Batch Auction Design for Cloud Container Services

Yu He[1], Lin Ma[1], Ruiting Zhou[2(✉)], and Chuanhe Huang[1]

[1] School of Computer Science, Wuhan University, Wuhan, China
{heyu,linma,huangch}@whu.edu.cn
[2] School of Cyber Science and Engineering, Wuhan University, Wuhan, China
ruitingzhou@whu.edu.cn

Abstract. Cloud containers represent a new, light-weight alternative to virtual machines in cloud computing. A user job may be described by a container graph that specifies the resource profile of each container and container dependence relations. This work is the first in the cloud computing literature that designs efficient market mechanisms for container based cloud jobs. Our design targets simultaneously incentive compatibility, computational efficiency, and economic efficiency. It further adapts the idea of batch online optimization into the paradigm of mechanism design, leveraging agile creation of cloud containers and exploiting delay tolerance of elastic cloud jobs. The new and classic techniques we employ include: (i) compact exponential optimization for expressing and handling non-traditional constraints that arise from container dependence and job deadlines; (ii) the primal-dual schema for designing efficient approximation algorithms for social welfare maximization; and (iii) posted price mechanisms for batch decision making and truthful payment design. Theoretical analysis and trace-driven empirical evaluation verify the efficacy of our container auction algorithms.

Keywords: Cloud container · Online auction

1 Introduction

Cloud computing offers cloud users with utility-like computing services in a pay-as-you-go fashion. Computing resources including CPU, RAM, disk storage and bandwidth can be leased in custom packages with minimal management overhead. Virtualization technologies help cloud providers pack cloud resources into a functional package for serving user jobs. Such packages used to be dominantly virtual machines (VMs), until the recent emergence of *cloud containers*, *e.g.*, Google Container Engine (largest Linux container) [6], Amazon EC2 Container Service (ECS) [2], Aliyun Container Service [1], Azure Container Service [3], and IBM Containers. Compared with general-purpose VMs, containers are more flexible and lightweight, enabling efficient and agile resource management. Applications are encapsulated inside the containers without running in a dedicated

operating system [25]. A representative cloud container is only megabytes in size and takes seconds to start [23,25], while launching a VM may take minutes. In the era of using VMs, VMs remain open throughout the life of the job. Because of the transient nature of a container, jobs could be separated into several containers, and resource allocation is more convenient.

A complex cloud job in practice is often composed of *sub-tasks.* For example, a social game server [19] typically consists of a front-end web server tier, a load balancing tier and a back-end data storage tier; a network security application may consist of an intrusion detection system (IDS), a firewall, and a load balancer. Different sub-tasks require different configurations of CPU, RAM, disk storage and bandwidth resources. Each sub-task can be served by a custom-made container following the resource profile defined by the cloud user [14]. Some cloud containers are to be launched after others finish execution, following the input-output relation of their corresponding tasks. Such a dependence relation among containers is captured by a *container (dependence) graph.* For example, in Amazon ECS, a cloud user submits a job definition including resource requirements, type of docker image, a container graph, and environment variables. ECS then provisions the containers on a shared operating system, instead of running VMs with complete operating systems [22] (Fig. 1).
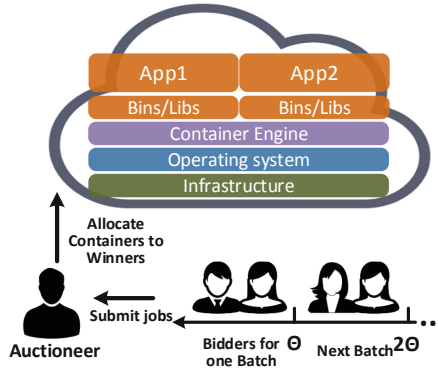

Fig. 1. Batch auction of cloud jobs running on containers.

In the growing cloud marketplace (*e.g.*, Amazon EC2 and ECS), fixed pricing mechanisms and auctions complement each other. While the former is simple to implement, the latter can automatically discover the market price of cloud services, and allocate resources to cloud users who value them the most [28]. A series of recent cloud auction mechanisms implicitly aim at non-elastic cloud jobs. These include both one-round cloud auctions [28] and online cloud auctions [21,29]. In both cases, the provider processes each bid immediately and commits to an irrevocable decision. Furthermore, even in the online auctions, users' service time window is *predefined* by start and finish times in the bid [21,29].

A large fraction of cloud jobs are elastic in nature, as exemplified by big data analytics and Google crawling data processing. They require a certain computing job to be completed without demanding always-on computing service, and may tolerate a certain level of delay in bid acceptance and in job completion. For example, since Sanger *et al.* published the first complete genome sequence of an organism in 1977, DNA sequencing algorithms around the globe currently produce 15 billion gigabytes of data per annum, for cloud processing [9]. A typical job of DNA testing takes 4 hours to complete, while the user is happy to receive the final result anytime in a few days after job submission [17].

Given that bids from cloud users can tolerate a certain level of delay in bid admission, it is natural to revise the common practice of immediate irrevocable decision making in online cloud auctions. We can group bids from a common time window into a batch, and apply batch bid processing to make more informed decisions on all bids from the same batch simultaneously. Actually, if one considers only online optimization and not online auctions, then such batch processing has already been studied in operations research, such as online scheduling to minimize job completion time [8], and scheduling batch and heterogeneous jobs with runtime elasticity in cloud computing platforms [16].

We study efficient auctions for cloud container services, where a bid submitted by a cloud user specifies: (i) the container dependence graph of the job; (ii) the resource profile of each container; (iii) the deadline of the job; and (iv) the willingness to pay (bidding price). Cloud containers can be agilely created and dropped to handle dynamic sub-tasks in cloud jobs; it becomes practically feasible to suspend and resume a sub-task. As long as a container is scheduled to run for a sufficient number of time slots, its sub-task will finish.

