Cloud-edge Collaboration for Cost-effective Video Service Provisioning

by

Yifei Zhu

M.Phil., Hong Kong University of Science and Technology, 2015
B.Sc., Xi'an Jiaotong University, 2012

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
School of Computing Science
Faculty of Applied Science

SIMON FRASER UNIVERSITY
Fall 2019

Approval

Name:	**Yifei Zhu**
Degree:	**Doctor of Philosophy (Computing Science)**
Title:	**Cloud-edge Collaboration for Cost-effective Video Service Provisioning**
Examining Committee:	**Chair:** Ouldooz Baghban Karimi Lecturer
	Jiangchuan Liu Senior Supervisor Professor
	Qianping Gu Supervisor Professor
	Mohamed Hefeeda Internal Examiner Professor School of Computing Science
	Jianwei Huang External Examiner Professor Department of Information Engineering The Chinese University of Hong Kong
Date Defended:	**December 10, 2019**

Abstract

The advances of personal computing devices and the prevalence of high-speed Internet access have pushed video streaming services into a new era. One of its representative examples is crowdsourced livecast services where numerous amateur broadcasters lively stream their video contents to viewers around the world. For video service providers, processing these multimedia contents is inherently resource-intensive, time-consuming, and consequently expensive. The demand for low latency to guarantee interactivity in these emerging services further challenges the prevalent cloud-based solutions. In this thesis, we start by revealing the potentials of offering cost-effective low-latency video services both at the cloud and the edge side through analyzing the traces collected from real-world applications. We then examine the feasibility of an instance subletting service at the cloud side, where idle cloud resources can be traded. The performance of such a service is examined from both theoretical and practical perspectives. To satisfy the low-latency requirement in the emerging interaction-rich video services, we propose a crowd transcoding solution, which fully relies on powerful users to finish transcoding. To further improve the stability of such a distributed computing system, we then propose a cloud-crowd collaborative solution, which combines redundant end viewers with the cloud to perform video processing tasks cost-effectively. Novel probabilistic auction mechanisms are designed to facilitate this solution with desirable economic properties guaranteed.

Keywords: Cloud computing; Edge computing; Multimedia; Resource allocation

Dedication

To my parents and my girlfriend(my future wife, too).

Acknowledgements

First and foremost, I am greatly grateful to my senior supervisor, Prof. Jiangchuan Liu for his tremendous support and invaluable guidance throughout my studies. Prof. Liu has offered me great freedom in research problem selection and has been instrumental in teaching me, to name just a few, how to conduct organized and efficient research, how to focus on the critical parts of a research problem. These along with other insightful guidance and vision sharing motivate me and prepare me to become an independent researcher.

I also owe sincere thanks to Prof. Dan Wang for his mentorship during my research assistantship at Hong Kong Polytechnic University. I had a great time there and learned a lot in research approaches from the discussion with Prof. Wang and Dr. Wei Bao from the University of Sydney. May Hong Kong join with mainland China to develop better.

I am grateful to the other members of my committee: Prof. Qianping Gu as my supervisor, Prof. Mohamed Hefeeda as my internal examiner, Prof. Jianwei Huang as my external examiner, and Dr. Ouldooz Baghban Karimi as the committee chair, who provided constructive comments to my thesis.

I would also like to thank Prof. Zhi Wang and Prof. Yong Cui from Tsinghua University, Prof. Feng Wang from the University of Mississippi who have provided invaluable feedbacks to some of my research papers. Their comments greatly helped me improve my manuscript.

I would like to give my gratitude to the colleagues in Prof. Liu's NetMedia lab, Silvery Di Fu, Dr. Cong Zhang, Dr. Xiaoyi Fan, Dr. Wei Gong, Dr. Xiaoqiang Ma, Dr. Lei Zhang, Miao Zhang, Fangxin Wang, Yutao Huang, Dr. Jihong Yu, Jia Zhao, Chi Xu, Yuchi Chen, Qiyun He, Dr. Ryan Shea. I will always cherish the brainstorm and discussion moments with them. These wonderful friends make my Ph.D. life colourful and inspire me constantly.

I would also love to express my sincere gratitude to my master degree's supervisor at Hong Kong University of Science and Technology, Prof. Bo Li, who brought me to networking research and introduced Simon Fraser University and Prof. Jiangchuan Liu to me. After my graduation, he kept offering me invaluable career advice, which I deeply appreciate.

Last but not least, I would like to express my earnest gratitude to my parents and my girlfriend, Dr. Wenqian Ren, for their constant support, love, and encouragement, without which I would not hold on and succeed through this mental and physical challenging research journey. I would like to thank my parents again for providing me with the best education possible.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Cost-effective Vidoe Service Provisioning

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 4.8 ZB by 2022 [11]. Among this, videos are the dominant data format, which stand for 82% of all consumer Internet traffic. Specifically, live video, contributed by the emerging crowdsourced live streaming services, is becoming the prominent video format chosen by users to share their lives with the world. Live videos will account for 17 percent of video traffic by 2022 and will have increased 15-fold from 2017 to 2022 [11]. 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, hosted over two million broadcasters per month and supported 9.7 million daily active viewers in 2016 [85].

Video service providers need to transcode same video content, either VoD or live videos, 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 [62]. These transcoding tasks are 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 [12]. 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.

However, the cost of cloud-based video processing is still high in that the massive number of videos need to be processed and current cloud-based solution is not efficient when facing temporally varying video workloads. 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 2 million unique monthly broadcasters in 2017, and 355 billion minutes of livecast has been watched. These video contents 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.

On the other hand, despite the fact that video service providers, as cloud users, are busy scaling up and out their rented cloud instances to meet their demands, the resource utilization of their instances is far from being efficient. This inefficiency, in turn, incurs extraneous expenditure to them. For example, Netflix reported one of its Amazon EC2 clusters had reserved 5x more instances overnight to support peak-time services; during off-peak times, more than 1500 `3.4xlarge` EC2 instances are mostly unused [3], translating into almost $2000 unnecessary costs per hour [1]. In general, instance underutilization can occur in both temporal and spatial domains [32]. In the former case, it occurs when cloud users purchase instances for a fixed amount of time (*i.e.*, the billing cycle, ranging from minutes to years) but make no use of them during certain time intervals. As a matter of fact, in reality, video providers usually choose long-term cloud instances, a.k.a reserved instances to save cost and guarantee resource provisioning. The latter may happen when the users are running workloads with heterogeneous demands on various resources (*e.g.,* CPU becomes underutilized when running memory-bound applications).

Jointly solving instance underutilization in both spatial and temporal domains is challenging for both users and cloud providers. Existing studies have focused on dynamically provisioning cloud resources [100][103], an approach that allows customized instance types to mitigate underutilization in the spatial domain only. Fine-grained pricing schemes have also been investigated in academia [54] and adopted by some cloud providers (*e.g.,* Microsoft Azure), in which instances are billed in a shorter time interval. Users in turn can timely turn off their underutilized instances to avoid extra charges. It unfortunately precludes users from purchasing long-term instances with significant price discounts from the wholesale market of the cloud providers [2].

Considering the great potential of underutilized instances owned by cloud users, like Netflix example we mentioned before, *Instance subletting* service has been suggested [22][14], which allows sporadic users to sublet their underutilized instances to others in need. If carefully designed, this service can make instance owners monetize their underutilized instances without introducing unnecessary downtime, and meanwhile make other users or users within same organization enjoy low-cost and high-quality computing resources. Being a complement and extension to the current cloud market, it has great potentials towards building an efficient and sustainable cloud ecosystem.

In addition to the low utilization of the resources and high cost of processing these videos, the requirement on streaming latency in the interactive environment created by the

emerging crowdsourced applications is even more stringent, as high latency severely affects the viewer-broadcaster, and viewer-viewer interactive experience [98]. While the cloud server may be far away from the live source, which inevitably causes higher streaming latency. We thus want to seek more cost-efficient, low-latency solutions to provision compute resources to better serve these videos.

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 [25]. 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 [86]. Looking deeper into these crowdsourced livecast service (CLS) platforms, we observe the abundant computational resources residing in edge viewers. Advanced processors (e.g., Intel i7), powered with dedicated video transcoding core like Intel Quick Sync, are becoming the main stream setting for desktops, especially among high-end game devices. According to Steam, 50% of its PCs have 4 physical CPUs; 8.93% of processors operate at 3.7 GHz and above; 74.76% of PCs runs DirectX 12 GPUs [10]. The computing power of mobile devices is increasing even more surprisingly, like Apple's newly released A10 Fusion chip achieves close to workstation-level performance in the single-thread situation [8]; Google leverages mobile devices to run text recommendation training in its Gboard app [68]. As a successful game-oriented CLS provider, 65% of Twitch viewers come from desktop computers [84]. Even if we only focus on its high end devices, the transcoding performance of these devices (e.g., i7 4 GHz, 4 cores, 16 GB, built-in GPU with Intel QuickSync) is already comparable to, sometimes even better than, compute-optimized instances (e.g., `c4.8xlarge`) in the public cloud [7]. In addition, unlike traditional live streaming services where viewers of popular streams tend to be evenly distributed [64], geo-distribution of viewers for a specific broadcaster is highly skewed (48% of broadcasters have their viewers totally in the same province/state [66]) in our studied CLS platform. Since most viewers that consume a channel are located in the same region as the broadcaster, allowing local viewers to transcode streams will greatly reduce the communication distance from broadcasters to viewers. Inspired by this observation and the massive viewer base in these services, combining these viewers (crowd) with the cloud to do transcoding seems to be a promising solution.

However, several challenges need to be handled in both cloud subletting and cloud-crowd transcoding solutions. In the cloud subletting services, A core issue is to allocate limited instances to be sublet efficiently and determine the trading price accordingly. An auction appears to be a natural candidate due to its efficiency in allocating scarce resources and its capability in deciding the market-based prices [104][102]. Instance subletting, however, presents a series of new challenges that are yet to be addressed in the existing auction designs. Previous auctions make decisions given a static known supply and only deal with

the online arrival of bidders [100][103][45]. In instance subletting, given the dynamic arrivals of underutilized instances, the supply of auction is largely unknown and fluctuates, too. In addition, instances cannot stay in the market forever. These instances also have distinct deadlines for subletting, after which they will be reclaimed by their original owners. As such, these extra time constraints also need to be dealt with in auction. Furthermore, by allowing multiple users to share a single instance, instances are no longer traded as the classical one-to-one exchange, but many-to-one exchange, which makes the pricing of co-located requests even more challenging.

In the cloud-crowd collaborative computing case, since transcoding inevitably costs extra power and bandwidth consumption, viewers' willingness to take on such tasks varies from person to person. Though we notice the formation of channel-oriented online communities, involving these viewers into taking computational intensive transcoding tasks still needs much stronger incentives to guarantee eventual performance. Thus we recruit qualified viewers to take on these tasks and incentivize them with monetary rewards. Providing this incentive in a cost-efficient way requires the valuations of viewers for taking these tasks, which unfortunately are private and depend on their own profiles in most cases. In addition, even if they accept these tasks, their uncertainty in completing the tasks presents another challenge since viewers may not guarantee continuous transcoding for the whole live session or fail to complete the task due to their heterogenous computing power. Furthermore, quantifying these uncertain behaviours may require private information which creates cost as well; namely, this uncertainty information is also private. The heterogeneity and uncertainty of viewers as well as their private status all present serious challenges in viewer selection and pricing.

1.2 Summary of Contributions

In this thesis, we start by revealing the potentials of offering cost-effective low-latency video services both at the cloud and edge side through analysing traces from Google, Twitch, and Yinke. We then present an instance subletting service at the cloud side to allow idle cloud resource trading. We systematically examine the instance subletting service from both theoretical and practical perspectives [107][106]. We start by revealing that the algorithmic side of our auction in the secondary market for subletting is a unique *multiple multi-dimensional knapsack problem* (MMKP)[1] that has yet to be addressed in existing auction mechanisms. We present an online auction mechanism with a carefully designed pricing function based on the real-time availability of resource usage. We show that the solution provides the best provable competitive ratio with static supply, and guarantees truthfulness and individual

[1]MMKP solely refers to multiple multi-dimensional knapsack problem in this paper, not multiple-choice multi-dimensional knapsack problem used in some application scenarios.

rationality simultaneously. Large scale simulations driven by real traces demonstrate that our solution can achieve significant performance gains in social welfare and cost. We then extend our algorithms to the dynamic supply scenario, where the supply of the underutilized instances in our system also changes over time. We improve our mechanisms by incorporating a discount strategy to mimic the effect of the earliest deadline first policy in real-time scheduling without sacrificing other economic properties. We further discuss the practical issues and demonstrate an EC2 based prototype implementation, which offers seamless and low-overhead operation thanks to the latest containerization techniques (Docker [4] in particular).

To satisfy the low-latency requirement, we further propose a crowd transcoding solution to allow viewers in the system to accomplish the transcoding tasks [108]. We designed an efficient data-driven threshold to filter out unstable viewers. Budget-constrained task assignment algorithms are then proposed for both offline and online scenarios.

To further offer reliable services, we devise redundancy strategies accordingly to different workload types. To pass the asymmetric information barrier in cost and uncertainty, we incorporate their statistical descriptions into our bidding language, design truthful auctions to recruit stable viewers and reward them with the right amount of incentive [109]. The mapping we present from our studied social welfare maximization problem to the min-cost flow problem not only guarantees optimal social efficiency but also is capable of absorbing different capacity requirements. We prove that our designed mechanisms guarantee social efficiency, incentive compatibility, and individual rationality in expectation.

1.3 Thesis Organization

The remainder of this thesis is organized as follows:

- Chapter 2 presents a brief introduction of resource allocation, pricing, and the differences of mechanism design, taking auction as an example, to classical game theory.
- Chapter 3 begins with the related work in providing low cost or high utilized cloud. It then presents the system model and problem formulation for a promising instance subletting services, where auction is used for price determination and resource allocation. Mechanisms are designed specifically for the static supply and dynamic supply situation in such a service.
- Chapter 4 presents a fully crowd solution to provide low cost transcoding services without violating the budget constraint.
- Chapter 5 presents a cloud-crowd solutions, where both cloud and redundant crowd are used to mitigate uncertainty. Novel mechanisms are designed to incorporate this uncertainty and guarantee economic properties.

- Chapter 6 summarizes our works, presents the lessons learned and pinpoints several interesting future directions.

Chapter 2

Preliminary

2.1 Resource Allocation and Pricing

Capacity and demand keep fighting with each other, determining how the services are provided. If there is always extra capacity, we can then adopt the most simplified approach to arrange resources so that the quality of services since congestion and delay is no longer a problem. But in most cases, even if the cloud service provider keeps adding capacity into its existing datacenter, the capacity is still behind the demand. It is also not efficient to prepare resources for the peak demand only. To efficiently use the limited resources, dedicated control measures need to be taken.

Pricing is intertwined with resource allocation in system design because current resource-consumption based pricing is particularly sensitive to how the cloud system utilizes its resources. Researchers in [92] identifies the importance of pricing in distributed system design and points out that optimizing profit from the provider's perspective may lead to sub-optimal system throughput.

Resource allocation problem usually can be formulated as the following optimization problem [71]:

$$\max \sum_{r} U_r(x_r), Rx \leq c \tag{2.1}$$

Normally, the problem strives to maximize certain objection function as a function of the allocation decision x_r. Tasks have resource constraints and the system has a capacity. The amount of resources a task eventually allocated (R) should not exceed the resource capacity C. Other constraints like placement constraint, budget constraint, QoS constraint are also possible. Placement constraint could be the specific kernel version or hardware architecture. This requirement arises because deep down there is heterogeneity in machines in the clusters or from the need for application optimization [78]. When we consider the market aspect, tasks can further have budget constraints. Different application types, like streaming applications, may also have specific QoS constraints to ensure their performance.

Fairness is a common goal of a scheduler in resource sharing clusters. Researchers work on policies to determine the fair share for each job or queues. Resource allocation follows first come first serve, a finished task will free its allocated resources, and these freed resources are then allocated to the waiting tasks in the queue to make sure that their allocated amount is the fair amount. Queues operating at the fair share or lower than fair share is not preemptive. For one dimension resources, there are also two types of fairness definitions: basic fairness where all players get an equitable share of resources is applicable when all players hold the same preference level towards the resources. The other type is extended fairness, cloud computing involves multiple types of resources. Thus the definition of fairness in multi-dimensional cases also varies [55] [46]. Existing cloud services, especially IaaS, still treat computing resources as a unidimensional resource, without allowing users to customize their VMs. These two approaches require different approaches.

How to define fairness in the multi-dimensional settings has been extensively studied in recent years due to the demand from the cloud computing scenarios. The seminal work [44] assumes the admission decision has been made, it strives to reach fairness and strategy-proof. Existing works on fairness assume jobs all follow a fixed pricing scheme and only consider fairness allocation. Researchers in [55] presents two sets of functions unifying efficiency and fairness together. Users can modify the parameter in the designed fairness functions to match their own efficiency-fairness tradeoff levels. The efficiency studied in this paper is the sum of all dominant share resources gained by all users. It assumes the knowledge of utility function and did not try to elicit it from users. In an auction, social efficiency is specially considered as we will see in the next section.

The resource allocation problem naturally comes with an economical interpretation, especially in linear programming scenarios. The following presents the formulation of the resource allocation problem in its simplest form, followed by its dual problem. [79].

$$\max \sum_i c_i x_i, \sum_i a_{i,j} x_i \leq b_j \tag{2.2}$$

$$\min \sum_j b_j y_j, \sum_j a_{i,j} y_j \geq c_i \tag{2.3}$$

where dual variable y can be interpreted as marginal value of resource i in production economy. We choose an item only if its hidden value is greater than its cost as stated in the constraint. While the connection between the pricing and allocation seems really straighforward in single dimension scenario, their connection becomes much more difficult to find especially when additional application-specific constraints. like budget constraints are added.

2.2 Mechanism Design: Auction as an Example

Auction is a branch of mechanism design, which serves as an integral part of modern microeconomics. Auction has a long history and even starts long before the formalisation of game theory. It is arguably the most successful application of game theory in the real world, and is widely used in selling antiques, houses, construction projects, spectrum, etc. The majority of game theory focuses on analysing the existence of equilibrium assuming the full complete information of all players. In contrast, an auction is used in the incomplete information scenario, namely, all players in the game only have knowledge of its payoff function without knowing that of others, so called private valuation. This uncertainty regarding values is an inherent feature of auctions and brings a substantial difference in how players in the auction play this game. Accompanied by it is the introduction of signalling, namely, what kind of information will each player reveal to others. This signal decision usually includes in a bid. Based on some assumptions in the distribution of signals or bids, auction theory studies whether equilibrium exists in an auction and how to reach this equilibrium.

There are two roles in an auction: bidder and auctioneer. Depending on the design of auction, bidders can be buyers or sellers or both, and submit bids to the auctioneer. Auctioneer collects these bids and makes two types of actions based on the rules designed by the system designer: allocation rules and payment rules. An allocation rule determines who will get the item of all collected bids. The payment rule determines how much this winner has to pay out of all collected bids. Based on the direct revelation principle, the designer just needs to design mechanisms to ask for users to directly report their private value. A good auction design also desires several properties: incentive compatibility, individual rationality, social efficiency, revenue maximization. Incentive compatibility means that the expected payoff of a bidder telling the truth is at least greater or equal to the expected payoff of a bidder reporting otherwise. Individual rationality means that the expected payoff of a user winning the auction is at least zero, which guarantees the incentive for the player to join the auction. Social efficiency guarantees that the eventual allocation maximizes the sum of all utility of all players. We also call such mechanism as an efficient mechanism. Revenue maximization guarantees that the eventual allocation maximizes the received payment of all players. We call such a mechanisms as an optimal mechanism.

In addition to auction approach, other approaches including nash bargaining solutions [96], dynamic pricing [53] [90], negotiation [16], shapley value [23] etc. have also been used in determine price and resource allocation. Recognizing benefit of price in allocating resources, the work in [53] combines dynamic pricing with inter-datacenter bandwidth allocation, targeting at maximizing social welfare. Shapley value in the cooperative game theory can also be used to determine the price and reach a sense of fairness, as opposed to auction. Researchers in [23] uses multi-dimensional bin-packing to maximize social welfare, then apply shapely value to split the social welfare determining the price for the buyers. Fairness and

other strategic manipulations, other than truthfulness can be guaranteed. It is not strategy-proof, instead it provides proximate fairness guarantees and shows a way to determine the allocation [23].

Chapter 3

Cloud Instance Subletting for Resource Utilization Optimization

3.1 Background and Related Work

In IaaS cloud computing, most of the current IaaS providers view the commodity unit as a single VM, in that the users just need to tell how many VMs it wants. This further combines with the time dimension. Users are charged by the per-hour fees (changed to per-minute basis in 2017 for on-demand Amazon AWS instances). Existing IaaS providers also provide several types of low-cost instance options other than their regular instances[1] to attract cost-conscious users. This pricing advantage is usually accompanied with constrained instance capability, which are further realized in two ways: (1) low-cost instances are interruptible and can be reclaimed by the provider at its will, like *spot instances* in EC2 and *preemptible instances* in Google Compute Engine (GCE). (2) low-cost instances are configured with small base capacity only and can burst to higher capacity opportunistically, like burstable instances[2] in GCE. Instance subletting services discussed in this paper are implemented using market-based auction mechanism and offers a different SLA in both time and spatial dimensions compared with existing instance options, making it a complementary part to the existing cloud market. For example, unlike preemptive spot instances, instance subletting services offer nonpreemptive instances where accepted instances are guaranteed to run without interruption during its requested period. Unlike small sized burstable instances, instances in instance subletting services have larger base capacity. The difference in these two service models fundamentally separates target user groups and presents new theoretical challenges in resource allocation. Fig.3.1 presents a more intuitive illustration to our design space and the position of our sublettable instance in it. In addition, as the first large scale attempt in applying market-based pricing on preemptible VM provisioning, pricing

[1] On-demand and reserved instances

[2] f1-micro instances in GCE get 0.2 of a vCPU and can burst up to a full vCPU for short periods.

in spot instance has been discovered to be not truly market-driven [13], which can induce complex strategic behaviour (e.g., untruthful bidding) of users. We strive to design auction mechanisms where truthful bidding is the dominant strategy for all users.

Attracted by the low cost of existing spot instances and reserved instances, novel cloud services complementing current cloud ecosystem have also become a focal point of the recent academic literature. Brokerage is a popular approach that takes benefit of reserved instances [50][93]. Wei *et al.* [93] propose to use a broker to resale the reserved instance bought from the cloud provider to the users. They focus on making reservation decisions at the supply side to reduce the total cost of a broker. Qiu *et al.*[76] study the interaction between private clouds and a broker, and formulate the trading problem as a Stackelberg game, which deprives the pricing power of the users. Yi *et al.* [97] incorporate fulfilment ratio requirements of batch jobs to maximize the revenue of the provider. Other works try to exploit spot instances through building a platform from a system's perspective without taking pricing issues and strategic behaviours into consideration [34][82]. Some preliminary results on subletting services have been presented with the focus only on the static supply situation. We study a more realistic scenario with the dynamic supply case and also examine the feasibility of subletting services from the practical perspective.

A large group of cloud providers, including the 5th IaaS provider Rackspace and some small niche cloud providers like, VPS.NET [3], still offer on-demand instances in an hourly rate or even longer. Users' continuous attraction to the personalized services provided there and the fear of vendor lock-in all make these users possible sources for the instance subletting service. The supply of our service is not restricted to on-demand instances only, rather any type of instances with explicit leasing period can come to our service. For example, reserved instances usually need commitment in 1 or 3 years. The marketplace for reserved instances trades instances in the unit of month, which are still too coarse for exploiting hourly or even minutely fluctuations.

Most of the previous literature focuses on studying dynamic resource provisioning from a single cloud service provider's perspective [100][103][102]. As one of the early works, Zhang *et al.* [99] propose an online auction mechanism for allocating a single type of resource in the cloud. Zhang *et al.* [102] further study the multi-dimensional resource provisioning problem. They consider the operation cost in this model, and design deterministic/randomized auction mechanisms to maximize social welfare and revenue. These works assume that the service provider has a static resource capacity on the supply side and consider at most the online arrival of instance requests. For multi-dimensional scenarios, they consolidate massive resources into a monolithic resource pool without either distinguishing the underlying barriers between different servers or considering the actual request placement process. While the computing resources in our system are contributed by distributed sporadic users, even

[3]VPS.NET: https://www.vps.net/

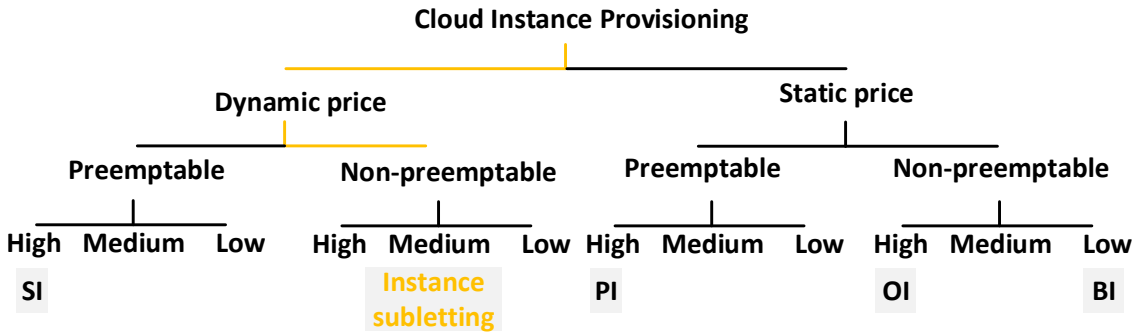


Figure 3.1: Design space and the position of sublettable instance: Maximum achievable capacity $\in \{High, Medium, Low\}$. SI for EC2 spot instances, PI for GCE preemptable instances, OI for on-demand intances, and BI for GCE burstable instance.

the resource capacity can be dynamically changing which increases the challenge in auction design further [19]. In addition, we can no longer ignore the underlying instance barrier to place the requests like before. Otherwise, severe resource fragmentation problems occurs to indivisible instances, leaving the system useless.

