushes the remaining most stable candidate into a priority queue with its reasonable payment as key (lines 10-16), and smaller keys means higher priority. Prior to each round, the scheduler checks whether the budget can afford the cheapest R candidates in the priority queue (lines 7-9), and assigns these candidates if affordable. Finally, if we could not afford R cheapest ones among all candidates, we select as many cheapest candidates as we can. As the priority queue is normally implemented with heap, and R usually is a small and constant number, the worst case time complexity of Algorithm 4 is $O(Nlog(N))$, where N is the total number of candidates. We next show Algorithm 4 can provide an optimal solution under the given payment rule in certain situations.

Algorithm 4. Comprehensive Scheduler with Strict Number of Transcoding Viewers

1: Input: Budget B_j
2: $p \leftarrow GetThreshold(B_j)$, $i \leftarrow 1$
3: Sort V in decreasing order of $S(v_i)$
4: Denote $\tilde{Q}$ as a priority queue, where items with smaller value will be at front
5: **while** $i \leq n$ **do**
6: $minCost \leftarrow$ the sum of top R reasonable prices in $\tilde{Q}$
7: **if** $minCost \leq B_j$ **then**
8: select these R viewers and return
9: **else**
10: $p_i = S(v_i) * p$
11: **if** $b_i < p_i$ **then**
12: push v_i into $\tilde{Q}$ with its reasonable price p_i as key
13: **end if**
14: $i \leftarrow i + 1$
15: **end if**
16: **end while**
17: select top R' viewers from $\tilde{Q}$ which is the maximum number of viewers B_j can afford, return

Notice that when a user abandons the transcoding task during the process of transcoding, the system will select the next possible candidate according to the price threshold. If the budget is running out or no users satisfy the threshold, cloud will be evoked to guarantee the stability of transcoding processes, which incurs higher cost.

Theorem 1. *Algorithm 4 can provide an optimal solution under the given payment rule, if the bids from viewers are randomly distributed and the number of viewers N is sufficiently large.*

Proof. We prove the above theorem by considering two cases, i.e., when the budget B_j is sufficiently large, and when it is not. In the first case, with a sufficiently large budget, the optimal solution would choose the top R most qualified candidates whose b_i is no more than its p_i, to maximize the total estimated quality. Since algorithm 4 attempts from the most stable candidates as well, it ends up with exactly the same schedule as the optimal one, as it terminates once it finds the top R candidates are affordable. In the second case, the budget is not able to afford the top R most stable candidates. Given the sufficiently large N and viewers' Pareto Distribution against their quality, the optimal solution will contain a set of candidates where the sum of their payment is exactly B_j, as the payment is proportional to the quality. On the other hand, Algorithm 4 will choose R candidates with quality $\frac{B_j}{R*p}$ where p is $GetThreshold(B_j)$. Again, the bids of these candidates are no more than their reasonable payment ($b_i \leq p_i$). The scheduling result will have the same total quality as optimal one. Such selection is also guaranteed to be made since bids from viewers are randomly distributed (so that at each quality some candidates have their bids larger than the reasonable payment and some others do not) and the number of viewers N is sufficiently large (so that we can find R candidates at the given quality level with its bid no more than its reasonable payment). Therefore, under the given condition, algorithm 4 will have an optimal result. □

3.3 Online Implementation

So far we have discussed the problem in an offline manner, which assumes that we have the whole knowledge of bid before the selection process. However, the real-world scenario could be more complex, we cannot wait until all bids are collected from the viewers and make the decision after that. In fact, in our interactive live streaming services, the arrivals of new channels require instant selection of viewers. Therefore, an online algorithm is needed to make the selection decision on-the-fly.

In this online situation, our objective remains the same, which is to find R affordable candidates who are as qualified as possible. However, we do not know the arriving time and order of the responding bids from candidates, yet we could not wait for all candidates to respond as it may take too long. We observe one key difference between our online scenario and most other classic online problems, e.g., generalized secretary problems, is that we indeed know how good these candidates are as we know their individual quality. Therefore, when receiving responses, we can place the corresponding candidates in an array according to their quality rank, and we set a rank threshold representing the lower bound of candidates we accept. The rank threshold is initialized to be extremely strict so that only top ranked (most qualified) candidates can be considered, and we loose it over time.

