Dynamic VM Scaling: Provisioning and Pricing through an Online Auction

Xiaoxi Zhang, *Student Member, IEEE,* Zhiyi Huang, *Member, IEEE,* Chuan Wu, *Senior Member, IEEE*
Zongpeng Li, *Senior Member, IEEE* and Francis C.M. Lau, *Senior Member, IEEE*

Abstract—Today's IaaS clouds allow dynamic scaling of VMs allocated to a user, according to real-time demand of the user. There are two types of scaling: horizontal scaling (scale-out) by allocating more VM instances to the user, and vertical scaling (scale-up) by boosting resources of VMs owned by the user. It has been a daunting issue how to efficiently allocate the resources on physical servers to meet the scaling demand of users on the go, which achieves the best server utilization and user utility. An accompanying critical challenge is how to effectively charge the incremental resources, such that the economic benefits of both the cloud provider and cloud users are guaranteed. There has been online auction design dealing with dynamic VM provisioning, where the resource bids are not related to each other, failing to handle VM scaling where later bids may rely on earlier bids of the same user. As the first in the literature, this paper designs an efficient, truthful online auction for resource provisioning and pricing in the practical cases of dynamic VM scaling, where: (i) users bid for customized VMs to use in future durations, and can bid again in the following time to increase resources, indicating both scale-up and scale-out options; (ii) the cloud provider packs the demanded VMs on heterogeneous servers for energy cost minimization on the go. We carefully design resource prices maintained for each type of resource on each server to achieve threshold-based online allocation and charging, as well as a novel competitive analysis technique based on submodularity of the offline objective, to show a good competitive ratio is achieved. The efficacy of the online auction is validated through solid theoretical analysis and trace-driven simulations.

Index Terms—Algorithm design, online auction, cloud computing

1 Introduction

Manual or auto scaling of resources has been widely supported in today's IaaS clouds. Most of the major cloud providers provide horizontal scaling (scale-out), to add more virtual machine (VM) instances to a user's application upon demand [1][2][3]. Some others enable vertical scaling (scale-up) by live addition of resources to VMs used by a user [4][5][6], using hot-plug technologies of CPU, memory, disk storage, etc. [7][8]. Scale-up can be potentially easier, but the resource increase cannot exceed the physical limit of a physical server; scale-out may involve data replication and be thus costly, but the resources that can be increased are potentially unlimited. The choice and preference of scaling modes are at user's call, depending on their respective resource need and program implementation.

Efficient timely allocation of resources on servers to accommodate time-varying demands of users is at the core of dynamic VM scaling. Practically, without knowing which VM is to be scaled up and which user will opt for scaling out, it is daunting to decide on which servers to place the VMs in the first place, even to just provide a somewhat good guarantee of resource availability for future scaling demands. The challenge escalates when we take server costs into consideration, striving to achieve high efficiency in power consumption and server utilization at the same time. An effective online solution is still missing, to optimize both user satisfaction and provider utility, *i.e.*, the social welfare.

An accompanying important challenge, economics wise, is how to price the incremental resources on the go, such that the benefits of both the cloud provider and users are guaranteed. Almost all the current IaaS offerings adopt fixed pricing [9], to charge a fixed unit price per preconfigured VM or per unit of resources, which does not change in the short term. There have been recent practices and proposals towards market-based pricing, for timely adaptation to demand-supply relation changes, that allocates resources to users who value the resources most and boosts both provider revenue and user utility. The Spot Instance market of Amazon EC2 is the pioneer production system adopting bidding-based dynamic VM pricing, but has been shown by studies not being a truly market-driven pricing system [10][11].

A number of online auction mechanisms have been proposed to achieve dynamic cloud resource allocation and pricing. They treat dynamically-arrival user demands as independent bids, ignoring possible connections among bids submitted at different times by a user. In the practical scenarios of VM scaling, a user may bid repeatedly after submitting his initial bid, to increase the resources needed, and hence later bids from the same user are related to earlier bids. With such time-coupling bids, the existing cloud auc-

- *X. Zhang, Z. Huang, C. Wu, and F. Lau are with the Department of Computer Science, The University of Hong Kong. E-mail: {xxzhang2, cwu, zhiyi, fcmlau}@cs.hku.hk*
- *Z. Li is with the Department of Computer Science, University of Calgary. E-mail:zongpeng@ucalgary.ca*

Manuscript received July 6, 2016; revised March 1, 2017. The research was supported in part by grants from Hong Kong RGC under the contracts HKU 717812, 718513, 17204715, 17225516, 27200214E, C7036-15G (CRF), a grant NSFC 61628209, a grant from Huawei Technologies Co. Ltd. (HO2016050002BE), a grant from the Natural Sciences and Engineering Research Council of Canada (NSERC), and a grant from Wedge Networks.

tion mechanisms are not applicable: our study has identified that the offline optimal resource allocation problem, whose solution any online decisions strive to approach, renders a completely different form from those studied in existing online cloud auctions. This brings significant difficulty in online mechanism design to approximate the offline optimum, and calls for new solutions.

Targeting market-driven, dynamic resource provisioning and pricing for VM scaling, this paper proposes a provenly efficient online auction mechanism. The following practical auction model is investigated: (i) Users bid for tailor-made VMs (with customized bundles of resources) to use in future durations, *e.g.*, based on empirical estimation/prediction of resource needs of their jobs; (ii) a user can bid again in the following time to increase resources, either before or after the start of his VM usage, when he learns better his resource need, and the bid indicates his preferences in scale-up or scale-out; (iii) the cloud provider packs the demanded VMs onto heterogeneous servers on the go, considering scaling preferences of users and striving for energy cost minimization on servers. Our online auction design aims to achieve truthfulness, individual rationality, computational efficiency, and competitiveness in social welfare, *i.e.*, a small ratio between the social welfare of the online mechanism over that of the offline optimum, computed assuming full knowledge over the system span.

We reveal the following key technical challenge in achieving such an efficient online mechanism. To guarantee competitiveness, the online allocation and pricing algorithm should typically be designed based on the structure of the offline resource allocation problem, and connections between the online decisions and increments of the offline social welfare should be established. To the best of our knowledge, the only well established technique for online auctions, where users bid for future resources with production (server) cost, is to exploit the online primal-dual framework [12][13], which requires that the corresponding offline optimization problem is convex with linear constraints. We identify that the offline optimization problem for VM scaling with time-coupling bids is non-standard, with a submodular objective function and non-linear constraints, such that none of the existing primal-dual frameworks is applicable.

Our contributions. We address the above challenge and design an efficient online auction as follows.

First, we construct an expressive bidding language to characterize different cases of dynamic demand scaling, which allows users to request and scale resources on different servers according to their preferences. We also uncover the underlying relation between users' online bidding behavior and the offline social welfare maximization problem. We identify an important property of the offline social welfare function, namely, submodularity, which plays a crucial role in our analysis.

Second, we design an efficient online auction mechanism based on carefully designed price functions. The prices for each unit of each resource on each server for each future time slot are maintained according to resource utilization on the server, and used to compute potential payments of users if their requested resources are allocated on the server. Such payments serve as thresholds to filter out low value bids, reserving resources for upcoming high value bids. Computed before taking in an accepted bid, such a payment is bid-independent, hence guaranteeing truthfulness of the mechanism.

Third, we design a novel competitive analysis technique which, based on submodularity of the objective function of the offline problem, obtains an upper bound of the offline optimum and shows a good competitive ratio achieved by our online mechanism. Currently, no competitive online algorithms exist for optimizing a submodular offline objective function under a complicated problem structure as in our case, *i.e.*,, complex non-linear constraints. Our online analysis is novel in the literature and may be useful for other online algorithm design problems with a submodular offline objective.

The rest of the paper is organized as follows. We discuss related work in Sec. 2, and define the auction model in Sec. 3. Sec. 4 presents our online auction design with analysis given in Sec. 4.2. We present simulation results in Sec. 5 and conclude the paper in Sec. 6.

2 Related Work

There have been a number of system work (*e.g.*, [14][15]) on resource scaling design in IaaS cloud, which exploit prediction of user demand for proactive resource scaling. They target at cost-effective practical schemes, without showing theoretical guarantee of the performance. Dynamic and efficient resource allocation is at the heart of resource scaling, and has been extensively investigated. For example, Alicherry *et al.* [16] study VM allocation in distributed cloud systems, taking into consideration the latency among the VMs. Joe-Wong *et al.* [17] seek to balance efficiency and fairness when allocating VMs to users. However, pricing of the allocated VMs is left out of the scope of these studies. Lin *et al.* [18] and Jiao *et al.* [19] design online algorithms for cloud resource provisioning considering switching cost of servers. They focus on minimizing total cost incurred while we aim to maximize social welfare which includes both valuation of user jobs and the operational cost of servers.

Towards market-driven pricing together with dynamic resource allocation, cloud auctions have been proposed in both offline/one-time settings [20][21] and online scenarios [22][23][24]. Most of the online auctions are built on different assumptions of the cloud system. Zhang *et al.* [22] consider a single type of cloud resource. Shi *et al.* [23] allow preemption of resources already occupied and paid by a user, which is typically not true in practice. In [24], users come and go over multiple rounds of the auction, while each user only occupies the allocated resources for one round. Moreover, none of these studies consider VM placement on physical servers and server cost minimization, which is in fact an important element in the social welfare that they aim to maximize. Therefore, these schemes cannot be easily extended to handle resource scaling, which relies heavily on whereabout of the VMs on different servers. A recent work [12] designs online auctions, where users bid for future resources and the cloud provider packs user-specified resource bundles onto heterogeneous servers on the fly, considering server cost in social welfare maximization and resource reusability. Unfortunately, the online primal-dual

framework for algorithm design that they applied is not applicable to an offline resource allocation problem with a non-convex objective function and non-linear constraints.

Recently, a sustainable body of theoretical literature has investigated submodular objective functions in both offline [25][26] and online [27][28] optimization. For example, Feldman *et al.* [28] relax the sectary problem to a model with a submodular objective, assuming elements arrive at a random order. Buchbinder *et al.* [27] study online algorithms for submodular function maximization in several models, allowing the algorithms to preempt previously accepted elements. We have identified that these models studied are structurally different from our model. Instead, we design novel competitive analysis techniques to prove the efficiency of our online mechanism.

3 Problem Model

3.1 Resource Scaling Auction

Consider an IaaS cloud with S servers. Let $[X]$ denote the set of integers $\{1, 2, \ldots, X\}$. A server $s \in [S]$ provisions K types of resources, *e.g.,* CPU, RAM and disks, with capacity C_{ks} of each resource $k \in [K]$. During a long span of T time slots, the cloud provider allocates the resources to N cloud users, who may come and go throughout the span, through an online auction. Each user as a bidder may demand a virtual machine (VM) of different resource composition from time to time, and use the requested VMs for variant lengths of time. The resource composition of each VM, as specified in a user bid, can vary over its usage span, according to time-varying need of the user's job on the VM. This can be enabled by dynamic addition and removal of resources to and from the VM, through "hotplug" technologies that adjust CPU cycles, memory and disks allocated to a running VM, supported in various virtualization environments [7][29].