3.2 System Model and Problem Formulation

3.2.1 System Model

The instance subletting service can be implemented either in the major IaaS market or in a secondary market operated by another third-party platform. In the former,IaaS users can directly purchase sublettable instances offered by the major cloud provider. They can also purchase instances, e.g., on-demand or reserved instances from the existing cloud markets, e.g., Amazon EC2, and sublet them in another secondary market, if in the latter case. In the instance subletting service, IaaS users purchase instances from cloud markets. As mentioned earlier, instances can be underutilized in the temporal and/or spatial domains. In order to allow them to compensate for their investment and increase utilization further, any users with underutilized instances can sublet and monetize their instances. The instance subletting service can be implemented either in the major IaaS market or in a secondary market operated by another third-party platform. In the former, IaaS users can directly purchase sublettable instances offered by the major cloud provider. They can also purchase instances, e.g., on-demand or reserved instances from the existing cloud markets, e.g., Amazon EC2, and sublet them in another secondary market, if in the latter case. As mentioned earlier, instances can be underutilized in the temporal and/or spatial domains. In order to allow them to compensate for their investment and increase utilization further, any users with underutilized instances can sublet and monetize their instances. We assume there are M instances for sublet in this market. Each instance consists of R types of resources (e.g., CPU, memory). We index each type of resources in an instance as r where $r \in R$. The capacity

of resource r in an instance $j \in M$ is denoted as C_j^r. Each instance j joins the market at time T_s^j and has a deadline T_{ddl}^j for sublet, after which the instance will be collected back to its owner.

We strive to design practical mechanisms for our instance subletting service. Especially, from the perspective of mechanism design, we need to find a balance between the complexity for users to obtain instances and the efficiency for a provider in allocating instances. Even after we choose auctions as our trading mechanism, we still want to minimize the number of involving parties in the trading process, e.g., bid collection, accounting, and fully maximize resource utilization. For rational subletters within a single subletting instance market, any further incomes towards their underutilized instances would compensate for their loss due to underutilization and correspondingly reduce their cost of ownership. Joining this market thus becomes the dominant strategy. The payment policy for the subletters then becomes orthogonal to our one-sided auction design. Situations may change when we have multiple subletting service providers. Subletters therein start to gain more bargaining power and have the need to further increase their payoffs by leveraging this competition. Under these situations, double auction may become reasonable as service providers try to shift some power towards the subletter side to encourage participation. However, the utilization under such conditions could be worse since some possible allocations may be hindered by the mismatch in bid value between the subletters and the buyers. Therefore, in this paper, to improve the utilization and reduce the complexity in the implementation, we adopt one-sided forward auctions.

In our one-sided auction, the instance subletting service provider acts as an auctioneer selling these M instances to N buyers through auctions. Fig.3.2 provides an overview of this auction market. Each buyer $i \in N$ comes to the market at time t_s^i and acts as a bidder bidding for an instance to meet its computing demand. This demand is specified from two aspects: resource configuration for its intended instance, and the minimum running time for this instance, denoted as $< \vec{d_i^r}, t_i >$ where $r \in R$. Each buyer i then attaches a bid valued at b_i along with its requirement $< \vec{d_i^r}, t_i >$. These bids are sent to the auctioneer. Since each bid corresponds to a request of a bidder, we use bid and request interchangeably in this paper.

After receiving a bid i, the auctioneer decides immediately whether to accept this bid or not, and the amount of price p_i this bid should be charged if accepted. Buyers behind the accepted bids will be allocated to the instance as specified in their bid. The minimum running time requirement t_i turns into a service level agreement (SLA) between the service provider and this buyer i. To be specific, our instance subletting service provider commits to make this instance available to this buyer within the bid-defined running time t_i without interruption. This commitment applies separately to each accepted bid.

In addition, the winning buyer i pays the corresponding price p_i to the auctioneer. We denote the private valuation of buyer i for having its intended instance as v_i. The utility

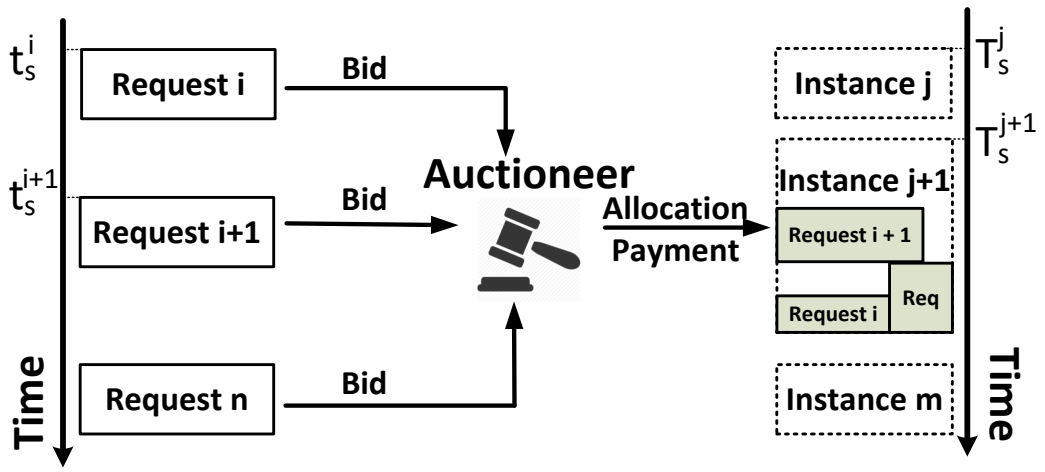


Figure 3.2: Auction market in our instance subleting system

of a buyer i follows commonly used quasi-linear utility form, which in turn is defined as $u_i = v_i - p_i$ if this buyer wins this instance with price p_i, and zero otherwise. Since we are studying a strategic environment, buyers in such environment may choose to misreport their bids other than their true valuation to improve their utility; *truthfulness* becomes the cornerstone in a well-managed auction. It is crucial because only after eliciting the true valuation from the buyers can a mechanism achieves *social efficiency*. In addition, a buyer's utility should be non-negative to guarantee their incentive to join the auction, known as *individual rationality* in the auction theory. We present the formal definitions of these three goals in the following:

Definition 1. *An auction achieves truthfulness if the dominant strategy for each player in the auction is to report its true valuation. Namely, reporting its true valuation generates the optimal utility:* $u(v_i) \geq u(b_i), \forall b_i \neq v_i$.

Definition 2. *An auction achieves individual rationality if the utility of the selected player is non-negative, namely,* $u(b_i) \geq 0$.

Definition 3. *An auction achieves social efficiency if the sum of utilities of all players and auctioneers are maximized.*

As can be seen, our instance subletting service only involves buyers directly participating in the auction in order to minimize users' effort to devise complex bidding mechanisms for their instances and maximize resource utilization. Correspondingly, since this is a one-sided auction, social welfare naturally is the sum of utilities of buyers and the utility of the auctioneer (also acts as the seller with online supply contributed by the subletting users). Social welfare becomes the sum of valuations of all buyers after the prices cancelled out.

Table 3.1: Table of notation

Symbol	Description
M	number of instances
N	number of requests
R	total types of resources
C_j^r	capacity of resource r in instance j
L_r, U_r	Lower and upper bound of per unit valuation for resource r
L, U	Lower and upper bound of L_r, U_r across all resources
t_i	minimum requested time of bid i
t_s^i	start time of request i
t_{min}	time unit in dynamic case
T_{ddl}^j	sublet deadline for instance j
d_i^r	demand of resource type r in a bid i
v_i	valuation of bid i
$\lambda_{r,j}$	marginal price of resource r in instance j
z_j^r	usage ratio of resource r in instance j
p_i	charging price for request i
u_i	utility of request i

Notice that the cost for the users to sublet their underutilized instances is the monetary cost determined once they purchase these instances. This cost becomes constant once they determined their own usage time and decide to join the subletting service to monetize their underutilized instances. Important notations are summarized in Table. 3.1 for clarity.

3.2.2 Auction in the Secondary Market: Problem Formulation

The implementation of the instance subletting system involves challenges from both the theoretical and the practical sides. We first examine its theoretical aspect. We start from analyzing its offline scenario with all the information available in advance, and focus on maximizing social welfare of the system, which is defined as the sum of utilities of all buyers and auctioneer, to ensure system-wide efficiency and stability. We introduce a binary variable $x_{i,j}$, which equals 1 if a request i is allocated to instance j or equals 0 if this request is rejected by our system. In the offline situation, our problem can be formally formulated as follows:

$$\max \quad \sum_{i \in N} \sum_{j \in M} x_{i,j} v_i \tag{3.1}$$

$$\text{s.t.} \quad \sum_{j \in M} x_{i,j} \leq 1, \forall i \in N \tag{3.2}$$

$$\sum_{i \in N} d_i^r x_{i,j} \leq C_j^r, \forall r \in R, \forall j \in M \tag{3.3}$$

$$t_i x_{i,j} \leq T_{ddl}^j - t_s^i, \forall i \in N, \forall j \in M \tag{3.4}$$

$$x_{i,j} \in \{0, 1\} \tag{3.5}$$

Under truthful bidding, we have the objective value as $\sum_i \sum_j x_{i,j} b_i$. Constraint (3.2) indicates that each user request can only be allocated to one instance; Constraint (3.3) indicates that the total requests allocated on an instance cannot consume more resources than the capacity of this instance in all resource types; and Constraint (3.4) indicates that the selected instance should have enough remaining time to guarantee the continuous running of its hosted requests.

Since understanding the structure of our problem is helpful for us to devise the auction mechanism in the following section, we here prove that our social welfare maximization problem is NP-hard in Theorem. 1.

Theorem 1. *The offline social welfare maximization problem, in its general form, is NP-hard.*

The offline problem studied here is essentially a MMKP problem. Both knapsacks (instances) and items (requests) are defined by multi-dimensional size vectors, and we have multiple knapsacks to choose from. We can easily reduce an instance of multi-dimensional knapsack problem, an NP-hard problem [58] into an instance of this problem to prove that our problem is also NP-hard. The detailed proof can be found in the Appendix.

Considering the online situation of our problem, it is known that the online knapsack problem is inapproximable to within any non-trivial multiplicative factor in general cases [20]. Fortunately, in our scenario, the value reflected in bid value under the truthful mechanism does not have to be arbitrarily large. We can interpret it as the willingness to pay for our subletting service, which is upper bounded by the on-demand instance price, since any clearing price higher than the price of this alternative will drive buyers to the on-demand instance. Under such condition, we present an $\alpha-$competitive mechanism in the following Sec. 3.3 and further extend it to handle dynamic supply situation in Sec. 3.4.

3.3 Mechanism Design under Static Supply

In this section, we first study the static supply scenario where all instances are already in the marketplace in the beginning and only buyers arrive over time. Since we have the static

supply situation here, there is no notion of time limit (leasing deadline) on each instance in this scenario any more. Otherwise, the lifetime of the whole market is determined by the instance with the longest leasing time, leaving our market no way to sustain.

Notice that simply integrating capacities of multiple knapsacks (instances) into a unified one, transforming our problem into the knapsack problem that previous works target at, does not solve our original problem. Consider the following small example: given two items with weight 1 and 3, the result of allocating these two items into two knapsacks with capacity 2, 2 respectively is obviously not the same as that of allocating these two into a single knapsack with capacity 4 after the simple capacity integration. Therefore, the internal barriers between different instances need to be handled properly.

A wide range of real world problems, like resource allocation problems [102][99], secretary problems [20], and keyword auctions [36], can be formulated into variants of knapsack problem. The online versions of these problems attract more attention among researchers due to their greater application potential. Buchbinder *et al.* in [27] propose an online deterministc algorithm for the fractional packing problem. Their result implies an $O(ln(U/L))$ competitive[4] ratio when there is only one packing constraint. Zhang *et al.* [99] propose an $O(ln(U/L))$-competitive algorithm for the packing problem in which the valuation type of user varies. Chakrabarty *et al.* [36] propose $O(ln(U/L))$-competitive algorithms for the knapsack problem and multi-choice knapsack problem, and later translate them to bidding strategies for budget constrained bidders in ad auction. Zhang *et al.* [102] reach a $O(ln(U/L))$-competitive algorithm for the knapsack problem with extra cost functions in its objective function. As a more complicated variant of knapsack, it is still unclear whether there exist online algorithms to solve our studied MMKP problem with a similar $O(ln(U/L))$ competitive ratio. Leveraging the primal-dual scheme, we design an effective algorithm and prove that it is $(\ln(U/L)+1)$-competitive with static supply. We then design pricing schemes to complement it into an auction mechanism, guaranteeing truthfulness and individual rationality. We present the detailed design of our mechanism based on the primal-dual scheme in the following.

3.3.1 Mechanism Design

The primal-dual approach has been widely used to reveal the pricing attribute in a problem from the economical perspective [74]. We leverage a primal-dual scheme with Lagrangian relaxation for the social welfare maximization and we show that the specific structure of this problem leads to a competitive online mechanism.

We introduce non-negative Lagrangian multipliers $\lambda = \{\lambda_{r,j}, r \in R, j \in M\}$ for all r, j pairs in constraints (3.3). Since our Lagrangian relaxation problem is the upper bound of

[4] U, L mentioned in this paragraph are all limits of certain input factors in their scenarios. Big O notation omits other constant factors considered in their scenarios.

our original problem, we have the Lagrangian dual problem as: $\min P(\lambda)$ s.t. $\lambda \geq 0$. where $P(\lambda)$ is defined as:

$$\begin{aligned} \max \quad & \sum_{i\in N}\sum_{j\in M} x_{i,j}b_i + \sum_{r\in R}\sum_{j\in M}\lambda_{r,j}(C_j^r - \sum_{i\in N} d_i^r x_{i,j}) \\ \text{s.t.} \quad & \sum_{j\in M} x_{i,j} \leq 1, \forall i \in N \qquad (3.6) \\ & x_{i,j} \in \{0,1\} \qquad (3.7) \end{aligned}$$

A subproblem then emerges for each request i from this dual problem.

$$\begin{aligned} \max \quad & \sum_{j\in M} x_{i,j}(b_i - \sum_{r\in R}\lambda_{r,j}d_i^r) \qquad (3.8) \\ \text{s.t.} \quad & \sum_{j\in M} x_{i,j} \leq 1 \qquad (3.9) \\ & x_{i,j} \in \{0,1\} \qquad (3.10) \end{aligned}$$

Observing this subproblem, we can interpret $\lambda_{r,j}$ as the unit price for each type of resource r in instance j. If we define $p_i = \sum_r \lambda_{r,j}d_i^r$, it indeed represents the price charged to request i from instance j. In other words, the subproblem essentially chooses the right instance to maximize the utility of a request i. Once we have interpreted the dual variable as the marginal price, the difficulty in online implementation lies in how to update this dual variable. In fact, the crux of designing a competitive algorithm lies in determining the appropriate threshold to absorb the worthwhile inputs so that the desired outcome can be reached. To be specific, we need to design a marginal price updating function such that the platform does not accept too many low-value bids in the beginning, leaving no room for those high-value bids coming in the future. It also should not be too conservative to leave the overall resources underutilized in the end.

We introduce a *usage ratio* z_j^r where $z_j^r = \frac{\sum_i x_{i,j}d_i^r}{R_j^r}$ to reflect the level of used resources of type r in current instance j. As we have mentioned, in our case, we can interpret the upper bound of a bidding price as the on-demand instance counterpart. Accordingly, we define L^r and U^r as the lower/upper bound of user's value per unit of resources respectively. We have $U = \max U^r, r \in R$. For each type of resources, we design our unit price updating function as follows:

$$\lambda(z_j^r) = (U^r e/L^r)^{z_j^r}(L^r/e). \qquad (3.11)$$

Our problem in static supply is similar to the previous dynamic resource provisioning problem . Different price updating function could lead to different competitive ratio. Our mechanism provides a cleaner price updating function and the best possible competitive ratio in our problem setting. The intuition behind defining a unit price updating function like this is that when the usage ratio z_j^r starts with zero, the marginal price is set to be

smaller than the unit price lower bound. It ensures that the price is low enough to accept as many bids as possible. With the increasing of z_j^r, instance becomes more and more conservative to admit new requests. This marginal price as a selection threshold built from previous selected bids also guarantees that only relatively high value bid can be admitted. Once z_j^r equals one, the marginal price is set to be the upper bound of the user's value per unit of resources. Under such circumstance, no bid can win the auction, guaranteeing the capacity constraint is satisfied. Otherwise, the charging price will be greater than their bid value, violating individual rationality.

After having the updated unit prices (dual variables), we can again make allocation and pricing decisions in the primal problem. We allocate request i to instance j iff i doesn't overfill the instance j, and instance j provides the maximum utility to request i. The detailed algorithm is presented in Algo. 8. Notice that when there is more than one instance offering the same utility to the request, we allocate the request to the most similar instance, where this similarity is calculated as the *dot product* of two examined resource vectors in Euclidean spaces, denoted as $Sim()$. This strives to keep the relative relationship of resources in the chosen instance, avoiding to use up a single type of resource, keeping the competence of a instance for future use. We next prove that our mechanism can guarantee truthfulness, individual rationality and a tight competitive ratio.

Algorithm 1 Online auction mechanism (OA)

Initiate $\lambda(z) = (Ue/L)^z(L/e), x_{i,j} = 0, L^r, U^r$
while Receiving bid i **do**
  Calculate utility:$u_i = b_i - \sum_r \lambda(z_j^r)d_i^r$
  $j^* = \arg\max_j(u_i), \forall j$
  if $u_i > 0$ **then**
    if $|j^*| > 1$ **then**
      $j^* = \arg\max_{j^*}(Sim(j^*, i))$
    end if
    $x_{i,j^*} = 1$
    $p_i = \sum_r \lambda(z_i^r)d_i^r$
  else
    $x_{i,j} = 0$
  end if
end while

3.3.2 Theoretical Analysis

Theorem 2. *Our online auction mechanism guarantees individual rationality and truthfulness in bid value.*

Our mechanism is truthful because its pricing scheme falls into the family of sequential posted price mechanisms [30]. The decision process also guarantees the individual rationality. Detailed proofs of all important theorems in the following can be found in the Appendix.

Theorem 3. *Our online pricing mechanism provides α-competitive in social welfare with $\alpha = \ln(U/L) + 1$.*

We prove the competitive ratio of our MMKP problem based on the monotonicity of our pricing function, which is derived from the primal-dual scheme. Detailed proof can be found in Appendix.

Theorem 4. *The competitive ratio of our online algorithm is tight.*

We prove the tightness of our algorithm by comparing it with the classical knapsack problem (one-dimension, one knapsack). Since our problem is a general problem of classical knapsack problem, based on the conclusion in [36] that any deterministic algorithms for solving classical knapsack problem (one-dimensional, one knapsack) cannot be better than $\ln(U/L) + 1$, we can prove that $\ln(U/L) + 1$ is the best possible competitive ratio for our MMKP problem. Detailed proof can be found in Appendix. Our mechanisms therefore improve the competitive ratio of the state-of-the-art deterministic auction mechanism [102], and provide a cleaner price updating function.

3.4 Mechanism Design under Dynamic Supply

We next extend the solution to consider dynamic supply situation in which even the capacity of instance pool is unknown and fluctuates. Users with idle instances arrive overtime. These instances are of different sizes and only available within the user specified time for subletting. Requests also come dynamically requiring different sizes of instances and time requirements.

Time plays a crucial role in this context, which also makes the problem much more complicated. Unlike other resources, the remaining lifetime (if regarded as a resource) in each instance is reusable as long as other resource requirement satisfy. Take the simple case in Fig.3.3 as an example. In the left subfigure, a request m is rejected due to the high price of instance i, while a feasible instance j may be available in the near future. In the right subfigure, when request m arrives, there are two instances, instance i and instance j available (satisfying both time and resource constraints). If we accept request m and allocate it to instance j, the unit price in instance j may increase so high that the later request n will be rejected by j; and due to SLA violation, instance i is unable to serve it, either.

The introduction of the time component makes our problem similar to the real-time scheduling problems in real-time systems, like in an operating system, different tasks with varying computation time and deadline compete to win the occupation of processors. Correspondingly, a line of research works focus on allocating single dimensional resource, e.g.,

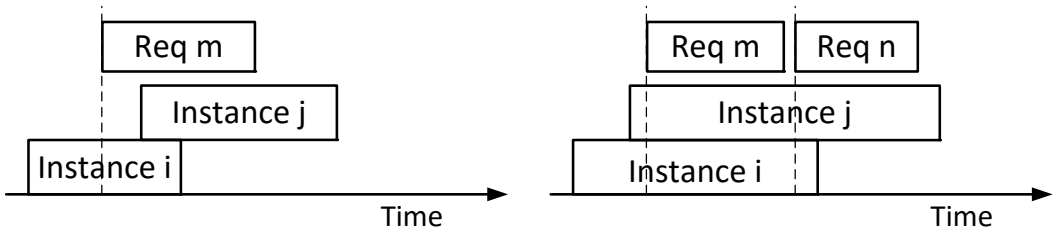


Figure 3.3: Adversary situations because of the time constraint

CPU computing time [52], [18], [29] or homogeneous instance[5] [90] in cloud environment. Jain *et al.* in [52] study preemptive job scheduling problems under parallelization limits in an offline fashion. Azar *et al.* in [18] study preemptive job scheduling problems under soft deadlines. Chawla *et al.* in [29] study the resource scheduling problem in stochastic settings with the known distributions on the demand side. Wang *et al.* in [90] study selling homogeneous instances with customer defined reservation time. These works all try to circumvent known lower bound by adding extra assumptions on either deadline, resource types, distributions, etc. They relate more to spot instance services or the job scheduling problem in Hadoop-like computing framework from the application perspective. The initial positioning of our services separates us with spot instance and these works in that: (1) preemption is not allowed; (2) users can customize the size of their multi-dimensional instances; (3) we request instant decision-making and resource accessing without any delay; (4) the proposed mechanism can handle the game theoretic environment and solve the pricing problem simultaneously; (5) most importantly, not only instance requests, those sublettable instances also arrive online. Unfortunately, we first have to present a negative result under our dynamic setting situation here.

We present a practical heuristic algorithm based on our previous mechanism after that.

Theorem 5. *No deterministic truthful mechanism can achieve better than* (U/L)*-approximation to social welfare.*

We prove this negative result by examining two adversary cases when both sides of our markets are online. Detailed proof can be found in Appendix.

3.4.1 Mechanism Design

Naturally, we need to incorporate time in our decision making process. To be specific, we plan to save those long-lived instances so that they can meet better bids, and fully exploit the capacity of those soon-expired ones. In the previous example, if we allocate request

[5]Number of instances is the only input in demand

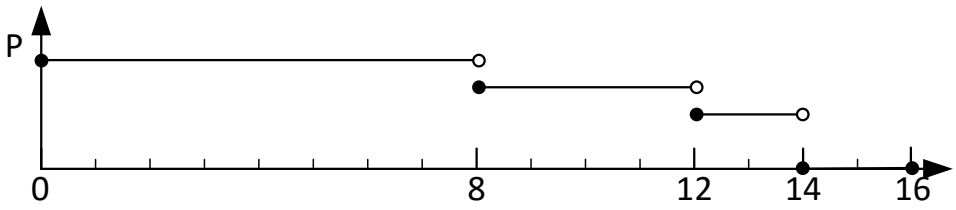


Figure 3.4: Illustration of price discount and stage separation (initial price P, $T_j = 16, t_{min} = 2$, no requests are admitted)

m to the soon expired instance i, the instance j may be able to serve request n as well. Such a design idea also echoes with the *earliest deadline first* policy in real-time scheduling. On the other hand, in order to guarantee truthfulness, we still need to select the instance with the largest utility with respect to a request. Nonetheless, naively allocating requests to the instance with the earliest deadline and with the largest utility could bring conflicting decisions.