This work advances the state-of-the-art in the literature of cloud auctions along two directions. **First**, while *batch algorithms* have been extensively studied in the field of online optimization, to the authors' knowledge, this work is the first that studies *batch auctions* in online auction design. **Second**, this work is the first cloud auction mechanism designed for container services, with expressive bids based on container graphs. Our mechanism design simultaneously targets the following goals: (i) *truthfulness*, *i.e.*, bidding true valuation for executing its job on the cloud maximizes a user's utility, regardless of how other users bid; (ii) *time efficiency*, we require that all components of the auction run in polynomial time, for practical implementation; (iii) *expressiveness*; the target auction permits a user to specify its job deadlines, desired cloud containers, and inter-container dependence relations; and (iv) *social welfare maximization*; *i.e.*, the overall 'happiness' of the cloud-ecosystem is maximized.

Corresponding to the above goals, our auction design leverages the following classic and new techniques in algorithm and mechanism design. For effectively expressing and handling user bids that admit deadline specification and container dependence graphs, we develop the technique of *compact exponential Integer Linear Programs (ILPs)*. We transform a natural formulation of the social welfare optimization ILP into a compact ILP with an exponential number of variables corresponding to valid container schedules. Although such a reformulation substantially inflates the ILP size, it lays the foundation for later efficient

primal-dual approximation algorithm design, helping deal with non-conventional constraints that arise from container dependence and job deadlines, whose dual variables are hard to interpret and update directly. A combinatorial sub-routine later helps identify good container schedules efficiently without exhaustively enumerating them.

Towards truthful batch auction design, we leverage the recent developments in *posted price auctions* [26]. At a high level, such an auction maintains an estimate of marginal resource prices for each resource type, based on expected supply-demand. Then upon decision making of each batch of bids, it chooses bids whose willingness to pay surpasses the estimated cost to serve them, based on resource demand of the container graph and projected marginal prices of resources. A winning user is charged with such estimated cost, which is independent from its bidding price. Truthfulness is hence guaranteed based on Myerson's celebrated characterization of truthful mechanisms [18].

The social welfare maximization problem in our container auction is NP-hard even in the offline setting, with all inputs given at once. A third key element of our cloud container auction is the classic primal-dual schema for designing efficient approximation algorithms, with rigorous guarantee on worst case performance. This is further integrated with the posted price framework, in that the marginal resource prices are associated with dual variables. The primal dual framework relies on a sub-routine that computes the optimal schedule of a given container graph, based on static resource prices (fixing dual variables, update primal solution). We apply dynamic programming [11] and graph traversal algorithms, for designing the sub-routine for (i) service chain type jobs from network function virtualization, and (ii) general jobs with arbitrary topologies in their container graphs. We evaluate the effectiveness of our cloud container auction through rigorous theoretical analysis and trace-driven simulation studies.

In the rest of the paper, we discuss related work in Sect. 2, and introduce the auction model in Sect. 3. The container auction is presented and analyzed in Sects. 4 and 5 separately. Section 6 presents simulation studies, and Sect. 7 concludes the paper.

2   Related Work

There exist a large body of studies in recent cloud computing literature on cloud auction design. Shi *et al.* [20] studied online auctions where users bid for heterogeneous types of VMs and proposed RSMOA, an online cloud auction for dynamic resource provisioning. Zhang *et al.* [27] propose COCA, a framework for truthfull online cloud auctions based on a monotonic payment rule and utility-maximizing allocation rule. These auction mechanisms are all confined to the solution space of immediately accepting or rejecting an arriving bid. To our knowledge, this work is the first that designs batch-type online auctions, both in the field of cloud computing and in the general literature of auction mechanism design.

In terms of batch-type online algorithms, Bitton *et al.* [7] study online scheduling in a batch processing system. Kumar *et al.* [16] design scheduling

mechanisms for runtime elasticity of heterogeneous workloads. They propose Delayed-LOS and Hybrid-LOS, two algorithms that improve an existing dynamic programming based scheduler. These work possess a resemblance to ours in terms of postponing immediate response for more informed decision making, although they focus on algorithm design only and do not consider payments or incentive compatibility.

Along the direction of posted price algorithms and mechanisms, Huang *et al.* [15] study online combinatorial auctions with production costs. They show that posted price mechanisms are incentive compatible and achieve optimal competitive ratios. Etzion *et al.* [10] present a simulation model to extend previous analytical framework, focusing on a firm selling consumer goods online using posted price and auction at the same time. This work was inspired in part by this line of recent developments on using posted prices to achieve effective resource allocation and bid-independent charges.

3   The Cloud Container Auction Model

We consider a public cloud in which the cloud provider (auctioneer) manages a pool of R types of resources, as exemplified by CPU, RAM, disk storage and bandwidth, and the capacity of resource-r is $\mathbb{C}_r$. Integer set $\{1, 2,..., X\}$ is denoted by $[X]$. There are I cloud users arriving in a large time span $\{1, 2, ..., T\}$, acting as bidders in the auction. Each user i submits a job bid that is 4-tuple:

$$\Pi_i = \{\mathcal{W}_i, t_i, d_i, B_i\}. \tag{1}$$

Here $\mathcal{W}_i$ is the workload of user i, t_i is arrival time of user i, and its required deadline for job completion is d_i. B_i is user i's overall willingness-to-pay for finishing its job by d_i (Fig. 2).
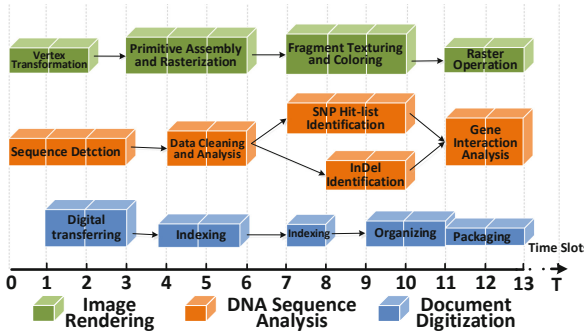

Fig. 2. Container graphs for cloud jobs from [4].