The above mechanism is illustrated in Fig. 2, and described in Algorithm 5 in detail. At the beginning (line 1), the scheduler sends requests to all available qualified candidates and waits for responses. It also initializes an empty array A of size N (line 2), where A[i] will be used to hold the ith ranked candidate when it responds. A dynamic array $bidArray$ is also initialized (line 3), which is initially empty and used keep inserted candidates in increasing order of their bids. The $rankThreshold$ is set to 1 at the beginning, meaning we only consider the first ranked candidate when we start. We loose the $rankThreshold$ by 1 every

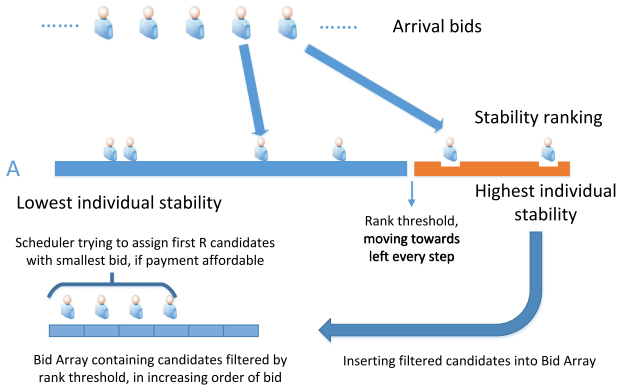


Fig. 2. Illustration of the online strategy.

time a new response is received (line 7). In the meanwhile, if the candidate at the new rank threshold position has already responded, insert it into the $bidArray$ (lines 8-10). Then we add the responding candidate into its corresponding position in A (line 11), and if its rank is smaller than the rank threshold, insert it into the $bidArray$ as well (lines 12-14). At the end of each round, we check if we can afford cheapest R currently considered candidates in the $bidArray$, and make such assignment if we can (lines 15-18). In terms of the payment, the main difference between the online scenario and the offline scenario is, we cannot know the cost performance, or the threshold, of all candidates, and therefore could not provide the reasonable payment to those selected candidates. Instead, for each selected candidate, we use the bid of next more expensive candidate in the $bidArray$ as the payment (lines 15-16), to maintain truthfulness. This pricing schemes falls into the generalized second price scheme, where the bidder in ith position pays the bid of the (i+1)th bidder [18]. Similar to the offline case, when users abandon the transcoding task, we choose the next affordable users in the bid array to fill in, and turn to cloud if no users satisfied the constraints.

Designing mechanisms to handle the strategic players in the auction usually boils down to designing algorithms in a certain restricted way. As can be seen from our algorithms, all above mentioned schedulers are 1) computationally efficient, given their non-exponential time complexity; 2) individually rational, as the payment is always higher than or equal to the bid, leaving selected viewers non-negative utility value; 3) budget feasible, since each assignment is made only after we make sure that the payment does not exceed the given budget; and 4) truthful, as the payment is either pre-determined (for the baseline and comprehensive schedulers), or an uncertain number larger than the bid (for the online scheduler). Being independent from the bid value of bidder itself, our payment scheme falls into the posted price schemes. These schedulers can also be easily deployed at reassignment time, and we only need to set R to 1 and set B_j to the left budget.

4 Performance Evaluation

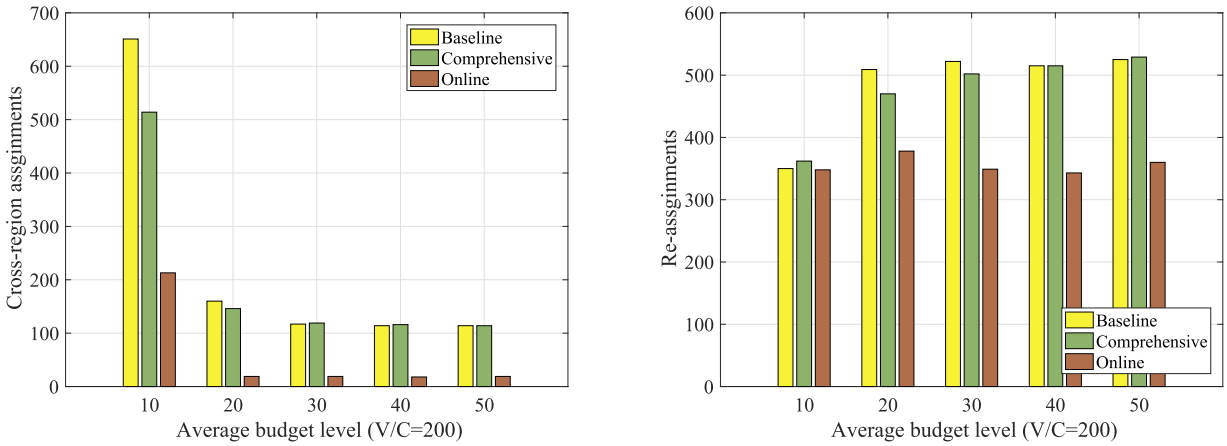
To evaluate our framework, we have conducted extensive simulations using large scale data captured by Twitch API. We first briefly introduce the selected datasets, and present the methodology as well as the evaluation results after that.