Hence, we modify the unit price updating function in a way that our market will discount the price of resources in an instance with the elapse of time. Through proper reducing the price as the function of the remainnig time, we could make the instances with the earliest deadline more easily to be selected under the utility maximization goal. In the meantime, reducing the unit price means we are lowering the threshold of admitting new requests. Without sacrificing social welfare too much, ideally, we want the discounts to be quite small at first. As times elapses, discounts become more and more aggressive. Inspired by equity evaluation in portfolio management [24], we incorporate a discount strategy here. We assume the minimum task time is t_{min} (it can also be set as the system unit time, here we study a general case). An instance can never accommodate any requests when the remaining lifetime of this instance becomes smaller than t_{min}. We separate an instance with the lifetime $T_j = T^j_{ddl} - T^j_s$ into $\lfloor \log_2 \frac{T_j}{t_{min}} \rfloor + 1$ stages in the time unit of t_{min}. For stage i where $i \in \{1, 2..\lfloor \log_2 \frac{T_j}{t_{min}} \rfloor\}$, unit price is discounted at time step t_i where $t_i = T_j - 2^{\lfloor \log_2 \frac{T_j}{t_{min}} \rfloor - i} t_{min}$ at the scale of $D_i = 2^i / 2^{\lfloor \log_2 \frac{T_j}{t_{min}} \rfloor}$ (as in percent off).

Therefore, we have our new unit price updating function as follows:

$$\lambda(z^r_i, t) = (U^r e / L^r)^z (L^r / e)(1 - D_i) \tag{3.12}$$

As can be seen, we increase discount exponentially as the elapse of time. In the last stage, when the remaining time is t_{min}, the price has become small enough to accommodate any requests if resources permit. Fig.3.4 illustrates how our design discounts the unit price

during the lifetime of an instance given no requests are admitted during its lifetime. Details of the algorithm for dynamic supply can be found in Algo. 2.

Theorem 6. *Our online auction mechanism guarantees individual rationality and truthfulness in bid value in the dynamic supply situation.*

The proved competitive ratio in the static case is not guaranteed here since we cannot guarantee that the value of the rejected bid is too low to all the instances in the total timespan and the uncertainty in instance supply. But our mechanism can still guarantee truthfulness and individual rationality, two crucial properties for a mechanism to handle strategic players. Furthermore, the simulation results illustrate that the social welfare is also improved in most cases, proving this to be an efficient heuristic improvement. The proofs of truthfulness and individual rationality are similar to the proofs of Theorem.2 since important principles used in proving these two properties are not violated. Thus we omit them here.

In addition, as for the truthfulness in lifetime, if the instance subletting service is provided by the same service provider selling the original instances (major market scenario), the service provider will have the complete information about the lifetime of the instances, leaving no room for misreporting in instance lifetime. In other cases, subletters will not submit lifetime claims larger than the true lifetime because it incurs another round of billing cycle. They also have no incentive for declaring their instance lifetime shorter than the true lifetime because it deprives the opportunity to further amortize the cost of owning these instances.

3.5 Evaluation

Experimental Settings

Our evaluation uses Google Cluster trace [5] consisting of 3,535,030 entries, reporting each tasks' ID, active time and resource demand (CPU, memory; normalized to values between 0 and 1) in an approximately 6 hours period. We identified 176,580 unique tasks after removing the reported anomalies and merging entries of the same task. We then sampled requests and instances from these task entries with varying sample rates. For each request, we derive the resource demands and time requirements directly from the entries of a task. Its bid value b_i is further calculated based on its resource demands d_i^r and a unit resource valuation variable $\mathbf{v}^r$ randomly generated from 0 to U^r, namely,

$$b_i = \sum_r d_i^r \mathbf{v}^r, \mathbf{v}^r \in [0, U^r] \tag{3.13}$$

For simplicity and consistency with the data trace, we assume an instance has the maximum allowable resource capacity specified in the trace (namely 1.0). To simulate the

Algorithm 2 OA under dynamic supply

while $t < T$ **do**
  while Receiving bid i **do**
    Calculate utility:$u_i = b_i - \sum_r \lambda(z_i^r, t) d_i^r$
    $j^* = \arg\max_j(u_i), \forall j$
    if $u_i > 0$ **and** $t_i \leq (T_{ddl}^j - t)$ **then**
      if $|j^*| > 1$ **then**
        $j^* = \arg\min_{j^*}(T_{ddl}^{j^*} - t)$
      end if
      $x_{i,j^*} = 1$
      $p_i = \sum_r \lambda(z_i^r, t) d_i^r$
    else
      $x_{i,j} = 0$
    end if
  end while
  while Receiving an instance j **do**
    Initiate $\lambda(z, t) = (L^r/e)$
  end while
  if $t = t_i$ **then**
    Update discount and time step: $D_i = 2D_i$, $t_i = T_{ddl}^j - 2^{\lfloor \log_2 \frac{T_j}{t_{min}} \rfloor - i} t_{min}$
    Update unit price: $\lambda(z_i^r, t) = (Ue/L)^z (L/e)(1 - D_i)$
  end if
end while

amount of available resources in each instance, i.e., how much resources the user wishes to sublet, we extract the CPU and memory usage information from a sampled task entry D_j, and apply the formula: $C_j^{cpu} = 1.0 - 1.0 \times D_j^{cpu}/(D_j^{cpu} + D_j^{mem})$ to get available CPU. We compute the available amount of memory likewise. The simulated sublet instance can in turn preserve the CPU-to-memory usage ratio information of the data trace. We do not include disk nor networking resources in our simulation, because unlike the pricing of CPU and memory, the pricing of disk and networking are usually decided by the resource consumption, not based on instance type [1]. We vary the total number of bids from 1000 to 8000 with an increasing number of instances to ensure the rejected requests on the baseline algorithm stay lower than 50%. In the static supply simulation, the simulator reads in requests chronologically based on their start time.

Services Compared and Performance Metrics

We compare our service with two dominant commercial services: the spot instance service and the reserved instance service, and two other close-related works on cloud pricing in [105] and [90]. Our solution is further compared with the optimal solutions obtained from an ILP solver[6]. Since the spot instance service actually has a proprietary pricing scheme, we adopt a straight forward implementation of its released approach[7]. Only when a request with a bid value greater than the current spot price can the request be admitted into an instance. Once we allocate a request to an instance, we set its charging price and the corresponding spot price of this instance as the value of the lowest bid residing in that instance[8]. The reserved instance service adopts a fixed pricing scheme. The fixed price for a reserved instance is set to be 70 percent of its on-demand instance based on the difference between the price of a `m4.large` instance (default reserved instance type) in 1-year term to its on-demand counterpart [1]. As for two other close related research works, they all determine the posted price based on the utilization of the resources in their contexts. Zhou *et al.* in [105] propose online auction mechanisms for dynamically Provisioning cloud resources with Soft Deadlines and operation costs. We directly implement their proposed mechanisms under hard deadline and zero provisioning costs situation as a comparison. Wang *et al.* in [90] propose mechanisms to Sell Reserved Instances in cloud, where users' request on provisioning deadline can be tight or delayable and prices for instances are determined based on how many instances have been sold (single dimension). We select their mechanisms under the tight deadline situation as a comparison. We further calculate the prices for our multi-dimensional subletting instances by summing the price of each resource

[6]PuLP: `https://pythonhosted.org/PuLP/`

[7]`http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/how-spot-instances-work.html`

[8]The spot price for the first request is its bid.

up based on their single-dimensional pricing scheme. For notation convenience, we refer to the first one as PSD and refer to the latter as SRI.

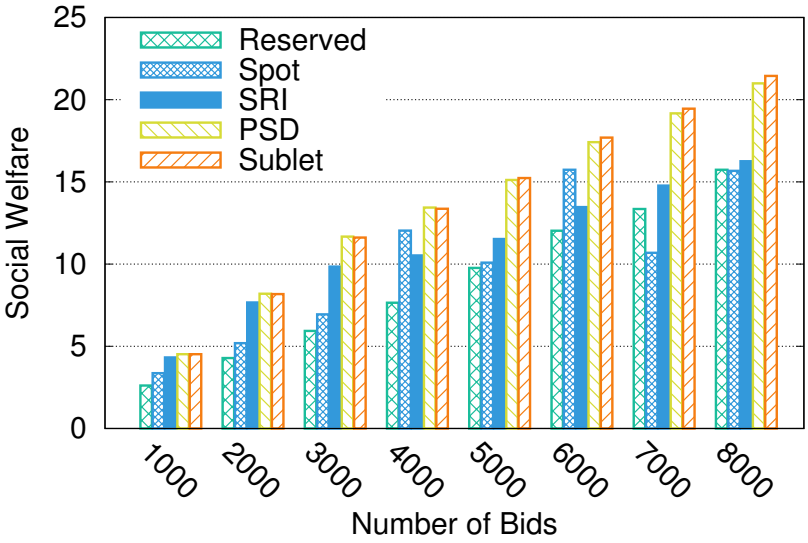
Since works in [105] and [90] all focus on proposing pricing schemes for the traditional instance trading scenarios with a fixed capacity from a theoretical perspective, we also examine their performances under two important economical metrics: social welfare and cost saving: social welfare and cost saving, where social welfare is the sum of valuations of all accepted requests and cost saving is the average cost reduction rate for users to finish their tasks compared with the on-demand counterparts. The former reflects how efficient the underlying mechanism in a service is in allocating limited resources; the latter measures how much benefit a service brings to bidders. To complement the study of our service, we further discuss the practical challenges in our subletting services including the system performance overhead in Section.3.6, which have not been covered in those works.

3.5.1 Trace-driven Simulations: the Case of Static Supply

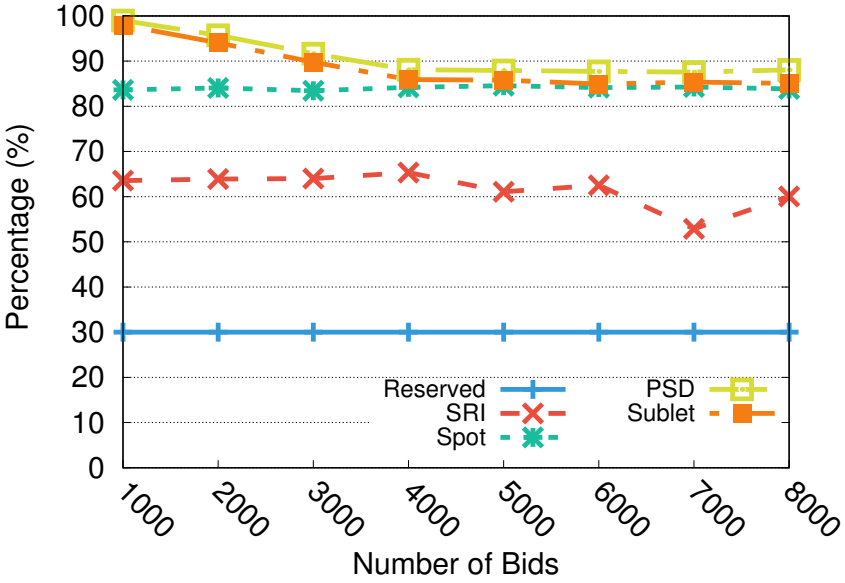
Fig.3.5a and Fig.3.5 illustrate the results in the static supply case. First, we compare the performance of our online mechanism with the offline optimal solution in Fig.3.5a. Overall the ratio tends to become smaller with the increase of the number of bids. It starts at slightly above 7 across all U values at 100 bids and plunges to 4 when the bids are 600. This is because the higher the total demand of resources, the more instances are provisioned; as shown in Eq. (3.8), an instance with the lowest price is chosen for each coming bid, hence the solution space becomes larger with more instances. As such, more allocation options are available, which leads to better performance.

Next, we compare the achieved social welfare of our instance subletting service with other instance options and research works in Fig.3.5a. As can be seen, the reserved instance service achieves considerably less social welfare than the other four services in most cases because its fixed pricing scheme cannot adapt to the changes in demand. To be more specific, we find that it rejects more requests than the other four; namely, it creates the overpricing problem for more requests than its alternatives. The rest four approaches presented in Fig.3.5a adopt dynamic pricing, which aims at efficiently reflecting market situations. Interestingly, the social welfare of the spot instance service drops at 7000 bids. It is because that, once the low-value bids are accepted, they directly affect the spot price. We find that instances directed by this low spot price accept 7% more low-value bids than that in other bid settings, leaving instances no room for future high-value ones. On the other hand, our instance subletting service still outperforms the spot instance service by 32.4% in social welfare when bid number reaches 8000. The underlying problems in static supply situation we study here is also similar to the scenarios that SRI and PSD targets at[9]. The instance subletting service

[9]Notice that differences also exist as we elaborated before. We implement the key ideas of both mechanisms and adapt them to our scenario.



(a) Social welfare with varying bid numbers



(b) Cost saving with varying bid numbers

Figure 3.5: Performance comparison in the static supply case

achieves 31.9% more social welfare than SRI in 8000 bids. However, the gap between our mechanism with the PSD is very small in the static supply situation. The close gap between these two originates from the similar primal-dual scheme and the price update function they choose. We will demonstrate later that our mechanisms outperform PSD in the dynamic supply case after the improvement.

We further compare the cost saving generated by these five services. Overall spot instances, PSD, and our instance subletting service all bring over 83% cost saving to users. Cost saving of SRI stays around 60%. Though our instance achieves less cost saving than PSD and spot instances at 8000 bids, it generates 28.3% more utility than spot price at this time. After comparing the transactions, we find that this difference is mainly because the instance subletting service at this setting accepts more bids than the spot instance one. For average utility per bidder, the instance subletting service is 18% less than the spot instance service. In fact, thanks to this extra number of accepted requests, the instance subletting service also attains 63.1% more revenue than the spot instance service at 8000 bids.

The fixed price adopted by reserved instances extracts more surplus from the bidder side, leaving the accepted bidders less utility. This extreme surplus turns out to bring the largest revenue to the seller side. In reality, our studied instance subletting service is expected to complement the fixed-price model, rather than fully replace it. Researchers in [38] have already proved that a hybrid market (coexistence of both spot instances and fixed-price instances) always maximizes the provider's profit, even if it decreases their revenue. This claim still holds true for our envisioned scenario, where both fixed-price instances and instance subletting services exist[10]. What is more, in a highly competitive public cloud market, social welfare would be preferred to guarantee user base.

3.5.2 Trace-driven Simulations: the Case of Dynamic Supply

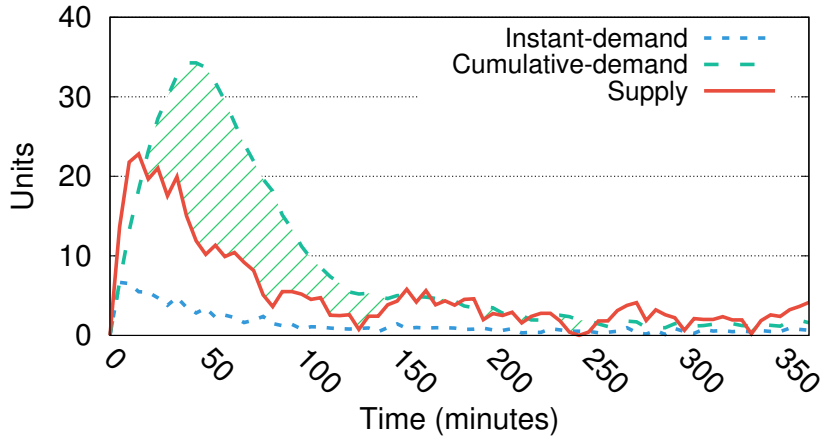
Experimental Settings

For the dynamic scenario, we use the same trace and settings except that we now incorporate the time attributes for all sublet instances and requests. Time unit in our simulation is set as the minimum time unit in the Google cluster trace, 300 seconds. A request or instance will be pushed into the simulator when its start time arrives. An instance will remain active, capable of accommodating requests, until its leasing deadline is reached.

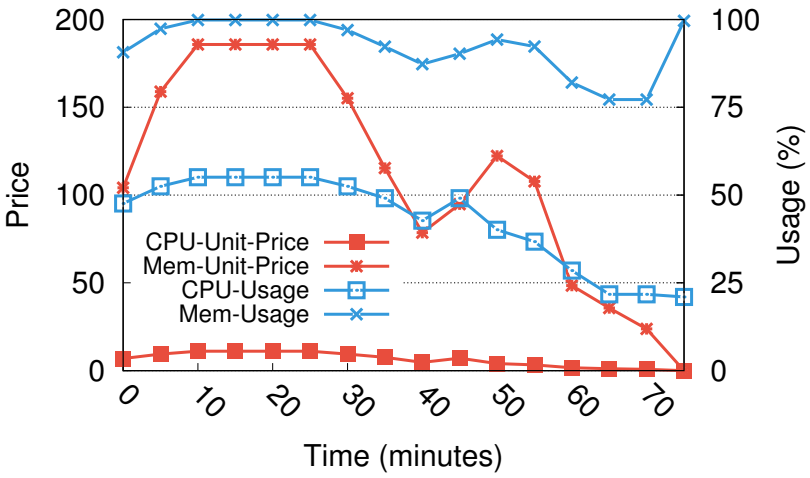
Results and Discussion

The dynamic arrivals of both requests and instances greatly complicate our scheduling. In Fig.3.6a, we first depict the demand and supply relationship in this dynamic supply situation using CPU as an example (N = 5000, M = 200). The solid line denotes the fluctuation of

[10]Preemption or not does not affect this general result



(a) Overview of demand/supply on CPU



(b) Dynamics of price and usage ratio in an instance

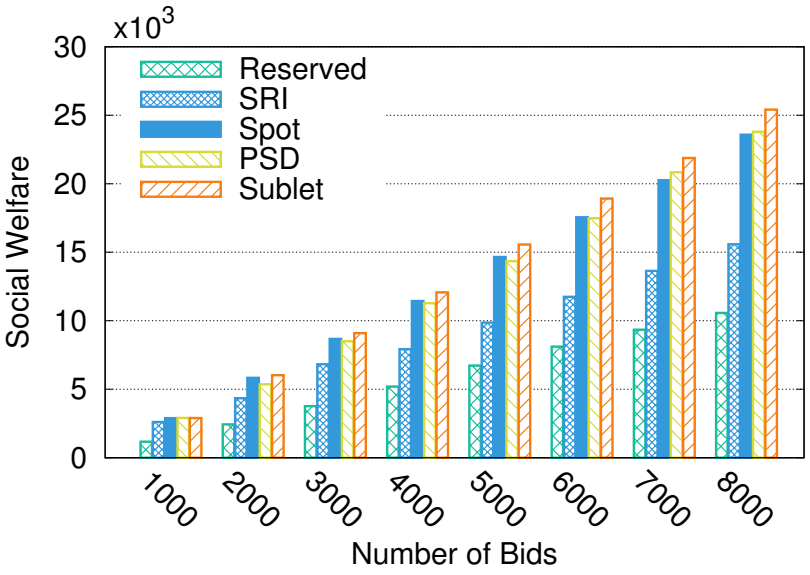
Figure 3.6: Dynamic demand and supply in our market and a snapshot of an instance

the total available CPU, i.e., the supply, in each time unit. Both dashed lines denote the requested demand of CPU, where the one sitting on the bottom represents instantaneous demand, the demand from the requests that just arrive in the current time unit; the other represents cumulative demand, the total demand that our platform is facing taking all active requests into account. The shaded area denotes the difference between cumulative demand and supply. The supply is almost always higher than the instantaneous demand, however, it is overwhelmed by the cumulative demand, especially during the 30-145 minutes timespan. As commonly occurs in reality, this exceeding amount of demand tests if an auction is well designed or not, because it needs to ensure that the highly scarce resources are well allocated.

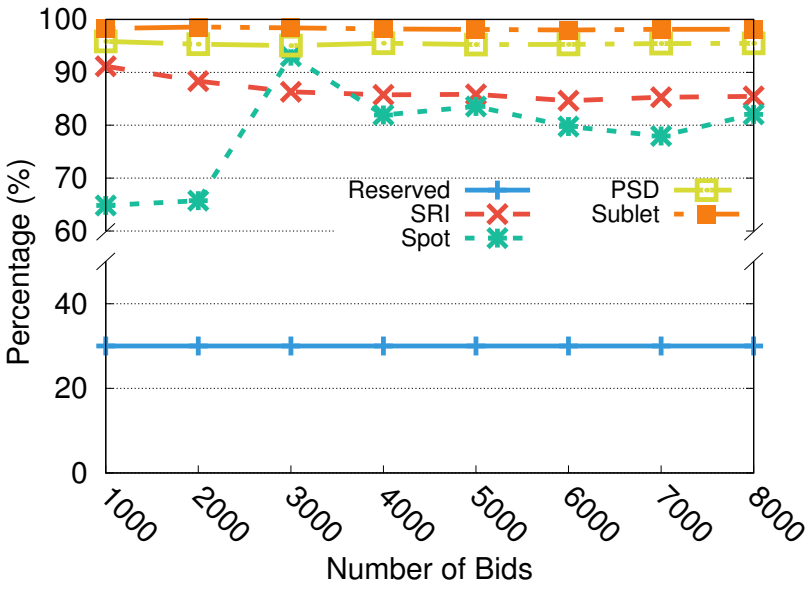
Fig.3.6b illustrates the interaction between prices and resources of a 75-minute-long instance in our instance subletting service. Its memory usage grows close to 100% between 5 minutes to 30 minutes. The unit price of memory, in turn, doubles in the first 10 minutes and remains at this high price for another 20 minutes, making the instance quite selective to admit new requests. The latter half of its life cycle is where our discount strategy starts to take effect. The memory usage fluctuates due to the admission of new requests, and price drops generally with the elapse of time to attract bids, which successfully allows memory to be fully utilized in the last 5 minutes. Similar trends also happen on the CPU usage. Memory is the obvious capping resource on this instance, thereby both the CPU's usage and price stay relatively low.

We now present the comparison of our instance subletting service with two dominant service models, reserved instances and spot instances, and two related studies, SRI and PSD, in this dynamic environment. In terms of social welfare (Fig.3.7a), the subletting service achieves the highest performance. The difference to spot instance service increases from 0.3% at 1000 bids to 7.7% at 8000 bids. The expansion of this gap illustrates the efficiency of our mechanism in picking out the valuable requests when the solution space is large. Recall that, to further handle the dynamics from the demand side, we devise the multi-stage discount strategy. It guides the requests to soon-expired instances and saves long-lived instances for the future using the invisible hand, price. Our result shows that the multi-stage discount based mechanism gains over 10% more social welfare than the discount-free version in all above 5000 bids situations. Overall, two research works for comparison all perform better than the dominant service models, reflecting a trade-off of complexity in allocation process and efficiency in allocation results happened in real world implementation. The difference to SRI increases from 11% at 1000 bids to 38% at 8000 bids. The instance subletting service also keeps achieving 7% more social welfare than the state of the art approach, PSD, in all the bid-over-2000 cases. This proves the efficiency of our multi-stage discount policy.

We demonstrate average cost savings of these five services in Fig.3.7b, which is defined as the savings of an instance over its on-demand counterpart. The cost saving of the reserved instance repeatedly stays at 30% due to its price setting. The cost saving of the spot



(a) Social welfare with varying bid numbers



(b) Cost saving with varying bid numbers

Figure 3.7: Performance comparison in the dynamic supply case

instances ranges from 64% to 93% with the average at 78%, which also accords with the official cost saving information about the spot instance service[11]. Our instance subletting service outruns two other commercial service models and related research works in all cases and steadily stays above 98%. The closing gap between the spot instance and our service is because the introduction of more bids brings higher possibility in having low spot price. It is worth noting that the presented performance of spot instances is the best possible case. The performance of spot instances in real life would be worse under the same rules. In reality, each type of instance in each region follows the same spot price. In our case, we create a spot price for each individual spot instance regardless of their region or type. According to the price discrimination principle in pricing theory, such a fine-grained pricing approach is better at adapting to the fluctuations of the demand and supply than the real-world spot instance setting. Similar to the performance in social welfare, our service keeps achieving over 10% more cost saving than SRI in all cases(except bid number = 1000), and 3% more cost saving than PSD.

As a conclusion, from both the static and dynamic simulations, our proposed online auction mechanisms can indeed attain high overall system performance under real-world workload patterns.

3.6 Practical Challenges and Prototype Validation

The proposed auction mechanism is a general framework: it is not tied to a specific implementation of instance subletting service. Still we are keen in pointing out a few practical concerns that we found critical in implementing such service. This will not only allow us to inspect our modeling assumptions in the theoretical part but also further complement our mechanism into a fully integrated service. To this end, we built a prototype of instance subletting service on Amazon EC2 public cloud. But before presenting our prototypes and experiment results, we will first list out the challenges and concerns in what follows.

The crux of implementing the subletting service lies in enabling multiple users to share an instance. Specifically, the platform should provide an API for hosting users to quantify the amount of resources they prepare to sublet and allocate the right amount of instance resources for each tenant user. (*Challenge 1*). In addition, the platform should maintain an isolated runtime environment for each tenant user, ensuring they can use only their own share of resources and preventing them from interfering each other (*Challenge 2*). Furthermore, the platform should be capable of managing a large cluster of instances, including the dynamics of instance joining and leaving, and providing a consistent view of the cluster states for the auction mechanism (*Challenge 3*). Given these challenges, we believe that the latest *container* technology is a promising tool for implementing the

[11] `https://aws.amazon.com/ec2/spot/`

subletting service. Container is a lightweight, application-oriented virtualization technology that is becoming increasingly popular among public cloud providers [28].