According to users workload, the detailed information will be obtained by cloud platform. Such as the number of sub-tasks of the job M, and each sub-task requires a container to process, thus m is also the number of containers. The

container graph G_i that describes the dependence among sub-tasks. The number of requested time slots for each sub-task N_{im}. Each sub-task can be suspended and resumed, as long as the total execution time accumulates to N_{im}. h_{im}^r is the resource configuration of container$m$ of user i.

A *(container) schedule* is a mapping from resources and time slots to cloud containers, serving accepted cloud jobs to meet their deadlines. We postpone immediate decision making on the bids, to judiciously exploit cloud jobs' tolerable delays in bid admission. We group bids from every θ time slots into a batch, resulting in Q batches within the large time span T. Let ρ_q be the number of users arriving within batch $q \in Q$. A binary variable x_i indicates whether user i's bid is accepted (1) or not (0). Another binary variable $z_{im}(t)$ indicates whether to execute user i's sub-task m at time slot t (1) or not (0); it encodes a schedule of user i's job. The cloud provider further computes a payment P_i to charge for a winning cloud user i. The holy grail of auction mechanism design is *truthfulness*, the property that greatly simplifies bidder strategy space and analysis of the auction mechanism (Table 1).

Table 1. List of notations

I	# of users
T	# of time slots
$\mathbb{C}_r$	capacity of type-r resource
M	# of sub-tasks/containers of one job
$\mathcal{W}_i$	Workload of user i
G_i	Dependence graph of user i's sub-tasks
N_{im}	# of time slots requested by user i's container m
h_{im}^r	Demand of type-r resource by user i's container m
t_i	User i's arrival time
d_i	Deadline of user i's bid
B_i	Bidding price of user i's bid
x_i	Accept the user i's bid(1) or not(0)
ρ_q	# of users arriving within batch q
$f_{ir}^S(t)$	Total type-r resource occupation of schedule in Γ_i for slot t
θ	# of time slots within one batch interval
$z_{im}(t)$	Allocated user i's container m at time slot t(1) or not(0)
$w_r(t)$	Amount of allocated type-r resource at time t
$y_r(t)$	Availablity of type-r resource at time slot t
$\kappa_r(t)$	Marginal price of type-r resource at time slot t
F_r	Minimum value of user's valuation per unit of type-r resource
D_r	Maximum value of user's valuation per unit of type-r resource
Γ_i	The set of valid schedules for each user
u_i	User i's utility

Lemma 1. *Let $Pr(B_i)$ denote the probability of bidder i winning an auction and B_{-i} be the bidding price except i. A mechanism is truthful if and only if the following hold for a fixed B_{-i} [12]:*

(1) $Pr(B_i)$ is monotonically non-decreasing in B_i;
(2) bidder i is charged by $B_i Pr(B_i) - \int_0^{B_i} Pr(B_i)dB$.

Lemma 1 can be explained in this orientation: the payment charged to bidder i for a fixed B_i is independent of B_i. We will use this mode to design a posted price function in Sect. 4. Since we meet the challenge that when we consider that online batch auction decisions are to be made based on hitherto information only. If user i's job is accepted, its utility is $u_i = v_i - P_i$, which equals $u_i = B_i - P_i$ under truthful bidding. The cloud provider's utility is $\sum_{i \in [I]} P_i$. The *social welfare* that captures the overall utility of both the provider and the users is $(\sum_{i \in [I]} B_i x_i - \sum_{i \in [I]} P_i) + (\sum_{i \in [I]} P_i)$. With payments cancelling themselves, the social welfare is simplified to $\sum_{i \in [I]} B_i x_i$.

Under the assumption of truthful bidding, the Social Welfare Maximization problem in our cloud container auction can be formulated into the following Integer Linear Program (ILP):

$$\text{maximize} \quad \sum_{i \in [I]} B_i x_i \tag{2}$$

subject to:

$$\theta \lceil \frac{t_i}{\theta} \rceil x_i \leq t z_{im}(t), \forall t, \forall m, \forall i : t_i \leq t, \tag{2a}$$

$$t z_{im}(t) \leq d_i x_i, \forall t, \forall m, \forall i : t_i \leq t, \tag{2b}$$

$$t z_{im}(t) \leq t' z_{im'}(t'), \tag{2c}$$

$$\forall t, t', \forall i : task \ m' \ arrives \ later \ than \ task \ m,$$

$$N_{im} x_i \leq \sum_{t \in [T]} z_{im}(t), \forall m, \forall i, \tag{2d}$$

$$\sum_{i \in [I]} \sum_{m \in [M]} h_{im}^r z_{im}(t) \leq \mathbb{C}_r, \forall r, \forall t, \tag{2e}$$

$$x_i, z_{im}(t) \in \{0, 1\}, \forall i, \forall t, \forall m. \tag{2f}$$

Constraints (2a) and (2b) ensure that user i's job is scheduled to execute only between its start time and deadline. (2c) enforces inter-task dependence of user i's sub-tasks, and (2d) makes sure that the total number of allocated time slots for each container is sufficient to finish the corresponding sub-task. Constraint (2e) states that the total amount of type-r resource utilized at time slot t is capped by system capacity.