Algorithm 5. Online Scheduler

1: Scheduler broadcast the transcoding request with the description of the job to all qualified stable viewers
2: Initialize an empty array A of size N, where N is the number of total qualified stable viewers
3: Initialize a dynamic array $bidArray$
4: Initialize $rankThreshold \leftarrow 1$
5: **while** $t < timeout$ **do**
6: Let $respondingCandidate$ be the candidate returning a response
7: $rankThreshold \leftarrow rankThreshold + 1$
8: **if** $A[rankThreshold]$ is not empty **then**
9: Insert $A[rankThreshold]$ into $bidArray$,
10: with $A[rankThreshold].bid$ as the sorting key
11: **end if**
12: $A[respondingCandidate.rank] \leftarrow respondingCandidate$
13: **if** $respondingCandidate.rank \leq rankThreshold$ **then**
14: insert $respondingCandidate$ into $bidArray$,
15: with $respondingCandidate.bid$ as sorting key
16: **end if**
17: **if** $sumOfBid(bidArray[2\ to\ 1+R]) \leq B_j$ **then**
18: Assign $bidArray[1\ to\ R]$, for each assigned candidate $bidArray[i]$,
19: provide payment $bidArray[i+1]$;
20: return
21: **end if**
22: **end while**
23: select top R' viewers from $bidArray$, where R' is the maximum number of viewers the budget B_j can afford, return

4.1 Trace-Driven Simulation Configurations and Metrics

For the simulation, we mainly used the channel-based viewer trace data captured with Twitch API containing the join/leave record of viewers in certain channels from January 25 to February 27, 2015. Each entry includes the viewer ID, time of the action and the action type ("Join" or "part"). In total, we collected 11,314,442 "JOIN" records and 11, 334,140 "PART" records. The selected part of the record contains 270,105 unique viewers, and this partial viewer trace has the same trend as that of the entire record we captured. Since current video transcoding services offered by Amazon, the dominant player in cloud industry, still could not offer live streaming transcoding. Interactive live streaming service providers just purchase instances in IaaS to implement their transcoding services. Therefore, we use the price of default instance type, m4.large instance, as the cost for transcoding a task. For each viewer, we randomly assign a bid between 0 and 2 times the price of cloud instance to represent the minimum reward this viewer is willing to receive in order to conduct the transcoding work. As for channels, since the top 10 percent channels attract around 98 percent of the total viewers, we only regard these top 10 percent channels as our targeted channels, which we will provide transcoding services to. We then scale up/down the channel trace to have different viewer to channel ratios (referred as viewer-to-channel ratio, or V/C ratio) based on the record of all channels we captured, which was recorded every five minutes, from February 2015 to June 2015. In such time-based record we have the detailed information of



(a) Cross-region assignment (b) Reassignment count with different budget

Fig. 3. Stability analysis under different budget levels when V/C=200.

all live channels during the captured period, such as the total channel number and viewer number, which are used to estimate the average channel to viewer ratio in our simulation. Also, each channel will be assigned with a random budget. To test the performance of our approach under different financial circumstances, we have generated 5 groups of test data at different budget levels. Generally, the budget of a channel is set to be proportional to the viewer number of that channel.

We use re-assignment number, cross region assignment number, and cost as our performance metrics. Re-assignment number changes when viewers abandon their transcoding tasks in the middle of streaming process, and the service provider has to find new viewers to take these tasks. Re-assignment brings system overhead, especially latency for reassigning and short absence of the target quality version during the reassigning period. Cross region assignment number changes when no qualified viewers can be found locally under the current budget level, and the service provider has to turn to cloud for help. Cost is defined as the sum of payments to finish all transcoding tasks. We thus want these metrics as small as possible. We evaluate our system performance under different budget constraints and viewer/channel ratios.

4.2 Evaluation Results

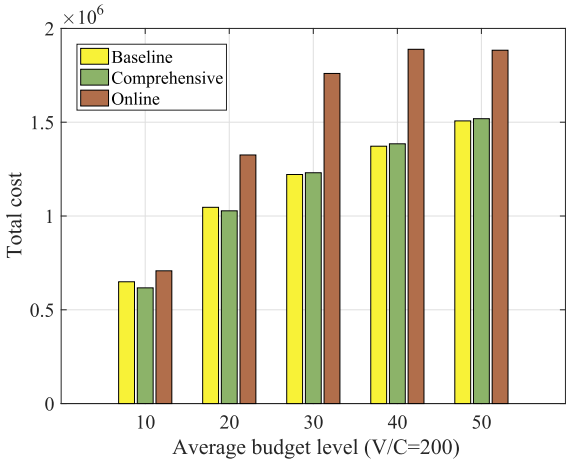
For comparison, we implemented the baseline scheduler, the comprehensive scheduler, and the online scheduler, under a variety of budget levels and V/C ratios. The existing industry solutions apply a pure cloud solution, which is far more expensive than our cloud-edge collaborative solutions. The difference in cost could reach over 80 percent, and thus we omit it here and only compare the performances of our solutions under different offline/online scenarios.

