Truthful Online Auction Toward Maximized Instance Utilization in the Cloud

Yifei Zhu, *Student Member, IEEE*, Silvery D. Fu, *Student Member, IEEE*, Jiangchuan Liu, *Fellow, IEEE*, and Yong Cui, *Member, IEEE*

***Abstract*— Although infrastructure as a service (IaaS) users are busy scaling up/out their cloud instances to meet the ever-increasing demands, the dynamics of their demands, as well as the coarse-grained billing options offered by leading cloud providers, have led to substantial instance underutilization in both temporal and spatial domains. This paper systematically examines an *instance subletting* service, where *sublettable instances* can be leased to others within predetermined periods when underutilized, from both theoretical and practical perspectives. The studied instance subletting service extends and complements the existing instance market of IaaS providers. We identify the unique challenges and opportunities in this new service, and design online auction mechanisms to make allocation and pricing decisions for the instances to be sublet. For static supplies of instances, our mechanism guarantees truthfulness and individual rationality with the best possible competitive ratio. We then incorporate a multi-stage discount strategy to gracefully handle dynamic supplies. Extensive trace-driven simulations show that our service achieves significant performance gains in both cost savings and social welfare. We further pinpoint the challenges in implementing such a service in the real-world system and validate our modeling assumptions through a container-based prototype implemented over Amazon EC2.**

***Index Terms*— Cloud computing, auction design, resource allocation, online algorithm, dynamic supply.**

I. Introduction

INFRASTRUCTURE as a service (IaaS) is one of the prominent service forms of current cloud services, where computing resources, such as CPUs and memory, are packed in the form of *virtual machines*, i.e. *instances*, and sold to users. As the fastest growing segment in the cloud ecosystem, the market of IaaS grew 31% in 2016 to total $22.1 billion [1]. Amazon EC2 as the leader in the IaaS market currently provides 57 types of pre-configured instances under 13 categories with pricing options ranging from prevalent fixed-priced on-demand instances to niche auction-based spot instances [2]. From the user side, it is estimated that 30% of applications will be migrated to the public cloud by the end of 2017, up from 14% in 2016 [3]. For both IaaS users and providers, the growing reliance on IaaS inevitably makes cost management their primary concern.

Although IaaS users are busy scaling up and out their cloud instances to meet their demands, the resource utilization of their instances is far from being efficient. This inefficiency, in turn, incurs extraneous expenditure for IaaS users. 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 [4], translating into almost $10 million unnecessary costs per year [2]. In general, instance underutilization can occur in both temporal and spatial domains [5]. In the former case, it occurs when IaaS 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 in previous Netflix case. The latter may happen when users are running workloads with heterogeneous demands on various resources. For instance, statistics show that the ratios of memory-to-CPU of tasks running in Google's data centers spread over more than three orders of magnitude [6]. CPU thus may become underutilized when running memory-bounded applications, and vice versa for memory.

Jointly solving instance underutilization in both domains is challenging. Existing studies have focused on dynamically provisioning cloud resources [7], [8], 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 [9] 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 idle instances to avoid extra charges. It unfortunately does little help to the underutilized instances, and discourages users from purchasing long-term instances with significant price discounts from the wholesale market of the cloud providers [10].

Recently, an *instance subletting* service has been suggested [11], [12], which allows underutilized instances to be sublet to others in need (in spatial and/or temporal domains).

Manuscript received May 25, 2017; revised December 2, 2017 and July 14, 2018; accepted July 16, 2018; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor A. Wierman. Date of publication September 10, 2018; date of current version October 15, 2018. This work was supported in part by the Industrial Canada Technology Demonstration Program Grant, in part by an NSERC Discovery Grant, and in part by an E. W. R. Steacie Memorial Fellowship. The work of Y. Cui was supported by the National Key Research and Development Program of China under Grant 2017YFB1010002. *(Corresponding author: Jiangchuan Liu.)*

Y. Zhu and J. Liu are with the School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada (e-mail: yza323@sfu.ca; jcliu@sfu.ca).

S. D. Fu was with Simon Fraser University, Burnaby, BC V5A 1S6, Canada. He is now with the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA 94720-1234 USA (e-mail: silvery@berkeley.edu).