Moreover, a user can (re)submit a bid later, to adjust his resource demands needed on the VMs requested earlier, according to the bidding results of previous VM requests (accepted or rejected), as well as his more precise estimation of future workload as the time draws near. In particular, if his previous bid fails, he can ask for the same VM again, or a VM with adjusted resource composition (increase or decrease) and/or usage span (expand or shrink); if the previous bid is successful, the user can ask for more resources either on an acquired VM (scale-up) or in the form of another VM on another server (scale-out). In the latter case, the new bid can be submitted either before or after an acquired VM has been launched: besides adding more resources in each time slot of VM usage, (i) if the VM is not yet running, the starting time of the VM can be advanced and the end time can be extended; (ii) if the VM has been launched, the end time can be postponed.

3.2 Bidding Language

We develop the following bidding language to model the above behavior of bidders. Let I_n be the total number of bids that a user $n \in [N]$ submits during the entire time span $[T]$, and B_{ni} denote the ith bid of user n, submitted in time slot t_{ni}.[1] Each bid can be expressed by

$$B_{ni} = (t^-_{ni}, t^+_{ni}, \{d_{nik}(t)\}_{k\in[K], t\in[t^-_{ni}, t^+_{ni}]}, \{b_{nis}\}_{s\in[S]}). \quad (1)$$

Here t^-_{ni} and t^+_{ni} represent the desired start time and end time of usage of the required resources, respectively. Naturally, we have $t_{ni} \le t^-_{ni} \le t^+_{ni}$. The third element in B_{ni} describes resources requested for each time slot during the usage interval, where $d_{nik}(t)$ is the demand of type-k resource in time slot t, set according to prediction of workload. If this is the very first bid of user n, or all previous bids of the user fail, $d_{nik}(t), \forall k \in [K]$, specify resource composition of the new VM at time t; if there exist successful bids before B_{ni} is submitted, $d_{nik}(t)$'s specify incremental resources to be added, either onto a VM that the user has successfully acquired, or in the form of a new VM.

User n indicates his choice or preferences of creating a new VM or adding resources onto an acquired VM using bidding prices $b_{nis}, \forall s \in [S]$. b_{nis} denotes the willingness-to-pay for the resources, if they are allocated on server s. If the bid is successful, resources are allocated on a server assigned with a previously-acquired VM of the same user, and that VM's running span overlaps with $[t^-_{ni}, t^+_{ni}]$, the incremental resources will be added to that VM (the scale-up case). If the bid is accepted on a server without such an already acquired VM, a new VM will be created and launched in due time (the scale-out case or deployment of the first-ever VM). A user can set b_{nis}'s according to concrete implementation of his program and detailed jobs running on the VMs. For example, if launching a new VM involves significant data replication, the user may prefer scale-up than scale-out, by setting $b_{ni\hat{s}}$ (where $\hat{s}$ is the server his previous VM is running on) larger than any other b_{nis}'s. If the user program is not implemented to exploit dynamically added CPU cores, but can smoothly handle distributed computation, the user may prefer scale-out and set a b_{nis} (where $s \neq \hat{s}$) larger than $b_{ni\hat{s}}$.

Note that "s" is used to differentiate bid prices for presentation clarity only: in practice, the cloud provider does not have to inform a user which actual servers his VMs are running on; the bidder can communicate with the provider by a message like "I am willing to pay *a* if my requested resources are added to the VM I acquired in time slot *b*, and I can pay *c* if they are put on another server". If the cloud provider can share more insider view of the data center network, a user can differentiate bid prices for provisioning his new VM on a server on the same rack (with one that runs an existing VM of his), on a different rack but sharing the same switch, etc.

We next give concrete examples to illustrate our bidding model. Suppose user 1 plans to run a MapReduce job. There are 3 servers. At time slot 1, he requests one VM with one unit of resource (considering one type of resource for simplicity) to run a Mapper during time slots 5 to 7, with the bid $B_{11} = (5, 7, \{1, 1, 1\}, \{1, 1, 1\})$, and requests a second VM to run a reducer during time slots 6 to 8 with $B_{12} = (6, 8, \{1, 1, 1\}, \{1, 1, 1\})$. He sets the same price of 1 in

1. We allow concurrent submission of multiple bids in the same time slot, i.e., $t_{ni} = t_{nj}$ for $j \neq i$, and randomly order simultaneous bids arriving at the same time.

both bids, no matter which server the VMs are provisioned on. At time slot 3, he realizes that the resource asked for the Mapper VM is insufficient. If B_{11} has been successful with the Mapper assigned to server 1, he can submit another bid $B_{13} = (5, 7, \{1, 2, 3\}, \{1, 0, 0\})$ to request adding 1, 2 and 3 units of resource to the same Mapper in its three running time slots, respectively, at the additional price of 1. Or he can submit $B_{13} = (5, 7, \{1, 2, 3\}, \{0, 1, 1\})$ instead, if he wants to run a new Mapper on either of the other two servers. If B_{11} has been rejected, he may resubmit a new bid in time slot 3 with boosted resources and prices, $B_{13} = (5, 7, \{2, 3, 4\}, \{2, 2, 2\})$. [2] If after the start of the Mapper acquired through B_{11}, at time slot 6, the user realizes that the resource for the Mapper is still insufficient, he may submit another bid $B_{14} = (7, 9, \{1, 5, 5\}, \{5, 0, 0\})$, to boost the VM resource to 5 from the next time slot (7) till time slot 9. An illustration of this example is shown in Fig. 1.

3.3 Goals of Online Auction Design

Upon receiving each bid B_{ni}, the cloud provider makes the following decisions on the spot. (i) **Resource provisioning**: $x_{nis}, \forall s \in [S]$, indicating whether resources requested in B_{ni} are to be provisioned on server s ($x_{nis} = 1$) or not ($x_{nis} = 0$). x_{nis}, being zero for all s implies rejection of B_{ni}. (ii) **Payment**: $\hat{p}_{ni}$, denoting the charge to user n if B_{ni} is accepted.

Our online mechanism design targets the following properties (which will be shown in Theorem 3): (1) *Truthfulness.* For any bidder, declaring his true information (start and end time of resource usage, resource needs) and true valuations of the resources in his bid always maximizes his utility (valuation minus payment), regardless of all the other bids. (2) *Individual rationality.* Each user always obtains a non-negative utility by submitting a bid in the auction. (3) *Computational efficiency*. Polynomial-time algorithms for resource allocation and payment calculation are needed for the auction, to run efficiently in an online fashion. (4) *Competitiveness in social welfare.* The social welfare over the system span $[T]$ is the sum of cloud provider's profit (aggregate payment of all winning bids minus total cost) and all users' utilities, which equals the aggregate user valuation minus the provider's cost. A cloud system operates at the maximal efficiency if social welfare is maximized over the running span, benefiting both the cloud provider and users. We aim to achieve a small competitive ratio, computed by dividing the offline optimal social welfare by the social welfare achieved by the online mechanism.

In computing cloud provider's profit, we consider the operational cost of servers, which is mainly due to the power cost decided by server power consumption. In practice, most cloud data centers keep their servers on, which remain in the low-power idle mode if no jobs are running, to avoid time-consuming booting up if switched completely off [30]. Decisions on turning servers on/off are usually made at a much larger time scale than those for VM allocation, *e.g.*, Amazon EC2 adjusts its server provisioning roughly once per month [12]. Therefore, we realistically assume that all S servers are turned on in the span T under our investigation. Each server consumes a basic amount of power with no VM running, and power usage increases with the increase of resource occupied on the server. Let $f_{ks}(\cdot)$ denote the cost function of server s on $y_{ks}(t)$, the amount of type-k resource used on the server at time slot t. We consider the following cost function:

$$f_{ks}(y_{ks}(t)) = \begin{cases} h_{ks} y_{ks}, & \text{if } y_{ks}(t) \in [0, C_{ks}] \\ +\infty, & y_{ks}(t) > C_{ks} \end{cases} \quad (2)$$

When the consumption of type-k resource, $y_{ks}(t)$, on server s does not exceed the capacity C_{ks}, the respective server cost increases linearly with the increase of resource usage; otherwise, the cost becomes infinity, which would never happen if the resource capacity constraint is respected. Such a linear cost has been shown through measurements: server power consumption increases roughly linearly with the utilization of CPU, memory, disk I/O and network I/O [31], if CPU DVFS (Dynamic Voltage Frequency Scaling) is not enabled.[3] h_{ks} indicates the relative weight of cost due to each type of resource in the overall server cost. It has also been shown that power consumption of memory, disk I/O and network I/O are significantly lower than that of the CPU, further ranked in a decreasing order among themselves [32], and the power usage due to different resources is additive [31]. We note that when $h_{ks} = 0$, the cost function becomes a zero-infinity function, which represents the special case of no server cost considered.

3.4 The Offline Social Welfare Maximization Problem

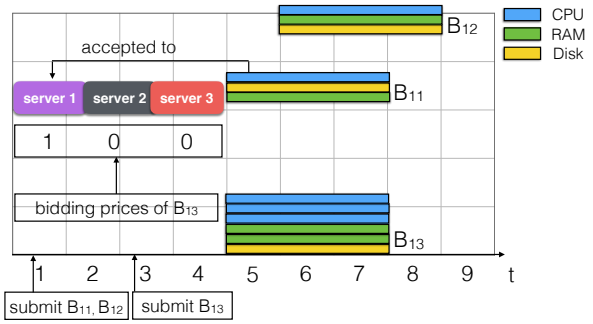
We next formulate the offline social welfare maximization problem, solving which provides the offline optimal decisions, assuming all user scaling demands in $[T]$ are known and truthful bidding is guaranteed. The offline optimum will serve as the basis, to compare our online algorithm with.

In participating in the auction, each user n has a value function $v_n : \{0, 1\}^{I_n \times S} \mapsto [0, +\infty)$. The user's overall value, for all I_n bids he submits, is denoted as $v_n(\mathbf{x}_n)$, which depends on $\mathbf{x}_n$, a feasible overall resource assignment scheme for user n in $[T]$. Here $\mathbf{x}_n = \{x_{nis}\}_{i \in [I_n], s \in S} \in \{0, 1\}^{I_n \times S}$, where $x_{nis} = 1$ if user n's bid i is accepted on server s and $x_{nis} = 0$ otherwise. The resource assignment is feasible if it satisfies $\sum_{s \in [S]} x_{nis} \leq 1$ for any $i \in [I_n]$, *i.e.*, any bid B_{ni} is accepted to at most one server.

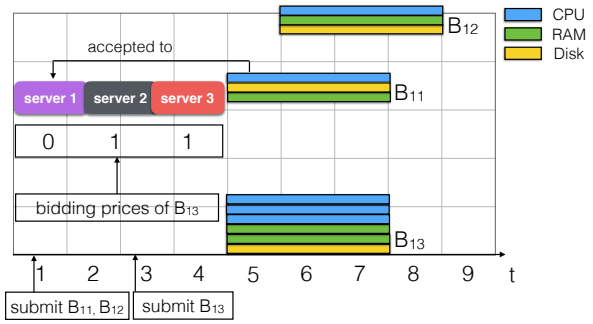
Let $D_{nik}(t)$ denote the overall demand of user n for type-k resource for future time slot t, which is his updated estimation of resource need for the future time slot when he submits his ith bid at t_{ni}. Note that this $D_{nik}(t)$ is the overall resource (type k) that user n wants to use in time slot t: some of the amount may have been allocated by accepting the user's earlier bids before B_{ni}, and some is requested through B_{ni}. The relationship between the asked resource $d_{nik}(t)$ in B_{ni} and this $D_{nik}(t)$ will be discussed soon after we formulate the offline problem, as follows.