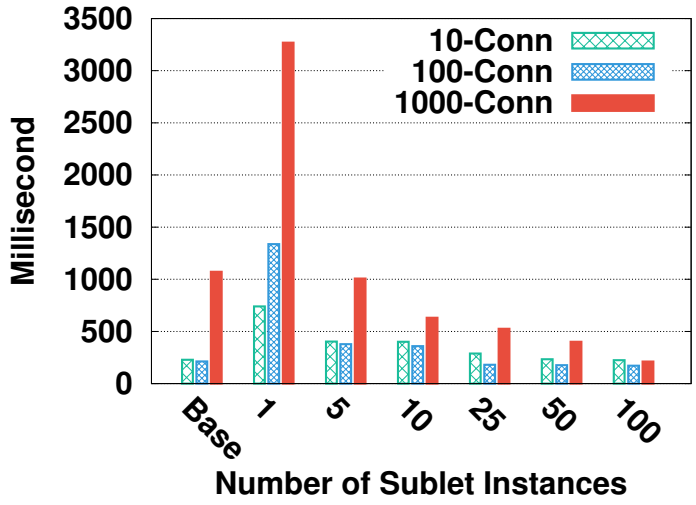
Container meets the design requirements of our auction mechanism (by addressing the above challenges); although other forms of implementation, like nested virtualization, would be possible, too. To address Challenge 1 and 2, the subletting system (in this case, our prototype) leverages `cgroups` module in the container to enforce each workload only use their own resource allocation share. The `cgroups` module defines a collection of kernel controllers for system resources including CPU, memory, network, etc. These controllers are assigned to the container runtime in the form of function hooking. As such, the hosting users can now specify how much resource share are made available to the tenant users, and the platform will refer to these specifications when it receives new resource requests. When the container starts running, those hooks ensure that the container does not use more than it is allowed to use. Meanwhile, every container will be associated with a unique set of resource identifiers for its PID, IPC, network, and file system etc., known as the `namespace isolation`, a feature that provides isolated runtime environments for in-container workloads. We use Docker [4] in our prototype to perform container-related operations.

To address Challenge 3, we leverage the Amazon EC2 container service (ECS), featuring a cluster state management module. Specifically, the module will run a consensus-based transactional journal to keep track of the cluster state information, maintaining a consistent view on the pool of instances. The ECS service also provides a customizable scheduler module allowing us to implement our auction mechanism.

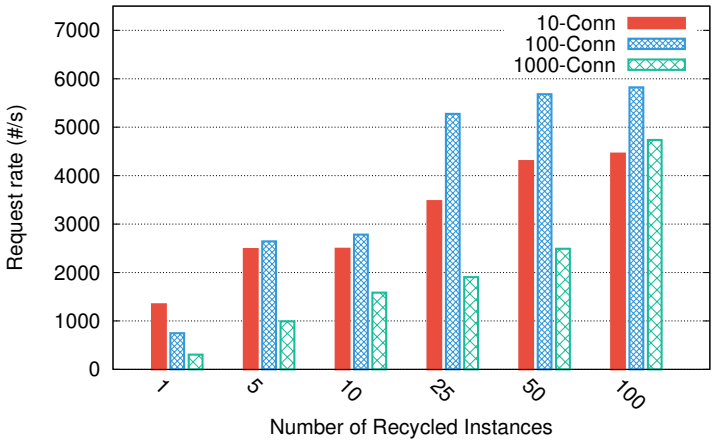
We preliminarily evaluate the prototype with two typical types of cloud workloads: multi-tier web applications and batch workloads. We chose the RuBBoS[12] multi-tier web server benchmark for the web application. We emulate the web client using `Apache Benchmark` and use a load balancer to dispatch the web requests to servers. We use `sysbench` as the batch workload, and the invocations of sysbench are independent from each other. On each sublet instance, we run these workloads inside containers, one for the web server and the other for the batch job.

For the web application, we consider the average request rate and the average web request completion time. These two metrics correspond to the throughput and latency performance of the web application. We are also interested in how these metrics change as the service scales out to having more sublet instances. To do this, we keep the cluster under load by constantly submitting buyer requests that comprise these two types of workloads. In fact, each round of our experiments can be regarded as the static supply case in Sec.3.3. We obtained the baseline performance for the web application by running a single web server on an on-demand instance without using container. The baseline performance for the batch

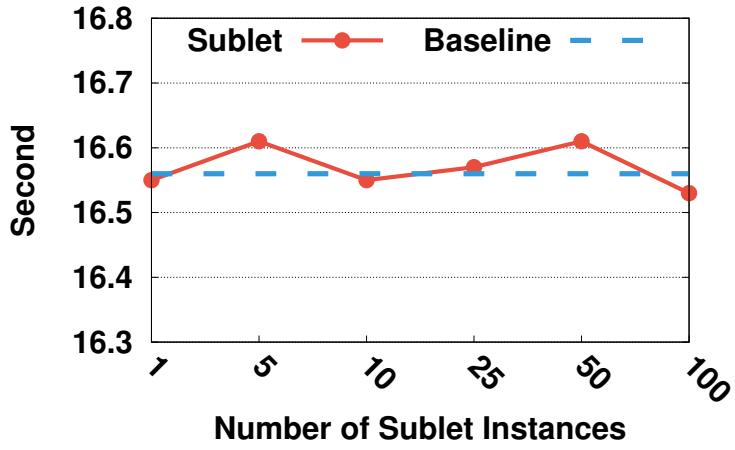
[12]RUBBoS Bulletin Board Benchmark: http://jmob.ow2.org/



(a) Avg. web request completion time



(b) Avg. web request rate



(c) Avg. batch task completion time

Figure 3.8: Prototype performance on web and batch workloads

workload is likewise obtained, except it is run inside a container with specified resource usage, which allows us to examine whether Challenge 1 and 2 are addressed.

The average request completion time in the web application is reported in Fig.3.8a. Compared to the baseline, the completion time of web requests is much increased under one sublet instance. This is an expected case for workloads using the instance subletting, because other running workloads on the same host are also consuming resources. Fortunately, this deficiency can be remedied by placing additional web servers when more instances are available. The performance of web application catches up with the baseline from 5 sublet instances. In Fig.3.8b, the throughput performance also sees similar improvement with the increase of sublet instance. While adding more web servers to sustain high performance may sound costly, the total monetary cost of acquiring these sublet instances can still be much lower than the original instance as described in Sec.3.5. Specifically, as a common challenge, application service providers would need to properly scale their services in the face of bursty workloads (e.g., a flash crowd). This is usually done through autoscaling at VM-granularity or overprovisioning VMs. In our instance-subletting platform, because containers allow finer-grain resource offerings that can be provisioned much faster than VMs, provisioning can happen just-in-time and with smaller resource consumption, e.g., when we detect surging queue-depth at the load balancer, and hence the cost of overprovisioning can be greatly reduced.

Note that with the increased throughput, though the latency of completing *all requests* will certainly be shortened, the *per-request* latency may not. For example, Fig.3.8a shows the 10 sublet instance setup can still give worse per-request latency performance than the baseline. This is because the load balancer can add an extra queuing delay to each request. Users can opt in more powerful load balancer to remedy such latency. Also, we observed that the 1000 connection setup yields lower performance than the 100 connection one as our emulated client becomes the bottleneck when the concurrent connection is high.

For the batch workload, as shown in Fig.3.8c, with sublet instances, the batch workload performance makes no statistically significant difference ($< 1\%$) as compared to the baseline. This shows that containerization did allow us to enforce the resources usage guarantees. Note that sysbench has a fairly stable resource requirements across its runs. In reality, a batch workload may have variable resource demands. Due to the use of container, if the batch workload running inside exceeds the resource usage, the workload will get throttled. Therefore, the owners of such workloads should tailor their container requests accordingly to achieve expected and meaningful performance.

Overall, these initial results are promising, suggesting that instance subletting with our online auction mechanism can indeed be built on current public cloud with minimal impact on users' perceived performance. Before concluding, it is worth discussing a few additional concerns that may limit the real-world deployment of instance subletting. Though we have yet to explore these issues on our current prototype, we do observe technological trends that

can help alleviate these issues. First, both container and nested virtualization solutions may result in OS tie-ins, because certainly not all OSes support these technologies. OS tie-ins could limit the types of workloads that can be run on the instance subletting service. Fortunately, standardization efforts in the container technology, e.g., the Open Container Initiative [6], demonstrate the on-going trend across OS vendors to support containers. In addition to Linux distributions, which natively support containers, other major OSes such as Windows are adding kernel supports for containers. This trend is thus one of the key reasons we chose containers in the prototype and the modelling assumption.

Meanwhile, it is also worth investigating how to effectively implement our allocation mechanism. Our prototype relies on Amazon ECS's default replication scheduler to allocate the benchmark workloads, which supports replacing the scheduling policy with user-supplied, customized ones. In our future work, we plan to port our simulator's allocation mechanism to the ECS scheduler and perform end-to-end, system-level evaluation over the instance subletting service.

In addition, though higher server utilization may lead to higher power usage to the providers, increasing it is critical to maximize the energy efficiency because server power consumption responds differently to varying utilization levels [21]. Higher resource utilization can help cloud providers to amortize their capital better. So overall, we believe an instance subletting service is promising as cloud providers can exploit subletting services as another form of differentiated, value-added service to attract diverse user groups, gain extra revenue.

3.7 Summary

In this chapter, we systematically examined *instance subletting*, a new cloud service that explores the idle resources from users, making them available to the public. Instance subletting offers a trading market for low-cost yet high-quality instances with enforced service level agreement on time. The market works with dynamic demand, and more importantly, it has a dynamic supply and time constraint on each item, which is not available in past products and studies. We presented an online auction mechanism with provable truthfulness and individual rationality, as well as the best possible competitive ratio with known supply information. We then extended it to cope with dynamic supply. Large scale simulations have indicated that our mechanism can achieve near optimal social welfare with significant cost savings. Its feasibility has been further validated through an Amazon EC2 based prototype.

Chapter 4

Exploiting Crowd for Low Latency Transcoding

In this chapter, we first examine the importance of latency in processing live videos. 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. Several budget constrained online and offline algorithms are proposed after that. Our PlatnetLab-based experiment and trace-driven simulations further proved the superiority of our online scheduler.

4.1 System Model and Problem Formulation

4.1.1 Why Delay Matters for Video Services in the New Era?

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 [60]. 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 [47]. 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

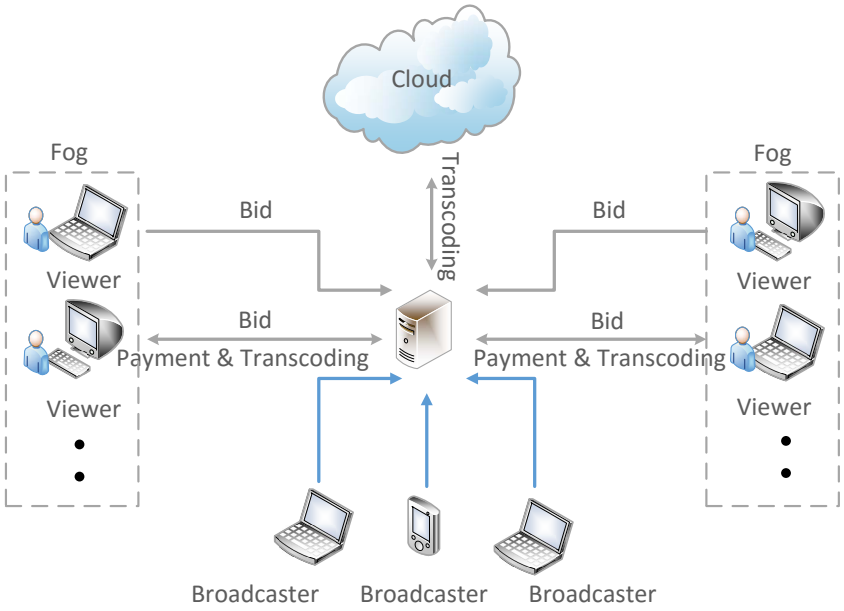


Figure 4.1: System overview

could generate false positive feedbacks to the broadcasters, which greatly affects the information value hidden in these messages [89]. 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 [91]. 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 [35]. Researchers therefore are actively seeking for new approaches or architectures to reduce the interaction delay for viewers.

4.1.2 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. 5.3 shows the overall design. Specifically, source live streaming contents are first collected by the regional server through protocols such as RTMP (Real Time Messaging Protocol). 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 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.

4.1.3 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 $C = \{c_1, c_2, ..., c_m\}$, and viewers as $V = \{v_1, v_2, ..., v_n\}$. 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 [48]. 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 [48]. 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

Table 4.1: Table of notation

Symbol	Description
v_i	viewer i
b_i	bid value of viewer i
$S(v_i)$	quality of viewer i
p_i	price charged to viewer i
B_j	budget allocated for channel j
l_min	maximum allowed transcoding time
l_i	transcoding time of viewer 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:

$$\max \quad \sum_i S(v_i)x_{i,j} \tag{4.1}$$

$$\text{s.t.} \quad \sum_i p_i t_i x_{i,j} \leq B_j, \forall B_j \tag{4.2}$$

$$\sum_j x_{i,j} \leq 1, \forall i \in V \tag{4.3}$$

$$l_i x_{i,j} < l_{min}, \forall x_{i,j} \in X \tag{4.4}$$

$$x_{i,j} \in \{0, 1\} \tag{4.5}$$

In constraint (4.2), the sum of all payments of a channel in the streaming period should be smaller than its planned budget. In constraint (4.3), 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 (4.4), the transcoding time of selected viewers should be less than the minimum requirement to guarantee high quality interactive live streaming. We further summarize the important notations used in this chapter for convenience in Table. 4.1.

4.2 Crowd-based 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.

4.2.1 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 [80], 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 4. The resulted threshold p represents the reasonable price per unit quality value. The main scheduling algorithm (shown in Algorithm 3) 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 (line 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.

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.

Algorithm 3 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 ← 0, index ← 0, X_j ← ∅
4: p ← GetThreshold(B_j)
5: while count ≤ R do
6:    index ← index + 1
7:    v_i ← V[index] and b_i is the bid of v_i
8:    if b_i ≤ S(v_i) * p ≤ B_j − Σ_{x∈X_j} p_x then
9:       p_i ← S(v_i) * p
10:          X_j ← X_j ∪ {v_i}
11:   else
12:      p_i ← 0
13:   end if
14: end while
15: return X_j
```

4.2.2 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 5) 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.

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

Algorithm 4 GetThreshold