Y. Cui is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: cuiyong@tsinghua.edu.cn).

Digital Object Identifier 10.1109/TNET.2018.2864726

If carefully designed, this service can make instance owners monetize their underutilized instances without introducing unnecessary downtime, and meanwhile make other users enjoy low-cost and high-quality computing resources. It can also help providers to gain profit by serving a wider range of potential users not yet covered by existing markets. Being a complement and extension to the current cloud market, it has great potentials towards building an efficient and sustainable cloud ecosystem.

Making instances sublettable however involves both practical and theoretical challenges. A core issue is to 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 market-based prices [13], [14]. Instance subletting, however, presents a series of new challenges that are yet to be addressed in the existing auction designs. Previous auctions make decisions either in the offline setting, assuming all bids are given at once, or only deals with the online arrival of bidders given a static known supply [8], [15], [16]. 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 this paper, we systematically examine the instance subletting service from both theoretical and practical perspectives. We start by revealing that the algorithmic side of our auction for subletting service 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 rationality simultaneously. It is then extended with a deadline-aware heuristic to handle dynamic supply. We include time constraints into service level agreements (SLA) of the subletting service to provide running time guarantee for buyers in this dynamic supply market. Pricing functions are also enhanced with a *multi-stage discount* strategy, agilely adapting to the elapse of time without sacrificing truthfulness and individual rationality. Large scale simulations driven by real traces demonstrate that our solution can achieve significant performance gains in social welfare and cost savings. We further discuss the practical issues in implementing the instance subletting service, including the platform API for resource bidding, resource isolation after request placement, and the resource management architecture under the dynamic supply. Accordingly, we design and implement an EC2 based prototype, which offers seamless and low-overhead operation thanks to the latest containerization techniques (Docker [17] in particular).

The remainder of this paper is organized as follows: Sec. II discusses related work; Sec. III presents the formal formulation of our problem; Sec. IV presents the design and analysis of our online auction mechanism in the static supply case, and Sec. V extends it to handle dynamic supply; extensive simulations are presented in Sec. VI, followed by discussions of practical challenges in Sec. VII; Sec. VIII concludes our work.

II. Background and Related Work

Most of the previous literature focuses on studying dynamic resource provisioning from a single cloud service provider's perspective [8], [14], [15]. As one of the early works, Zhang *et al.* [18] propose an online auction mechanism for allocating a single type of resource in the cloud. Zhang *et al.* [14] 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 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.

Existing IaaS providers also provide several types of low-cost instance options other than their regular instances[2] 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[3] 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.

[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.

[2]On-demand and reserved instances

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

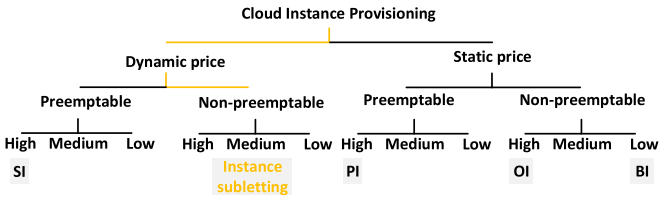


Fig. 1. Design space and the position of sublettable instance: Maximum achievable capacity $\in \{High, Medium, Low\}$. SI for EC2 spot instances, PI for GCE preemptable instances, OI for on-demand intances, and BI for GCE burstable instance.

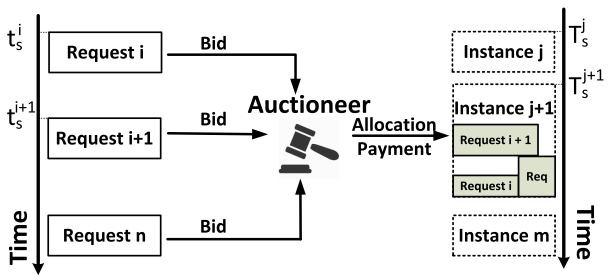


Fig. 2. Auction market in our instance subleting system.

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.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 in spot instance has been discovered to be not truly market-driven [20], 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 [21], [22]. Wang *et al.* [22] 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.* [23] 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.* [24] 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 [25], [26]. Some preliminary results on subletting services have been presented in [27] 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,[4] 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.