2. Note that in this case the demands and valuations of B_{13} become larger since B_{13} has to include the demand and valuations of the rejected bid B_{11}. In this sense, for each user, previously accepted bids are fewer, the resource demands and valuations (or bidding prices) are larger.

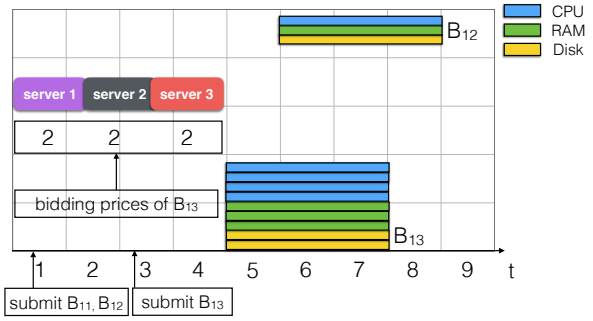
3. We leave the case of super-linear CPU cost function due to DVFS for future studies.



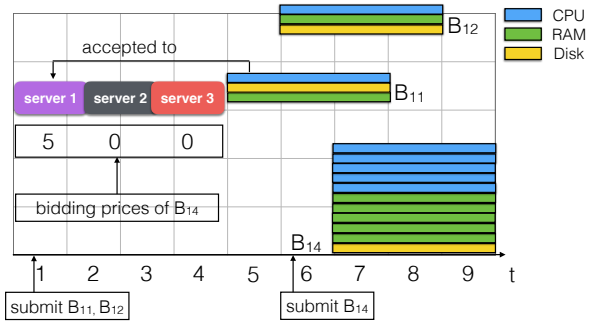
Case 1: B_{11} is accepted to server 1; B_{13} requires to scale up



Case 2: B_{11} is accepted to server 1; B_{13} requires to scale out



Case 3: B_{11} is rejected; B_{13} increases resource demands and bidding prices



Case 4: B_{11} has started running; B_{14} requires to scale up

Fig. 1: An Illustration for four example cases

$$\text{maximize:} \quad \sum_{n\in[N]} v_n(\mathbf{x}_n) - \sum_{t\in[T]}\sum_{s\in[S]}\sum_{k\in[K]} f_{ks}(y_{ks}(t)) \qquad (3)$$

subject to:

$$\sum_{s\in[S]} x_{nis} \le 1, \qquad \forall n\in[N], i\in[I_n] \ (3a)$$

$$\sum_{n\in[N]}\sum_{i\in[I_n]} x_{nis}(D_{nik}(t) - \max_{\substack{1\le j\le i-1:\\ t^-_{nj}\le t\le t^+_{nj}}} \{D_{njk}(t)\sum_{s\in[S]} x_{njs}\}) \le y_{ks}(t),$$
$$\forall k\in[K], s\in[S], t\in[T] \ (3b)$$

The offline problem aims to maximize social welfare over system span T, *i.e.*, the sum of the overall values of all users minus the total server cost over $t \in [T]$. We will discuss the connection between user's b_{nis}'s and his overall value v_n soon. Constraint (3a) ensures that each bid B_{ni} is accepted to at most one server. Constraint (3b) connects x_{nis}'s and $y_{ks}(t)$': $D_{nik}(t) - \max_{1\le j\le i-1:t^-_{nj}\le t\le t^+_{nj}}\{D_{njk}(t)\sum_{s\in[S]} x_{njs}\}$ is the additional amount of resource k allocated to user n for t, if B_{ni} is accepted; the maximization operation is to come up with the overall allocated resource to the user before B_{ni}. Hence summing up all such resource allocation for all accepted bids on server s, we obtain the total amount of type-k resource allocated on s at t, *i.e.*, $y_{ks}(t)$. Note that the capacity constraint, $y_{ks}(t) \le C_{ks}$, is implicitly satisfied by any feasible solution achieving non-trivial social welfare, due to our definition of the cost function in (2). Based on our best attempts, constraint (3b) cannot be formulated into an equivalent linear constraint, leading to significant difficulty in computing both the offline and online solutions. We next show the objective function is submodular, after giving relations between our offline formulation and online bids of the users.

Relation between online and offline problems. In the *online* auction, a bid of a user B_{ni} depends on his previous bids in two ways: (1) the amount of resources requested and, thus, the bid prices may depend on which of the previous bids are accepted (recalling the example in the last paragraph of Sec. 3.2) (2) the bid prices may also depend on the servers to which the previous accepted bids are assigned (if the user has a preference between scaling out and scaling up). As a reasonable assumption, the *online* bid prices can be formulated by the *offline* value function (assuming truthful bidding) which embodies the auction's decisions on previous bids. In particular, an *online* bid price b_{nis} is set as the additional value user n can obtain, if the bid B_{ni} is accepted to server s, as follows:

$$b_{nis} = v_n(\mathbf{x}_n^{(i-1)} \vee \mathbf{e}^{(is)}) - v_n(\mathbf{x}_n^{(i-1)}) \qquad (v_n(\mathbf{x}_n^0) = 0) \ . \qquad (4)$$

$v_n(\mathbf{x}_n^{(i-1)} \vee \mathbf{e}^{(is)})$ is the overall value of user n up to the time after submitting bid B_{ni} if the bid is accepted on server s, and $v_n(\mathbf{x}_n^{(i-1)})$ is the overall value of user n so far otherwise. Here $\mathbf{x}_n^{(i-1)}$ denotes user n's resource assignment vector by the online auction after handling its $(i-1)$th bid, *i.e.*, for

any $j < i$, $x_{njs'}^{(i-1)} = 1$ if the online auction accepted bid j on server s' and 0 otherwise, and for any $j \geq i$ and any s', $x_{njs'} = 0$. $\mathbf{e}^{(is)}$ is an $I_n \times S$ vector with $\mathbf{e}_{is}^{(is)} = 1$ and all other entries being zero, and $\mathbf{x} \vee \mathbf{x}'$ denotes a vector whose entries are equal to the max of the corresponding entries in $\mathbf{x}$ and $\mathbf{x}'$, i.e., $(\mathbf{x} \vee \mathbf{x}')_i = 1$ if $x_i = 1$ or $x'_i = 1$. So $\mathbf{x}_n^{(i-1)} \vee \mathbf{e}^{(is)}$ equals $\mathbf{x}_n^{(i)}$ with bid i accepted to server s. In addition, the amount of resource k for time $t \in [t_{ni}^-, t_{ni}^+]$ that user n requests in his bid B_{ni}, should be the incremental amount to meet his total resource demand $D_{nik}(t)$, depending on whether his previous bids are accepted or not:

$$d_{nik}(t) = D_{nik}(t) - \max_{1 \leq j \leq i-1 : t_{nj}^- \leq t \leq t_{nj}^+} \{D_{njk}(t) \sum_{s \in [S]} x_{njs}\} \quad (D_{n0k}(t) = 0). \quad (5)$$

To give a better understanding of the relation of online and offline problem, we introduce the "pure" value part of the value function $v(\cdot)$. The other part will be defined later to handle the user's preference for scaling up/out. The "pure" value function represents the value of resources required by the bid, without considering scaling-up/out. Practically, a larger amount of any type of resource leads to a higher "pure" value. In the offline setting, each total demand $D_{nik}(t)$ and the corresponding total "pure" value of each bid is fixed. They represent the input that can be known in advance by the offline problem. Suppose in an extreme case, user n has won all his previous bids, then each demand of each bid i will be $D_{nik}(t) - D_{n(i-1)k}(t)$ ($D_{n0k}(t) = 0$). However, in many other cases, we can not say the online submitted demand $d_{nik}(t)$ is equal to $D_{nik}(t) - D_{n(i-1)k}(t)$. *For any of the rejected bids, the user will require the demands again, adding the demands to the the demands of the next bid.* That is the fundamental problem which makes the online input (both online demands and the corresponding bid prices) relying on the previous auction solutions. Finally, we relate online $d_{nik}(t)$ to offline $D_{nik}(t)$ by (5). For the offline value function $v(\cdot)$, we do not need to restrict the specific function of the "pure" value part. The basic idea is that: *For any user, the online demands hence the online "pure" value of a later bid are* **larger** *if his previously accepted bids are* **fewer**. Since the marginal increase of the offline "pure" value equals to the online "pure" value, the above basic idea directly proves that offline "pure" value function is submodular according to the definition (see Definition 1. in the following). It is also one of the fundamental ideas through the proof of submodularity in Lemma 1 and Theorem 1. Since the submodularity is a general property that a "pure" value function possesses, the mechanism with the competitive analysis works for other studies where value functions could be more arbitrary than this work if they do not consider the user's preference for scaling up or out and the server cost.

The value function. We consider the value v_n of user n consists of two parts: a monotone "pure" value function $u_n(\mathbf{x}_n)$ for the resources allocated to him, minus a cost function $\Phi_n(\mathbf{x}_n)$ specifying the decreased value due to scaling up/out. The "pure" value $u_n(\mathbf{x}_n)$ depends on acceptance of user n's bids. After the ith bid, user n may have a unit value δ_{nik} for each unit of resource k in interval $[t_{ni}^-, t_{ni}^+]$ if his ith bid is accepted. His "pure" value u_n is simply the sum of the unit values of the resources obtained from the accepted bids in the entire span [T]. More correctly, readers may think of the "pure" value function to be the one in the above example throughout the rest of the paper. Nonetheless, note that a much richer family of "pure" value functions satisfy submodularity. A user n may be subjected to a cost when his bids i and j are accepted on the same server, or on different servers. Let $\phi_{n(ij)}^{up}$ denote the scale-up cost of user n when i and j are accepted on the same server, for all $i < j$, which may represent the delay of resource hotplug. Let $\phi_{n(ij)}^{out}$ denote his scale-out cost if i and j are accepted on different servers, *e.g.*, data replication and communication cost among the VMs. The total cost of user n in $[T]$ due to scaling up/out between accepted bids can be computed as

$$\Phi_n(\mathbf{x}_n) = \sum_{i<j} \sum_{s \in [S]} \phi_{n(ij)}^{up} x_{nis} x_{njs} + \sum_{i<j} \sum_{s \neq s'} \phi_{n(ij)}^{out} x_{nis} x_{njs'} \quad (6)$$

Note that $\phi_{n(ij)}^{up}$ and $\phi_{n(ij)}^{out}$ are only introduced for defining the value function v_n. A user n does not need to specify them in his bids. Since the decisions for previous bids and the costs are known to the user, he just submits bidding prices b_{nis} calculated according to (4).

Assumption 1 (rational unit value). The unit value δ_{nik} of user n for each unit of resource k in interval $[t_{ni}^-, t_{ni}^+]$ after his ith bid is accepted, is greater than the cost h_{ks} for each unit of resource k on any server s.

The assumption is reasonable as otherwise, serving the bid on any server will lead to a negative profit gain of the cloud provider. Let $w(\mathbf{x})$ denote the social welfare in (3) as a function of the resource assignment $\mathbf{x}$ of all users. Based on Assumption 1, we establish submodularity of $w(\mathbf{x})$.