We first conduct simulation to see the number of cross-region assignments under various conditions. Regarding to different V/C ratios, the results of three schedulers are similar in respect to their relative performance. Thus we only report the results when $V/C = 200$ in Fig. 3a. Notably, the advantage of the comprehensive scheduler under low budget level, compared to the baseline scheduler, is more remarkable at higher V/C ratio. From Fig. 3a, we can easily see that both offline approaches (baseline and comprehensive) perform similarly at high budget level. On the other hand, the difference becomes obvious when the budget is scarce, in which case the comprehensive scheduler has better performance. The online scheduler however has much better performance, as it only has $1/6$ to $1/3$ cross-region assignments of the offline strategies. Interestingly, this however is mainly because the online mechanism does not have the requirement on the cost performance for each candidate, so that all candidates with different bid value (even those with too high bid) can be chosen.

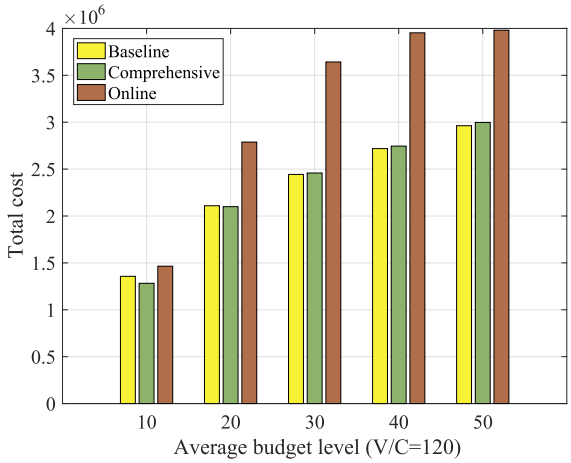
In terms of the reassignment, as can be seen from Fig. 3b, the results of the baseline scheduler and the comprehensive scheduler again are similar, and are around 50 percent higher than that of the online scheduler, except the case where the budget level is extremely low. The lower reassignment count at budget level 10 is mainly caused by the high cross-region assignment count, such that many assignments are not initially made locally and thus will not trigger reassignment later. The results of the online scheduler however remain the same across all budget levels, indicating that it has more stable performance once the assignment is made.

We also measure the cost of our approaches under different budget levels and V/C ratios. As shown in Figs. 4a, 4b and 5a, once again we see the results being very similar under different V/C ratios. The baseline scheduler and comprehensive scheduler have similar costs under different budget levels, while the online scheduler has upto 50 percent higher cost than the previous two. Notice that the online scheduler achieves lower reassignment and cross-region assignment counts than other offline approaches, but has higher cost. This is because the payment made to the viewers are determined differently in offline scheduler and online scheduler. On the other hand, offline schedulers can decide the optimal cost with all bids information in place. While for the online scheduler, since there is no oracle knowledge on the arrival of bids, the payment is greatly influenced by the arriving patterns, which makes the cost higher than the offline situations.

Additionally, for the online scheduler in particular, we compare its average result with the optimal case and worst case. The optimal case is where the responding candidate come in decreasing order of quality, while the worst case is

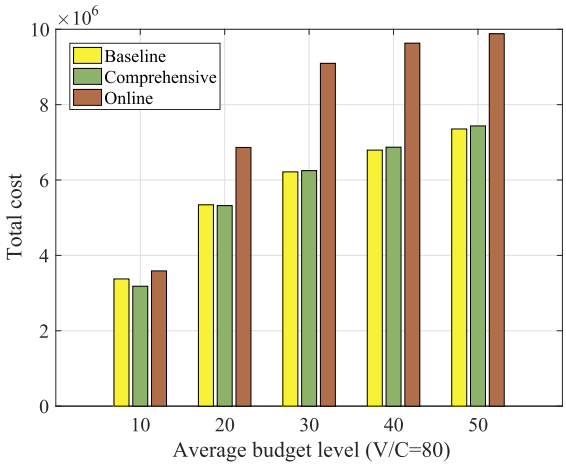


(a) Cost of different approaches (V/C=200)

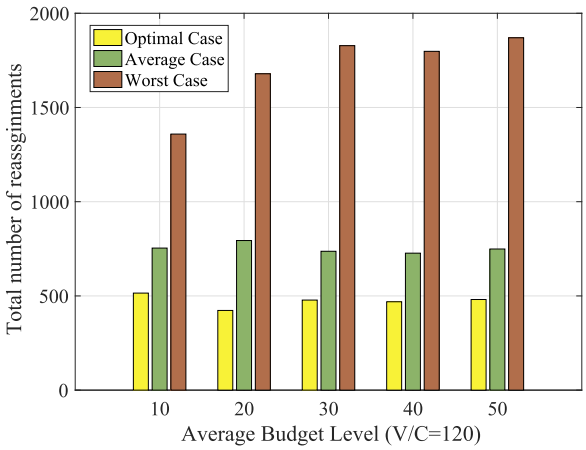


(b) Cost of different approaches (V/C=120)

Fig. 4. Cost comparison under different V/C values.