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

III. Instance Subletting: System Model and Problem Formulation

A. 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. In the latter case, they can also purchase sublettable instances purchased from the existing cloud markets, e.g., Amazon EC2 on a third-party platform. 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. In our one-sided auction, the instance subletting service provider acts as an auctioneer selling these M instances to N buyers through auctions. Fig.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

TABLE I
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

buyer i for having its intended instance as v_i. The utility 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. 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. I for clarity.

B. Auction in the Subletting Service: 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 and discuss its practical challenges in Sec. VII. 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. Social welfare is a commonly-used objective in resource allocation literatures and can be regarded as the generalization of utilization maximization to a setting with utility-weighted requests. 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:

$$\max \sum_{i \in N} \sum_{j \in M} x_{i,j} v_i \tag{1}$$

$$\text{s.t.} \sum_{j \in M} x_{i,j} \leq 1, \forall i \in N \tag{2}$$

$$\sum_{i \in N} d_i^r x_{i,j} \leq C_j^r, \forall r \in R, \forall j \in M \tag{3}$$

$$t_i x_{i,j} \leq T_{ddl}^j - t_s^i, \forall i \in N, \forall j \in M \tag{4}$$

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

Under truthful bidding, we have the objective value as $\sum_i \sum_j x_{i,j} b_i$. Constraint (2) indicates that each user request can only be allocated to one instance; Constraint (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 (4) indicates that the selected instance should have enough remaining time to guarantee the continuous running of its hosted requests.

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 [28] into an instance of this problem to prove that our problem is also NP-hard.

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 [29]. 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[5] mechanism in the following Sec. IV.

[5] α is the competitive ratio we will reveal in Sec. IV-B

IV. 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 [14], [18], secretary problems [29], and keyword auctions [30], 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. 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.

A. Mechanism Design

The primal-dual approach has been widely used to reveal the pricing attribute in a problem from the economical perspective [31]. 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). Since our Lagrangian relaxation problem is the upper bound of our original problem, we have the Lagrangian dual problem as: $\min P(\lambda)$ s.t. $\lambda \geq 0$. where $P(\lambda)$ is defined as:

$$\max \sum_{i\in N}\sum_{j\in M} x_{i,j}b_i + \sum_{r\in R}\sum_{j\in M}\lambda_{r,j}(C_j^r - \sum_{i\in N} d_i^r x_{i,j})$$

$$\text{s.t.} \sum_{j\in M} x_{i,j} \leq 1, \forall i \in N \tag{6}$$

$$x_{i,j} \in \{0,1\} \tag{7}$$

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

$$\max \sum_{j\in M} x_{i,j}(b_i - \sum_{r\in R}\lambda_{r,j}d_i^r) \tag{8}$$

$$\text{s.t.} \sum_{j\in M} x_{i,j} \leq 1 \tag{9}$$

$$x_{i,j} \in \{0,1\} \tag{10}$$

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:

$$\lambda(z_j^r) = (U^r e/L^r)^z (L^r/e). \tag{11}$$

Our problem in static supply is similar to the previous dynamic resource provisioning problem [32]. 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. 1. 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

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

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.

B. 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 [33]. The decision process also guarantees the individual rationality. Detailed proofs of all important theorems in the following can be found in the supplementary material .

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.

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).

V. 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 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.

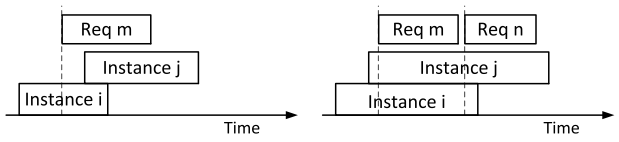


Fig. 3. Adversary situations because of the time constraint.

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., CPU computing time [34]–[36] or homogeneous instance[6] [37] in cloud environment. Jain *et al.* [34] study preemptive job scheduling problems under parallelization limits in an offline fashion. Azar *et al.* [35] study preemptive job scheduling problems under soft deadlines. Chawla *et al.* [36] study the resource scheduling problem in stochastic settings with the known distributions on the demand side. Wang *et al.* [37] 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.