Submodularity If Ω is a finite set, a function $h : \{0,1\}^\Omega \to \mathbb{R}$ is submodular if for every $\mathbf{x}, \mathbf{y} \in \{0,1\}^\Omega$ with $x_\omega \leq y_\omega$ for all $\omega \in \Omega$ and any $\omega' \in \Omega$ such that $y_{\omega'} = 0$, we have

$$h(\mathbf{x} \vee \mathbf{e}^{\omega'}) - h(\mathbf{x}) \geq h(\mathbf{y} \vee \mathbf{e}^{\omega'}) - h(\mathbf{y})$$

Here, $\{0,1\}^\Omega$ is the power set of Ω, *i.e.,* the set of all subsets of Ω. In the functions that we define which will be shown to be submodular (*e.g.,* $w(\mathbf{x}), g(\mathbf{x})$), Ω denotes the set of decision variables, *i.e.,* $\{x_{nis}\}_{s \in [S], i \in [I], n \in [N]}$, each of which is either 0 or 1.

So far, $w(\mathbf{x})$ is defined only for the set of feasible assignment vectors: $\mathcal{X} = \{\mathbf{x} \mid \sum_{s \in [S]} x_{nis} \leq 1, \forall n \in [N], \forall i \in [I_n]\}$. To show that $w(\mathbf{x})$ is submodular, we must extend the definition of $w(\mathbf{x})$ to all binary vectors. We will decompose the social welfare into two parts and handle them separately. Denote the "pure" value minus server cost as

$$g(\mathbf{x}) = \sum_{n \in [N]} u_n(\mathbf{x}_n) - \sum_{t \in [T]} \sum_{s \in [S]} \sum_{k \in [K]} f_{ks}(y_{ks}(t)) \ . \quad (7)$$

We have $w(\mathbf{x}) = g(\mathbf{x}) - \Phi(\mathbf{x})$ (with $\Phi(\mathbf{x}) = \sum_{n \in [N]} \Phi_n(\mathbf{x}_n)$). Since $\Phi(\mathbf{x})$ is well defined for all binary vectors, it suffices to extend the definition of $g(\mathbf{x})$ to all binary vectors as follows:

$$g(\mathbf{x}) = \max_{\mathbf{x}' \in \mathcal{X} : \forall i, \forall s, x'_{nis} \leq x_{nis}} g(\mathbf{x}')$$

TABLE 1: Key Notation

N	# of users	T	# of time slots
K	# of resource types	S	# of servers
B_{ni}	bid i of user n	t_{ni}	arrival time of B_{ni}

I_n	# of bids that user n submits
C_{ks}	capacity of type-k resource on server s
$t_{ni}^-(t_{ni}^+)$	start (end) time of resource usage in B_{ni}
$d_{nik}(t)$	demand of type-k resource at t of B_{ni}
x_{nis}	serve B_{ni} on server s (1) or not (0)
$D_{nik}(t)$	offline overall demand of type-k resource at t when user n submits the ith bid
δ_{nik}	value per unit resource k demand of B_{ni}
$\phi_{n(ij)}^{up}$	the cost incurred by winning bid B_{ni} and B_{nj} on a same server (scale-up).
$\phi_{n(ij)}^{out}$	the cost incurred by winning B_{ni} and B_{nj} on different servers (scale-out).
b_{nis}	bidding price if B_{ni} is accepted to server s
U_k	$\max_{s\in[S],i\in[I_n],n\in[N],t\in[t_{ni}^-,t_{ni}^+]} \frac{b_{nis}}{d_{nik}(t)}$
L_k	$\min_{s\in[S],i\in[I_n],n\in[N]} \frac{b_{nis}}{K\sum_{t\in[t_{ni}^-,t_{ni}^+]} d_{nik}(t)}$
$y_{ks}(t)$	allocation amount of resource k on s at t
$p_{ks}(t)$	marginal price of resource k on s at t
$\hat{p}_{ni}$	payment of bid B_{ni}
$g(\mathbf{x})$	pure value minus server cost given an assignment $\mathbf{x}$
$w(\mathbf{x})$	social welfare function given an assignment $\mathbf{x}$
$\hat{w}_{nis}(\mathbf{x})$	marginal welfare of accepting a user n's bid i given an assignment $\mathbf{x}$
$\hat{\phi}_{nis}(\mathbf{x})$	marginal scaling cost of accepting a user n's bid i given an assignment $\mathbf{x}$

Lemma 1. *The "pure" value minus server cost $g(\mathbf{x})$ defined in (7) is submodular.*

The detailed proof is given in Appendix A.

Theorem 1. *The social welfare function $w(\mathbf{x})$ (which equals the objective in* (3)*) is submodular.*

The detailed proof is given in Appendix B.

Assumption 2 (small scale up/out costs). Suppose $\mathbf{x}$ is a feasible resource assignment that does not accept bid i of user n (i.e., $x_{nis'} = 0$ for all $s' \in [S]$). Then, the increase in $g(\mathbf{x})$ in (7) for accepting the bid on any server is at least α times the increase in the total scale up/out cost $\Phi_n(\mathbf{x}_n)$, where α is a large enough value (e.g., $\alpha \geq 3$).

Since the social welfare equals $g(\mathbf{x}) - \Phi(\mathbf{x})$, this assumption states that accepting a bid increases the social welfare substantially, which is reasonable in practice as well.

4 Online Auction for Social Welfare Maximization

In this section, we introduce our online auction design.

4.1 Online Auction ($Toast$)

Resource provisioning. Let us first explain how the auctioneer decides whether to accept a bid B_{ni} or not and to which server it is assigned if accepted. We maintain a price for each unit of resource of each type k on each server s for each future time slot t, denoted by $p_{ks}(t)$, according to reserved allocation of the respective resource on the respective server at that time due to already accepted bids. We compute the potential payment of B_{ni}, if the requested resources are to be provisioned on server s, as $P_{nis} = \sum_{t\in[t_{ni}^-,t_{ni}^+]}\sum_{k\in[K]} d_{nik}(t)p_{ks}(t)$, $\forall s \in [S]$. Hence the corresponding utility gain received by user n, if the bid is accepted, would be $b_{nis} - P_{nis}$ (bidding price minuses payment). We check if the user's utility is positive on some servers. If so, we choose the server giving the largest utility gain, i.e., $s^* = \arg\max_{s\in[S]}(b_{nis} - P_{nis})$, to serve the bid, *i.e.*, $x_{nis^*} = 1$ and $x_{nis} = 0$ for all $s \neq s^*$, and increase the amount of allocated resources $y_{ks^*}(t)$ by $d_{nik}(t)$ on server s^* for all resources $k \in [K]$ and all time slots $t \in [t_{ni}^-, t_{ni}^+]$. If no server provides a positive utility, reject the bid, *i.e.*, $x_{nis} = 0$ for all $s \in [S]$. The above utility-maximizing approach for each bid leads to truthfulness and approximate social welfare maximization for some carefully chosen prices which we will explain next.

Payment design. We design a price function $p_{ks}(t)$ that depends on the amount of allocated resource, $y_{ks}(t)$, for all $k \in [K], s \in [S], t \in [T]$, as follows:

$$p_{ks}(t) = \frac{L_k - h_{ks}}{2KS}\left(\frac{2KS(U_k - h_{ks})}{L_k - h_{ks}}\right)^{\frac{y_{ks}(t)}{C_{ks}}} + h_{ks} \tag{8}$$

where
$$L_k = \min_{s\in[S],i\in[I_n],n\in[N]} \frac{b_{nis}}{K\sum_{t\in[t_{ni}^-,t_{ni}^+]} d_{nik}(t)}$$
$$U_k = \max_{s\in[S],i\in[I_n],n\in[N],t\in[t_{ni}^-,t_{ni}^+]} \frac{b_{nis}}{d_{nik}(t)}$$

Here U_k is an upper bound of the unit price of resource k in the sense that no user would want his bid to be accepted at price U_k. L_k is a lower bound on the unit price of type-k resource in the sense that even the lowest bid would be accepted if the unit price for resource k is L_k for all k. The first term of the above price design ensures that (i) the cloud would accept even the lowest bid at the beginning, but (ii) the price grows exponentially as the demand increases so the cloud would not allocate all resources to low value bids that come early at the risk of having no capacity left for high value bids in the future. The second term ensures that (iii) the gain in social welfare when a bid is served outweighs the loss in total server cost. Indeed, designing the online pricing rules is the key to obtain a good competitive ratio in social welfare. The price of serving each B_{ni} on server s^* is computed by summing up the products of resource demands and current unit prices on server s^*, over all k and $t \in [t_{ni}^-, t_{ni}^+]$, i.e.,

$$\hat{p}_{ni} = \sum_{t\in[t_{ni}^-,t_{ni}^+]}\sum_{k\in[K]} d_{nik}(t)p_{ks^*}(t) \ .$$

Online Auction. We summarize the online auction mechanism, referred to as $Toast$, in Alg. 1. Note that the payment (line 10) is computed based on unit resource prices on the selected server s^* before counting in resources requested in the bid (line 2). Hence, the payment is independent of the

ALGORITHM 1: The Online Auction Mechanism ($Toast$)

Input: S, K, $\mathbf{C}$, $\mathbf{U}$, $\mathbf{L}$
Output: $\mathbf{x}, \mathbf{p}$
Initialize: $\mathbf{x} = \mathbf{0}$, $\mathbf{y} = \mathbf{0}$, $\mathbf{p}(\mathbf{y}) = \mathbf{p}(\mathbf{0})$, $\hat{\mathbf{p}} = \mathbf{0}$
Upon arrival of bid B_{ni} **according to** (1):
Compute $P_{nis} = \sum_{t\in[t_{ni}^-,t_{ni}^+]} \sum_{k\in[K]} d_{nik}(t)p_{ks}(t), \forall s \in [S]$;
$u_{ni} = \max_{s\in[S]}(b_{nis} - P_{nis})$;
if $u_{ni} > 0$ **then**
  Accept B_{ni};
  $s^* = \text{argmax}_{s\in[S]}(b_{nis} - P_{nis})$;
  Update $x_{nis^*} = 1$, and allocate each resource k at amount of $d_{nik}(t)$ on server s^*, for each $t \in [t_{ni}^-, t_{ni}^+]$;
  Update $y_{ks^*}(t) = y_{ks^*}(t) + d_{nik}(t)$, $\forall k \in [K], t \in [t_{ni}^-, t_{ni}^+]$;
  Update $p_{ks^*}(t)$ in (8), $\forall k \in [K], t \in [t_{ni}^-, t_{ni}^+]$;
  Charge B_{ni} **the payment of** $\hat{p}_{ni} = P_{nis^*}$;
else
  Reject B_{ni};

bid, which is the key to guarantee truthfulness. In addition, although our online auction requires U_k and L_k as input, whose exact values are not known before all bids have arrived, we can adopt estimated values of these upper and lower bounds as input to our online algorithm, *e.g.*, based on past experience. We will show in Sec. 5 the impact of the estimation accuracy.

Theorem 2. *Alg.* 1 *outputs a feasible solution of* (3).

The detailed proof is given in Appendix C.

Theorem 3. *The online auction in Alg.* 1 *is truthful in valuation, bid arrival time and VM execution duration, achieves individual rationality for each bidder and the cloud provider, and processes each bid in* $O(KST)$ *time.*

The detailed proof is given in Appendix D.

4.2 Competitive Analysis of Algorithm $Toast$

We next analyse the competitive ratio achieved by our online auction. Let $\mathbf{x}(A)$ be the assignment vector of the algorithm and $\mathbf{x}^\star$ be the offline optimal assignment. In order to show competitiveness of the algorithm, we need to show that $w(\mathbf{x}(A))$ is comparable to $w(\mathbf{x}^\star)$. However, it is difficult to compare them directly. We adopt a two-step approach from the literature of submodular maximization (e.g., [27]): (i) first show that $w(\mathbf{x}(A))$ is comparable to $w(\mathbf{x}(A) \vee \mathbf{x}^\star)$, which is the welfare of accepting both the bids accepted by our algorithm and those accepted by the offline optimum; then (ii) show that $w(\mathbf{x}(A) \vee \mathbf{x}^\star)$ is comparable to $w(\mathbf{x}^\star)$, *i.e.,* the offline optimum.