(a) Cost of different approaches (V/C=80)



(b) Extreme cases of the online scheduler

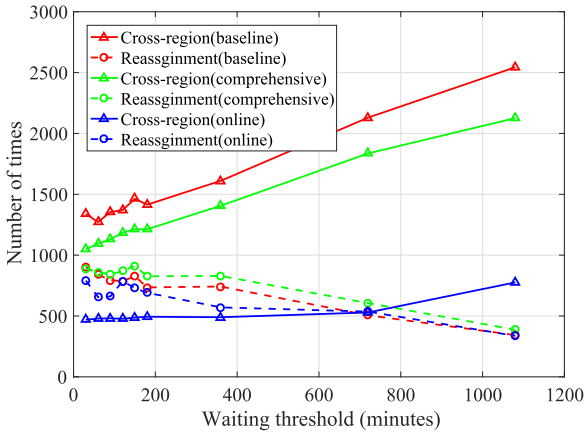
Fig. 5. Impact of budget on stability and cost.

the opposite. Fig. 5b shows the results of reassignment count. We see the average result is much closer to the optimal case than to the worst case, especially at high budget level. This also confirms the overall great performance of the online scheduler. Note that, the optimal case here can also be regarded as an offline deployment of the online scheduler.

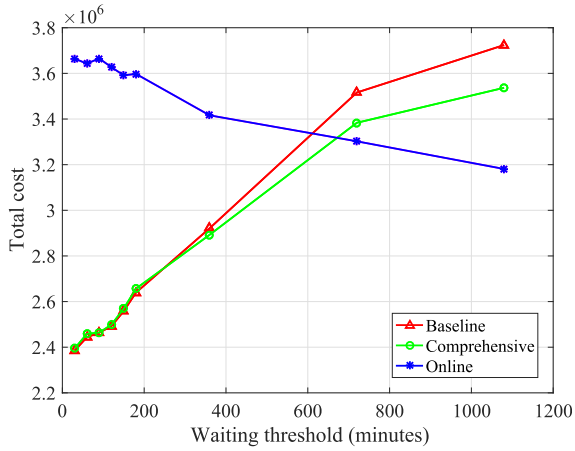
In short, the online scheduler has significantly better performance compared to the other two, in terms of reassignment count and cross-region assignment count, while without having extra cost overall. Interestingly, the online scheduler however benefits from being not able to calculate the threshold p in the online manner, such that it has more freedom to choose more stable candidate, even with partially overmuch payment. The results also indirectly reveal that the threshold-based mechanism of making reasonable assignment/payment based on cost performance, which is suitable for crowdsourced work in general, may not be suitable in our scenario.

We further measures the reassignment and cross-region assignment count with low budget and median V/C ratio, under different waiting threshold, as such settings provide the most obvious result. From Fig. 6a, we clearly see the trade-off between cross-region assignment and reassignment for each scheduler, as in all results they always go towards the opposite directions. The baseline scheduler and comprehensive scheduler have similar results, while the comprehensive scheduler has sightly higher reassignment count but much lower cross-region assignment count. The online scheduler has much better performance in terms of both metrics. With higher the budget level, the results remain similar overall but the difference between the baseline and comprehensive approach become smaller.

Additionally, we present the total cost of different approaches with median budget level in Fig. 6b. Again, we see the baseline and comprehensive approach have similar results, which increase as the waiting threshold goes up. This is mainly because with a higher waiting threshold, the selected candidates have higher quality and thus higher reasonable payment, leading to higher cost overall. The online scheduler however, has lower cost as waiting threshold increases. This is mainly caused by the increasing number of cross-region assignment, which is the dominant factor in this scenario. In conclusion, the large viewer base and V/C ratio allow us to have a larger waiting threshold upto 200 minutes, and the online scheduler is much less affected by the change of waiting threshold compared to the other two schedulers.