A. Mechanism Design

Naturally, we need to incorporate time in our decision making process. To be specific, we plan to save those long-lived

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

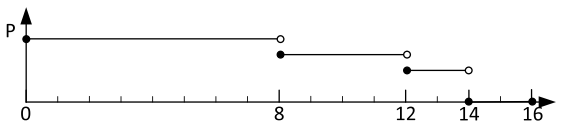


Fig. 4. Illustration of price discount and stage separation (initial price P, $T_j = 16, t_{min} = 2$, no requests are admitted).

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 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.

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. 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 [38], 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:

$$\lambda(z^r_i, t) = (U^r e / L^r)^z (L^r / e)(1 - D_i) \tag{12}$$

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.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.

Algorithm 2 OA Under Dynamic Supply

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

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.

VI. Performance Evaluation

A. Trace-Driven Simulations: The Case of Static Supply

1) Experimental Settings: Our evaluation uses Google Cluster trace [39] 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^r_i and a unit resource valuation

variable $\mathbf{v}^r$ randomly generated from 0 to U^r, namely,

$$b_i = \sum_r d_i^r \mathbf{v}^r, \mathbf{v}^r \in [0, U^r] \tag{13}$$

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 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 [2]. 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.

2) 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 [32] and [37]. Our solution is further compared with the optimal solutions obtained from an ILP solver.[7] Since the spot instance service actually has a proprietary pricing scheme, we adopt a straight forward implementation of its released approach.[8] 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.[9] 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 [2]. 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.* [32] 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.* [37] 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 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.

[7]PuLP: https://pythonhosted.org/PuLP/

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

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

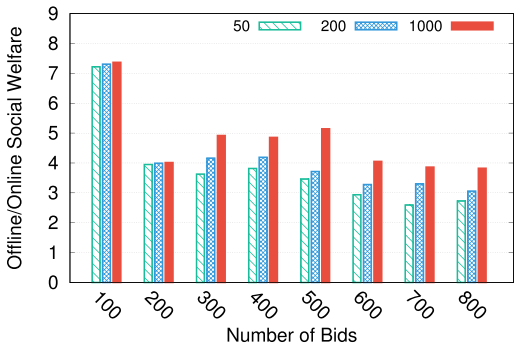


Fig. 5. Comparison with the optimal objective value under different bidding upper bounds (U).

Since works in [32] and [37] 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.VII, which have not been covered in those works.

3) Results and Discussion: Fig.5 and Fig.6 illustrate the results in the static supply case. First, we compare the performance of our online mechanism with the offline optimal solution in Fig.5. 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. (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.6a. 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.6a 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

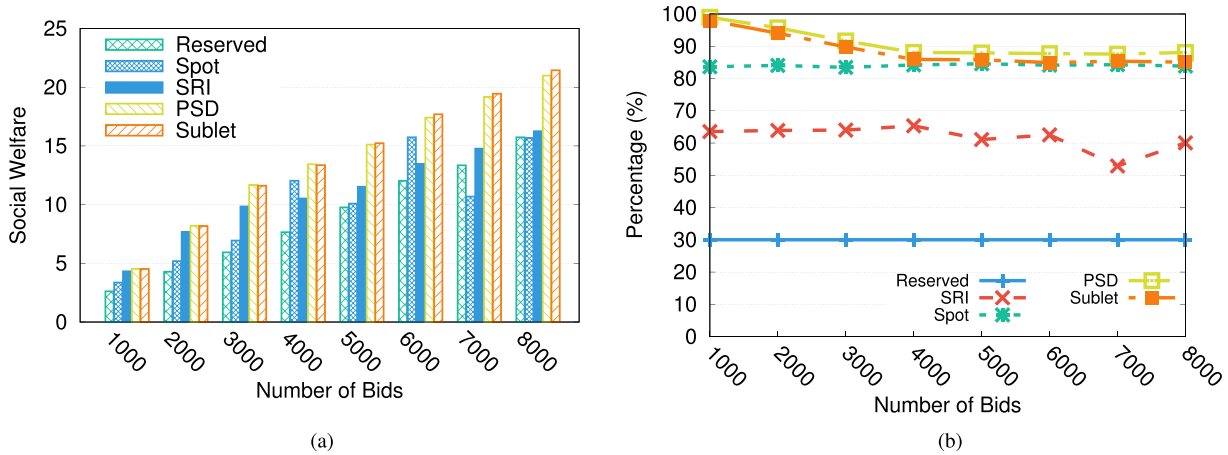


Fig. 6. Performance comparison in the static supply case. (a) Social welfare with varying bid numbers. (b) Cost saving with varying bid numbers.