To show (i), imagine we start with $w(\mathbf{x}(A))$ and add one by one the bids that are accepted by the offline optimum in $\mathbf{x}^\star$ but not by the algorithm in $\mathbf{x}(A)$. We need to upper bound the marginal value of accepting each such bid. Similarly, to show (ii), imagine we start from $w(\mathbf{x}^\star)$ and add one by one the bids that are accepted by the algorithm in $\mathbf{x}(A)$ but not by the optimum in $\mathbf{x}^\star$. We need to lower bound the (potentially negative) marginal value of accepting each such bid.

Let $\hat{w}_{nis}(\mathbf{x})$ denote the marginal welfare of accepting a user n's bid i on server s given an assignment $\mathbf{x}$, i.e.,

$$\hat{w}_{nis}(\mathbf{x}) = w(\mathbf{x} \vee \mathbf{e}^{(nis)}) - w(\mathbf{x}). \tag{9}$$

Similarly, let

$$\hat{\Phi}_{nis}(\mathbf{x}) = \sum_{j\neq i} \phi_{n(ij)}^{up} x_{njs} + \sum_{j\neq i} \sum_{s'\neq s} \phi_{n(ij)}^{out} x_{njs'} \tag{10}$$

denote the additional scale-up/out costs for accepting a user n's bid i on server s given $\mathbf{x}$. As a corollary of Assumption 2, the following lemma holds.

Lemma 2. *Suppose* $\mathbf{x}$ *satisfies that* $x_{nis'} = 0$ *for all* $s' \in [S]$, *and* $\sum_{s'\in[S]} x_{njs'} \leq 1$ *for all* j. *Then, we have*

$$\hat{\Phi}_{nis}(\mathbf{x}) \leq \frac{1}{2}\hat{w}_{nis}(\mathbf{x})$$

Proof of Lemma 2 is given in Appendix E.

As a further corollary, we have the following lemma.

Lemma 3. *Suppose* $\mathbf{x}$ *satisfies that* $x_{nis'} = 0$ *for all* $s' \in [S]$, *and* $\sum_{s'\in[S]} x_{njs'} \leq 1$ *for all* j *Then* $\hat{w}_{nis}(\mathbf{x}) \geq 0$

In case the bid has been accepted to some other servers in $\mathbf{x}$, we lower bound the changes in the welfare function with the following lemma, which follows by monotonicity of $g(\mathbf{x})$. Note that the algorithm can gurantee that each bid is accepted to at most one server while in the analysis we temporarily allow multiple entries of $\mathbf{x}$ to be 1. The reason is that a well-defined sumodular function should not be restricted by such a constraint $\sum_{s\in[S]} x_{nis} \leq 1$. The following lemma essentially shows that if we choose more than one servers for an accepted bid, the decreased social welfare ($-\hat{w}_{nis}(\mathbf{x})$) will be no more than the additional scale-up/out costs incurred.

Lemma 4. *Suppose* $\mathbf{x}$ *satisfies that there exists* $s' \neq s$ *such that* $x_{nis'} = 1$. *Then,* $\hat{w}_{nis}(\mathbf{x}) \geq -\hat{\Phi}_{nis}(\mathbf{x})$.

Proof of Lemma 4 is given in Appendix F.

To upper bound the marginal values of accepting a bid that is accepted by the offline optimum in $\mathbf{x}^\star$ but not by the algorithm in $\mathbf{x}(A)$, w.r.t. $\mathbf{x}(A)$, we divide such bids into two subsets.

$$\mathcal{S}_1^\star = \{(n,j,s) \mid x_{njs}^\star = 1, \forall s' \neq s, x(A)_{njs'} = 0\} \tag{11}$$
$$\mathcal{S}_2^\star = \{(n,j,s) \mid x_{njs}^\star = 1, \exists s' \neq s, x(A)_{njs'} = 1\} \tag{12}$$

Similarly, divide the bids that are accepted by the algorithm in $\mathbf{x}(A)$ but not by the optimal in $\mathbf{x}^\star$ into two subsets

$$\mathcal{S}_1 = \{(n,j,s) \mid x(A)_{njs} = 1, \forall s' \neq s, x_{njs'}^\star = 0\} \tag{13}$$
$$\mathcal{S}_2 = \{(n,j,s) \mid x(A)_{njs} = 1, \exists s' \neq s, x_{njs'}^\star = 1\} \tag{14}$$

The following lemma is to show that $w(\mathbf{x}^\star \vee \mathbf{x}(A))$ is comparable to $w(\mathbf{x}^\star)$ as we stated in step (ii) at the beginning of this subsection.

Lemma 5. *The social welfare of the offline optimum is upper bounded by*

$$w(\mathbf{x}^\star) \leq w(\mathbf{x}^\star \vee \mathbf{x}(A)) + \textstyle\sum_{(n,i,s)\in\mathcal{S}_2} \hat{\Phi}_{nis}(\mathbf{x}^\star \vee \mathbf{x}(A))$$

Proof of Lemma 5 is given in Appendix G.

The following lemma is to show that $w(\mathbf{x}(A))$ is comparable to $w(\mathbf{x}^\star \vee \mathbf{x}(A))$ as we stated in step (i) at the beginning of this subsection.

Lemma 6. *We have*

$$w(\mathbf{x}^\star \vee \mathbf{x}(A)) \leq w(\mathbf{x}(A)) + \sum_{(n,i,s)\in\mathcal{S}_1^\star} \hat{w}_{nis}(\mathbf{x}(A))$$

Proof of Lemma 6 is given in Appendix H.

Lemma 7. *We have*

$$\sum_{(n,i,s)\in\mathcal{S}_2} \hat{\Phi}_{nis}(\mathbf{x}^\star \vee \mathbf{x}(A)) \leq \frac{1}{2}w(\mathbf{x}^\star) + \frac{1}{2}w(\mathbf{x}(A))$$

Proof of Lemma 7 is given in Appendix I.

Lemma 8. *We have*

$$\textstyle\sum_{(n,i,s)\in\mathcal{S}_1^\star} \hat{w}_{nis}(\mathbf{x}(A)) \leq \sum_{k,s,t} \big[p_{ks}(Y_{ks}(t)) - h_{ks}\big] C_{ks}$$

Proof of Lemma is given in Appendix J.

Lemma 9. *Suppose* $w(\mathbf{x}^\star) \geq \sum_{k,s,t} \frac{2(L_k - h_{ks})C_{ks}}{KS}$. *Then,*

$$\sum_{t\in[T]}\sum_{s\in[S]}\sum_{k\in[K]} \big[p_{ks}(Y_{ks}(t)) - h_{ks}\big] C_{ks}$$
$$\leq \max_{s\in[S],k\in[K]} \ln\left(\frac{2KS(U_k - h_{ks})}{L_k - h_{ks}}\right) w(\mathbf{x}(A)) + \frac{1}{4}w(\mathbf{x}^\star)$$

Proof of Lemma 9 is given in Appendix K.

Theorem 4. *Suppose* $w(\mathbf{x}^\star) \geq \sum_{k,s,t} \frac{2(L_k - h_{ks})C_{ks}}{KS}$. *Then, our designed online auction* $Toast$ *has a competitive ratio of* $O\big(\max_{s\in[S],k\in[K]} \ln(\frac{2KS(U_k - h_{ks})}{L_k - h_{ks}})\big)$.

Proof. Putting Lemma 5 –9 together proves the theorem. □

Recall that L_k is a lower bound of the bidding prices per unit of demand among all the bids. Thus $L_k - h_{ks}$ is the minimum social welfare contributed by a bid if it is served by server s and $(L_k - h_{ks})C_{ks}$ is the minimal social welfare if the capacity of k-type resource on server s is fully occupied. Thus, the assumption in Theorem 4 is essentially that the workload at each time slot is high enough to fully occupy at least one resource on two servers or two resources on one server, which is easy to fulfill in practice.

5 Performance Evaluation

5.1 Experimental Setup

We evaluate our online auction using trace-driven simulations, exploiting Google cluster-usage data [33], which contains resource capacity of servers and job information submitted to the Google cluster. A job comprises multiple tasks, each of which is accompanied by its resource requirement (CPU, RAM and disk). We associate each job with a user and translate each of the job's tasks into a VM bid of the user, requesting $K = 3$ types of resource demands extracted from the traces. Especially, the overall resource demand of user n when submitting the ith bid ($D_{nik}(t)$) equals the total demand of the resource in the first i tasks of the job; the resource requirement specified in the ith bid ($d_{nik}(t)$) is set according to (5). The number of bids per user (I_n) is within $[1, 50]$.

There are in total $T = 1000$ time slots and each time slot is 10 seconds long. Google cluster data provides the arrival time of each job, and the start time and end time of each task in a job. We scale the duration of Google jobs down to our time scale, and set the time a user submits his first bid according to the job arrival time in Google data. We also order tasks in a job according to their start time, and set start (end) time of VM usage in the bids, t_{ni}^- (t_{ni}^+), according to the start (end) time of the respective tasks. The arrival time of each bid i ($i \geq 2$) of a user n is set uniformly within $[t_{n(i-1)}, t_{ni}^-]$. We set the bidding prices in each bid according to (4). We uniformly randomly set $\phi_{n(ij)}^{up}$'s and $\phi_{n(ji)}^{out}$'s within $[0, \sum_{t\in[t_{ni}^-, t_{ni}^+]} \sum_{k\in[K]} d_{nik}(t)\delta_{nik}/I_n]$ (note scaling up/out cost of each bid is not always as small as Assumption 2 requires). The unit values δ_{nik} of users are set within the upper and lower bounds of users' value per unit of resource per unit of time, U_k and L_k, which will be varied in different experiments. By default, $U_k = 50$, $L_k = 1$. We simulate servers with heterogenous resource capacities (C_{ks}) following the distribution of server configurations summarized from the Google data as follows (CPU and Memory units are normalized so that the maximum capacity is 1):

percentage of machines	53%	30%	8%	6%	3%
CPU	0.50	0.50	0.50	1.00	0.25
Memory	0.50	0.25	0.750	1.00	0.25

Since the Google data does not provide disk configurations, we set the disk storage capacity of servers randomly within $[320, 800](GB)$. The total capacity of each type of resource to provision, and hence the number of servers to simulate, is roughly according to the total amount of demand from all bids multiplying a random number in $[0.4, 0.8]$. h_{ks} is uniformly distributed within $[0.4, 0.6]$ for CPU (different for different servers s), and within $[0.005, 0.02]$ for RAM and disk, roughly following the percentage measured in [32].

We compare our online auction with the offline optimum, as well as two heuristic schemes, *Twice-the-Cost* (TC) and *Twice-the-Index* (TI) (we identified a lack of comparable approaches from the cloud auction literature). The two heuristics share the same steps with our online auction, except for adopting different price functions [34]: For TC, $p_{ks}(t) = 2f_{ks}^{'}(y_{ks}(t))$, *i.e.*, the marginal price at t is twice of the marginal cost for t; for TI, $p_{ks}(t) = f_{ks}^{'}(2y_{ks}(t))$, *i.e.*, the marginal price for t is the marginal cost on twice of the resource usage at t.