```
1: Input: Budget B_j
2: Sort V in decreasing order of S(v_i)/b_i
3: X ← ∅, i ← 0
4: while b_i ≤ S(v_i)B/S(X ∪ v_i) do
5:     X ← X ∪ v_i
6:     i ← i + 1
7: end while
8: p ← B/S(X)
9: return p
```

in Algorithm 6. Similar to the baseline scheduler, the comprehensive scheduler calculates the price threshold and orders all candidates (line 2-3). Then, in each round, it pushes the remaining most stable candidate into a priority queue with its reasonable payment as key (line 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 (line 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 6 is $O(Nlog(N))$, where N is the total number of candidates. We next show Algorithm 6 can provide an optimal solution under the given payment rule in certain situations.

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 7. *Algorithm 6 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 6 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 6 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

Algorithm 5 Psudo-polynomial optimal approach

$p \leftarrow GetThreshold(B_j)$
for i from 1 to N **do**
  $p_i = p * S(v_i)$
end for
initialize *table* as a three-dimensional table
for B from 1 to B_j **do**
  for i from 1 to N **do**
    for r from 1 to R **do**
      if $b_i > p_i$ or $p_i > B$ or $r == 0$ **then**
        $table[i, B, r] \leftarrow table[i-1, B, r]$
      else
        $v[i, B, r] \leftarrow max(table[i-1, B, r], S(v_i) + table[i-1, B-p_i, r-1])$
      end if
      record such transition in a backtrack table
    end for
  end for
end for
backtrack and return the whole schedule

Algorithm 6 Comprehensive Scheduler with Strict Number of Transcoding Viewers

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

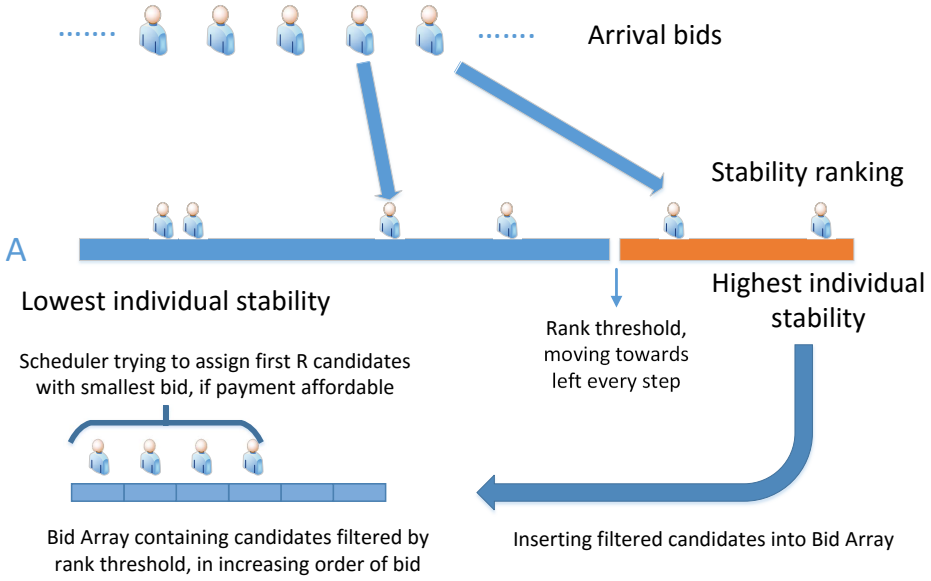


Figure 4.2: Illustration of the online strategy

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 6 will have an optimal result. □

4.2.3 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. 4.2, and described in Algorithm 7 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 i^{th} 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 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 $bidArray$ (line 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 (line 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 (line 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 (line 15-16), to maintain truthfulness. This pricing schemes falls into the generalized second price scheme, where the bidder in i-th position pays the bid of the $(i+1)$-th bidder [87] . 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.

Algorithm 7 Online Scheduler

Scheduler broadcast the transcoding request with the description of the job to all qualified stable viewers
Initialize an empty array A of size N, where N is the number of total qualified stable viewers
Initialize a dynamic array $bidArray$
Initialize $rankThreshold \leftarrow 1$
while $t < timeout$ **do**
  Let $respondingCandidate$ be the candidate returning a response
  $rankThreshold \leftarrow rankThreshold + 1$
  if $A[rankThreshold]$ is not empty **then**
    Insert $A[rankThreshold]$ into $bidArray$,
      with $A[rankThreshold].bid$ as the sorting key
  end if
  $A[respondingCandidate.rank] \leftarrow respondingCandidate$
  if $respondingCandidate.rank \leq rankThreshold$ **then**
    insert $respondingCandidate$ into $bidArray$,
      with $respondingCandidate.bid$ as sorting key
  end if
  if $sumOfBid(bidArray[2\ to\ 1 + R]) \leq B_j$ **then**
    Assign $bidArray[1\ to\ R]$, for each assigned candidate $bidArray[i]$,
      provide payment $bidArray[i + 1]$;
    return
  end if
end while
select top R' viewers from $bidArray$, where R' is the maximum number of viewers the budget B_j can afford, return

4.3 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.

4.3.1 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% channels attract around 98% of the total viewers, we only regard these top 10% 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 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.

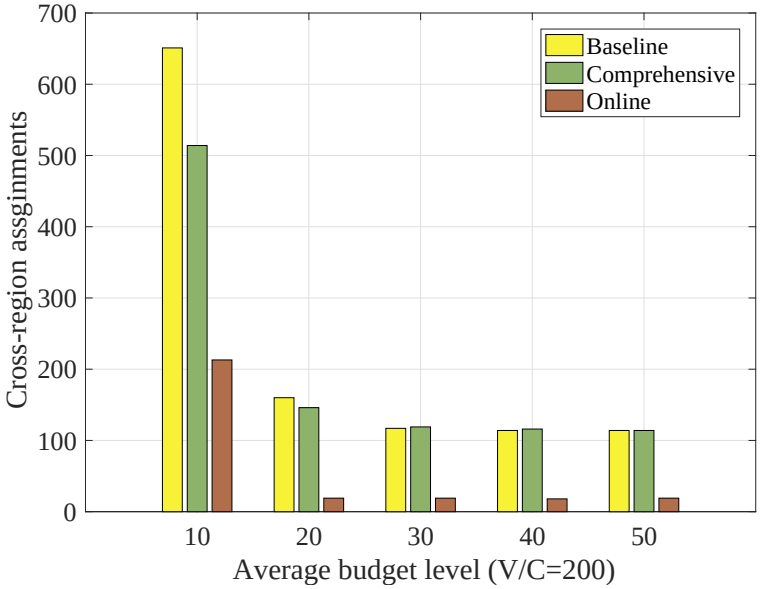
4.3.2 Evaluation Results

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%, and thus we omit it here and only compare the performances of our solutions under different offline/online scenarios.

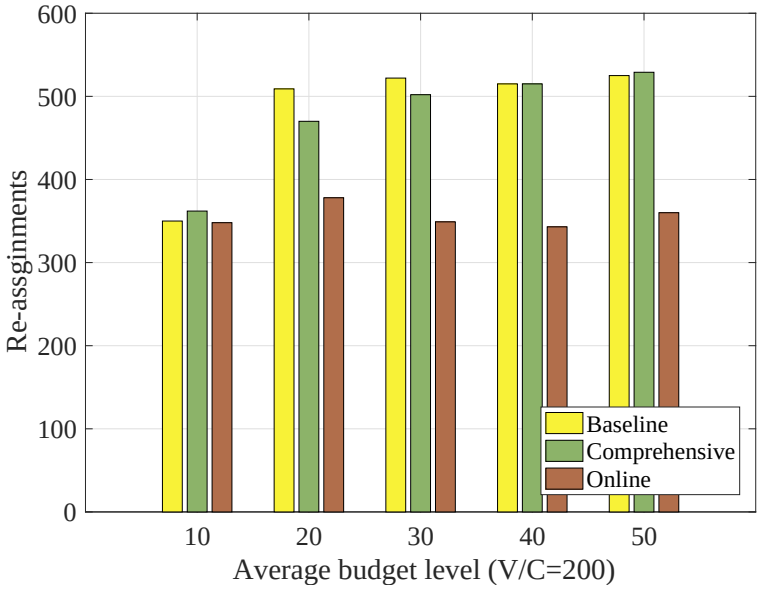
We first conduct simulation to see the number of cross-region assignments under various conditions. Regarding to different V/C ratios, the results of three schedulers are similar in respect to their relative performance. Thus we only report the results when $V/C = 200$ in Fig. 4.3a. Notably, the advantage of the comprehensive scheduler under low budget level, compared to the baseline scheduler, is more remarkable at higher V/C ratio. From Fig 4.3a, 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. 4.3b, the results of the baseline scheduler and the comprehensive scheduler again are similar, and are around 50% 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 Fig. 4.4a, Fig. 4.4b and Fig. 4.5a, 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% 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



(a) Cross-region assignment



(b) Reassignment count with different budget

Figure 4.3: Stability analysis under different budget levels when V/C=200

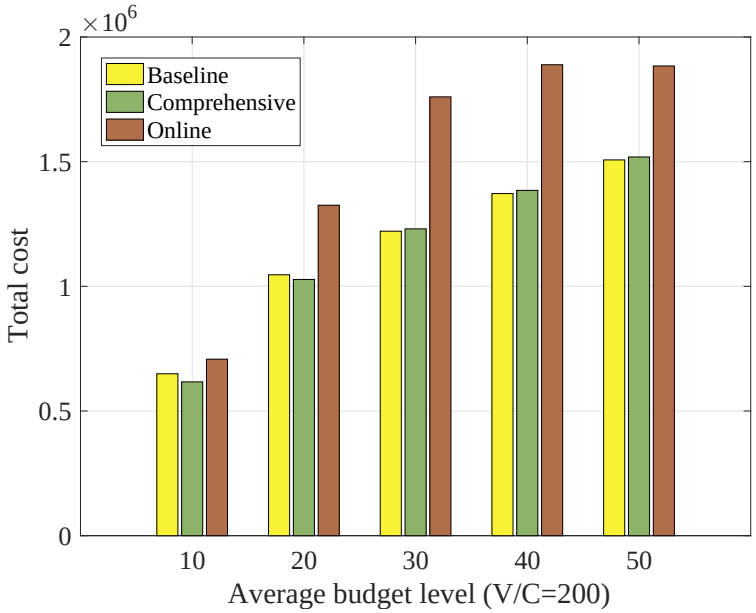
knowledge on the arrival of bids, the payment is greatly influenced by the arriving patterns, which makes the cost higher than the offline situations.

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 the opposite. Fig. 4.5b 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.

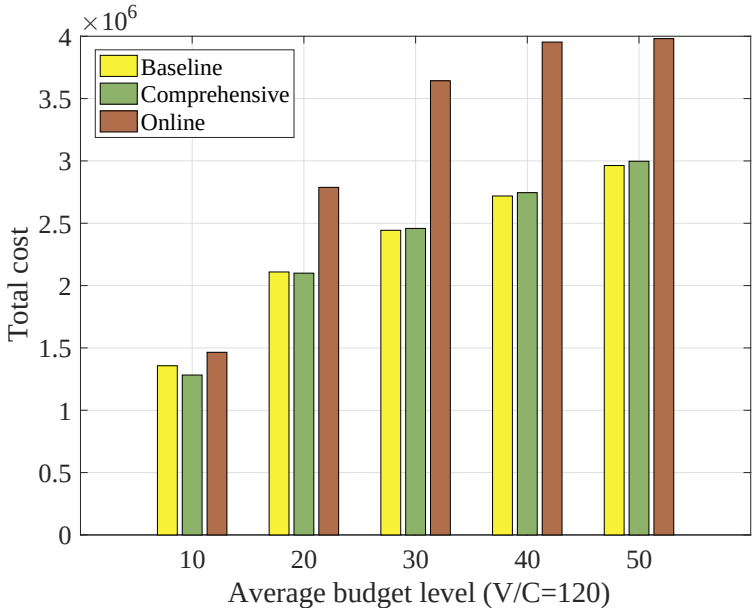
In short, the online scheduler has significantly better performance compared to the other two, in terms of reassignment count and cross-region assignment count, while without having extra cost overall. Interestingly, the online scheduler however benefits from being not able to calculate the threshold 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. 4.6a, 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. 4.6b. 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.

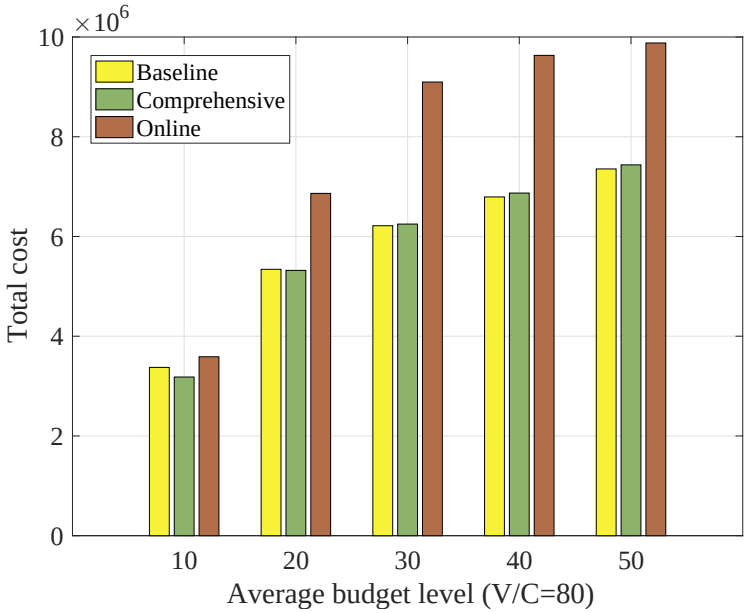


(a) Cost of different approaches (V/C=200)

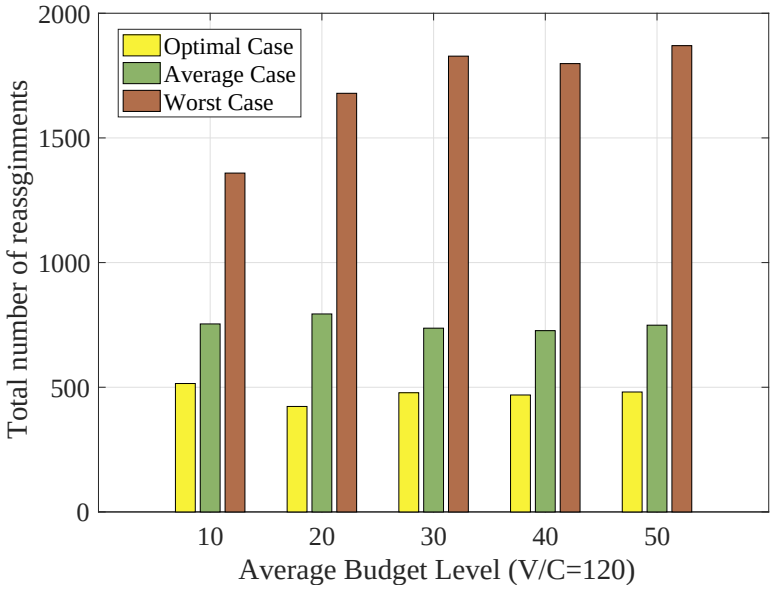


(b) Cost of different approaches (V/C=120)

Figure 4.4: Cost comparison under different V/C values

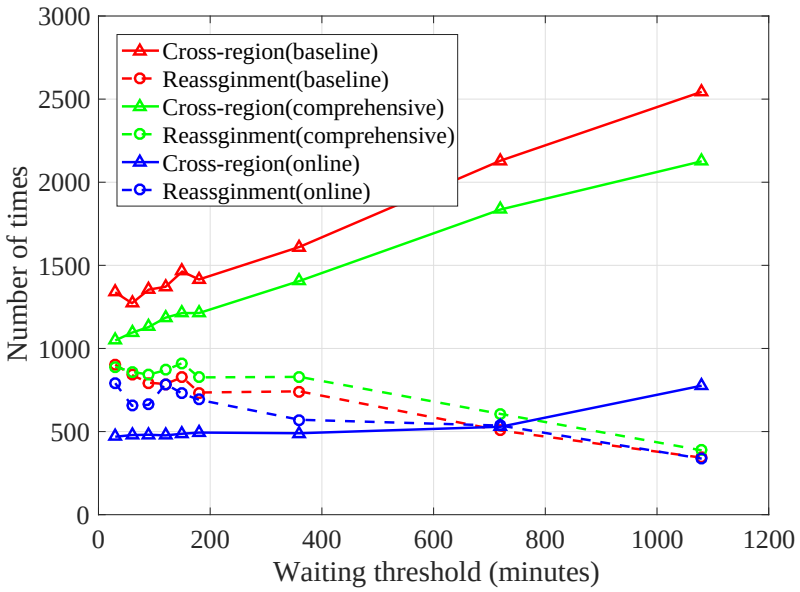


(a) Cost of different approaches (V/C=80)

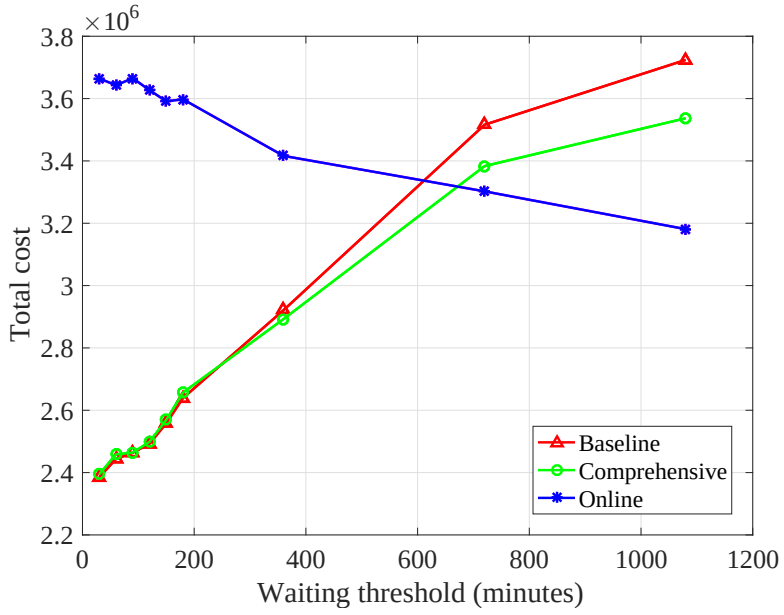


(b) Extreme cases of the online scheduler

Figure 4.5: Impact of budget on stability and cost



(a) Different waiting threshold with low budget



(b) Total cost under different waiting threshold

Figure 4.6: Impact of waiting threshold on stability and cost

4.4 System-level Evaluation

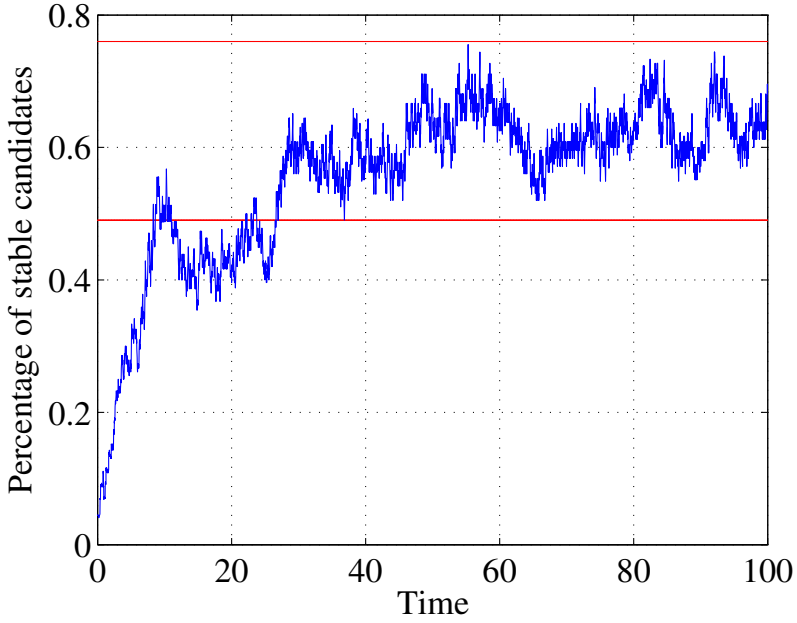
4.4.1 Prototype Setup

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 3.5Mbps 1080p video as our source video and set 2.5Mbps 720p as the target quality. Each video slice is of 1 second.

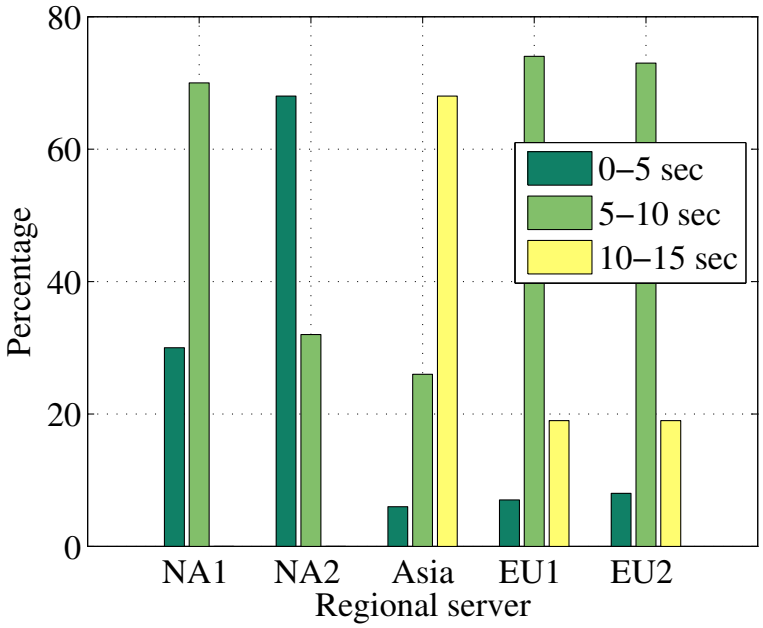
4.4.2 System Performance Results

We first measure the percentage of stable viewers in the system after it runs for 60 minutes. As shown in Fig. 4.7a, with a 60-minute waiting threshold, around 60% 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% and 76%, 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. 4.7b 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.



(a) Stable viewer ratio along measured time



(b) Streaming delay in different regions

Figure 4.7: Stability and delay analysis in the Planetlab-based prototype

4.5 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) [57] [47] [83] and the technical perspective (e.g., streaming performance, user experience) [17] [75]. 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. [31] 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. [48] 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. [80] 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 [51] . Maharjan et al. [67] 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[109]. 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.

4.6 Summary

In this chapter, 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.

Chapter 5

Towards Reliable Cloud-Crowd Collaborative Transcoding

In our previous chapter, involving one viewer for taking one transcoding task may still be unstable considering the dynamics of viewers as can be observed in the evaluation results. In this chapter, we explore the possibility of recruiting multiple viewers to take one task, increasing its reliability through redundancy. Mechanisms need to be carefully designed for such a many-to-one mapping situation, especially considering the uncertain behaviours of viewers.

In the following chapter, Section 5.1 discusses the challenges for having a cloud-crowd collaborative system. Section 5.2 presents the formal formulation of our problem and other desirable goals. Section 5.3 presents the detailed mechanism design for broadcasting workloads. We evaluate our design through trace-driven simulations in Section 5.4. Section 5.6 summarizes the whole chapter.

5.1 Challenges and Principles

5.1.1 Why a Cloud-Crowd Collaborative System?

Most CLS platforms start primarily as a place to watch livestreamed contents and recently have began to diversify their content sources. Take Twitch for example, broadcasters normally livestream their game content to viewers and interact with them through chat

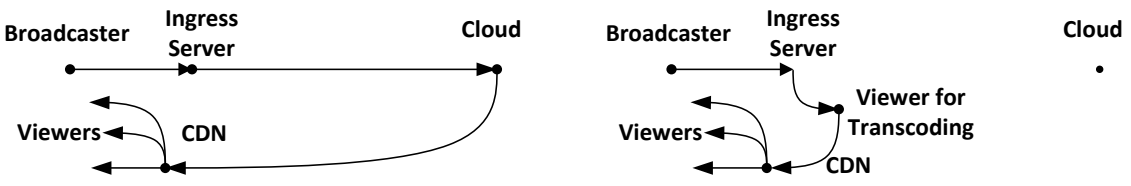


Figure 5.1: Effects of local popularity on latency reduction

messages. Besides these *broadcasting workloads*, Twitch also enables "Uploads" in late 2016, allowing broadcasters to upload pre-recorded content and publish later [9], to which we refer as *pre-recorded video workloads*. For both types of workloads, the heterogeneous source formats and network/viewer device configurations require videos to be transcoded into multiple representations. However, they have different requirements for transcoding processes. For broadcasting workloads, each live session requires non-stop transcoding service with low latency, so that viewers can receive their desired video formats with small startup delay and keep interacting with broadcasters with no interruptions. Pre-recorded workloads, however, are interruption-tolerant as long as the user-defined playback deadline is met. The diversity of service types without a doubt increases the demand for computational resources and calls for more efficient cost management to meet heterogeneous Quality of Service (QoS) requirements.

For CLS, involving crowd into the transcoding processes is expected to greatly reduce latency. Though cloud providers keep opening up new service regions in recent years, the nature of centralized datacenter-based cloud determines that the distribution of broadcasters and viewers in existing CLS platforms is far more diverse than these datacenters. Therefore, the distance between broadcasters to the closest cloud datacenter can still be too far to meet latency requirements. 90% users could have an over 200 ms interaction latency with the naive deployment on cloud [91]. What is more, after carefully examining the viewer-broadcaster trace we have, unlike the traditional live streaming services where viewers of popular streams tend to be evenly distributed [64], geo-distribution of viewers for a specific broadcasters is highly skewed (48% of broadcasters have their viewers totally in the same region) in our studied CLS services[1]. Since most viewers that consume a stream locate in the same region as the broadcasters, allowing local viewers transcode the same stream will greatly reduce the communication distance from broadcasters to viewers. Fig.5.1 illustrates how latency reduction can be achieved by allowing local crowd to transcode the local stream thanks to the local popularity of some channels. In addition to latency reduction, reducing cost is another benefit that a Cloud-Crowd collaborative system could bring. Current cloud providers possessing strong pricing power offer rigid pricing schemes for cloud users. It is difficult to incorporate new pricing polices or bargain with the cloud to get reasonable prices for different use cases. While hiring viewers gives us flexible pricing power, viewers with different levels of willingness to pay can be guided into much lower trading prices under the proper pricing mechanism without hurting their own interest at the same time.

If we want to totally rely on distributed crowds, their reliability still can be the top concern like previous peer-based distributed systems. Without the strict Service Level Agree-

[1]We suspect that it is because the content in CLS is more user-generated rather than the big event focused live streaming like before. Viewers are more interested in knowing the broadcasters in their proximity driven by the social purpose.

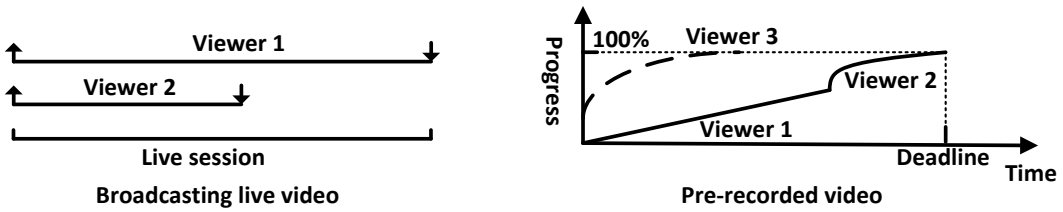


Figure 5.2: Redundancy illustration: simultaneous redundancy for broadcasting live video workloads and sequential redundancy for pre-recorded video workloads

ments (SLA) like they formed between cloud providers and cloud users, it is hard to build a reliable system, especially for our CLS. As a practical system, facing tasks with QoS requirements and dynamic supply of computational power from viewers, it is still a wise choice to combine crowd with cloud. This hybrid mode allows us to extend the cloud computing paradigm to network edges with end user clients serving as a complementary component of cloud computing.

5.1.2 Why Redundancy Helps?

Although assigning transcoding tasks to viewers may help reduce cost and latency, involving these viewers actually presents unique challenges for each workload type. First, to lower the acceptance barrier of such system, we envision that viewers would only do transcoding when they are surfing CLS platforms, which means that their duration for transcoding varies. Unlike entirely relying on cloud where instances are guaranteed by strict SLA, analysis of our trace shows that the duration of viewers for staying in the platform follows Pareto distribution with varying parameters. This heterogeneous participation behaviour greatly challenges the broadcasting transcoding workloads, since assigning tasks to unstable viewers definitely makes the transcoding inconsistent. Second, task execution performances of viewers (e.g., total task execution time) greatly vary due to their heterogeneous computing power. The diverse task execution time greatly challenges pre-recorded video workloads since assigning tasks to random viewers may make tasks miss their deadlines.

The redundancy principle has been widely applied to mitigate uncertainty and improve system performances in distributed systems [81, 43, 77, 88]. In our scenario, we devise different redundancy strategies for different types of workloads. Broadcasting live video workloads value continuous transcoding without any interruptions; Pre-recorded video workloads value finishing the transcoding before a overall deadline. Correspondingly, we recruit redundant viewers simultaneously to mitigate viewers' uncertainty in broadcasting workloads so that redundant viewers can instantly compensate for the loss brought by the free rider, and recruit redundant viewers sequentially to mitigate viewers' uncertainty in pre-recorded video