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.[10] The instance subletting service 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. Dierks and Seuken [40] 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.[11] What is more, in a highly competitive public cloud market, social welfare would be preferred to guarantee user base.

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

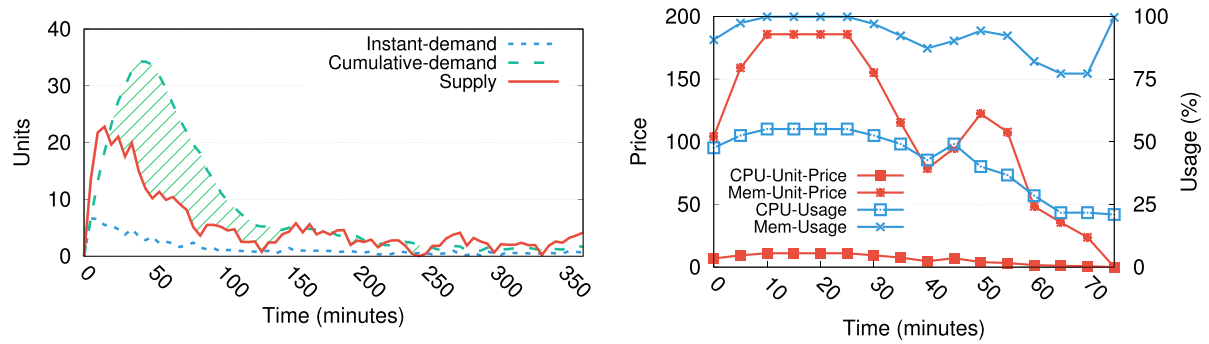
B. Trace-Driven Simulations: The Case of Dynamic Supply

1) 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 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.

2) Results and Discussion: The dynamic arrivals of both requests and instances greatly complicate our scheduling. In Fig.7a, 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 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.

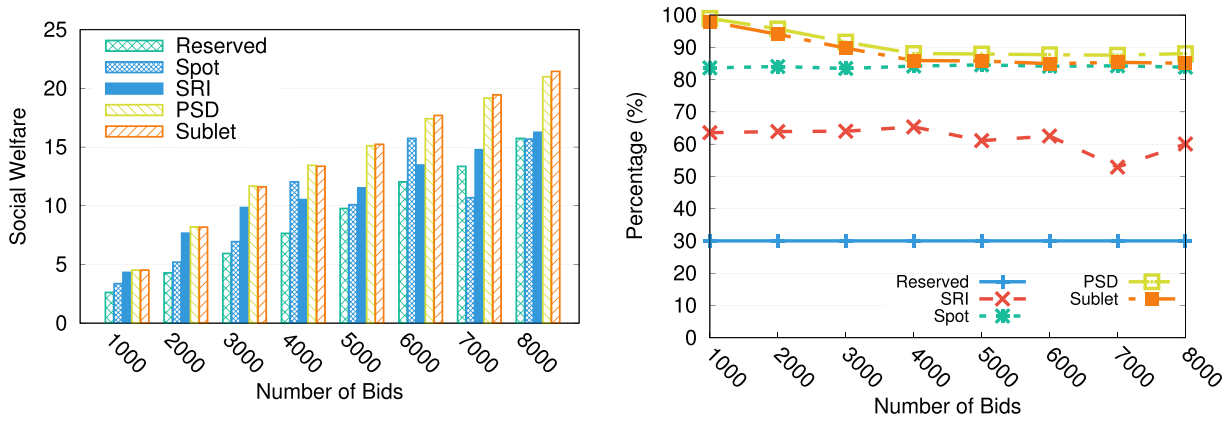
Fig.7b illustrates the interaction between prices and resources of a 75-minute-long instance in our instance subletting service. Its memory usage grows close to 100% between 5 minutes to 30 minutes. The unit price of memory, in turn,

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



(a) (b)

Fig. 7. Dynamic demand and supply in our market and a snapshot of an instance. (a) Overview of demand/supply on CPU. (b) Dynamics of price and usage ratio in an instance.