Even in the offline setting with all inputs given, ILP (2) is still NP-hard. This can be verified by observing that with constraints (2e) and (2f) alone, and ILP (2) degrades into the classic knapsack problem known to be NP-hard. We resort to the classic primal-dual schema [24] for efficient algorithm design. We

first reformulate ILP (2) into an equivalent *compact exponential* version, to hide the non-conventional constraints that arise from container dependence and job deadlines, whose dual variables would be hard to interpret and to update:

$$\text{maximize} \quad \sum_{i \in [I]} \sum_{S \in \Gamma_i} B_i x_{iS} \tag{3}$$

subject to:

$$\sum_{i \in [I]} \sum_{S : t \in S} f_{ir}^S(t) x_{iS} \leq \mathbb{C}_r, \forall r \in [R], \forall t \in [T], \tag{3a}$$

$$\sum_{S \in \Gamma_i} x_{iS} \leq 1, \forall i \in [I], \tag{3b}$$

$$x_{iS} \in \{0, 1\}, \forall i \in [I], \forall S \in \Gamma_i. \tag{3c}$$

In the compact exponential ILP above, Γ_i represents a set of valid schedules for sub-tasks that meet constraints (2a), (2b), (2c) and (2d). B_{iS} represents the bidding price of user i for schedule $S \in \Gamma_i$. Since a time slot can serve two or more containers, we let $f_{ir}^S(t)$ represent the total type-r resource occupation of user i's schedule S in t. Constraints (3a) and (3b) correspond to (2e) and (2f) in ILP (2). We relax the integer constraints $x_i \in \{0, 1\}$ to $x_i \geq 0$, and introduce dual variable vectors u_i and $\kappa_r(t)$ to constraints (3a) and (3b) respectively, to formulate the dual of the LP relaxation of ILP (3).

$$\text{minimize} \sum_{i \in [I]} u_i + \sum_{t \in [T]} \sum_{r \in [R]} \mathbb{C}_r \kappa_r(t) \tag{4}$$

subject to:

$$u_i \geq B_i - \sum_{r \in [R]} \sum_{t \in S} f_{ir}^S(t) \kappa_r(t), \forall i \in [I], \forall S \in \Gamma_i, \tag{4a}$$

$$\kappa_r(t), u_i \geq 0, \forall i \in [I], \forall r \in [R], \forall t \in [T]. \tag{4b}$$

While the reformulated ILP (3) is compact in its form, it has an exponential number of variables that arise from the exponential number of feasible job schedules. Correspondingly, the dual problem (4) has an exponential number of constraints. Even there are exponential number of schedule options are available, we only select polynomial number of them to compute the approximately optimal objective through a sub-algorithm (Sect. 4.2). We next design an efficient auction algorithm that efficiently solves the primal and dual compact exponential ILPs simultaneously, pursuing social welfare maximization (in the primal solution) while computing payments (in the dual solution).

4   Batch Auction Algorithm for Social Welfare Maximization

4.1   The Batch Algorithm

Departing from traditional online auctions that make immediate and irrevocable decisions, our auction mechanism takes a batch processing approach to handle

user bids. In each batch, we aim to choose a subset of bids to accept, and to dynamically provision containers, through choosing a feasible assignment of the primal variable x_{iS}. We let $x_{iS} = 1$, if user i's bid with schedule S is accepted, then allocate time slots according to the schedule, and update the amount of resources occupied.

We now focus on batch bid processing and container provisioning for social welfare maximization. A set of dual constraints exists for each primal variable x_{iS}. We minimize the increase of the dual objective and maintain dual feasibility (4a) by leveraging *complementary slackness*. Once the dual constraint (4a) is tight with user i's schedule S (KKT conditions [29]), the primal variable x_{iS} is updated to 1. According to constraint (4b), the dual variable $u_i \geq 0$. Therefore, we let u_i be the maximum of 0 and the RHS of (4a). If $u_i = 0$, the bid is rejected.

$$u_i = \max\{0, \max_{s \in \Gamma_i}(B_i - \sum_{r \in [R]} \sum_{t \in S} f_{ir}^S(t)\kappa_r(t))\}, \forall i \in \rho_q \tag{5}$$

$\kappa_r(t)$ can be viewed as the marginal price per unit of type-r resource at t. Consequently, $\sum_{r \in [R]} \sum_{t \in S} f_{ir}^S(t)\kappa_r(t))$ represents the cost of serving user i by schedule S, and $\{B_i - \sum_{r \in [R]} \sum_{t \in S} f_{ir}^S(t)\kappa_r(t)\}$ is the utility of user i's bid. The above assignment (5) chooses the schedule which can maximize the job's utility.

Our auction strives to reserve a certain amount of resource for potential high-value bids in the future. Careful implementation of such an intuition through dual price design is crucial in guaranteeing a good competitive ratio of the auction.

Let D_r and F_r represent the maximum and minimum user valuation per unit of type-r resource respectively. $w_r(t)$ denotes the amount of allocated type-r resource at t. We define the marginal price $\kappa_r(t)$ to be an increasing function of $w_r(t)$:

$$\kappa_r(w_r(t)) = \frac{\sigma F_r}{k}(\frac{kD_r}{\sigma F_r})^{\frac{w_r(t)}{\mathbb{C}_r}} \tag{6}$$

$$\text{where } D_r = \max_{i \in [I]} \frac{B_i}{\sum_{m \in [M]} N_{im}h_{im}^r}; F_r = \min_{i \in [I]} \frac{B_i}{\sum_{m \in [M]} N_{im}h_{im}^r}.$$

The initial price of each type-r resource should be low enough such that any user's bid can be accepted; otherwise there might be a large amount of idle resource. Thus we decrease the starting price by a coefficient k, satisfying: $k - 1 = \max_{r \in [R]} ln(\frac{kD_r}{\sigma F_r})$ and $k > 1$. The detailed explanation of k is given in Theorem 5. For all $w_r(t) < \mathbb{C}_r$, $\kappa_r(t) < D_r$, and it will reach D_r when $w_r(t) = \mathbb{C}_r$. In that case, the cloud provider will not further allocate any type-r resource. The parameter is defined as the minimum occupation rate of all kinds of resources within slots T, *i.e.*,

$$\sigma = \min_{r \in R} \frac{\sum_{i \in [I]} \sum_{m \in [M]} h_{im}^r N_{im}x_i}{\mathbb{C}_r T}$$

We assume that there are enough cloud users to potentially exhaust resources within each slot. Thus the resource occupation rate σ is close to 1.