5.2 Comparison with Offline Optimum

We study the ratios computed by dividing the offline optimal social welfare by the social welfare achieved by our online algorithm under different settings. The offline optimal social welfare is obtained by solving the offline problem (3) exactly with a brute force approach, by enumerating all possible solutions, due to the hardness of solving the non-convex offline problem. Due to the high time complexity of the method, we limit the largest number of bids from all users to be 30 in this set of experiments and the number of servers to very small as well. Fig. 2 shows that the ratios are around 2 with varying numbers of servers.

5.3 Performance with Under/Over-Estimation of U_k and L_k

We evaluate performance of our online auction at different levels of under-estimation and over-estimation of the actual

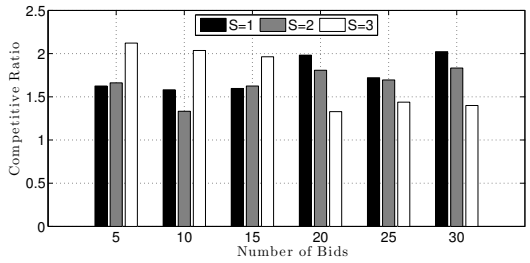


Fig. 2: Competitive ratio of $Toast$ with different server numbers

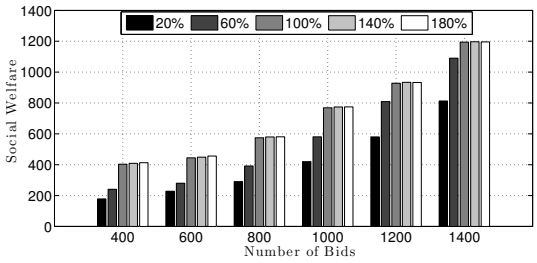


Fig. 3: Social welfare achieved by $Toast$ with different estimations of L_k

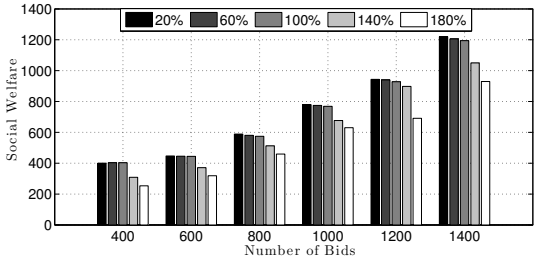


Fig. 4: Social welfare achieved by $Toast$ with different estimations of U_k

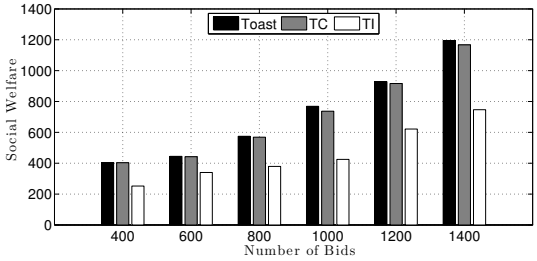


Fig. 5: Comparison among $Toast$, TC, and TI

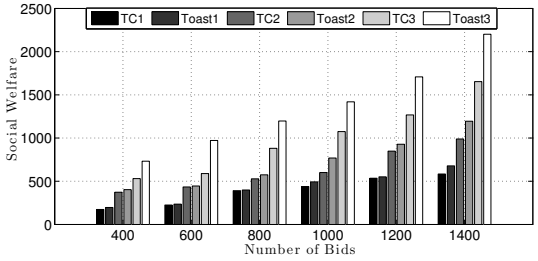


Fig. 6: Comparison between $Toast$ and TC with varying U_k/L_k

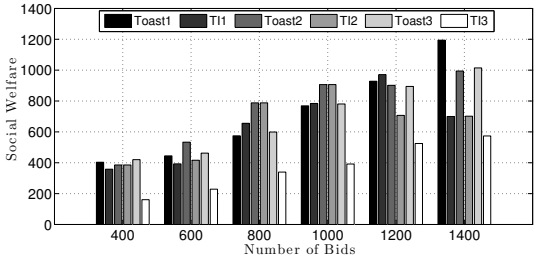


Fig. 7: Comparison between $Toast$ and TI with varying U_k/L_k

values (the case of 100%) of U_k and L_k, which are input in the price functions in $Toast$. Fig. 3 and Fig. 4 illustrate how the social welfare achieved is influenced by their under-estimation or over-estimation, with over-estimation more preferred for L_k and under-estimation more preferred for U_k.

5.4 Comparison with Two Heuristics

Fig. 5 shows that $Toast$ outperforms TI significantly and achieves slightly better performance than TC. In Fig. 6, we further compare $Toast$ and TC by running the algorithms under different values of U_k/L_k. Especially, U_k/L_k is 10 for $TC1$ and $Toast1$, 50 for $TC2$ and $Toast2$, and 250 for $TC3$ and $Toast3$ in the figure. We observe that when U_k/L_k is larger, $Toast$ outperforms TC more. This can be explained by that the price function of TC is only related to the marginal server cost which underperforms in filtering out low value bids when the range of bidding price per unit of demand is large.

We also further compare $Toast$ and TI in Fig. 7, by setting the ratio of total resource capacity/total demand of each type of resource in the system at each time within different ranges. The range is $[0.6, 0.8]$, $[0.4, 0.6]$, and $[0.2, 0.4]$ for $Toast1$ ($TI1$), $Toast2$ ($TI2$), and $Toast3$ ($TI3$), respectively. We observe that when the resource is more scarce, our mechanism outperforms TI more. This is expected since TI rejects more bids when the server utilization is high, because its prices surge when resource allocation reaches half of the server capacity.

6 Concluding Remarks

This work designs a truthful and efficient online auction for dynamic resource scaling and pricing, where cloud users repeatedly bid for resources into the future with increased amounts, according to their scale-up/out preferences. We consider server energy cost minimization in social welfare maximization, and reveal an important property, submodularity, of the objective function in the resulting significantly more challenging offline problem. A novel competitive analysis framework is established for submodular function optimization with non-linear constraints, demonstrating a good competitive ratio of our online mechanism. The current work focuses on dynamic increase of resources, which

is supported in today's IaaS clouds. On the other hand, resource scale-down and scale-in involve selling resources back to the provider. Designing efficient online mechanisms for a bi-directional market is significantly more challenging, which we seek to investigate in our future work, possibly exploiting double auctions.

APPENDIX A
PROOF OF LEMMA 1

Proof. Informally, (7) is submodular because the marginal increase of (7) due to accepting a bid i (of user n) is proportional to the amount of resources obtained through the bid, which decreases if more of the other bids are accepted.

Formally, we need to show that for any $n \in [N]$, $i \in [I_n]$, $s \in [S]$, and any $\mathbf{x}, \mathbf{x}'$ such that $x_{nis} = x'_{nis} = 0$ and $x_{njs'} \geq x'_{njs'}$ for all $j \in [I_n]$ and $s' \neq s$, the marginal increase in "pure" value minus server cost of accepting bid i on server s satisfies

$$g(\mathbf{x} \vee \mathbf{e}^{(nis)}) - g(\mathbf{x}) \leq g(\mathbf{x}' \vee \mathbf{e}^{(nis)}) - g(\mathbf{x}')$$

Since $g(\mathbf{x}) = \sum_{n\in[N]} u_n(\mathbf{x}_n) - \sum_{k,s,t} f_{ks}(y_{ks}(t))$, we have

$$g(\mathbf{x}\vee\mathbf{e}^{(nis)}) = \sum_{n\in[N]} \sum_{k\in[K]} \sum_{t\in[t^-_{ni},t^+_{ni}]} (\delta_{nik} - h_{ks}) d_{nik}(t)$$

According to Assumption 1, $\delta_{nik} \geq h_{ks}$. And recall that

$$d_{nik}(t) = D_{nik}(t) - \max_{1\leq j\leq i-1: t^-_{nj} \leq t^+_{nj}} \{D_{njk}(t) \sum_{s\in[S]} x_{njs}\}.$$

Since we have $x_{njs'} \geq x'_{njs'}$ for all $j \in [I_n]$ and $s \in [S]$, $d_{nik}(t)$ under $\mathbf{x} \vee \mathbf{e}^{(nis)}$ is no larger than that under $\mathbf{x}' \vee \mathbf{e}^{(nis)}$. Thus the lemma follows. □

APPENDIX B
PROOF OF THEOREM 1

Proof. We need to show that for any $n \in [N]$, $i \in [I_n]$, $s \in [S]$, and any $\mathbf{x}, \mathbf{x}'$ such that $x_{nis} = x'_{nis} = 0$ and $x_{njs'} \geq x'_{njs'}$ for all $j \in [I_n]$ and $s' \neq s$, the marginal increase in social welfare of accepting bid i on server s satisfies

$$w(\mathbf{x} \vee \mathbf{e}^{(nis)}) - w(\mathbf{x}) \leq w(\mathbf{x}' \vee \mathbf{e}^{(nis)}) - w(\mathbf{x}')$$

Since $w(\mathbf{x}) = g(\mathbf{x}) - \Phi(\mathbf{x})$, it suffices to show that

$$g(\mathbf{x} \vee \mathbf{e}^{(nis)}) - g(\mathbf{x}) \leq g(\mathbf{x}' \vee \mathbf{e}^{(nis)}) - g(\mathbf{x}') \quad (15)$$

and

$$\Phi(\mathbf{x} \vee \mathbf{e}^{(nis)}) - \Phi(\mathbf{x}) \geq \Phi(\mathbf{x}' \vee \mathbf{e}^{(nis)}) - \Phi(\mathbf{x}') \quad (16)$$

Inequality (15) holds by our assumption of g's being submodular. Next, we show (16). By (6), we have that for any $\mathbf{x}$, $\Phi(\mathbf{x} \vee \mathbf{e}^{(nis)}) - \Phi(\mathbf{x})$ is equal to

$$\sum_{j\neq i: x_{njs}=1} \phi^{up}_{n(ij)} + \sum_{j\neq i: \exists s'\neq s, x_{njs'}=1} \phi^{out}_{n(ij)} \ ,$$

which is non-decreasing in $\mathbf{x}$ since $\phi^{up}_{n(ij)}$'s and $\phi^{out}_{n(ij)}$'s are non-negative. So (16) holds because $x_{njs'} \geq x'_{njs'}$ for all $j \in [I_n]$ and $s \in [S]$.

□

APPENDIX C
PROOF OF THEOREM 2

Proof. *VM consolidation constraint (3a).* This follows by the definition of our online auction (line 8 and 9 of Alg. 1). *Capacity constraint (3b).* By our payment design in (8), we have $p_{ks}(t) = U_k$ when $y_{ks}(t) = C_{ks}$, which is high enough to ensure negative utility for any bid. So the algorithm assigns resources within the server capacities. □

APPENDIX D
PROOF OF THEOREM 3

Proof. *(Truthfulness in bidding price)* The marginal prices that the cloud provider presents to bid B_{ni} depend only on the demands of resources before the arrival of bid B_{ni} and the demands of bid B_{ni}, thus, are independent on B_{ni}'s bidding price. Further, the cloud provider always assigns bids to servers to maximize each bid's utility given the current marginal prices. Assuming the bidders are myopic, i.e., a bidder only cares about the utility in his current bid. So it falls into the family of sequential posted price mechanisms (e.g., [35]) and, thus, a bidder cannot improve its utility by lying about the bidding price of any of his bids.