(a) (b)

Fig. 8. Performance comparison in the dynamic supply case. (a) Social welfare with varying bid numbers. (b) Cost saving with varying bid numbers.

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.8a), 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.8b, 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 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.[12] 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

[12]https://aws.amazon.com/ec2/spot/

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.

VII. 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 subletting service. Container is a lightweight, application-oriented virtualization technology that is becoming increasingly popular among public cloud providers [41].

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 [17] 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[13] 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.IV. 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 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.9a. 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.9b, the throughput performance also sees similar improvement with the increase of sublet instance. While adding more web servers to sustain high

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

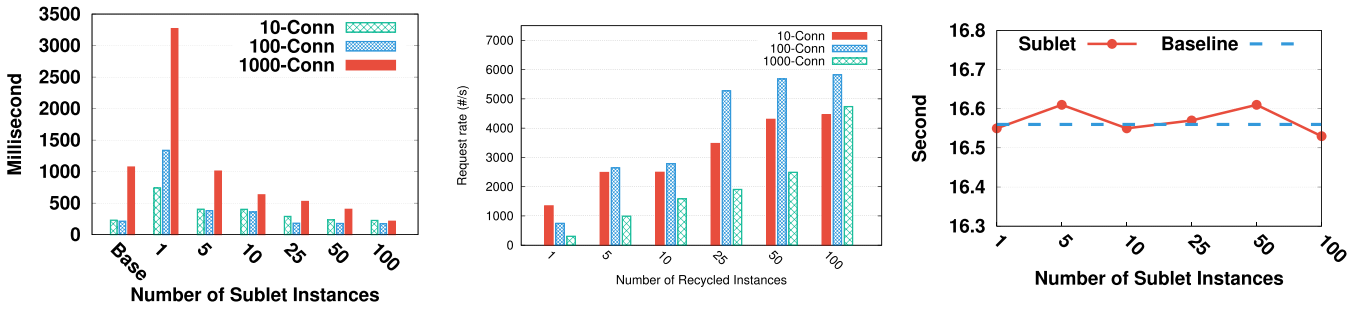


Fig. 9. Prototype performance on web and batch workloads. (a) Avg. web request completion time. (b) Avg. web request rate. (c) Avg. batch task completion time.

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.VI. 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.9a 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.9c, 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 [42], 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 [43]. 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.

VIII. Conclusion

In this paper, 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.

REFERENCES

[1] *Gartner Forecast: Public Cloud Service, Wordwide, 2013–2019*. Accessed: Aug. 21, 2018. [Online]. Available: http://www.gartner.com/newsroom/id/3188817

[2] *Amazon EC2 Pricing*. Accessed: Jun. 21, 2018. [Online]. Available: https://aws.amazon.com/ec2/pricing/

[3] (2016). *Morgan Stanley CIO Survey*. [Online]. Available: https://goo.gl/5NDrjK

[4] *Creating Your Own EC2 Spot Market*. Accessed: Aug. 21, 2018. [Online]. Available: http://techblog.netflix.com/2015/09/creating-your-own-ec2-spot-market.html

[5] L. Chen and H. Shen, "Consolidating complementary VMs with spatial/temporal-awareness in cloud datacenters," in *Proc. IEEE INFOCOM*, Apr./May 2014, pp. 1033–1041.

[6] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proc. ACM SoCC*, 2012, Art. no. 7.

[7] W. Shi, L. Zhang, C. Wu, Z. Li, and F. C. M. Lau, "An online auction framework for dynamic resource provisioning in cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 4, pp. 2060–2073, Aug. 2016.

[8] X. Zhang, C. Wu, Z. Li, and F. C. M. Lau, "A truthful (1-ε)-optimal mechanism for on-demand cloud resource provisioning," in *Proc. IEEE INFOCOM*, Apr./May 2015, pp. 1053–1061.

[9] H. Jin, X. Wang, S. Wu, S. Di, and X. Shi, "Towards optimized fine-grained pricing of IaaS cloud platform," *IEEE Trans. Cloud Comput.*, vol. 3, no. 4, pp. 436–448, Oct./Dec. 2015.