We design a batch auction algorithm A_{batch} in Algorithm 1 with container scheduling algorithm A_{sub} in Algorithm 2 or Algorithm 3, which can select optimal container scheduling under different circumstances. A_{batch} defines the posted price function and initializes the primal and dual variables in line 1. Upon the arrival of ρ_q users within batch q, we first select the schedule that maximize users' utility through the dual oracle (lines 4–6). $\sum_{r\in[R]}\sum_{t\in[s_i]}f_{ir}^s(t)\kappa_r(t)$ in line 7 is viewed as the weighted total resource demand by user i, thus $\frac{B_i}{\sum_{r\in[R]}\sum_{t\in[s_i]}f_{ir}^s(t)\kappa_r(t)}$ can be interpreted as the value for a unit resource of user i, and we select the bid μ with the maximum unit resource value. If user μ obtains positive utility, we update the primal variable x_μ and dual variable $\kappa_r(t)$ according to μ's schedule s_μ (lines 9–16).

Algorithm 1. A Primal-dual Posted Price Auction A_{batch}

1: Initialize $x_i = 0$, $z_{im}(t) = 0$, $w_r(t) = 0$, $u_i = 0$, $\kappa_r(t) = \frac{\sigma F_r}{k}$, $\forall i \in [I], r \in [R], t \in [T], S \in \Gamma_i, \psi = \varnothing$;

2: **Group a set of ρ_q users within θ time slots**;

3: **while** $\psi \neq \rho_q$ **do**

4:    **for all** $i \in \rho_q \setminus \psi$ **do**

5:       $(u_i, S_i, cost_i, \{f_{ir}^S(t)\}) = A_{sub}(\{\Pi_i\}, \{\mathbb{C}_r\}, \{w_r(t)\}, \{\kappa_r(t)\})$;

6:    **end for**

7:    $\mu = argmax_{i\in\rho_q\setminus\psi}\{\dfrac{B_i}{\sum_{r\in[R]}\sum_{t\in[s_i]}f_{ir}^s(t)\kappa_r(t)}\}$;

8:    **if** $u_\mu > 0$ **then**

9:       $x_\mu = 1$;

10:      Accept user μ's bid, allocate resources according to S_i, and charge $cost_i$ for user i;

11:      **update:** $\psi = \psi \bigcup \{\mu\}$;

12:      **for all** $t \in S_\mu$ **do**

13:         $w_r(t) = w_r(t) + f_{\mu r}^S(t)$;

14:         $\kappa_r(t) = \frac{\sigma F_r}{k}(\frac{kD_r}{\sigma F_r})^{\frac{w_r(t)}{\mathbb{C}_r}}, \forall r \in [R]$;

15:      **end for**

16:   **else**

17:      Reject user μ's bid, and delete user μ from the set ρ_q.

18:   **end if**

19: **end while**

4.2   Sub-algorithm of Auction Mechanism

Our container scheduling algorithms A_{sub} only selects utility-maximizing schedules for each job, rather than an exponential number of schedules. Therefore, we compute a schedule that minimizes the cost of serving the job.

In our auction mechanism, dependence graph of user tasks is complicated to handle. We first focus on a relatively small, yet representative case of jobs from Network Function Virtualization [13], where each container graph is a service chain. We exploit the sequential chain structure to design A_{sub1} Algorithm 2

with polynomial time complexity, based on dynamic programming. By choosing time slots that can ensure right operating sequence and minimum payment for each sub-task, the first two nested `for` loops select minimum-cost schedule for containers (lines 3–10). Then the second `for` loop updates the cost and schedule for each container m (lines 11–15); line 17 updates the cost and utility of user i's schedule S_i at the end.

Container graphs in practice can be more complex than a chain structure. For general jobs with arbitrary container graph topology, the container scheduling problem is NP-hard, as proven in Theorem 1; we design A_{sub2} in Algorithm 3 to solve the optimization. Lines 2–8 in Algorithm 3 sort available time slots by $c_m(t)$. Then A_{sub2} employs Depth-First Search (DFS) (line 9). We adapt the DFS procedure with improvements to select available time slots with minimum cost in a recursive process that decides a container schedule. Truthfulness requires solving the problem exactly, and our algorithm runs in exponential time to the number of sub-tasks in a job, which is mostly small and can be viewed as a constant in practice.