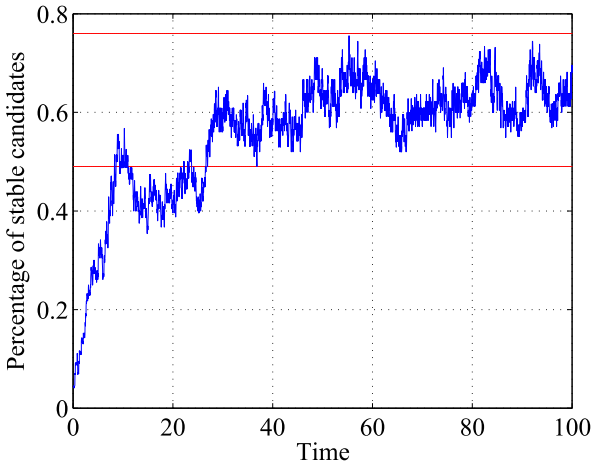


(a) Different waiting threshold with low budget

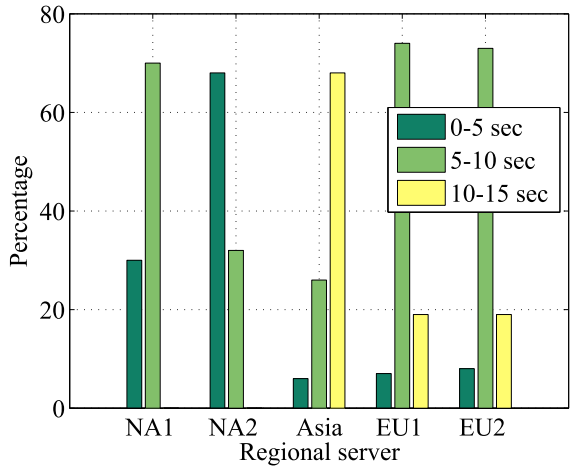


(b) Total cost under different waiting threshold

Fig. 6. Impact of waiting threshold on stability and cost.



(a) Stable viewer ratio along measured time



(b) Streaming delay in different regions

Fig. 7. Stability and delay analysis in the Planetlab-based prototype.

5 System-level Evaluation

5.1 Prototype Setup

We implemented a prototype of our framework with online scheduler on the PlanetLab. In our prototype, we use 5 PlanetLab nodes with similar network conditions as regional servers, 2 in North America (NA), 1 in Asia and 2 in Europe (EU), and other nodes as viewers. In total 377 nodes are used. During the experiment, each viewer node imitates an actual viewer behavior by joining the nearest regional server at a random time, staying for a duration according to the Pareto distribution, and leaving. Each viewer also submits a bid randomly. The regional server keeps track of all stable candidates, and assign transcoding assignment with a payment calculated at the same time. The selected candidates use *ffmpeg* to transcode a high quality video sent from the regional server using TCP, into a low quality versions, which are then sent back to the server. We use a 3.5 Mbps 1080p video as our source video and set 2.5 Mbps 720p as the target quality. Each video slice is of 1 second.

5.2 System Performance Results

We first measure the percentage of stable viewers in the system after it runs for 60 minutes. As shown in Fig. 7a, with a 60-minute waiting threshold, around 60 percent of the online viewers are stable, after the system stabilized at around 30 minutes. The two red lines show that the portion of stable viewers is always between 49 percent and 76 percent, which also confirms our simulation results. The experiment scheduling result, in terms of reassignment, cross-region assignment and cost, is similar to that of our large-scale simulation. This is expected given that in both simulations and experiments the viewers statistically follow the same shifted Pareto Distribution. We therefore focus on the streaming performance. Fig. 7b shows the streaming delay variance perceived by five regional servers. We see that the streaming latency in North America is much lower than that in other regions. This reveals that viewers watching distinct quality versions of the same channel may perceive highly different delay, which severely affects the online interaction between broadcasters and viewers. For example, viewers with lower latency may become spoilers describing a scene (in chat) which is going to be watched by others a few seconds later. Also, given the delay variance within the same region, the reassignment even itself may introduce a new delay difference, which may freeze the streaming for a short period. A solution to such issue is to introduce a short pre-set delay for each channel, as well as a

penalty to the scheduler when assigning candidates with highly variant delays.

In short, our experiment shows the feasibility of the proposed framework, and confirms the simulation results. It on the other hand also indicates the potential issues caused by the candidate delay variance, which calls for further enhancement as mentioned above.

6 Related Work

The wide popularity of Twitch-like applications indicates the emergency of novel interactive live streaming services, where massive immature broadcasters stream their contents through ordinary video devices over the Internet. Some pioneer works have studied such emerging system from both social perspective (e.g., online community) [8], [10], [19] and the technical perspective (e.g., streaming performance, user experience) [20], [21]. Attracted by the elasticity in computing power and "pay-as-you-go" billing model, cloud naturally becomes the choice for supporting such service. Chen et al. [22] design cloud leasing strategies to optimize cloud site allocation for geo-distributed broadcasters. However, for CLS platforms with massive broadcasters but charge nothing from the viewers, cloud-based approaches are expensive. He et al. [16] have studied the potential of edge viewers in the CLS systems, and put forward a viewer transcoding framework based on voluntary participation of viewers. Nevertheless, altruistic assumption or naive fixed price incentive approach could not fully motivate users to take these computation-intensive tasks and truly reduce cost.

Auctions have been widely used to solve the incentive problems in various scenarios, like crowdsourcing, spectrum sharing, P2P networks, etc. For crowdsourced tasks, Singer et al. [17] have used auctions for providing incentive with budget constraints. The context-specific requirements in our personal livecast applications make their approaches cannot be directly applied to our scenario. For instance, they target at minimizing payment or maximizing tasks, while we focus on maximizing the quality of users for conducting the transcoding services. For P2P networks, a taxation-based incentive mechanism is proposed to guarantee fairness, efficiency, and incentive in layered video coding [23]. Maharjan et al. [24] studies cloud-enabled multimedia crowdsourcing, and drive the optimal reward for recruiting broadcasters to do multimedia-based tasks based on the knowledge of utility functions. Unlike this utility based approach, our auction approach does not rely on the knowledge of utility function and tries to make decisions using the bid information as well as the user history statistics. There are few research works on addressing incentive problems in interactive streaming context. A recent work studies the same cloud-edge transcoding services in personal livecast applications[25]. They focus on applying redundancy principle to improve the reliability of this service and maximizing the expected social welfare. Different from works in the pure distributed networks and the recent works in livecast transcoding, we studied the incentive issue with budget constraint to maximize the quality of users directly to improve stability and reduce latency.