workloads so that redundant viewers can fill in to guarantee finishing the task before the deadline. For instance, as illustrated in Fig.5.2, for a broadcasting task in the left subfigure, assume that each viewer i has its own probability p_i for leaving the system within a given session period. Introducing one more redundant viewer, viewer 2, to work along with the initial viewer, viewer 1, changes the probability of successfully transcoding a live session without interruption from $1 - p_1$ to $1 - p_1 p_2$. For pre-recorded video workloads in the right subfigure, assume that each worker has an individual probability for finishing the task before deadlines reflecting its heterogeneous computing power and load situations. Assigning a pre-recorded video task to a viewer, viewer 1, with smaller capability (usually with smaller cost as well) first, followed by a more powerful one, viewer 2, may be more cost efficient than simply assigning this task to the viewer with the strongest computing power, viewer 3. It is also worth noting that deploying redundant instances at different regions to guarantee the continuity of the transcoding has already become an option in major streaming hosting service platforms, like Wowza. But apart from the simultaneous redundancy, we also consider the sequential redundancy for pre-recorded workloads, more importantly, how to incorporate these solutions into our auction design when interacting with the independent, self-interest viewers.

5.2 System Model and Problem Formulation

5.2.1 System Model

Globally, our system is divided into multiple regions, where each region has its own regional datacenter (referred to as "ingress server") for ingesting source videos, assigning transcoding tasks to viewers or cloud. The streaming segments after being transcoded are forwarded to CDN for further delivery. Fig. 5.3 illustrates the overall design of our cloud-crowd transcoding system. Specifically, source broadcasting contents are first collected by the ingress server through protocols such as RTMP (Real Time Messaging Protocol). Several viewers will then be selected for transcoding according to certain criteria. Unmatched tasks after this selection or during the broadcasting process will be sent to cloud if no further satisfiable transcoding viewers can be found locally. The selected viewers and cloud servers transcode video contents into different quality versions and send them to CDN serving all viewers.

A live session with source format 1080p60fps will be transcoded into 720p60fps, 720p30fps, 540p30fps, etc., according to current twitch settings. Each representation of this live session represents a task that needs to be transcoded in our scenario. We leverage similar streaming test module like Twitch Inspector[2] in the existing Twitch platform to evaluate the network performance of viewers. Only viewers passing the minimum bandwidth requirement can be added into the candidate pool. After this initial capability evaluation, we have a viewer

[2] `https://inspector.twitch.tv`

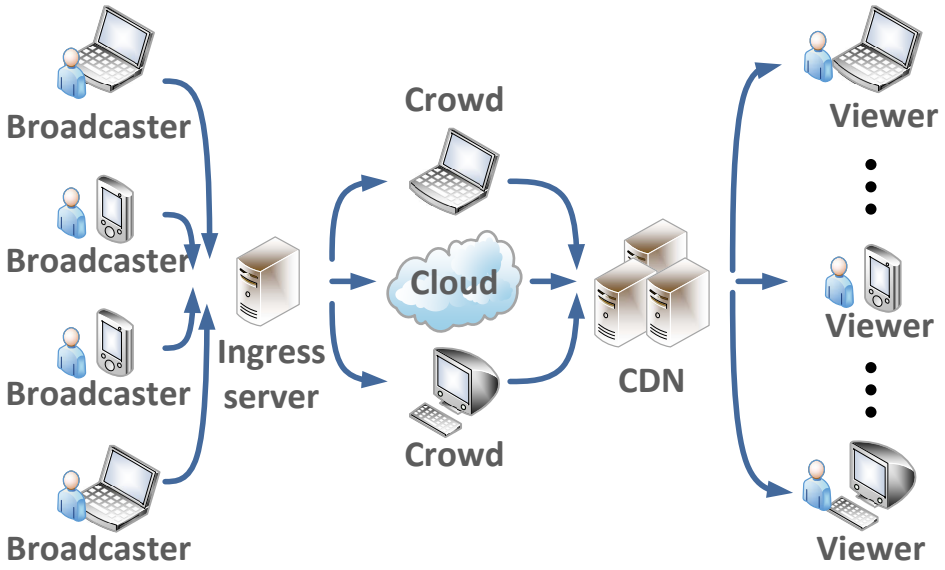


Figure 5.3: Cloud-Crowd system overview

set $N = \{1, 2..., |N|\}$ and a transcoding task set $M = \{1, 2, ..., |M|\}$ in our system. Let $c^l_{i,j}$ denote the cost of viewer $i \in N$ for taking a transcoding task $j \in M$ in broadcasting workloads, and $p^l_{i,j}$ denote the probability for failing to complete task j, namely viewer i leaves the platform before the end of its serviced transcoding task j. Similarly, we have $c^d_{i,j}$ for the cost of viewer i in taking the pre-recorded video task j and $p^d_{i,j}$ for the probability of completing its assigned task before the deadline. The cost of viewers reflecting their willingness to accept the task can be influenced by various factors, like their regional electricity price, making their willingness vary from person to person.

Each viewer thus has a type $\theta_i = (c^\tau_i, p^\tau_i)$ where $\tau \in \{l, d\}$ represents the total type set of workload. Specifically, $(c^l_i, p^l_i) = \{(c^l_{i,j}, p^l_{i,j}), \forall j \in M\}$ for broadcasting tasks; $(c^d_i, p^d_i) = \{(c^d_{i,j}, p^d_{i,j}), \forall j \in M\}$for pre-recorded video tasks. Let $\hat{\theta}_i = (\hat{c}^\tau_i, \hat{p}^\tau_i)$ denote the declared type (bid) of viewer i since selfish viewers may misreport their types to gain better utility. Let $\theta_{-i} = (\theta_1, ..., \theta_{i-1}, \theta_{i+1}, ..., \theta_N)$ be the profile types of all viewers except i, and (θ_i, θ_{-i}) completes the whole profile of all viewers, θ. We adopt the direct revelation principle in mechanism design to design auctions since it provides clear input information and allows us to focus on devising direct mechanisms in our auction. Following the direct revelation principle, a mapping function π, which includes an allocation mechanism (also known as social choice function) f and a payment mechanism λ, maps collected types to result R. $\theta_1 \times \theta_2 \times ... \times \theta_n \rightarrow R$ in our auction mechanism. Given one collected type input, the allocation and payment results of all viewers, $(f_1, f_2, ...f_{|N|}, \lambda_1, \lambda_2, ..., \lambda_{|N|})$, form one resulting result $r \in R$. Since the selected viewer may leave the transcoding task during the transcoding process or fail to complete the transcoding task before the deadline probabilistically, we

Table 5.1: Table of important notations

Symbol	Definition
π	auction mechanisms, including f and λ
f, λ	allocation policy, payment policy
τ	workload type set
$\theta_i = (p, c)$	type of viewer i, including uncertainty p and cost c
V_j	valuation for task j
$\overline{w}(\pi(\theta))$	expected social welfare under current profile type θ
B_j	redundancy capacity of task j
$\boldsymbol{\gamma}$	eventual execution result
$c_{i,j}$	cost of viewer i in taking task j

denote the eventual task execution result as a vector $\boldsymbol{\gamma}$, where $\gamma_j \in \boldsymbol{\gamma}$ is 1 if the task j allocated to viewers is completed, 0 otherwise. The utility for viewer i follows commonly used quasi-linear utility form and is denoted as $u_i = \mathbf{E}\{\lambda(\theta_i, \boldsymbol{\gamma}) - c_i(f_i(\theta_i))\}$. Utility for the ingest server, acting as the auctioneer, is $\mathbf{E}\{\sum_{j \in M} V_j(\theta_i, \boldsymbol{\gamma}) - \sum_{i \in N} \lambda(\theta_i, \boldsymbol{\gamma})\}$. The valuation for a transcoding task j, V_j, can be defined as any valuable metrics that are important to service provider measured in currency, like possible revenue of this channel. The social welfare in turn is $\overline{w}(\pi(\theta)) = \mathbf{E}\{\sum_{j \in M} V_j(\theta, \boldsymbol{\gamma}) - \sum_{i \in N} c_i(f_i(\theta_i))\}$. Social welfare can be regarded as a generalization of common performance metrics, such as utilization, to a setting with utility-weighted tasks and operation costs.

A good mechanism is expected to satisfy social efficiency, individual rationality and incentive compatibility. We summarizes important notations in Table 5.1 and present the formal definition of these desirable properties in the following for clarification and proof purpose.

Definition 4. *A mechanism is ex-post incentive compatible, if for every bidder i and valuation type $\theta_i = (c_i^\tau, c_i^\tau)$, declaring their true type θ_i is the best response given all other players declare their true type θ_{-i}. Namely,*

$$u_i(\pi(c_i^\tau, c_{-i}^\tau, p_i^\tau, p_{-i}^\tau)) \geq u_i(\pi(\hat{c}_i^\tau, c_{-i}^\tau, \hat{p}_i^\tau, p_{-i}^\tau))$$

Definition 5. *A mechanism is ex-post individual rationality, if for every bidder i, any profile type $\theta_i = (c_i^\tau, c_i^\tau)$ and θ_{-i}, its expected utility is always non-negative if being selected. Namely,*

$$\mathbf{E}\{u_i(\pi(c_i^\tau, c_{-i}^\tau, p_i^\tau, p_{-i}^\tau))\} \geq 0$$

Definition 6. *A mechanism achieves ex-post social efficiency, if, given any profile types θ , its allocation policy f^* maximizes the sum of utilities of all bidders and auctioneer in the system in expectation. Namely,*

$$\mathbf{E}_{f^*}\{\overline{w}(\pi(\theta_i, \theta_{-i}))\} \geq \mathbf{E}_{f'}\{\overline{w}(\pi(\theta_i, \theta_{-i}))\}$$

5.2.2 Social Welfare Maximization for Cloud-Crowd Collaboration

In our studied cloud-crowd collaborative system, we aim at maximizing the system wide utility, social welfare, in expectation given the private cost and uncertainty for finishing different type of workloads. To help describe our formulation, we introduce a set of 0-1 variable $x_{i,j}^{\tau}$ for each pair of viewer and task. Variable $x_{i,j}^{\tau}$ equals one if viewer v_i is assigned for task j of type τ. B is maximum number of viewers allowed for each task j. We formally formulate our problem in the following:

$$\max \quad \mathbf{E}\{\sum_{j\in M} V_j(\theta,\boldsymbol{\gamma}) - \sum_{i\in N} c_i(f_i(\theta_i))\} \tag{5.1}$$

$$\text{s.t.} \quad \sum_{i} x_{i,j}^{\tau} \leq B_j, \forall j \in M, \forall \tau \tag{5.2}$$

$$x_i \in \{0,1\}, \forall i, B_j \in \mathbb{Z}_+^*, \forall j \tag{5.3}$$

Constraint (5.2) denotes that the number of viewers selected for a task j should be smaller than the required redundancy capacity; Previous works in cloud-based transcoding propose methods to estimate the transcoding resources needed by each transcoding task, and assign tasks to instances within their capability during the task assignment process[17]. Thanks to the abundant candidate pool from the crowd, we design our system to require each viewer only transcode one task of one type at one time to simplify the complex estimation process. If we choose to multiplex multiple tasks on a single viewer, we can easily relax this constraint to other values which does not affect the mechanism we proposed to get the optimal solution.

5.3 Mechanism Design for Cloud-Crowd Collaborative Transcoding

5.3.1 Transcoding the Broadcasting Workloads in C2

We first demonstrate how to design mechanisms to handle broadcasting workloads and apply the redundancy principle to improve social welfare further. For broadcasting workloads, each live session requires non-stop transcoding service with low latency, so that viewers can constantly receive their desired video formats. To guarantee this continuity in transcoding service, it is suboptimal to just select one viewer for transcoding one representation of a channel and start selecting another new viewer after the previous viewer abandons task abruptly. In contrast, having multiple viewers transcode one representation of a channel simultaneously can greatly improve social welfare and guarantee availability of the live session at all representations.

Following direct revelation principle, if we want to involve uncertain viewers into our transcoding process, explicitly asking for their probabilities of completing those transcoding

tasks seems to be the most straightforward approach. However, self-interested viewers may misreport their probability, claiming to be more competent than they really are, to win the rewards. Even worse, the classical auction mechanism, like VCG mechanism, cannot be directly applied to our scenario to guarantee truthfulness in bidding due to its deterministic nature. Naively adding probability into it can jeopardize the truthfulness of the whole mechanism [71]. In our game theoretic scenario, each viewer knows its own probability for finishing a task and the auctioneer knows the actual execution result of each task. Our objective hence is to design a probabilistic auction mechanism to guarantee desirable economic properties, like incentive compatibility in cost and likelihood for finishing the transcoding and maximize social welfare at the same time.

Our mechanism, presented in Algorithm 8, has two parts: the allocation mechanism and the payment determination mechanism. For the allocation part, we select viewers to transcode our tasks so that the expected social welfare of the whole system can be maximized. We need to solve our problem optimally, since any non-exact solution can harm the incentive compatibility and individual rationality of our mechanism as we will see in the proof. After observing that every scheduling decision binds with a quantifiable cost, we plan to adopt a graph-based declarative description to our problem. However, finding the appropriate graph structure to encode the constraints and our goals is not that straightforward. Bipartite matching with N and M on each side can describe one-to-one channel-to-task assignments, but fails to capture the redundancy design in our mechanism. Simply adding dummy task on one side cannot reflect different contribution to the social welfare due to different combinations of viewers-task pair. By adding another set of vertex other than source/sink, N, and M, we manage to represent our social welfare maximization problem into a min-cost flow problem.

We refer to the viewers that being selected for a task as the viewer group setting for this task. Given a set of viewers N, a set of transcoding task M, and the constraints defined in formulation, we construct a flow network $G = \{N \cup D \cup M \cup \{s, t\}, E\}$, where D represents possible combinations for all tasks $(v_1, v_2, ..., v_{|B|}), \forall v_i \in N$. The supply of s and the demand of t are all $|M|$ in value. The capacity of any edge between D and M is set to be 1. The cost of any edge between D and M is set to be $-\overline{u}_e = V_j(1 - \prod p^l_{i,j}) - \sum_i c_{i,j}, \forall j \in N, \forall i \in D$. Notice that normally a min-cost flow problem works when all costs are non-negative. We can transform edges with negative cost to edges with non-negative cost using edge reversal transformation techniques. After transforming to min-cost flow problem, based on the integral flow theorem [15], we know that our min-cost flow problem can be solved optimally. Equivalently, our social welfare maximization problem can be optimized, given the truthful telling from viewers.

Achieving incentive compatibility in the uncertainty of viewer behaviour thus becomes the key challenge of our whole mechanism. We solve this by binding payment for a viewer with the actual outcome of its assigned transcoding task, namely, whether the allocated

task for this viewer has been successfully transcoded or not. Specifically, the payment for a selected viewer i would be the expected social welfare of others (excluding the cost of i) minus the expected social welfare without the existence of viewer at all, $\overline{w}(\pi(c^l_{-i}, p^l_i, \theta_{-i})) - \overline{w}(\pi(\theta_{-i}))$, if viewer i finished this task. The first item in this payment formula excludes the cost of its own to avoid manipulation. The dependence of payment on real execution result further guarantees incentive compatibility. If the selected task is not successfully transcoded, our mechanism incurs a penalty for this viewer, which equals the expected social welfare without the existence of viewer, $-\overline{w}(\pi(\theta_{-i}))$. This penalty is necessary for us to prove the incentive compatibility in uncertainty. Since some tasks may fail to find desirable viewers and uncertain viewers may still leave the task during the transcoding process, we use cloud to guarantee a continuous transcoding process. Cloud thus is treated as a special bidder in our system whose cost is a public information.

Algorithm 8 The auction mechanism for C2