[10] *Amazon EC2 Reserved Instance Marketplace*. Accessed: Aug. 21, 2018. [Online]. Available: http://goo.gl/myvRvr

[11] A. Bestavros and O. Krieger, "Toward an open cloud marketplace: Vision and first steps," *IEEE Internet Comput.*, vol. 18, no. 1, pp. 72–77, Jan. 2014.

[12] O. A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, "The rise of RaaS: The resource-as-a-service cloud," *Commun. ACM*, vol. 57, no. 7, pp. 76–84, 2014.

[13] L. Zheng, C. Joe-Wong, C. W. Tan, M. Chiang, and X. Wang, "How to bid the cloud," in *Proc. ACM SIGCOMM*, 2015, pp. 71–84.

[14] X. Zhang, Z. Huang, C. Wu, Z. Li, and F. C. M. Lau, "Online auctions in IaaS clouds: Welfare and profit maximization with server costs," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 1034–1047, Apr. 2017.

[15] L. Zhang, Z. Li, and C. Wu, "Dynamic resource provisioning in cloud computing: A randomized auction approach," in *Proc. IEEE INFOCOM*, Apr./May 2014, pp. 433–441.

[16] G. Goel, V. Mirrokni, and R. P. Leme, "Clinching auctions with online supply," in *Proc. ACM-SIAM SODA*, 2013, pp. 605–619.

[17] *Docker*. [Online]. Available: https://www.docker.com/

[18] H. Zhang *et al.*, "A framework for truthful online auctions in cloud computing with heterogeneous user demands," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 1510–1518.

[19] M. Babaioff, L. Blumrosen, and A. Roth, "Auctions with online supply," in *Proc. ACM EC*, 2010, pp. 13–22.

[20] O. A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, "Deconstructing Amazon EC2 spot instance pricing," *ACM Trans. Econ. Comput.*, vol. 1, no. 3, 2013, Art. no. 16.

[21] A. A. Hossain and E.-N. Huh, "Refundable service through cloud brokerage," in *Proc. IEEE CLOUD*, Jun./Jul. 2013, pp. 972–973.

[22] W. Wang, D. Niu, B. Li, and B. Liang, "Dynamic cloud resource reservation via cloud brokerage," in *Proc. IEEE ICDCS*, Jul. 2013, pp. 400–409.

[23] X. Qiu, C. Wu, H. Li, Z. Li, and F. C. M. Lau, "Federated private clouds via broker's marketplace: A Stackelberg-game perspective," in *Proc. IEEE CLOUD*, 2014, pp. 296–303.

[24] X. Yi, F. Liu, Z. Li, and H. Jin, "Flexible instance: Meeting deadlines of delay tolerant jobs in the cloud with dynamic pricing," in *Proc. IEEE ICDCS*, Jun. 2016, pp. 415–424.

[25] N. Chohan *et al.*, "See spot run: Using spot instances for mapreduce workflows," in *Proc. USENIX HotCloud*, 2010, p. 7.

[26] S. Subramanya, T. Guo, P. Sharma, D. Irwin, and P. Shenoy, "SpotOn: A batch computing service for the spot market," in *Proc. ACM SoCC*, 2015, pp. 329–341.

[27] Y. Zhu, S. Fu, J. Liu, and Y. Cui, "Truthful online auction for cloud instance subletting," in *Proc. IEEE ICDCS*, Jun. 2017, pp. 2466–2471.

[28] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Berlin, Germany: Springer-Verlag, 2004.

[29] M. Babaioff, N. Immorlica, D. Kempe, and R. Kleinberg, "Online auctions and generalized secretary problems," *ACM SIGecom Exchanges*, vol. 7, no. 2, 2008, Art. no. 7.

[30] Y. Zhou, D. Chakrabarty, and R. Lukose, "Budget constrained bidding in keyword auctions and online knapsack problems," in *Proc. WWW*, 2007, pp. 1243–1244.

[31] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Chelmsford, MA, USA: Courier Corporation, 1982.

[32] R. Zhou, Z. Li, C. Wu, and Z. Huang, "An efficient cloud market mechanism for computing jobs with soft deadlines," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 793–805, Apr. 2017.

[33] S. Chawla, J. D. Hartline, D. L. Malec, and B. Sivan, "Multi-parameter mechanism design and sequential posted pricing," in *Proc. ACM STOC*, 2010, pp. 311–320.