7 Conclusion

In this paper, we proposed low-latency, cost-efficient mechanisms for transcoding big video data in the personal livecast applications. Specifically, we examined the potential of involving end viewers into transcoding big video data from massive broadcasters to lower the computation cost and reduce latency. Our mechanisms assign qualified viewers to channels for transcoding tasks and determine the right amount of money to motivate them under the budget constraints. Our large-scale trace-driven simulation proved the superiority of the online scheduler, while our PlanetLab-based experiment further revealed a series of potential issues and suggested corresponding enhancements for practical deployment. As for further work, involving one viewer for taking one transcoding task may still be unstable considering the dynamics of viewers. We could recruit more viewers to take one task, increasing its reliability through redundancy. To further reduce cost, we could also consider incorporating spot instances from the cloud to take the transcoding tasks. Providing periodic transcoding services to increase viewers' utility or providing stable transcoding services through active instance migration can all be studied further.

Acknowledgments

This research is supported by a Canada NSERC Discovery Grant. B. Li's work was supported in part by RGC GRF grants under the contracts 16211715 and 16206417, and a RGC CRF grant under the contract C7036-15G. Y. Hu's work was supported by the National Key Research and Development Program of China under contract 2016YFC0501801, and Science and Technology Planning Project of Qinghai Province, China, under contract 2017-ZJ-730.

References

[1] Twitch, "Twitch 2016 Retrospective," 2016. [Online]. Available: https://www.twitch.tv/year/2016

[2] B. Li, Z. Wang, J. Liu, and W. Zhu, "Two decades of internet video streaming: A retrospective view," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 9, no. 1s, 2013, Art. no. 33.

[3] V. K. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z.-L. Zhang, "Unreeling netflix: Understanding and improving multi-cdn movie delivery," in *Proc. IEEE INFOCOM*, 2012, pp. 1620–1628.

[4] C. Zhang and J. Liu, "On crowdsourced interactive live streaming: A twitch. tv-based measurement study," in *Proc. 25th ACM Workshop Netw. Operating Syst. Support Digit. Audio Video*, 2015, pp. 55–60.

[5] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. 1st Edition MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.

[6] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 27–32, 2014.

[7] G. Nascimento, M. Ribeiro, L. Cerf, N. Cesário, M. Kaytoue, C. Raïssi, T. Vasconcelos, and W. Meira, "Modeling and analyzing the video game live-streaming community," in *Proc. 9th Latin Ameri. Web Congr.*, 2014, pp. 1–9.

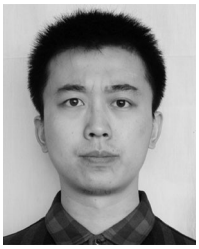
[8] M. Kaytoue, A. Silva, L. Cerf, W. Meira Jr, and C. Raïssi, "Watch me playing, i am a professional: A first study on video game live streaming," in *Proc. 21st Int. Conf. World Wide Web*, 2012, pp. 1181–1188.

[9] S. S. Krishnan and R. K. Sitaraman, "Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs," *IEEE/ACM Trans. Netw.*, vol. 21, no. 6, pp. 2001–2014, Dec. 2013.

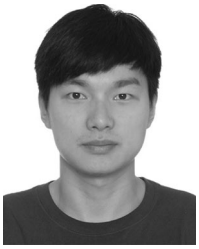
[10] W. A. Hamilton, O. Garretson, and A. Kerne, "Streaming on twitch: Fostering participatory communities of play within live mixed media," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2014, pp. 1315–1324.

[11] B. Wang, X. Zhang, G. Wang, H. Zheng, and B. Y. Zhao, "Anatomy of a personalized livestreaming system," in *Proc. Internet Meas. Conf.*, 2016, pp. 485–498.