```
//The allocation mechanism
for all task j in M do
    Construct a flow network G = {N ∪ M ∪ D ∪ {s,t}, E}
    Select B_j viewers for each task j based on the solution of min-cost flow problem
    arg min_{j∈N}(∑ cost(e)), ∀e ∈ E
end for
//The payment mechanism:
while a live session ends or a channel reaches its publishing deadline do
    if γ_j = 1 then
        λ_i = w̄(π(c^l_{-i}, p^l_i, θ_{-i})) − w̄(π(θ_{-i})), ∀i ∈ {i|f_i = j}
    else
        λ_i = −w̄(π(θ_{-i})), ∀i ∈ {i|f_i = j}
    end if
end while
```

Before the formal proof, we use a small, simplified example to help understand how our mechanism works in one task scenario, even though our previous mechanism is designed to work under multiple tasks scenarios. Suppose we have one broadcasting task of value 10, two viewers, A and B, with type (2, 0.3) and (4, 0.2) respectively, where the second term is the probability of this viewer leaving the task within the assigned session. The redundancy level of task is 1. The optimal strategy thus is to choose viewer A with the expected social welfare 5 (social efficiency). The payment for viewer A is $10 - (0.8 \times 10 - 4) = 6$ if it succeeded, and -4 otherwise. Its expected utility becomes $0.7 \times 6 - 0.3 \times 4 - 2 = 0.1$, which is greater than 0 (individual rationality). Even if B lies about its probability as 0, claiming that it will stick to the transcoding to the end, it actually has a negative expected utility: $0.8 \times 5 - 0.2 \times 5 - 4 = -1$, which prevents it from doing that (incentive compatibility).

Theorem 8. *Our proposed auction mechanism guarantees social efficiency in expectation.*

This theorem is obvious since we solve our problem optimally.

Theorem 9. *Our proposed auction mechanism guarantees ex-post individual rationality.*

Proof. We start by again observing the expectation of utility of a viewer i, when declaring (c_i^l, p_i^l),

$$\overline{u}_i = \mathbf{E}\{\overline{\mathbf{w}}(\pi(c_{-i}^l, p_i^l, \theta_{-i}), \boldsymbol{\gamma}) - \overline{w}(\pi(\theta_{-i})) - c_i(f_i(\theta_i))\} =$$

$$\mathbf{E}\{\overline{\mathbf{w}}(\pi(\mathbf{c_i^l}, \mathbf{p_i^l}, \theta_{-\mathbf{i}}), \boldsymbol{\gamma})\} - \mathbf{E}\{\overline{w}(\pi(\theta_{-i}))\}.$$

Due to the optimality of our allocation mechanism, we have $\overline{\mathbf{w}}(\pi(c_i^l, p_i^l, \theta_{-i}), \boldsymbol{\gamma}) \geq \overline{w}(\pi(\theta_{-i}))$, which proves that the expected utility of a viewer is non negative. □

Theorem 10. *Our proposed auction mechanism guarantees ex-post incentive compatibility in expectation.*

Proof. Since we are targeting at ex-post incentive compatibility, assume other viewers all submit true types (c_{-i}^l, p_{-i}^l), we need to show that for viewer i, its expected utility when declaring (c_i^l, p_i^l) is not less than the utility when declaring $(\hat{c}_i^l, \hat{p}_i^l)$. The utility of i, when declaring (c_i^l, p_i^l), is $u_i = \mathbf{E}\{\lambda(\theta_i, \boldsymbol{\gamma}) - c_i(f_i(\theta_i))\} = \mathbf{E}\{\overline{\mathbf{w}}(\pi(\mathbf{c_{-i}^l}, \mathbf{p_i^l}, \theta_{-\mathbf{i}}), \boldsymbol{\gamma}) - \overline{\mathbf{w}}(\pi(\theta_{-\mathbf{i}})) - c_i(f_i(\theta_i))\}$. Social welfare component based only on the assignment of $\pi(\theta_{-i})$ is independent of i. We therefore focus on the rest, $\mathbf{E}\{\overline{w}(\pi(c_{-i}^l, p_i^l, \theta_{-i}, \boldsymbol{\gamma})) - c_i^l\}$ which just forms the expectation of the expected social welfare $\mathbf{E}\{\overline{w}(\pi(c_i^l, p_i^l, \theta_{-i}))\}$. Since social welfare depends on the eventual task execution result and our mechanism calculate the optimal result, we have $\mathbf{E}\{\overline{w}(\pi(c_i^l, p_i^l, \theta_{-i})) \geq \mathbf{E}\{\overline{w}(\pi(\hat{c}_i^l, \hat{p}_i^l, \theta_{-i}))\}$. That competes the proof. □

5.3.2 Transcoding the Pre-recorded Video Workloads in C2

We then demonstrate how our mechanism can handle pre-recorded video workloads with a different redundancy strategy. Unlike broadcasting videos, the uploaded short videos only need to be transcoded before a certain deadline. Specifically, we define the SLA on how quickly the uploaded videos need to be available in defined representations as the duration of a video/constant time [61]. The transcoding time of each video depends on various factors like its duration, complexity of scenes, computating power of viewers. Due to efficient performance isolation and mature virtualization techniques, most of previous works assume the performance of instances are homogeneous and stable. In contrast, viewers in C2 obviously have heterogenous computing power, not to mention that the competition with local workloads also makes the transcoding time even more unpredictable. This heterogeneous task execution time greatly challenges decision making in assigning pre-recorded video workloads. Most of previous works deterministically estimate the transcoding time to do further allocation, where methods range from simple linear model [59] to complex neural networks [40]. Considering the difficulty in deterministically estimating transcoding time in practice, Zhang et.al in [101] take empirical approaches that treat the transcoding time as a distribution. In this paper, we also assume the transcoding time on a viewer follows its

own distribution from a statistical perspective, given its device configurations, a video task and its targeted resolution.

Recall that $c^d_{i,j}$ denotes the cost of viewer i in taking the pre-recorded video task j and $p^d_{i,j}$ for the probability of completing its assigned task before the deadline. Facing this private information, our goal is to select viewers to complete the task before the deadline. Since each selected viewer may finish the task any time and transcoding pre-recorded videos allows interruptibility, it is suboptimal to select redundant viewers to transcode the same video concurrently in the beginning anymore. On the contrary, since a viewer may not finish the task before the deadline, it could help if we invoke another viewer when we think that the previous viewer will miss the deadline. In short, our problem becomes which viewers to select, and when redundant viewers can be selected to help finishing the task. The order of viewers for each task also matters now, we thus denote the permutation of all viewer grouping setting options as D', replacing the combination result D in the previous broadcasting workloads. For pre-recorded video tasks, the expected social welfare is

$$\overline{w}(\pi(\theta_i, \theta_{-i})) = \sum_{j \in N} (V_j(1 - \prod(1 - p^d_{i,j})) - \sum_{i \in D'} c_{i,j} \prod_{k=1}^{i-1} (1 - p^d_{k,j}))$$

In our scenario, if the distribution of transcoding time is public information and has the memoryless property, like the exponential distribution, we may derive a closed form solution for invocation time given viewer group settings. However, it turns out that distributions of transcoding time do not necessarily have this property, like the Gamma distribution found in [101]. Without having distribution with memoryless property or even without knowing the distribution information of viewers at all, calculating which viewers to choose and when to select redundant viewers is NP-hard, while solving it sub-optimally endangers our incentive compatibility. To guarantee our mechanism is still incentive compatibility, we design our auction to operate in discrete time slots, where the duration of one time slot is determined according to the duration of channels. In addition, due to limited bandwidth capacity, management complexity, and diminishing marginal gain by having extra viewers, each task j in our system is upper bounded by B_j. These two settings in turn reduce the feasible solution space. According to theoretical results in [70], it is then possible to run the optimal algorithm on this restricted solution space to achieve the incentive compatibility. Based on this, we can now find out the optimal viewer group setting and the time to invoke redundant viewers in polynomial time. Calculating payment for each selected viewer still follows the policy we defined in the previous section only with different calculations for expected social welfare.

Theorem 11. *After reducing the solution space through time discretion and bounding redundancy level, our proposed auction mechanism still guarantees ex-post incentive compatibility in expectation.*

Since the determination of time slot and redundancy level is independent of viewers' type, and our mechanism still finds out the optimal solution under this restricted solution space (range), namely, our mechanisms find the maximal solution in its range, according to [70], no viewer can increase its social welfare by misreporting. Our mechanism still guarantees ex-post incentive compatibility.

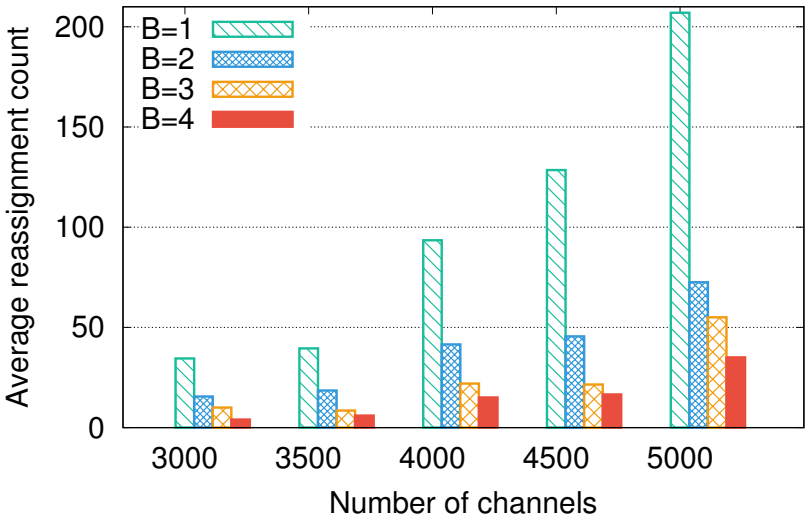
We will use another toy example to demonstrate how our mechanism work under pre-recorded video workloads. Suppose we have one transcoding task of value 10, lasting 40 mins. In a 1/2 SLA requirement, this task is expected to be transcoded within 20 mins after upload. The time slot duration in our system is set to be 10 mins. We also have three viewers, A, B, and C with type $(2, 0.3), (4, 0.8), (3, 0.6)$, where the probability in the second item reflects the probability of finishing the task within 10 mins. When we just greedily select one viewer with the largest expected social welfare to fulfil the task, viewer B with expected social welfare $10 \times 0.8 - 4 = 4$ is our choice. However, if we introduce one more redundant viewer, the optimal choice becomes either BC or CB, all with expected social welfare 4.6. Take BC as example, the expected social welfare of selecting these two is $10 \times (1 - 0.2 \times 0.4) - 4 - 3 \times 0.2 = 4.6$. This in turn gives us larger social efficiency. Notice that viewer C will be invoked only if viewer B failed to finish the task at the end of the first time slot.

The known worst-case complexity bound on the min-cost flow problem for a graph with E edges and V nodes is $\mathcal{O}\{ElogV(E+Vlog(V))\}$[73]. Since our system has far more viewers than broadcasting channels, the overall complexity for running the mechanism in broadcasting workloads is $\mathcal{O}\{N^B log(N)\}$. Similarly, for pre-recorded video workloads, covering at most T time slots, we have the overall complexity equal $\mathcal{O}\{|N|^B T^{B-1}\}$.

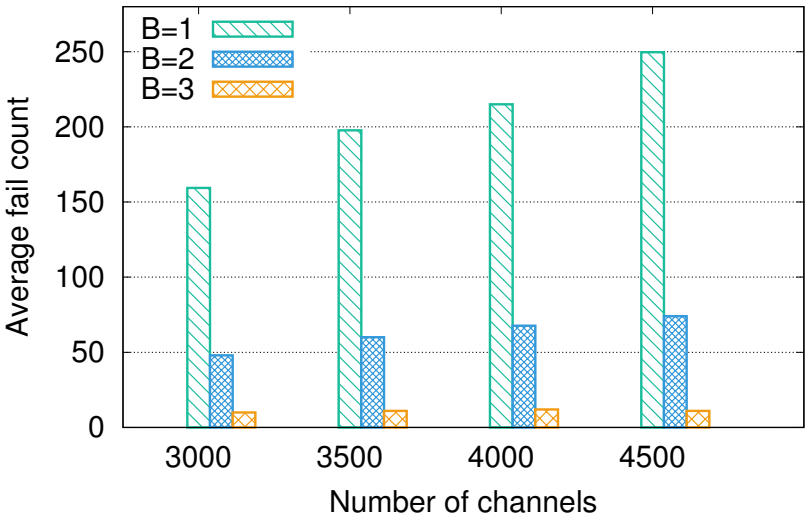
5.4 Evaluation

5.4.1 Dataset and Experiment Settings

We collected the channel/broadcaster trace through Twitch's public API from January 25 to February 27, 2015. This public API provides the game name, viewer number, and some other related information of every broadcast channel. In the mean time, we further captured the channel-based viewer trace through connecting to the Internet Relay Chat (IRC) interface offered by Twitch. This trace contains over 10 million join/leave records of viewers in certain channels in the same period. We use the price of the instance type, *m4.large*, as the cost for transcoding a task in the cloud. *m4.large* instance is also the default instance in the transcoding benchmark tests of major streaming hosting providers. Bids are generated randomly between 0 and this price. Viewers with the bid value higher than the cloud cost lose its competitive advantage with the cloud as the system can be return back to the existing cloud-based solution and we are looking for a cost-efficient solution. The value of a channel is calculated proportional to its accumulated popularity, number

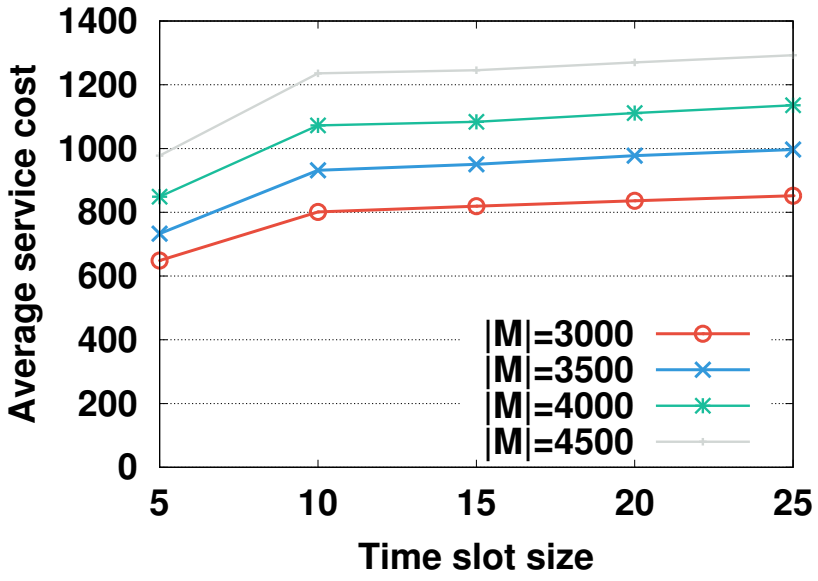


(a) Average reassignment count in broadcasting workloads

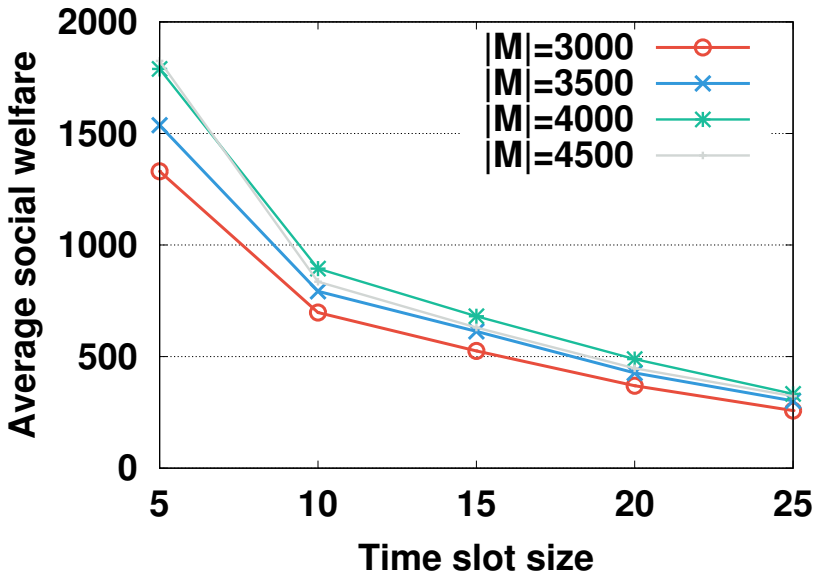


(b) Average fail count in pre-recorded video workloads

Figure 5.4: Impact of redundancy level (varying B)



(a) Average service cost at different channels



(b) Average social welfare at different channels

Figure 5.5: Impact of timeslot size

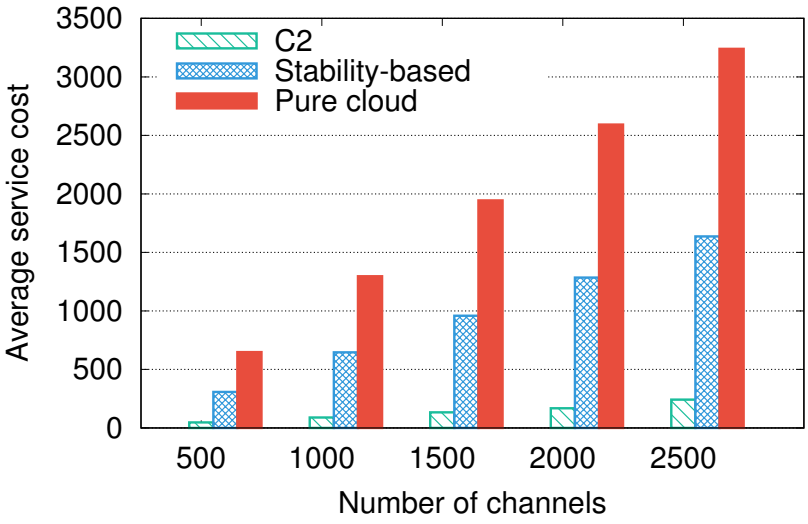
of viewers watching this channel over the entire time span. Default SLA on transcoding deadline in pre-recorded video workloads is defined to be 1, namely a 150-minute duration video uploaded at time t is expected to be available in $t + 150$ time.

Uncertain behaviour For broadcasting workloads, the probability for transcoding an entire live session is derived from the Pareto distribution with its parameters varying from 0.5 to 0.9 as observed in our trace. This varying likelihood for completing the task reflects the heterogenous behaviour of viewers in taking our tasks. For pre-recorded video workloads, we follow the same statistical model as in [101] where transcoding time of a specified file is determined by its size and a random variable $\mathbf{t}$ reflecting the transcoding time of a unit size segments. The distribution of $\mathbf{t}$ can be modelled by a Gamma distribution function in their study, with the shape parameter ranges from 4.868 to 5.209, and the scale parameter ranges from 0.101 to 0.145. Since not all viewers are capable of taking the transcoding tasks, we sample only 1% of viewers from our enormous viewer pool to simulate the eligible viewers [3].

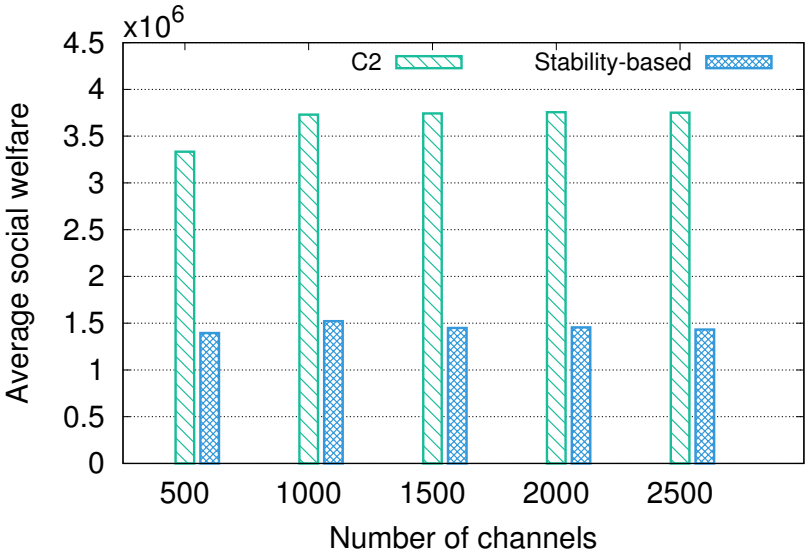
Baseline For broadcasting workloads, we compare our solution with a fully cloud-based approach and a stability-based method where viewers are selected greedily based on its average historical duration per cost [49]. Similarly, our baseline algorithm in pre-recorded video workloads is another greedy algorithm, greedy-efficiency, which selects viewers with the largest probability for finishing the task within the deadline per cost. We set a static price as thirty percent of the cloud counterpart to motivate viewers in the stability-based and greedy-efficiency approach.

Metrics To examine the performance of C2, we measure two important metrics: social welfare and total service cost for both two types of workloads. The first one is the ultimate optimization goal in our mechanism, reflecting the overall system efficiency. The second one is the overall cost for covering the transcoding demand of all channels. In addition, specifically for broadcasting workloads, we also measure reassignment count, which measures the number of times all transcoding viewers leave the assigned transcoding process. This reflects the stability of our system. Similarly, for pre-recorded video workloads, we measure the fail count, which denotes the number of times a channel missed the transcoding deadline. The reassignment count and the fail count can also be regarded as the metric for the quality of services for our transcoding service, as the reconnection due to reassigment in the broadcasting workloads could lead to livecast freezing and failing to transcode the video before the deadline in the pre-recorded video workloads violates the service-level-agreement promised by the service.

[3]According to the Steam and Twitch statistics, roughly 2% Twitch users have PCs with 4-core CPUs operating at 3.7GHz and above, and DirectX 12 GPUs. Sampling 1% of viewers here is a conservative choice to test the performance of C2.



(a) Average service cost



(b) Average social welfare

Figure 5.6: Performance comparison for broadcasting workloads (B = 2)

5.4.2 Sensitivity Analysis

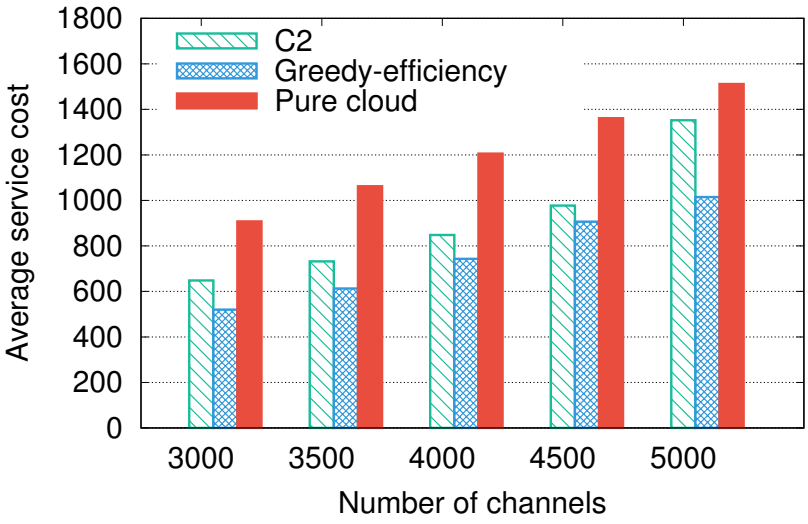
We first conduct sensitivity analysis to see the impact of time slot size and redundancy level on system performance, and then compare the performance of C2 with baseline approaches.

Impact of redundancy level Fig. 5.4 presents the impact of the redundancy level on different workload types. For broadcasting workloads in Fig. 5.4a, introducing one more redundant viewer can already reduce up to 65% reassignment than the baseline situation. This advantage sustains over 55% in the $B = 2$ cases. This gain diminishes as the redundant level increases. The average gain for increasing one more redundant viewers changes from 56% ($B = 1$ to $B = 2$) to around 25% ($B = 2$ to $B = 3$), and eventually drops to 5% ($B = 3$ to $B = 4$). For pre-recorded video workloads in Fig. 5.4b, introducing one more redundant viewer achieves around 70% less fail count reduction. The marginal benefit of introducing extra redundant viewers also decreases from 70% to 25% on average. In reality, small redundancy level like (B=2) may be suffice already considering the extra bandwidth cost involved for having one more redundant viewer per task. Different tasks with different values can be set with different redundancy level to guarantee the balance between cost and stability.

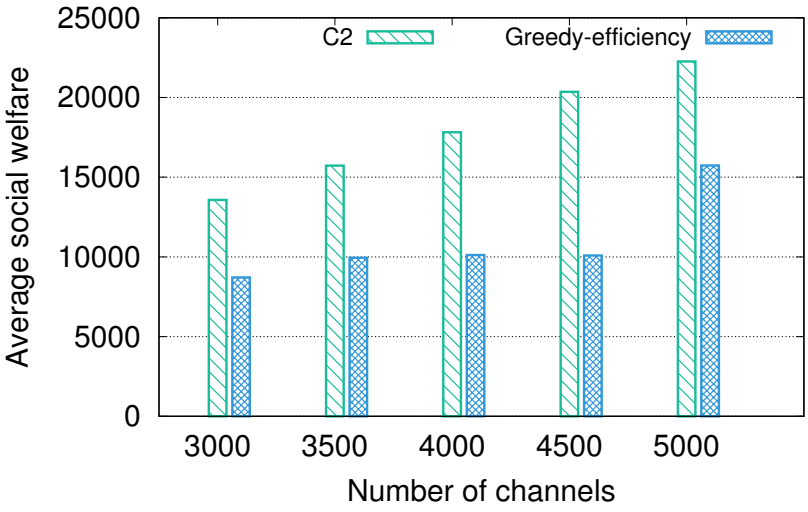
Impact of time slot size We next examine the impact of time slot size on service cost and social welfare in pre-recorded video workloads in Fig. 5.5. Service cost for transcoding the specified number of channels increases with the increase of time slot size. Correspondingly, social welfare of C2 decreases with the increase of time slot size. A smaller time slot means we can determine when to select redundant viewers in much finer granularity, which consequently leads to higher social welfare. Similar to the effect of redundant viewers, we also observe the decreasing of marginal performance gain in service cost and social welfare cases. On the other hand, setting time slot size too small will greatly increase the solution space, which could further increase the scheduling overhead. Therefore, it also suggests that sequential redundancy with coarse time slot granularity is suffice to guarantee the quality of service of the pre-recorded video workloads.

5.4.3 Comparison with Baseline Methods

Fig. 5.6a presents the service cost for transcoding specified numbers of channels in broadcasting workloads. C2's exploitation of normal viewers significantly reduces cost when compared with pure cloud implementation (e.g., by $\approx$ 93% compared to EC2 on-demand instance) and outperforms static pricing schemes used in stability approach (by $\approx$ 86%). This demonstrates the superiority of dynamic pricing in incentivizing viewers without introducing severe underpricing or overpricing problems. Fig. 5.6b demonstrates that C2 achieves 58% to 62% more social welfare than the stability-based approach. We omit the graphical presentation of reassignment count in these situations due to the excess advantage in C2. The reassignment count of the stability-based approach ranges from 220 times in 480 channels to 877



(a) Average service cost



(b) Average social welfare

Figure 5.7: Performance comparison for pre-recorded video workloads (B =2)

times in 2500 channels. On the other hand, C2 just experiences reassignment 8 times in the 2500 cases. It is because that C2 does not rely on average duration history to filter reliable viewers, instead it directly elicits true probability of finishing the task from viewers.

For pre-recorded video workloads, C2 achieves over 28% service cost reduction in most cases than the pure cloud approach except when channel number reaches 5000 in Fig. 5.7a. The average service cost of C2 usually is around 10% more than that of greedy-efficiency approach. Notably, C2 in this setting procures two viewers for each transcoding task. With a little more investment in recruiting redundant viewers, C2 achieves at least 41% more social welfare than the greedy-efficiency approach in Fig. 5.7b. The social welfare gain even reaches 100% more than the greedy-efficiency approach when channel number is 4500. Thanks to the efficient invocation of redundant viewers, C2 is capable of finishing much more transcoding tasks than the greedy-efficiency approach when comparing the average fail count. When the system has 2500 channels, C2 only misses the deadline for 30 times, as compared to 344 times in the greedy-efficiency approach. Overall, C2 keeps achieving 5 times less fail count than the greedy approach.

5.5 Related Works

Attracted by the elasticity in computing power and the “pay-as-you-go" billing model, cloud naturally becomes the choice for supporting video transcoding services [41, 17, 65]. Aparicio-Pardo et al. in [17] study the appropriate target representations for transcoding to maximize viewers' satisfaction. Chen et al. in [31] propose a generic cloud renting framework to minimize leasing cost through service migration. Gao et al. in [41] present a dynamic resource provisioning algorithm to minimize the transcoding cost based on task preemption. Li et al. in [63] propose a on-demand video transcoding system to save up the transcoding cost while maintaining a robust QoS for viewers.

Driven by the high cost in cloud transcoding, researchers have also studied novel architectures to help reduce transcoding cost [94, 61]. Krishnappa et al. in [61] outsource transcoding tasks to CDN and leverage online transcoding to improve user experience. The involvement of viewers brings unprecedented level of uncertainty and heterogeneity to computing resources which seldom appears in previous SLA guaranteed clouds. The altruistic assumption or naive fixed price incentive approaches also could not fully motivate users to take these computation-intensive tasks and truly reduce cost. In addition, peer transcoding in P2P systems shares some familiarity with our work in leveraging peer nodes to do transcoding [95]. However, these works mainly focus on how a tree structure can be constructed by peer information exchange in a distributed way. Our system still keeps a global control plane on task scheduling; essentially, each viewer still works as a powerful node to transcode alone. We focus on addressing the heterogeneity and uncertainty issues by using auction to do task allocation and incentive provisioning. What is more, cloud component is

indispensable to our system due to its critical role in keeping stability and availability for our applications. Our solution echoes with the emerging paradigm of edge computing [42], while we do not rely on extra deployment of dedicated edge servers. Our mechanism also offers insights to scheduling in other distributed computing infrastructures with heterogenious computing power and under the control of different entities.

Auctions have been widely used to solve the resource allocation problem and incentive issues in varies application scenarios, like crowdsourcing [39], spectrum allocation [33], etc. It strives to elicit the private information from players through bidding and determines the appropriate reward as incentive. Since most previous auction mechanisms treat user behaviour as a static status, they are no longer applicable to our scenario due to the stochastic viewer computing behavior. We instead take a statistical description towards this uncertainty and incorporate it into our bidding language. There are few research works on addressing incentive problems in video streaming context. We tailor our mechanisms to explicitly meet the requirements of our studied applications.

The most related work [49] studies a fully crowd transcoding approach. They select stable altruistic viewers based on the knowledge of viewer distribution. An extension of this work [108] incorporates auction to provide incentive, but they only apply the existing mechanism and neglect the uncertain behaviour of viewers after taking the tasks, which could lead to frequent reconnection and reassignment. We propose a cloud-crowd solution to better guarantee the practicality of the system and assume no knowledge in viewers' distribution. Novel auction mechanisms incorporating the redundancy principle are designed further to improve stability.

5.6 Summary

In this chapter, we presented a novel transcoding system, C2, with a hybrid architecture which combines the computing power of cloud and crowd together to accomplish transcoding tasks in the emerging crowd-interactive livecast systems. Facing the heterogeneity of viewers and the asymmetric information situation, we designed truthful auction mechanisms to select stable viewers for transcoding and tailor redundancy strategies for different types of workloads. We further proved theoretically that our proposed mechanism achieves social efficiency, individual rationality, and ex-post incentive compatibility. The trace-driven simulation demonstrated that our system achieves higher social welfare and lower service cost than the pure cloud solution.

Chapter 6

Conclusion, Lessons Learned, and Future Work

In this chapter, we conclude this thesis by summarizing its contribution in Section 6.1, reflect the lessons learned from the design and implementation of our solutions and discuss the possible future directions for cost-effective multimedia service provisioning in Section 6.2.

6.1 Summary of Contributions

In this thesis, we designed systems to offer cost-effective multimedia services, especially transcoding services, via exploiting the hidden compute power located in the hand of independent users. We reveal the potential in these users to improve resource utilization, cost-efficiency, and latency through examining the traces from the real-world applications. We apply an algorithmic, game-theoretic approach to jointly provide incentive and system performance. Specifically,

In cloud instance subletting, we examine the feasibility of subletting underutilized instances to other users with small demand, so that the resource utilization could be improved further. Our mechanisms provide the best possible competitive ratio in the static case and beat the state-of-the-art commercial solutions in the more practical dynamic case. We further prototyped such a service based on the emerging container techniques, demonstrating negligible performance loss in such a nested design.

In crowdtranscoding, we involve unstable viewers into the system to provide low latency transcoding services demanded by the emerging crowdsourced livecast services. Our proposed algorithms successfully select stable users without violating the budget constraint.

In cloud-edge collaborative transcoding, we further incorporate the uncertainties of viewers into the mechanism design process and mitigate the effect of such uncertainty on the systems by applying the redundancy principle. Our proposed probabilistic auction mecha-

nisms successfully improve the stability of a cloud-edge collaborative transcoding system. Eventually, it delivers a cost-effective stable transcoding service.

6.2 Lessons Learned and Future Works

We now first present several lessons learned over the process of conducting this research. We then present our thoughts on possible directions in the future.

No universal solution Our cloud subletting services and the collaborative transcoding services all demonstrate the great potential of dynamic pricing in cost reduction, resource utilization improvement in the cloud scenarios. Such a potential can only be unleashed by carefully designing the corresponding mechanisms to provide enough incentive and satisfy the context-dependent constraints simultaneously (*e.g.*, dynamic multiple multiple knapsacks in cloud subletting, uncertain viewers in the collaborative transcoding cases). The selected economic properties in our thesis are usually economic efficiency, individual rationality, and truthfulness. There are other important properties like collusion-free, fairness. We regard mechanisms that satisfy these extra properties as the future work.

Existing researchers usually solve resource allocation problems and pricing problems independently. Their pricing solutions mainly aim at maximizing the provider's revenue or profit given the knowledge of user's utility function. Their resource allocation solutions mainly focus on allocating resources fairly among multiple admitted users or allocate resource requests into the minimum number of servers to increase resource utilization. On the other hand, pricing has always been advocated by economists as a powerful tool to lead the market, influences demand and supply, and allocate resources more efficiently. In previous communication networks, services are governed by several monopolies, and the growth of demand is quite predictable. Price thus is usually determined based on potential competition, not real competition. In contrast, the convenience in accessing computing power through thes cloud has freed the demand of users; advanced communication technology enables users to signal their voice, more flexible pricing schemes become a possible option to confront the exponential growth in demand. Separation pricing from engineering scheduling problem gives up the opportunity to influence users' demand to achieve greater engineering goals like more efficiency in resource utilization, alleviating server burden, etc. Obviously, there is no panacea of mechanisms that can handle all incentive-involving problems. Identifying the unique characteristics in different scenarios, formulating them into mathematical languages, and solving them using algorithmic game theory, is challenging and interesting, and serve as the key contribution to this field.

Need for simplicity In the short term, we believe that what possibly could barrier novel pricing schemes from being deployed is its complexity in rules. A pricing scheme with desirable economic properties could have a bigger impact if it is also relatively simple and easy to understand to human beings in the current stage. For most users, they are more