(Truthfulness in arrival time) Since the marginal prices are non-decreasing in the amount of allocated resources, which is non-decreasing over the resource allocation time, *i.e.*, let $y_{ks}(t_{ni}, t)$ denote the amount of resource k on s in t which has been allocated by t_{ni} ($t_{ni} \leq t$) and $y_{ks}(t_{ni}, t)$ is non-decreasing over t_{ni}. Hence a bidder cannot decrease the total price of the resource that it requests by delaying its arrival. Note that the arrival time of a bid is the first time the bidder is aware of his latest updated demands so the arrival time can not be earlier.

(Truthfulness in VM execution duration) Dropping part of the true resource occupation duration in the request would risk failing to complete the job. So no bidder would do that. On the other hand, the marginal prices are non-negative according to (8). So requesting a superset of the true VM occupation duration increases a bidder's payment and decreases her utility.

(Individually rational) According to line 4 of Alg. 1, the utility in any bid of a bidder is always non-negative. The profit of the provider is also non-negative based on the formulation of $\hat{p}_{ni}$ in Sec. 4 which implies $p_{ks}(y_{ks}(t)) > f'_{ks}(y_{ks}(t))$.

(Polynomial running time) To process a bid B_{ni}, the algorithm first sums up the marginal prices for all requested resources over the occupation duration for each server $s \in [S]$ to compute the payment that bid B_{ni} should pay if it would be served on server s. This step runs in $O(KST)$ time. Then, the algorithm computes u_{ni} and decides the allocation and payment of bid B_{ni} by checking the utility of the bid if it would be served on each server s, which can be done in $O(S)$ time. Finally, the algorithm updates the amount of allocated resources $y_{ks}(t)$ and the respective marginal prices $p_{ks}(t)$, which can be done in $O(KST)$ time.

□

APPENDIX E
PROOF OF LEMMA 2

Proof. According to *Assumption* 2, the increase in $g(\mathbf{x})$ for turning x_{nis} from 0 to 1 is at least α (≥ 3) times the corresponding increase in $\Phi_n(x_n)$. Moreover, we have $g(\mathbf{x}) = w(\mathbf{x}) + \Phi(\mathbf{x})$. Thus we have $\hat{w}_{nis}(\mathbf{x}) + \hat{\Phi}_{nis}(\mathbf{x}) \geq 3\hat{\Phi}_{nis}(\mathbf{x})$, which the lemma follows. □

APPENDIX F
PROOF OF LEMMA 4

Proof. According to Lemma 1, $g(\mathbf{x})$ is monotone, *i.e.*, the increase in $g(\mathbf{x})$ for turning x_{nis} from 0 to 1 is always nonnegative. Therefore, $\hat{w}_{nis}(\mathbf{x}) + \hat{\Phi}_{nis}(\mathbf{x}) \geq 0$, which the lemma follows. □

APPENDIX G
PROOF OF LEMMA5

Proof. Let $\mathbf{e}^{\mathcal{S}_1}$ be a vector with ones in the entries in $\mathcal{S}_1$ and zeros in other entries. Define $\mathbf{e}^{\mathcal{S}_2}$ similarly. Then, we have $\mathbf{x}^\star \vee \mathbf{x}(A) = \mathbf{x}^\star \vee \mathbf{e}^{\mathcal{S}_1} \vee \mathbf{e}^{\mathcal{S}_2}$. By submodularity, we have

$$\begin{aligned} &w(\mathbf{x}^\star \vee \mathbf{x}(A)) = w(\mathbf{x}^\star \vee \mathbf{e}^{\mathcal{S}_1} \vee \mathbf{e}^{\mathcal{S}_2}) \\ &\geq w(\mathbf{x}^\star) + \sum_{(n,i,s)\in\mathcal{S}_1} \hat{w}_{nis}(\mathbf{x}^\star \vee \mathbf{e}^{\mathcal{S}_1}) \\ &+ \sum_{(n,i,s)\in\mathcal{S}_2} \hat{w}_{nis}(\mathbf{x}^\star \vee \mathbf{e}^{\mathcal{S}_1} \vee \mathbf{e}^{\mathcal{S}_2}) \end{aligned}$$

Note that $\mathbf{x}^\star \vee \mathbf{e}^{\mathcal{S}_1}$ satisfies the conditions in Lemma 3. So the second term on the RHS of the above inequality is at least 0. By Lemma 4, we have $\hat{w}_{nis}(\mathbf{x}^\star \vee \mathbf{e}^{\mathcal{S}_1} \vee \mathbf{e}^{\mathcal{S}_2}) = -\hat{\Phi}_{nis}(\mathbf{x}^\star \vee \mathbf{e}^{\mathcal{S}_1} \vee \mathbf{e}^{\mathcal{S}_2}) = -\hat{\Phi}_{nis}(\mathbf{x}^\star \vee \mathbf{x}(A))$. So the lemma follows. □

APPENDIX H
PROOF OF 6

Proof. Define $\mathbf{e}^{\mathcal{S}_1^\star}$ and $\mathbf{e}^{\mathcal{S}_2^\star}$ similarly as in the previous proof. Then, we have $\mathbf{x}^\star \vee \mathbf{x}(A) = \mathbf{x}(A) \vee \mathbf{e}^{\mathcal{S}_1^\star} \vee \mathbf{e}^{\mathcal{S}_2^\star}$. By submodularity, we have

$$\begin{aligned} &w(\mathbf{x}^\star \vee \mathbf{x}(A)) = w(\mathbf{x}(A) \vee \mathbf{e}^{\mathcal{S}_1^\star} \vee \mathbf{e}^{\mathcal{S}_2^\star}) \\ &\leq w(\mathbf{x}(A)) + \sum_{(n,i,s)\in\mathcal{S}_1^\star} \hat{w}_{nis}(\mathbf{x}(A)) \\ &+ \sum_{(n,i,s)\in\mathcal{S}_2^\star} \hat{w}_{nis}(\mathbf{x}(A) \vee \mathbf{e}^{\mathcal{S}_1^\star}) \end{aligned}$$

By Lemma 4, the last term is less than or equal to 0. □

APPENDIX I
PROOF OF 7

Proof. Note that $\hat{\Phi}_{nis}(\mathbf{x})$ is linear in $\mathbf{x}$ (Eqn. (10)). We get that $\hat{\Phi}_{nis}(\mathbf{x}^\star \vee \mathbf{x}(A)) \leq \hat{\Phi}_{nis}(\mathbf{x}^\star) + \hat{\Phi}_{nis}(\mathbf{x}(A))$. Since $\phi^{up}_{nii} = \phi^{out}_{nii} = 0$, we have that $\hat{\Phi}_{nis}(\mathbf{x}(A)) = \hat{\Phi}_{nis}(\mathbf{x}(A) \wedge \mathbf{e}^{(nis)})$. Further, by Lemma 2, we have $\hat{\Phi}_{nis}(\mathbf{x}(A) \wedge \mathbf{e}^{(nis)}) \leq \frac{1}{2}\hat{w}_{nis}(\mathbf{x}(A) \wedge \mathbf{e}^{(nis)})$. Here $\mathbf{x} \wedge \mathbf{x}'$ denotes a vector whose entries are equal to the min of the corresponding entries in $\mathbf{x}$ and $\mathbf{x}'$, i.e., $(\mathbf{x} \wedge \mathbf{x}')_i = 1$ if both $x_i = 1$ and $x'_i = 1$ hold. By submodularity, we get that $w(\mathbf{x}(A)) \geq \sum_{(n,i,s):x_{nis}=1} \hat{w}_{nis}(\mathbf{x}(A) \wedge \mathbf{e}^{(nis)}) \geq \sum_{(n,i,s)\in\mathcal{S}_2} \hat{w}_{nis}(\mathbf{x}(A) \wedge \mathbf{e}^{(nis)})$, where the second inequality holds because $\mathcal{S}_2$ is a subset of the (n,i,s)'s that are accepted by $\mathbf{x}$ (i.e., $x_{nis} = 1$) and each term in the sum is non-negative (Lemma 3). Thus $\sum_{x_{nis}\in\mathcal{S}_2} \hat{\Phi}_{nis}(\mathbf{x}(A)) \leq \frac{1}{2}w(\mathbf{x}(A))$.

Moreover, for each $x_{nis} \in \mathcal{S}_2$, there exists a $s' \neq s$ such that $x_{nis'} \in \mathbf{x}^\star$, *i.e.*, $x^\star_{nis'} = 1$. According to Assumption 2, we have $\sum_{k\in[K],t\in[t^-_{ni},t^+_{ni}]}(\delta_{nik} - h_{ks'})d_{nik}(t) \geq 3\hat{\Phi}_{nis'}(\mathbf{x}^\star)$ and $\sum_{k\in[K],t\in[t^-_{ni},t^+_{ni}]}(\delta_{nik} - h_{ks'})d_{nik}(t) \geq 3\hat{\Phi}_{nis}(\mathbf{x}^\star)$. Thus we have $\sum_{k\in[K],t\in[t^-_{ni},t^+_{ni}]}(\delta_{nik} - h_{ks'})d_{nik}(t) - \hat{\Phi}_{nis'}(\mathbf{x}^\star) \geq \frac{2}{3}\sum_{k\in[K],t\in[t^-_{ni},t^+_{ni}]}(\delta_{nik} - h_{ks'})d_{nik}(t) \geq 2\hat{\Phi}_{nis}(\mathbf{x}^\star)$, *i.e.*, $\hat{\Phi}_{nis}(\mathbf{x}^\star) \leq \frac{1}{2}\hat{w}_{nis'}(\mathbf{x}^\star)$. Since $w(\mathbf{x}^\star) \geq \sum_{(nis'):x_{nis'}=1} \hat{w}_{nis'}(\mathbf{x}^\star)$, we have $\sum_{x_{nis}\in\mathcal{S}_2} \hat{\Phi}_{nis}(\mathbf{x}^\star) \leq \frac{1}{2}w(\mathbf{x}^\star)$. Putting together the inequalities shows the lemma. □

APPENDIX J
PROOF OF 8

Proof. Note that the marginal valuation $\hat{w}_{nis}(\mathbf{x}(A))$ represents the valuation of bid B_{ni} if it is served on server s conditioned on the decisions on all other bids being the same as those by our algorithm $Toast$. For each $x_{nis} \in \mathcal{S}_1^\star$, bid B_{ni} is rejected by our algorithm. Therefore, the marginal valuation of accepting bid B_{ni} on server s when the bid arrived (which, by submodularity, is greater than or equal to $\hat{w}_{nis}(\mathbf{x}(A))$), is smaller than the payment on that server s. Let $y^{(ni)}_{ks}(t)$ denote the allocated amount of resource k at t on s just before the arrival of B_{ni}. Then we have

$$\begin{aligned} \hat{w}_{nis}(\mathbf{x}(A)) &\leq \sum_{t\in[t^-_{ni},t^+_{ni}]} \sum_{k\in[K]} \big[p_{ks}(y^{(ni)}_{ks}(t)) - h_{ks}\big]d_{nik}(t) \\ &\leq \sum_{t\in[t^-_{ni},t^+_{ni}]} \sum_{k\in[K]} \big[p_{ks}(Y_{ks}(t)) - h_{ks}\big]d_{nik}(t) \end{aligned}$$

where the second inequality holds because $p_{ks}(y_{ks}(t))$ is monotone.