Algorithm 2. A_{sub1}: Container Graph Scheduling - Service Chains

Input: bidding language $\{\Pi_i\}, \{\mathbb{C}_r\}, \{\kappa_r(t)\}, \{w_r(t)\}$;
Output: u_i; S_i, $cost_i$, $\{f_{ir}^S(t)\}$;

1: Initialize $S_i = \varnothing$; $f_{ir}^S(t) = 0, \forall t \in [T]$;
2: **for all** $m \in [M]$ **do**
3:     **for all** $t_s \in [\theta\lceil\frac{t_i}{\theta}\rceil + \sum_1^{m-1} N_{im}, d_i - \sum_m^M N_{im}]$ **do**
4:         **for all** $t_e \in [t_s + N_{im}, d_i - \sum_{m+1}^M N_{im}]$ **do**
5:             $c_m(t) = \sum_{r \in [R]} h_{im}^r \kappa_r(t), \forall t \in [t_s, t_e]$;
6:             Select N_{im} slots with minimum $c_m(t)$ and $w_r(t) + h_{im}^r \le \mathbb{C}_r, \forall r \in [R]$ to τ_m;
7:             $\Delta_m = [\Delta_m \ \tau_m]$;
8:             $p_m(t_s, t_e) = \sum_{t \in \tau_m} c_m(t)$;
9:         **end for**
10:    **end for**
11:    **for all** $t_s \in [\theta\lceil\frac{t_i}{\theta}\rceil + \sum_1^{m-1} N_{im}, d_i - \sum_m^M N_{im}]$ **do**
12:        $pay = min_{t_e < t_s}\{p_{m-1}(:, t_e)\}, \tau_m \in [\Delta_m]$;
13:        $p_m(t_s, t_e) = p_m(t_s, t_e) + pay$;
14:        $S_i = [S_i \ \tau_m], \ f_{ir}^S(t) = f_{ir}^S(t) + h_{im}^r$;
15:    **end for**
16: **end for**
17: **Update:** $cost_i = min_{t_s, t_e}(p_m(t_s, t_e))$; $u_i = B_i - cost_i$;

Theorem 1. *In each batch of container based auction, given fixed resource prices, choosing the schedule of sub-tasks with minimum cost with a general container graph is NP-hard.*

Proof: We construct a polynomial-time reduction to sub-task scheduling from the classic NP-hard problem *subset sum*: $\max_{x_i} \sum_{i=1}^n c_i x_i$, subject to $\sum_{i=1}^n c_i x_i \le V, x_i \in \{0, 1\}$.

Given a set $\{c_1, c_2, ..., c_n\}$ and a objective V, our problem reduces to an instance of $K = (|M| = n, h_{im}^r = c_i, \mathbb{C}_r = V)$, in which each user's job has M types of containers with 1 slot requirement, and the resource pool contains one type of resource. We should put as many containers in one slot with lowest price as possible. If a polynomial-time algorithm solves the capacitated container scheduling problem K, it will solve the corresponding subset sum problem as well, and vice versa. Consequently, the *subset sum* problem can be viewed as a special case of the sub-task scheduling problem, which must be NP-hard as well. □

5   Analysis of Auction Mechanism

5.1   Truthfulness of the Batch Algorithm

Theorem 2. *The batch auction in Algorithm 1 that computes resource allocation and payment is truthful.*

Proof: In Algorithm 1, upon the arrival of user i and our posted price mechanism, the payment P_i that user i needs to pay to the cloud provider (if its bid is accepted) depends only on the amount of resources that has been allocated and user i's demand. Which means, user i's bidding price does not affect its payment. Therefore, leveraging Lemma 1, our online batch auction is truthful.   □

Algorithm 3. A_{sub2}: Container Graph Scheduling - General Topology

Input: bidding language $\{\Pi_i\}$, $\{\mathbb{C}_r\}$,$\{\kappa_r(t)\}$, $\{w_r(t)\}$;
Output: u_i; S_i, $\{f_{ir}^S(t)\}$, $cost_i$;
 1: Initialize $S_i = \varnothing$; $f_{ir}^S(t) = 0, \forall t \in [T]$; c_{min}=INF;
 2: **for all** $m \in [M]$ **do**
 3:      **for all** $t \in [\theta\lceil \frac{t_i}{\theta} \rceil, d_i]$ **do**
 4:          $c_m(t) = \sum_{r \in [R]} h_{im}^r \kappa_r(t), \forall t \in [\theta\lceil \frac{t_i}{\theta} \rceil, d_i]$;
 5:          Sort slots with $w_r(t) + h_{im}^r \leq \mathbb{C}_r, \forall r \in [R]$ according to $c_m(t)$ to τ_m;
 6:          $p_m(t_s, t_e)$=$\sum_{t \in \tau_m} c_m(t)$;
 7:      **end for**
 8: **end for**
 9: Calling **Depth-First Search(m)** to find the container schedule S_i and resource allocaton $\{f_{ir}^S(t)\}$ with minimum cost c_{min};
10: **Update:** $cost_i = c_{min}$;$u_i = B_i - cost_i$;

5.2   Solution Feasibility of the Batch Algorithm

Theorem 3. *Algorithm 1 computes a feasible solution to ILP (2).*

Proof: x_i is initialized to 0 and updated to 1 only (line 10 in Algorithm A_{batch}), so the solution of our algorithm is binary valued, and satisfies constraint (2f). Container scheduling algorithms A_{sub1} and A_{sub2} guarantee that the schedule S

for each user's bid satisfies constraints (2a), (2b), (2c) and (2d). For container provisioning and scheduling, both A_{sub1} and A_{sub2} select time slots satisfying resource capacity limits, $f_{ir}^S(t) + w_r(t) \leq \mathbb{C}_r, \exists t \in [T]$. Hence constraints (2e) is satisfied. In summary, the solution we obtain is feasible for ILP (2).   □

Theorem 4. *The computational complexity of Batch Algorithm 1 to ILP (2) is polynomial time.*