[12] H. Wang, R. Shea, X. Ma, F. Wang, and J. Liu, "On design and performance of cloud-based distributed interactive applications," in *Proc. IEEE 22nd Int. Conf. Netw. Protocols*, 2014, pp. 37–46.
[13] S. Choy, B. Wong, G. Simon, and C. Rosenberg, "The brewing storm in cloud gaming: A measurement study on cloud to end-user latency," in *Proc. IEEE NetGames*, 2012, Art. no. 2.
[14] E. Nygren, R. K. Sitaraman, and J. Sun, "The akamai network: A platform for high-performance internet applications," *ACM SIGOPS Operating Syst. Rev.*, vol. 44, no. 3, pp. 2–19, 2010.
[15] Twitch, "Twitch deploys more transcode servers," 2015. [Online]. Available: https://blog.twitch.tv/boom-more-transcode-servers-427bcbfb519#.njykno ph5
[16] Q. He, C. Zhang, and J. Liu, "Utilizing massive viewers for video transcoding in crowdsourced live streaming," in *Proc. IEEE Cloud*, 2016, pp. 2524–2532.
[17] Y. Singer and M. Mittal, "Pricing mechanisms for crowdsourcing markets," in *Proc. 22nd Int. Conf. World Wide Web*, 2013, pp. 1157–1166.
[18] H. R. Varian, "Position auctions," *Int. J. Ind. Org.*, vol. 25, no. 6, pp. 1163–1178, 2007.
[19] J. C. Tang, G. Venolia, and K. M. Inkpen, "Meerkat and periscope: I stream, you stream, apps stream for live streams," in *Proc. CHI Conf. Human Factors Comput. Syst.*, 2016, vol. 16, pp. 4770–4780.
[20] 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 Multimedia Syst. Conf.*, 2015, pp. 49–60.
[21] K. Pires and G. Simon, "Dash in twitch: Adaptive bitrate streaming in live game streaming platforms," in *Proc. ACM VideoNEXT*, 2014, pp. 13–18.
[22] F. Chen, C. Zhang, F. Wang, and J. Liu, "Crowdsourced live streaming over the cloud," in *Proc. IEEE INFOCOM*, 2015, pp. 2524–2532.
[23] H. Hu, Y. Guo, and Y. Liu, "Peer-to-peer streaming of layered video: Efficiency, fairness and incentive," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 21, no. 8, pp. 1013–1026, Aug. 2011.
[24] S. Maharjan, Y. Zhang, and S. Gjessing, "Optimal incentive design for cloud-enabled multimedia crowdsourcing," *IEEE Trans. Multimedia*, vol. 18, no. 12, Dec. 2016.
[25] Y. Zhu, J. Liu, Z. Wang, and C. Zhang, "When cloud meets uncertain crowd: An auction approach for crowdsourced livecast transcoding," in *Proc. ACM Multimedia Conf.*, 2017, pp. 1372–1380.



Yifei Zhu (S'15) received the BE degree from Xi'an Jiaotong University, Xian, China, in 2012, and the MPhil degree from Hong Kong University of Science and Technology, Hong Kong, in 2015. He is working toward the PhD degree in the School of Computing Science, Simon Fraser University, British Columbia, Canada. His areas of interests include cloud computing, multimedia networking, internet-of-things, and crowdsourcing. He is a student member of the IEEE.



Qiyun He received the BEng degree from Zhejiang University, Zhejiang, China, the BSc degree from Simon Fraser University, British Columbia, Canada, both in 2015, and the MSc degree from Simon Fraser University, in 2017, all in the field of computer science. His research interests include cloud computing, social media, multimedia systems and networks.

Jiangchuan Liu (S'01-M'03-SM'08-F'17) received the BEng. (cum laude) from Tsinghua University, Beijing, China, in 1999, and the PhD degree from The Hong Kong University of Science and Technology, in 2003, both in computer science. He is currently a full professor (with University Professorship) with the School of Computing Science, Simon Fraser University, British Columbia, Canada. He is a steering committee member of the *IEEE Transactions on Mobile Computing*, and an associate editor of the *IEEE/ACM Transactions on Networking*, the *IEEE Transactions on Big Data*, and the *IEEE Transactions on Multimedia*. He is a co-recipient of the Test of Time Paper Award of IEEE INFOCOM (2015), ACM TOMCCAP Nicolas D. Georganas Best Paper Award (2013), and ACM Multimedia Best Paper Award (2012). He is a fellow of the IEEE and an NSERC E.W.R. Steacie memorial fellow.

Bo Li (F'11) received the BEng (summa cum laude) degree in computer science from Tsinghua University, Beijing, and the PhD degree in electrical and computer engineering from the University of Massachusetts at Amherst. He is currently a professor with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. He was the chief technical advisor with Chinacache Corp. (a NASDAQ listed company), the leading CDN operator in China. He was a Cheung Kong visiting chair professor with Shanghai Jiao Tong University (2010-2013) and an adjunct researcher with Microsoft Research Asia (1999-2007) and with the Microsoft Advance Technology Center (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 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. He 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 received six Best Paper Awards from the IEEE. He received the Young Investigator Award from the Natural Science Foundation of China (NFSC) in 2005, and the State Natural Science Award (2nd Class) from China in 2011. He is a fellow of the IEEE.

Yueming Hu received the PhD degree in soil geography from Zhejiang University. He is a professor of land information technology with South China Agricultural University. He is also the director of the Guangdong Province Key Laboratory of Land Use and Consolidation, and the chairman of the Guangdong Mapping and Geoinformation Industry Technology Innovation Alliance.