Moreover, the bids in $\mathcal{S}_1^\star$ are a subset of the bids accepted by the offline optimal. So the total demand of these bids for any resource on any server does not exceed the capacity of the server. Thus, summing up over all $(n,i,s) \in \mathcal{S}_1^\star$, we have

$$\sum_{(n,i,s)\in\mathcal{S}_1^\star} \hat{w}_{nis}(\mathbf{x}(A)) \leq \sum_{t\in[T]} \sum_{s\in[S]} \sum_{k\in[K]} \big[p_{ks}(Y_{ks}(t)) - h_{ks}\big]C_{ks}$$

□

APPENDIX K
PROOF OF 9

Proof. For any bid B_{ni} that is accepted by the algorithm on some server s, i.e., $x_{nis} = 1$, its marginal value is at least the payment. Thus, we have

$$w(\mathbf{x}(A)) \geq \sum_{t\in[T]}\sum_{s\in[S]}\sum_{k\in[K]}\int_0^{Y_{ks}(t)}[p_{ks}(y_{ks}(t))-h_{ks}]\mathrm{d}y_{ks}(t) \tag{17}$$

Next, we lower bound each term in the above sum as follows:

$$\begin{aligned}
&\int_0^{Y_{ks}(t)}[p_{ks}(y_{ks}(t))-h_{ks}]\mathrm{d}y_{ks}(t)\\
=&\int_0^{Y_{ks}(t)}[\frac{L_k-h_{ks}}{2KS}\left(\frac{2KS(U_k-h_{ks})}{L_k-h_{ks}}\right)^{\frac{y_{ks}(t)}{C_{ks}}}]\mathrm{d}y_{ks}(t)\\
=&\frac{(L_k-h_{ks})C_{ks}}{2KS\ln(\frac{2KS(U_k-h_{ks})}{L_k-h_{ks}})}(\frac{2KS(U_k-h_{ks})}{L_k-h_{ks}})^{\frac{Y_{ks}(t)}{C_{ks}}}\\
&-\frac{(L_k-h_{ks})C_{ks}}{2KS\ln(\frac{2KS(U_k-h_{ks})}{L_k-h_{ks}})}\\
=&\frac{1}{\ln(\frac{2KS(U_k-h_{ks})}{L_k-h_{ks}})}\left([p_{ks}(Y_{ks}(t))-h_{ks}]C_{ks}-\frac{(L_k-h_{ks})C_{ks}}{2KS}\right)
\end{aligned}$$

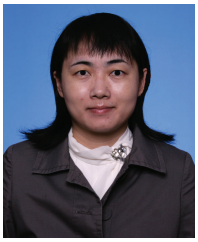
where the last equality holds by our payment design in (8). The lemma then follows by summing over all $k\in[K]$, $s\in[S]$, and $t\in[T]$ and using the assumption that the optimal welfare is at least $w(\mathbf{x}^\star)\geq\sum_{k,s,t}\frac{2(L_k-h_{ks})C_{ks}}{KS}$. □

REFERENCES

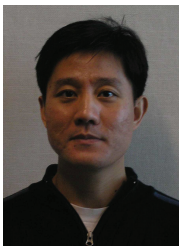
[1] Amazon. Amazon Auto Scaling. http://aws.amazon.com/autoscaling/ 2014.
[2] Azure. Autoscaling and Microsoft Azure. https://msdn.microsoft.com/en-us/library/hh680945(v=pandp.50).aspx 2012.
[3] Google. Autoscaler - Compute Engine Google Cloud Platform. https://cloud.google.com/compute/docs/autoscaler/ 2015.
[4] ProfitBricks. ProfitBricks Live Vertical Scaling. https://www.profitbricks.de/sites/default/files/PB_LiveVerticalScaling_by_ProfitBricks_EN.pdf 2012.
[5] Lunacloud. lunacloud. http://www.lunacloud.com 2014.
[6] Ninefold. ninefold. https://ninefold.com 2013.
[7] Linux-kvm. KVM CPU Hotplug. http://www.linux-kvm.org/page/CPUHotPlug.
[8] Backdrift. Xen Memory Hotplug. http://backdrift.org/xen-memory-hot-add-and-remove.
[9] Al-Roomi May, Al-Ebrahim Shaikha, Buqrais Sabika, and Ahmad Imtiaz. Cloud Computing Pricing Models: A Survey. *International Journal of Grid and Distributed Computing*, 6(5):93–106, 2013.
[10] Orna Agmon Ben-Yehuda, Muli Ben-Yehuda, Assaf Schuster, and Dan Tsafrir. Deconstructing Amazon EC2 Spot Instance Pricing. In *Proc. of IEEE CloudCom*, 2011.
[11] Qian Wang, Kui Ren, and Xiaoqiao Meng. When Cloud meets eBay: Towards Effective Pricing for Cloud Computing. In *Proc. of IEEE INFOCOM*, 2012.
[12] Xiaoxi Zhang, Zhiyi Huang, Chuan Wu, Zongpeng Li, and Francis C.M. Lau. Online Auctions in IaaS Clouds: Welfare and Profit Maximization with Server Costs. In *Proc. of ACM SIGMETRICS*, 2015.
[13] Zhiyi Huang and Anthony Kim. Welfare Maximization with Production Costs: a Primal Dual Approach. In *Proc. of the ACM-SIAM SODA*, 2015.
[14] Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. CloudScale: Elastic Resource Scaling for Multi-tenant Cloud Systems. In *Proc. of ACM SoCC*, 2011.
[15] Hiep Nguyen, Zhiming Shen, Xiaohui Gu, Sethuraman Subbiah, and John Wilkes. Agile: Elastic Distributed Resource Scaling for Infrastructure-as-a-Service. In *Proc. of the USENIX ICAC*, 2013.
[16] M Alicherry and T V Lakshman. Network Aware Resource Allocation in Distributed Clouds. In *Proc. of IEEE INFOCOM*, 2012.
[17] Carlee Joe-Wong, Soumya Sen, Tian Lan, and Mung Chiang. Multi Resource Allocation: Fairness-Efficiency Tradeoffs in a Unifying Framework. In *Proc. of IEEE INFOCOM*, 2012.
[18] L. L. Andrew E. Thereska M. Lin, A. Wierman. Dynamic Right-Sizing for Power-Proportional Data Centers. In *Proc. of IEEE INFOCOM*, 2011.
[19] J. Llorca Y. Jin A. Sala L. Jiao, A. Tulino. Smoothed Online Resource Allocation in Multi-Tier Distributed Cloud Networks. In *IPDPS*, 2016.
[20] Sharrukh Zaman and Daniel Grosu. Combinatorial Auction-based Allocation of Virtual Machine Instances in Clouds. *Journal of Parallel and Distributed Computing*, 73(4):495–508, 2013.
[21] Linquan Zhang, Zongpeng Li, and Chuan Wu. Dynamic Resource Provisioning in Cloud Computing: A Randomized Auction Approach. In *Proc. of IEEE INFOCOM*, 2014.
[22] Hong Zhang, Hongbo Jiang, Bo Li, Fangming Liu, Athanasios V. Vasilakos, and Jiangchuan Liu. A Framework for Truthful Online Auctions in Cloud Computing with Heterogeneous User Demands. In *Proc. of IEEE INFOCOM*, 2013.
[23] Weijie Shi, Chuan Wu, and Zongpeng Li. RSMOA: A Revenue and Social Welfare Maximizing Online Auction for Dynamic Cloud Resource Provisioning. In *Proc. of IWQoS*, 2014.
[24] Weijie Shi, Linquan Zhang, Chuan Wu, Zongpeng Li, and Fracis C.M. Lau. An Online Auction Framework for Dynamic Resource Provisioning in Cloud Computing. In *Proc. of ACM SIGMETRICS*, 2014.
[25] Jeff Bilmes Rishabh Iyer. Submodular Optimization with Submodular Cover and Submodular Knapsack Constraints. In *Proc. of NIPS*, 2013.
[26] Uriel Feige and Jan Vondr ak. Approximation algorithms for allocation problems: Improving the factor of 1 - 1/e. In *Proc. of IEEE FOCS*, 2006.
[27] Niv Buchbinder, Moran Feldman, and Roy Schwartz. Online submodular maximization with preemption. In *Proc. of ACM-SIAM SODA*, 2015.
[28] Moran Feldman, Joseph (Seffi) Naor, , and Roy Schwartz. Improved competitive ratios for submodular secretary problems. In *Proc. of APPROX/RANDOM*, 2011.
[29] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. In *Proc. of ACM SOSP*, 2003.
[30] Hao Chen, Michael C. Caramanis, and Ayse K. Coskun. Reducing the Data Center Electricity Costs Through Participation in Smart Grid Programs. In *Proc. of IGCC*, 2014.
[31] Bhavani Krishnan, Hrishikesh Amur, Ada Gavrilovska, and Karsten Schwan. VM Power Metering: Feasibility and Challenges. *ACM SIGMETRICS Performance Evaluation Review*, 38(3):56–60, 2010.
[32] JAman Kansal, Feng Zhao, Jie Liu, Nupur Kothari, and Arka A. Bhattacharya. Virtual Machine Power Metering and Provisioning. In *Proc. of ACM SoCC*, 2010.
[33] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. Google cluster-usage traces: format + schema. Technical report. http://code.google.com/p/googleclusterdata/wiki/TraceVersion2.
[34] A. Blum, A. Gupta, Y. Mansour, and A. Sharma. Welfare and Profit Maximization with Production Costs. In *Proc. of IEEE FOCS*, 2011.
[35] Shuchi Chawla, Jason D Hartline, David L Malec, and Balasubramanian Sivan. Multi-Parameter Mechanism Design and Sequential Posted Pricing. In *Proc. of ACM STOC*, 2010.

1 . Xiaoxi Zhang received her B.E. degree in 2013, from Department of Electronics and Information Engineering, Huazhong University of Science and Technology. She is currently a Ph.D. candidate in Department of Computer Science, The University of Hong Kong. Her research interests include cloud computing, network economics, and online algorithms.

2 . Zhiyi Huang received his B.E. degree in 2008 from Department of Computer Science and Technology, Tsinghua University, and his Ph.D. degree in 2013 from Department of Computer and Information Science, University of Pennsylvania. He is currently an assistant professor of Computer Science at the University of Hong Kong, China. His research interests include theoretical computer science, algorithmic game theory, differential privacy, and online algorithms.



3 . Chuan Wu received her B.E. and M.E. degrees in 2000 and 2002 from Department of Computer Science and Technology, Tsinghua University, China, and her Ph.D. degree in 2008 from the Department of Electrical and Computer Engineering, University of Toronto, Canada. She is currently an associate professor in the Department of Computer Science, The University of Hong Kong, China. Her research interests include cloud computing and online/mobile social network.



4 . Zongpeng Li received his B.E. degree in Computer Science and Technology from Tsinghua University in 1999, his M.S. degree in Computer Science from University of Toronto in 2001, and his Ph.D. degree in Electrical and Computer Engineering from University of Toronto in 2005. He is currently a professor in the Department of Computer Science in the University of Calgary. His research interests are in computer networks, particularly in network optimization, multicast algorithm design, network game theory and network coding.

5 . Francis C.M. Lau received his Ph.D. degree from the Department of computer science from University of Waterloo. He is currently a professor in computer science in The University of Hong Kong. He is the editor-in-chief of the Journal of Interconnection Networks. His research interests include computer systems, networks, programming languages, and application of computing in arts.