accustomed to easy pricing schemes (e.g., fixed pricing), which are easy for users to finish cost management and planning. Though co-designing pricing with resource allocation has great potential in improving social welfare and resource utilization. Researchers in [72] have already argued that, in communication networks, when service costs decrease, providers will primarily consider improving usage, which usually requires simple pricing schemes. This motivation prevails over the need to operate a network at high utilization levels. This may still hold true for cloud computing.

The author is not indicating that flexible pricing (e.g., auction-based pricing) is useless in real life. In fact, we can easily find such systems in ride-sharing, ad selling for search engines, and even cloud computing (Amazon AWS). However, it may not be reasonable to assume that such a dynamic pricing scheme will replace the static ones. Rather, the author believes that novel pricing schemes like these would serve as the complement to the mainstream pricing schemes. Even further, with the decreasing barriers in signalling for users due to the increasing advances in communication techniques. Providers can access real, accurate feedback from users so that their preference can be revealed like in Uber case. With the advances in machine learning techniques, the software agent that automate this complex bidding or bargaining process may further make these complicated, yet effective, pricing schemes more prevalent.

Need for fairness Though auction demonstrates its great capability in efficiently allocating resources and determining the price. There is a concern that the allocation outcome determined by this utilitarian approach resulting in a large difference in pay-off among agents. If considering the repetition of auctions, when a subgroup of users keeps experiencing failure in each round, namely, starvation for some players, it may further affect their motivation in joining the further auction, which may degenerate the competitiveness of the environment. Therefore, fairness (envy-free) starts to attract researchers' attention even from an auction approach perspective.

For future work, the evolve of the cloud services and the form of the multimedia services all generate new opportunities and problems that calls for attention.

Practical cloud-edge systems for low-latency services The emerging interaction-rich or mission-critical applications, including AR, autonomous driving, etc., all require low-latency support from the computing and networking infrastructure. This direction could further be divided into two dimensions: cloud-side optimization and cloud-edge collaboration. In the first one, cloud itself is evolving from lower-level Infrastructure-as-a-service (e.g., virtual machines) to higher-level function-level services (e.g., serverless computing). Serverless computing is expected to dominant cloud computing services due to its better auto-scaling, platform flexibility, and fine-grained pricing schemes. The application of serverless computing to the emerging low-latency applications, like live video analytics services still needs further studying. One of the key challenges is that cloud providers provide serverless computing at a higher abstraction level, which makes performance modeling and prediction

more difficult. This further hinders the performance improvement for serverless computing to support low-latency services. In the second dimension, edge computing has been widely recognized as a promising solution to offer low-latency services. However, considering the limited computation capacity and the uncertain network environment, how to provide stable, efficient computing services to those computation-hungry, low-latency applications also needs further studying.

Data-driven mechanism design The convenient of signalling users' preference makes it convenient for researchers to collect users' preference. Based on data collected, it is possible for researchers to incorporate data driven stochastic model into their mechanism, and design Bayesian based mechanism where approximate truthfulness can be guaranteed. Furthermore, emerging applications present new requirement for resource provisioning. Among these, the temporal requirement is the most critical one [56]. How to design mechanisms to guarantee these new requirements is also interesting and challenging. Combinatorial auction in the context of cloud computing is still an open challenge. Besides, apart from auctions, more user-friendly mechanisms are still needed to offer high quality service. Recent works in auction incorporate the idea of learning approach to predict user's hidden valuation. Nazerzadeh et.al in [26] utilize machine learning to derive the whole valuation space. Combinatorial mechanisms are further adapted to guarantee efficiency. The profit maximization problem is modelled as a multi-armed bandits problem in [69]. Deng et.al in [37] discuss the trade-off between exploitation and exploration when learning to achieve revenue maximization.

System and networking for machine-centric video applications As machines now has the capability to process the data and make decisions on their own, previous protocols that target at improving the quality of service for human users cannot perfectly serve the requirements from these machine-centric systems anymore. For example, previous Dynamic Adaptive Streaming over HTTP (DASH) protocol aims at providing smooth video playback at the client side via adaptively streaming video chunks with maximum allowable bitrate under the constraint of varying network conditions. For live video analytics systems deployed in smart city scenario, the ultimate goal is to answer content-based queries (e.g., searching for a target for Amber Alert) with high accuracy, low cost, and low latency. Transmitting a video chunk from a camera with the maximum allowable bitrate may not be necessary if this video chunk is similar with previously transmitted video chunks from the same camera or similar chunks from other cameras in the proximity. This decision could reduce workloads, save more cost and reduce latency, without introducing too much accuracy loss. Therefore, algorithms and protocols need to be tailored further to serve the coming machine-centric intelligent system.

Bibliography

[1] Amazon ec2 pricing. https://aws.amazon.com/ec2/pricing/.

[2] Amazon EC2 Reserved Instance Marketplace. http://goo.gl/myvRvr.

[3] Creating your own EC2 spot market. http://techblog.netflix.com/2015/09/creating-your-own-ec2-spot-market.html.

[4] Docker. https://www.docker.com/.

[5] Google cluster data. http://googleresearch.blogspot.com/2010/01/google-cluster-data.html.

[6] Open Container Initiative. https://www.opencontainers.org/.

[7] Wowza Transcoder Performance Benchmark. https://www.wowza.com/docs/wowza-transcoder-performance-benchmark, 2015.

[8] The iPhone's new chip should worry Intel. http://www.theverge.com/2016/9/16/12939310/iphone-7-a10-fusion-processor-apple-intel-future, 2016.

[9] Uploads Open Beta: Help shape the future of Uploads on Twitch. https://blog.twitch.tv/uploads-open-beta-now-live-aec0e1925989?sf37518783=1, 2016.

[10] Steam Hardware Survey: March 2017. http://store.steampowered.com/hwsurvey, 2017.

[11] Cisco Visual Networking Index. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html, 2019.

[12] Vijay Kumar Adhikari, Yang Guo, Fang Hao, Matteo Varvello, Volker Hilt, Moritz Steiner, and Zhi-Li Zhang. Unreeling netflix: Understanding and improving multi-cdn movie delivery. In *IEEE INFOCOM*, 2012.

[13] Orna Agmon Ben-Yehuda, Muli Ben-Yehuda, Assaf Schuster, and Dan Tsafrir. Deconstructing amazon ec2 spot instance pricing. *ACM Transactions on Economics and Computation*, 1(3):16, 2013.

[14] Orna Agmon Ben-Yehuda, Muli Ben-Yehuda, Assaf Schuster, and Dan Tsafrir. The rise of raas: the resource-as-a-service cloud. *Commun. of the ACM*, 57(7), 2014.

[15] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. Network flows: theory, algorithms, and applications. 1993.

[16] Bo An, Victor R. Lesser, David E. Irwin, and Michael Zink. Automated negotiation with decommitment for dynamic resource allocation in cloud computing. In *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), Toronto, Canada, May 10-14, 2010, Volume 1-3*, pages 981–988, 2010.

[17] Ramon Aparicio-Pardo, Karine Pires, Alberto Blanc, and Gwendal Simon. Transcoding live adaptive video streams at a massive scale in the cloud. 2015.

[18] Yossi Azar, Inna Kalp-Shaltiel, Brendan Lucier, Ishai Menache, Joseph Seffi Naor, and Jonathan Yaniv. Truthful online scheduling with commitments. In *Proc. ACM EC 2015*.

[19] Moshe Babaioff, Liad Blumrosen, and Aaron Roth. Auctions with online supply. In *Proc. ACM EC*, pages 13–22, 2010.

[20] Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. Online auctions and generalized secretary problems. *ACM SIGecom Exchanges*, 7(2), 2008.

[21] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40(12), 2007.

[22] Azer Bestavros and Orran Krieger. Toward an open cloud marketplace: Vision and first steps. *IEEE Internet Comput.*, 18(1), 2014.

[23] Gideon Blocq, Yoram Bachrach, and Peter Key. The shared assignment game and applications to pricing in cloud computing. In *Proc. international conference on Autonomous agents and multi-agent systems*, pages 605–612. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[24] Zvi Bodie, Alex Kane, and Alan J Marcus. *Essentials of investments*. McGraw-Hill, 2013.

[25] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[26] Gianluca Brero, Benjamin Lubin, and Sven Seuken. Probably approximately efficient combinatorial auctions via machine learning. In *AAAI*, pages 397–405, 2017.

[27] Niv Buchbinder and Joseph Naor. Online primal-dual algorithms for covering and packing problems. In *European Symposium on Algorithms*, pages 689–701. Springer, 2005.

[28] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, omega, and kubernetes. *Commun. of the ACM*, 59(5), 2016.

[29] Shuchi Chawla, Nikhil R Devanur, Alexander E Holroyd, Anna R Karlin, James B Martin, and Balasubramanian Sivan. Stability of service under time-of-use pricing. In *Proc. ACM STOC 2017*.

[30] Shuchi Chawla, Jason D Hartline, David L Malec, and Balasubramanian Sivan. Multi-parameter mechanism design and sequential posted pricing. In *Proc. ACM STOC 2010*.

[31] Fei Chen, Cong Zhang, Feng Wang, and Jiangchuan Liu. Crowdsourced live streaming over the cloud. In *Proc. IEEE INFOCOM*, 2015.

[32] Liuhua Chen and Haiying Shen. Consolidating complementary vms with spatial/temporal-awareness in cloud datacenters. In *Proc. IEEE INFOCOM*, 2014.

[33] Yanjiao Chen, Jin Zhang, Kaishun Wu, and Qian Zhang. Tames: A truthful auction mechanism for heterogeneous spectrum allocation. In *Proc. IEEE INFOCOM*, pages 180–184, 2013.

[34] N. Chohan et al. See spot run: Using spot instances for mapreduce workflows. In *Proc. USENIX HotCloud*, 2010.

[35] Sharon Choy, Bernard Wong, Gwendal Simon, and Catherine Rosenberg. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In *Proc. IEEE NetGames*, page 2. IEEE Press, 2012.

[36] Deeparnab, Yunhong Zhou, and Rajan Lukose. Budget constrained bidding in keyword auctions and online knapsack problems. In *Proc. WWW*. 2007.

[37] Xiaotie Deng, Tao Xiao, and Keyu Zhu. Learn to play maximum revenue auction. *IEEE Transactions on Cloud Computing*, 2017.

[38] Ludwig Dierks and Sven Seuken. Cloud pricing: the spot market strikes back. In *The Workshop on Economics of Cloud Computing*, 2016.

[39] Zhenni Feng, Yanmin Zhu, Qian Zhang, Lionel M Ni, and Athanasios V Vasilakos. Trac: Truthful auction for location-aware collaborative sensing in mobile crowdsourcing. In *Proc. IEEE INFOCOM*, pages 1231–1239, 2014.

[40] Guanyu Gao, Han Hu, Yonggang Wen, and Cedric Westphal. Resource provisioning and profit maximization for transcoding in clouds: A two-timescale approach. *IEEE Trans. Multimedia*, 2016.

[41] Guanyu Gao, Yonggang Wen, and Cedric Westphal. Dynamic resource provisioning with qos guarantee for video transcoding in online video sharing service. In *Proc. ACM MM*, pages 868–877, 2016.

[42] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere. Edge-centric computing: Vision and challenges. *ACM SIGCOMM Computer Communication Review*, 45(5):37–42, 2015.

[43] Kristen Gardner, Samuel Zbarsky, Sherwin Doroudi, Mor Harchol-Balter, and Esa Hyytia. Reducing latency via redundant requests: Exact analysis. *ACM SIGMETRICS Performance Evaluation Review*, 43(1):347–360, 2015.

[44] A. Ghodsi et al. Dominant resource fairness: Fair allocation of multiple resource types. In *Proc. USENIX NSDI*, volume 11, pages 24–24, 2011.

[45] Gagan Goel, Vahab Mirrokni, and Renato Paes Leme. Clinching auctions with online supply. *Games and Economic Behavior*, 2015.

[46] Ajay Gopinathan, Zongpeng Li, and Chuan Wu. Strategyproof auctions for balancing social welfare and fairness in secondary spectrum markets. In *INFOCOM, 2011 Proceedings IEEE*, pages 3020–3028. IEEE, 2011.

[47] William A Hamilton, Oliver Garretson, and Andruid Kerne. Streaming on twitch: fostering participatory communities of play within live mixed media. In *ACM CHI*, 2014.

[48] Qiyun He, Cong Zhang, and Jiangchuan Liu. Utilizing massive viewers for video transcoding in crowdsourced live streaming. In *IEEE Cloud*, 2016.

[49] Qiyun He, Cong Zhang, and Jiangchuan Liu. Crowdtranscoding: Online video transcoding with massive viewers. *IEEE Transactions on Multimedia*, 19(6):1365–1375, 2017.

[50] Ashraf Anwer Hossain and Eui-Nam Huh. Refundable service through cloud brokerage. In *Proc. IEEE CLOUD*, 2013.

[51] Hao Hu, Yang Guo, and Yong Liu. Peer-to-peer streaming of layered video: Efficiency, fairness and incentive. *IEEE Transactions on Circuits and Systems for video Technology*, 21(8):1013–1026, 2011.

[52] Navendu Jain, Ishai Menache, Joseph Seffi Naor, and Jonathan Yaniv. Near-optimal scheduling mechanisms for deadline-sensitive jobs in large computing clusters. *ACM Transactions on Parallel Computing*, 2(1):3, 2015.

[53] Virajith Jalaparti, Ivan Bliznets, Srikanth Kandula, Brendan Lucier, and Ishai Menache. Dynamic pricing and traffic engineering for timely inter-datacenter transfers. *SIGCOMM*, 2016.

[54] Hai Jin, Xinhou Wang, Song Wu, Sheng Di, and Xuanhua Shi. Towards optimized fine-grained pricing of iaas cloud platform. *IEEE Trans. Cloud Comput.*, 3(4), 2015.

[55] Carlee Joe-Wong, Soumya Sen, Tian Lan, and Mung Chiang. Multiresource allocation: Fairness-efficiency tradeoffs in a unifying framework. *IEEE/ACM Transactions on Networking (TON)*, 21(6):1785–1798, 2013.

[56] Ian A Kash and Peter B Key. Pricing the cloud. *IEEE Internet Computing*, 20(1):36–43, 2016.

[57] Mehdi Kaytoue, Arlei Silva, Loïc Cerf, Wagner Meira Jr, and Chedy Raïssi. Watch me playing, I am a professional: a first study on video game live streaming. In *ACM WWW*, 2012.

[58] Hans Kellerer, Ulrich Pferschy, and David Pisinger. Knapsack problems. 2004.

[59] Seungcheol Ko, Seongsoo Park, and Hwansoo Han. Design analysis for real-time video transcoding on cloud systems. In *Proc. ACM SAC*, pages 1610–1615, 2013.

[60] S Shunmuga Krishnan and Ramesh K Sitaraman. Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. *IEEE/ACM Transactions on Networking*, 21(6):2001–2014, 2013.

[61] Dilip Kumar Krishnappa, Michael Zink, and Ramesh K Sitaraman. Optimizing the video transcoding workflow in content delivery networks. In *Proc. ACM MMSys*, pages 37–48, 2015.

[62] Baochun Li, Zhi Wang, Jiangchuan Liu, and Wenwu Zhu. Two decades of internet video streaming: A retrospective view. *ACM Trans. on Multimedia Comput., Commun., and Appl.*, 9(1s):33, 2013.

[63] Xiangbo Li, Mohsen Amini Salehi, Magdy Bayoumi, Nian-Feng Tzeng, and Rajkumar Buyya. Cost-efficient and robust on-demand video transcoding using heterogeneous cloud services. *IEEE Transactions on Parallel and Distributed Systems*, 29(3):556–571, 2018.

[64] Zhenyu Li, Gaogang Xie, Mohamed Ali Kaafar, and Kave Salamatian. User behavior characterization of a large-scale mobile live streaming system. In *Proc. ACM WWW*, pages 307–313, 2015.

[65] He Ma, Beomjoo Seo, and Roger Zimmermann. Dynamic scheduling on video transcoding for mpeg dash in the cloud environment. In *Proc. ACM MMSys*, 2014.

[66] Ming Ma, Lei Zhang, Jiangchuan Liu, Zhi Wang, Weihua Li, Guangling Hou, and Lifeng Sun. Characterizing user behaviors in mobile personal livecast. In *ACM NOSSDAV*, pages 43–48, 2017.

[67] Sabita Maharjan, Yan Zhang, and Stein Gjessing. Optimal incentive design for cloud-enabled multimedia crowdsourcing. *IEEE Trans. Multimedia*, 18(12):2470–2481, 2016.

[68] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282, 2017.

[69] Hamid Nazerzadeh, Renato Paes Leme, Afshin Rostamizadeh, and Umar Syed. Where to sell: Simulating auctions from learning algorithms. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, pages 597–598. ACM, 2016.

[70] Noam Nisan and Amir Ronen. Computationally feasible vcg mechanisms. *J. Artif. Intell. Res.(JAIR)*, 29:19–47, 2007.

[71] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani. *Algorithmic game theory*, volume 1. Cambridge University Press Cambridge, 2007.

[72] Andrew Odlyzko. Internet pricing and the history of communications. *Computer networks*, 36(5-6):493–517, 2001.

[73] James B Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations research*, 41(2):338–350, 1993.

[74] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1982.

[75] Karine Pires and Gwendal Simon. Dash in twitch: Adaptive bitrate streaming in live game streaming platforms. In *ACM VideoNext*, 2014.

[76] Xuanjia Qiu, Chuan Wu, Hongxing Li, Zongpeng Li, and Francis Lau. Federated private clouds via broker's marketplace: A stackelberg-game perspective. In *Proc. IEEE CLOUD*, 2014.

[77] Nihar B Shah, Kangwook Lee, and Kannan Ramchandran. When do redundant requests reduce latency? *IEEE Trans. Commu.*, 64(2):715–722, 2016.

[78] Bikash Sharma, Victor Chudnovsky, Joseph L Hellerstein, Rasekh Rifaat, and Chita R Das. Modeling and synthesizing task placement constraints in google compute clusters. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 3. ACM, 2011.

[79] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.

[80] Yaron Singer and Manas Mittal. Pricing mechanisms for crowdsourcing markets. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1157–1166. International World Wide Web Conferences Steering Committee, 2013.

[81] S Stein, EH Gerdinga, AC Rogersa, K Larson, and NR Jennings. Algorithms and mechanisms for procuring services with uncertain durations using redundancy. *Artificial Intelligence*, 175:2021–2060, 2011.

[82] Supreeth Subramanya, Tian Guo, Prateek Sharma, David Irwin, and Prashant Shenoy. Spoton: A batch computing service for the spot market. In *Proc. ACM Eurosys*, 2015.

[83] John C Tang, Gina Venolia, and Kori M Inkpen. Meerkat and periscope: I stream, you stream, apps stream for live streams. In *Proc. ACM CHI*, volume 16, pages 4770–4780, 2016.

[84] Twitch. Twitch retrospective 2015. `https://www.twitch.tv/year/2015`, 2015.

[85] Twitch. Twitch 2016 Retrospective. `https://www.twitch.tv/year/2016`, 2016.

[86] Luis M Vaquero and Luis Rodero-Merino. Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM Computer Communication Review*, 44(5):27–32, 2014.

[87] Hal R Varian. Position auctions. *international Journal of industrial Organization*, 25(6):1163–1178, 2007.

[88] Ashish Vulimiri, Philip Brighten Godfrey, Radhika Mittal, Justine Sherry, Sylvia Ratnasamy, and Scott Shenker. Low latency via redundancy. In *Proc. ACM CoNEXT*, pages 283–294, 2013.

[89] Bolun Wang, Xinyi Zhang, Gang Wang, Haitao Zheng, and Ben Y Zhao. Anatomy of a personalized livestreaming system. In *Proc. ACM IMC*, pages 485–498, 2016.

[90] Changjun Wang, Weidong Ma, Tao Qin, Xujin Chen, Xiaodong Hu, and Tie-Yan Liu. Selling reserved instances in cloud computing. In *IJCAI*, 2015.

[91] Haiyang Wang, Ryan Shea, Xiaoqiang Ma, Feng Wang, and Jiangchuan Liu. On design and performance of cloud-based distributed interactive applications. In *Proc. IEEE ICNP*, pages 37–46, 2014.

[92] Hongyi Wang, Qingfeng Jing, Bingsheng He, Zhengping Qian, and Lidong Zhou. Distributed systems meet economics: pricing in the cloud. 2010.

[93] Wei Wang, Di Niu, Baochun Li, and Ben Liang. Dynamic cloud resource reservation via cloud brokerage. In *Proc. IEEE ICDCS*, 2013.

[94] Zhi Wang, Lifeng Sun, Chuan Wu, Wenwu Zhu, and Shiqiang Yang. Joint online transcoding and geo-distributed delivery for dynamic adaptive streaming. In *Proc. IEEE INFOCOM*, 2014.

[95] Jui-Chieh Wu, Polly Huang, Jason J Yao, and Homer H Chen. A collaborative transcoding strategy for live broadcasting over peer-to-peer iptv networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(2):220–224, 2011.

[96] Xiaomeng Yi, Fangming Liu, Zongpeng Li, and Hai Jin. Flexible instance: Meeting deadlines of delay tolerant jobs in the cloud with dynamic pricing. In *Proc. IEEE ICDCS*, 2016.

[97] Xiaomeng Yi, Fangming Liu, Zongpeng Li, and Hai Jin. Flexible instance: Meeting deadlines of delay tolerant jobs in the cloud with dynamic pricing. In *Proc. IEEE ICDCS*, 2016.

[98] Cong Zhang and Jiangchuan Liu. On crowdsourced interactive live streaming: a twitch. tv-based measurement study. In *ACM NOSSDAV*, 2015.

[99] Hong Zhang, Bo Li, Hongbo Jiang, Fangming Liu, Athanasios V Vasilakos, and Jiangchuan Liu. A framework for truthful online auctions in cloud computing with heterogeneous user demands. In *Proc. IEEE INFOCOM*. 2013.

[100] Linquan Zhang, Zongpeng Li, and Chuan Wu. Dynamic resource provisioning in cloud computing: A randomized auction approach. In *Proc. IEEE INFOCOM*, 2014.

[101] Weiwen Zhang, Yonggang Wen, Jianfei Cai, and Dapeng Oliver Wu. Toward transcoding as a service in a multimedia cloud: Energy-efficient job-dispatching algorithm. *IEEE Trans. vehicular technology*, 63(5):2002–2012, 2014.

[102] Xiaoxi Zhang, Zhiyi Huang, Chuan Wu, Zongpeng Li, and Francis CM Lau. Online auctions in iaas clouds: Welfare and profit maximization with server costs. In *IEEE/ACM Transactions on Networking*, 2017.

[103] Xiaoxi Zhang, Chuan Wu, Zongpeng Li, and Francis C M Lau. A truthful (1-ε)-optimal mechanism for on-demand cloud resource provisioning. In *Proc. IEEE INFOCOM*, 2015.

[104] Liang Zheng, Carlee Joe-Wong, Chee Wei Tan, Mung Chiang, and Xinyu Wang. How to bid the cloud. In *Proc. ACM SIGCOMM*, 2015.

[105] Ruiting Zhou, Zongpeng Li, Chuan Wu, and Zhiyi Huang. An efficient cloud market mechanism for computing jobs with soft deadlines. *IEEE/ACM Trans. Netw.*, 25(2):793–805, 2017.

[106] Y. Zhu, S. Fu, J. Liu, and Y. Cui. Truthful online auction for cloud instance subletting. In *Proc. IEEE ICDCS 2017*.

[107] Yifei Zhu, Silvery D Fu, Jiangchuan Liu, and Yong Cui. Truthful online auction toward maximized instance utilization in the cloud. *IEEE/ACM Transactions on Networking*, 26(5):2132–2145, 2018.

[108] Yifei Zhu, Qiyun He, Jiangchuan Liu, Bo Li, and Yueming Hu. When crowd meets big video data: Cloud-edge collaborative transcoding for personal livecast. *IEEE Transactions on Network Science and Engineering*, 2018.

[109] Yifei Zhu, Jiangchuan Liu, Zhi Wang, and Cong Zhang. When cloud meets uncertain crowd: An auction approach for crowdsourced livecast transcoding. In *Proc. ACM MM*, pages 1372–1380, 2017.

Appendix A

Proof

A.1 Proof of Theorem 1

We present a polynomial-time reduction from the multidimensional 0-1 Knapsack Problem, an NP-hard problem [58]. The multidimensional Knapsack problem is defined as:

$$\max \sum_i x_i c_i \quad \text{s.t.} \sum_i a_{i,j} x_i \leq b_j, \forall j; \; x_i \in \{0,1\}. \tag{A.1}$$

Consider a special case of our problem. When we have only one instance and this instance stays in the market long enough to satisfy all time requirements, our problem becomes:

$$\max \sum_i x_i v_i \quad \text{s.t.} \sum_i d_i x_i \leq C_j, \forall j; \; x_i \in \{0,1\}. \tag{A.2}$$

Therefore, if there exists an algorithm that can solve our problem optimally and efficiently, we can also solve this Knapsack problem in the same way, which is, however, NP-hard.

A.2 Proof of Theorem 2

The unit price for bid i only depends on the past demand usage, and independent of current bid value itself. These are all generalized necessary conditions to ensure truthfulness [71]. Besides that, upon receiving each request, our selection process always maximize the utility for each bidder and accepts this bid if its utility is greater than zero as can be seen in our dual subproblem. Therefore, our mechanism falls into the family of sequential posted price mechanisms, where the auctioneer posts the price to a bidder and bidder responds accordingly to maximize its utility. Under such take-it-or-leave-it pricing scheme, a bidder can not improve its utility further after our generated maximized utility option by untruthful bidding [30]. Specifically, if the bidding price is higher than its true valuation, this bidder may receive a negative utility, which prohibits it from doing it. If the bidding price is lower than the true valuation, it may not pass the threshold, leading to possible failure in auction. Therefore, bidders have no incentive to misreport its bid value. Our mechanism in turn is

truthful. As for individual rationality, because we let $x_{i,j} = 1$ only if the utility for each bidder is non-negative, our pricing scheme thus naturally guarantees individual rationality.

A.3 Proof of Theorem 3

In the static supply scenario, our studied problem is a *multiple multi-dimensional knapsack problem*, where instances are the knapsacks. Given an input sequence η, we have our algorithm (OA) terminate filling $\mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3, ...\mathcal{Z}_m$ percent of M instances, respectively. Notice that each $\mathcal{Z}_i (i \in M)$ is comprised of $\{Z_i^r, r \in R\}$. We denote S be the set of requests picked by the algorithm, this corresponds to $S_1, S_2 ... S_n$ for each instance respectively. Similarly, we use S^* denote the request set selected by the optimum. Let $W_i^r = \sum_{j \in S_i \cap S^*} d_j^r$ be the common weight (resource demand) of requests j in type r in instance i, and $P_i = \sum_{j \in S_i \cap S^*} v_j$ be the common value of requests in instance i. According to our algorithm, if a request j is not admitted, then its value is less than $\sum_r \lambda(z_i^r) d_j^r \leq \sum_r \lambda(Z_i^r) d_j^r$, for all i. We further let $v(S_i \backslash S_i^*) = \sum_{j \in (S_i \backslash S_i^*)} v_j$ in instance i.

Due to the monotonicity of function λ, we have

$$\frac{OPT(\eta)}{OA(\eta)} \leq \frac{\sum_i (P_i + \sum_r \lambda(Z_i^r)(C_i^r - W_i^r))}{\sum_i (P_i + v(S \backslash S^*))} \tag{A.3}$$

Since we have for all i, $P_i \geq \sum_r \sum_{j \in S \cap S^\star} \lambda(z_{i,j}^r) d_j^r$. We thus have

$$\frac{OPT(\eta)}{OA(\eta)} \leq \frac{\sum_i (\sum_r \sum_{j \in S \cap S^\star} \lambda(z_{i,j}^r) d_j^r + \sum_r \lambda(Z_i^r)(C_i^r - W_i^r))}{\sum_i (\sum_r \sum_{j \in S \cap S^\star} \lambda(z_{i,j}^r) d_j^r + v(S \backslash S^*))} \tag{A.4}$$

The formular on the right hand side (RHS) is less than and equal to $\frac{\sum_i \sum_r \lambda(Z_i^r) C_i^r}{\sum_i \sum_r \sum_{j \in S_i} \lambda(z_i^r) d_j^r}$. Each item in the denominator $\sum_{j \in S_i} \lambda(z_{i,j}^r) \frac{d_j^r}{C_i^r} \approx \lambda(z_{i,j}^r) \triangle z_j^r = \frac{\lambda(Z_i^r)}{\ln(U^r/L^r)+1}$ via an integration. Therefore, we have the RHS in (A.4) less than and equal to $\ln(U/L) + 1$, where $U = \max U_r, \forall r \in R$. That completes our proof.

A.4 Proof of Theorem 4

Our studied MMKP problem is a generalization of online knapsack problem. Chakrabarty in [36] *et al.* already prove that the lower bound of any deterministic online algorithm for online knapsack problem is at least $\ln(U/L)+1$ based on Yao's minimax principle. If we have a online algorithm that can achieves a competitive ratio smaller than $\ln(U/L) + 1$ for our studied problem, we can also achieve the same lower bound for the online knapsack problem. By contradiction, we know that $\ln(U/L) + 1$ is also the lower bound for our studied MMKP problem. Thus the competitive ratio of our online algorithm is tight. That completes the proof.

A.5 Proof of Theorem 5

We prove this negative result by examining all possible decisions a deterministic mechanism can take towards the adversary cases when two sides of our markets all online. Given an arriving instance and an arriving request, if the capacity of this instance is allowed to take the request, any deterministic mechanisms have to decide if this request is allocated to the instance or not, a binary decision without probability. If the mechanism allocates this request to the instance, it could happen that the next arriving request with U/L times more value is rejected since no more instances arrive and capacity is not big enough to accommodate one more request. Similarly, if the mechanism decides not to allocate this request to the instance, it could happen that the first arriving request has the largest valuation (U/L times more) than the rest following requests, and no more instances arrive further. Therefore, any deterministic mechanism cannot handle these two adversarial situations simultaneously. This completes the proof. Randomized mechanisms or mechanisms with extra knowledge in supply/demand distributions may help further improve the situation. We plan to study these two directions in our future work.