Proof: We first consider the case of service chains ($A_{sub} = A_{sub1}$). Line 1 in Algorithm 1 takes linear time to initialize the price function, primal and dual variables. According to user arrivals, the `while` loop iterates ρ_q times to find user μ with maximum unit resource value, then updates the primal and dual variables in linear time. In the `for` loop (lines 4–6), Algorithm A_{sub1} iterates ρ_q^2 times to select the best schedule of users with maximum utility. Then each A_{sub1} in Algorithm 2 takes $\eta = (d_i - t_s - \sum_{m \in [M]} N_{im})^2$ steps to compute the price of each time slot and examine resource capacity limits for each container. Thus it takes $O(M\eta^2)$ to choose the utility maximization schedule for user i. In summary, the running time of A_{batch} with A_{sub1} is $O(M\eta^2\rho_q^2)$. We next consider the case of general container graphs ($A_{sub} = A_{sub2}$). The complexity of A_{sub2} is exponential to the number of containers in the container graph, which is mostly small and an be viewed as a constant.   □

5.3   Competitive Ratio of the Batch Algorithm

The *competitive ratio* is an upper-bound ratio of the optimal social welfare achieved by ILP (2) to the social welfare achieved by our batch algorithm. The primal-dual framework in our batch algorithm design enables a competitive ratio analysis based on LP duality theory [24]. Let P_i and D_i be the primal objective value (3) and dual objective value (4) after accepting user i's job, respectively. Then we let P_0 and D_0 be the initial objective values of primal (3) and dual (4) programs, and $P_0 = 0$. P_I and D_I are the final primal and dual objective values achieved by our algorithm A_{batch}. Let OPT_1 and OPT_2 be the optimal objective values of (2) and (3), respectively. Since the compact exponential ILP is equivalent to the original ILP, we have $OPT_1 = OPT_2$, which is hereafter referred to as OPT.

Lemma 2. *According to the initial marginal price of each time slot, the initial dual objective value D_0 is at most $\frac{1}{k}OPT$.*

Proof: We first show a lower bound on the optimal social welfare:

$$OPT \geq \sigma \sum_{r \in [R]} \sum_{t \in [T]} F_r \mathbb{C}_r.$$

Recall that we let σ denote the minimum resource occupation rate within slots T. F_r can be interpreted as the minimum social welfare generated by a job per unit of type-r resource and per unit of time. Therefore, $\sigma \sum_{r \in [R]} \sum_{t \in [T]} F_r \mathbb{C}_r$ is the minimum social welfare generated by all users.

According to dual (4) and marginal price function (6):

$$D_0 = \sum_{t \in [T]} \sum_{r \in [R]} \mathbb{C}_r \kappa_r(0) = \sum_{t \in [T]} \sum_{r \in [R]} \mathbb{C}_r \left(\frac{\sigma F_r}{k} \right)$$
$$= \frac{1}{k} \sum_{t \in [T]} \sum_{r \in [R]} F_r \mathbb{C}_r \sigma \leq \frac{1}{k} OPT$$

Therefore, the initial dual objective value D_0 is bounded by $\frac{1}{k} OPT$. $\qquad \square$

Lemma 3. *If there is a constant $\alpha > 1$, and the primal and dual objective values increased by handling each user i's job satisfy $P_i - P_{i-1} \geq \frac{1}{\alpha}(D_i - D_{i-1})$, then the batch algorithm is $\frac{k}{k-1}\alpha$-competitive.*

Proof: Since the inequality is satisfied for all users, we sum up the inequality of each user i:

$$P_I = \sum_i (P_i - P_{i-1}) \geq \frac{1}{\alpha} \sum_i (D_i - D_{i-1}) = \frac{1}{\alpha}(D_I - D_0).$$

According to weak duality and Lemma 2, $D_I \geq OPT$ and $D_0 \geq \frac{1}{k} OPT$. Therefore,

$$P_I \geq \frac{k-1}{k\alpha} OPT_1 = \frac{k-1}{k\alpha} OPT_2,$$

with the fact that $P_0 = 0$. Our batch algorithm is $\frac{k}{k-1}\alpha$-competitive. $\qquad \square$

Next we will define an *Allocation Price Relation* to identify this α. If the Allocation Price Relation is satisfied by α, the objective values achieved by our algorithm A_{batch} guarantee the inequality in Lemma 3.

Definition 1. *The Allocation Price Relation for $\alpha \geq 1$ is that $\kappa_r^{i-1}(t)(w_r^i(t) - w_r^{i-1}(t)) \geq \frac{1}{\alpha}\mathbb{C}_r(\kappa_r^i(t) - \kappa_r^{i-1}(t)), \forall i \in [I], \forall r \in [R], \forall t \in [s]$, where $\kappa_r^i(t)$ represents the price of type-r resource after processing user i's job. $w_r^i(t)$ is the total amount of allocated type-r resource after accepting user i.*

Lemma 4. *For a given $\alpha \geq 1$, if the price function $\kappa_r(t)$ satisfies $\kappa_r^{i-1}(t)(w_r^i(t) - w_r^{i-1}(t)) \geq \frac{1}{\alpha}\mathbb{C}_r(\kappa_r^i(t) - \kappa_r^{i-1}(t)), \forall i \in [I], \forall r \in [R], \forall t \in [l]$, then Algorithm A_{batch} $P_i - P_{i-1} \geq \frac{1}{\alpha}(D_i - D_{i-1}), \forall i \in [I]$.*

Proof: If bid i is rejected, $P_i - P_{i-1} = D_i - D_{i-1} = 0$. Then we assume that bid i is accepted and let s be the job schedule of user i. Knowing that our algorithm accepts a bid when constraint (4a) is tight, $B_{is} = u_i + \sum_{r \in [R]} \sum_{t \in s} f_{ir}^s(t)\kappa_r^{i-1}(t)$. So the increase of primal objective is:

$$P_i - P_{i-1} = u_i + \sum_{r \in [R]} \sum_{t \in s} \kappa_r^{i-1}(t)(w_r^i(t) - w_r^{i-1}(t))$$