[34] N. Jain, I. Menache, J. Naor, and J. Yaniv, "Near-optimal scheduling mechanisms for deadline-sensitive jobs in large computing clusters," *ACM Trans. Parallel Comput.*, vol. 2, no. 1, 2015, Art. no. 3.

[35] Y. Azar *et al.*, "Truthful online scheduling with commitments," in *Proc. ACM EC*, 2015, pp. 715–732.

[36] S. Chawla *et al.*, "Stability of service under time-of-use pricing," in *Proc. ACM STOC*, 2017, pp. 184–197.

[37] C. Wang *et al.*, "Selling reserved instances in cloud computing," in *Proc. IJCAI*, 2015, pp. 224–230.

[38] Z. Bodie, A. Kane, and A. J. Marcus, *Essentials of Investments*. New York, NY, USA: McGraw-Hill, 2013.

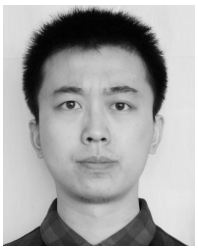
[39] *Google Cluster Data*. Accessed: Feb. 21, 2017. [Online]. Available: http://googleresearch.blogspot.com/2010/01/google-cluster-data.html

[40] L. Dierks and S. Seuken, "Cloud pricing: The spot market strikes back," in *Proc. Workshop Econ. Cloud Comput.*, 2016, pp. 1–28.

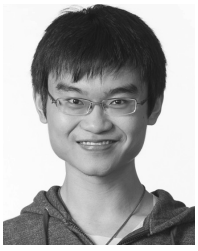
[41] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Commun. ACM*, vol. 59, no. 5, pp. 50–57, 2016.

[42] *Open Container Initiative*. Accessed: Aug. 21, 2018. [Online]. Available: https://www.opencontainers.org/

[43] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.



Yifei Zhu (S'15) received the B.E. degree from Xi'an Jiaotong University, Xian, China, in 2012, and the M.Phil. degree from The Hong Kong University of Science and Technology in 2015. He is currently pursuing the Ph.D. degree with the School of Computing Science, Simon Fraser University, British Columbia, Canada. His areas of interest are cloud computing, multimedia networking, Internet of Things, and crowdsourcing.



Silvery D. Fu (S'15) received the B.Sc. and B.Eng. degrees from the dual degree program of Zhejiang University, China, and Simon Fraser University, Canada, in 2016, and the M.Sc. degree from Simon Fraser University in 2017. He is currently pursuing the Ph.D. degree at the University of California, Berkeley. He is interested in system and networking research.

Jiangchuan Liu (S'01–M'03–SM'08–F'17) received the B.Eng. degree *(cum laude)* in computer science from Tsinghua University, Beijing, China, in 1999, and the Ph.D. degree in computer science from The Hong Kong University of Science and Technology in 2003. He is currently a Full Professor (with the University Professorship) with the School of Computing Science, Simon Fraser University, BC, Canada. He is also a Steering Committee Member of the IEEE TRANSACTIONS ON MOBILE COMPUTING. He is an IEEE Fellow and an NSERC E. W. R. Steacie Memorial Fellow. He was a co-recipient of the ACM Multimedia Best Paper Award (2012), the ACM TOMCCAP Nicolas D. Georganas Best Paper Award (2013), and the Test of Time Paper Award of IEEE INFOCOM (2015). He is an Associate Editor of the IEEE/ACM TRANSACTIONS ON NETWORKING, the IEEE TRANSACTIONS ON BIG DATA, and the IEEE TRANSACTIONS ON MULTIMEDIA.

Yong Cui received the B.E. and Ph.D. degrees in computer science and engineering from Tsinghua University, China, in 1999 and 2004, respectively. He is currently a Full Professor with the Computer Science Department, Tsinghua University. He has authored over 100 papers in refereed conferences and journals with several best paper awards. He has also co-authored seven Internet standard documents (RFC) for his proposal on IPv6 technologies. His major research interests include mobile cloud computing and network architecture. He is currently the Working Group Co-Chair of the IETF. He served or serves on the Editorial Boards for the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE TRANSACTIONS ON CLOUD COMPUTING, and the IEEE INTERNET COMPUTING.