According to dual (4), the increase of dual objective is:

$$D_i - D_{i-1} = u_i + \sum_{r \in [R]} \sum_{t \in s} \mathbb{C}_r(\kappa_r^i(t) - \kappa_r^{i-1}(t))$$

Since we have $u_i \geq 0$, $\alpha \geq 1$ and $\kappa_r^{i-1}(t)(w_r^i(t) - w_r^{i-1}(t)) \geq \frac{1}{\alpha}\mathbb{C}_r(\kappa_r^i(t) - \kappa_r^{i-1}(t))$:

$$
\begin{aligned}
P_i - P_{i-1} &= u_i + \sum_{r \in [R]} \sum_{t \in s} \kappa_r^{i-1}(t)(w_r^i(t) - w_r^{i-1}(t)) \\
&\geq u_i + \frac{1}{\alpha} \sum_{r \in [R]} \sum_{t \in s} \mathbb{C}_r(\kappa_r^i(t) - \kappa_r^{i-1}(t)) \\
&\geq \frac{1}{\alpha}(u_i + \sum_{r \in [R]} \sum_{t \in s} \mathbb{C}_r(\kappa_r^i(t) - \kappa_r^{i-1}(t))) \\
&= \frac{1}{\alpha}(D_i - D_{i-1})
\end{aligned}
$$

$\square$

We next try to find the α_r for type-r resource that satisfies the Allocation Price Relationship. Thus the α is the maximum value among all α_r. Since the capacity of type-r resource is larger than a user demand, we let $dw_r(t)$ denote $w_r^i(t) - w_r^{i-1}(t)$. We first prepare with the following definition.

Definition 2. *The Differential Allocation Price Relation for A_{batch} with a given parameter $\alpha_r \geq 1$ $\kappa_r(t)dw_r(t) \geq \frac{1}{\alpha_r}\mathbb{C}_r d\kappa_r(t), \forall i \in [I], \forall r \in [R], \forall t \in [s]$.*

Lemma 5. *The marginal price defined in (5) satisfies the Differential Allocation Price Relation, and we can get $\alpha_r = ln(\frac{kD_r}{\sigma F_r})$.*

Proof: The derivative of the marginal price function is:

$$
d\kappa_r(t) = \kappa_r'(w_r(t))dw_r(t) = \frac{\sigma F_r}{k}(\frac{kD_r}{\sigma F_r})^{\frac{w_r(t)}{\mathbb{C}_r}} \frac{1}{\mathbb{C}_r} ln(\frac{kD_r}{\sigma F_r})dw_r(t).
$$

Therefore:
$$
\begin{aligned}
\frac{\sigma F_r}{k}(\frac{kD_r}{\sigma F_r})^{\frac{w_r(t)}{\mathbb{C}_r}} &\geq \frac{\mathbb{C}_r}{\alpha_r} \frac{\sigma F_r}{k}(\frac{kD_r}{\sigma F_r})^{\frac{w_r(t)}{\mathbb{C}_r}} \frac{1}{\mathbb{C}_r} ln(\frac{kD_r}{\sigma F_r}) \\
&\geq \frac{1}{\alpha_r}(\frac{\sigma F_r}{k}(\frac{kD_r}{\sigma F_r})^{\frac{w_r(t)}{\mathbb{C}_r}} ln(\frac{kD_r}{\sigma F_r})), \Rightarrow \alpha_r \geq ln(\frac{kD_r}{\sigma F_r})
\end{aligned}
$$

Thus we can obtain $\alpha_r = ln(\frac{kD_r}{\sigma F_r})$. $\square$

Lemma 6. *The batch auction Algorithm A_{batch} is $\frac{k}{k-1}\alpha$-competitive in social welfare with $\alpha = max_{r \in [R]}ln(\frac{kD_r}{\sigma F_r})$.*

Proof: Lemma 5 implies that $\alpha = max_{r \in [R]}ln(\frac{kD_r}{\sigma F_r})$ satisfies the Differential Allocation Price Relation of all kinds of resources. Since the above mentioned, $dw_r(t) = w_r^i(t) - w_r^{i-1}(t)$,

$$
\begin{aligned}
d\kappa_r(t) &= \kappa_r'(w_r(t))dw_r(t) = \kappa_r'(w_r(t))(w_r^i(t) - w_r^{i-1}(t)) \\
&= \kappa_r^i(t) - \kappa_r^{i-1}(t).
\end{aligned}
$$

Thus, we can obtain $\alpha = max_{r \in [R]}ln(\frac{kD_r}{\sigma F_r})$ due to the Allocation Price Relationship. $\square$

Theorem 5. *If* k *satisfies* $k - 1 = \max_{r \in [R]} ln(\frac{kD_r}{\sigma F_r})$ *and* $k > 1$, *the competitive ratio of batch auction algorithm is minimum, and is equal to* k.

Proof: We assumpt that $\varpi = max_{r \in [R]}(\frac{D_r}{\sigma F_r})$. By Lemma 5, the competitive ratio of our batch algorithm is $\frac{k}{k-1}\alpha = \frac{k}{k-1}ln(\frac{kD_{r*}}{\sigma F_{r*}}) = \frac{k}{k-1}ln(k\varpi)$, thus the competitive ratio is a function of k. Differentiating $\frac{k}{k-1}ln(k\varpi)$ on k is:

$$(\frac{k}{k-1}ln(k\varpi))' = \frac{k - 1 - ln(k\varpi)}{(k-1)^2}$$

It suffices to show that $(\frac{k-1-ln(k\varpi)}{(k-1)^2})'$ is positive as $k \in [1, \propto]$. When k satisfies $k - 1 = ln(k\varpi)$ and $k > 1$, we can obtain the minimum competitive ratio:

$$\frac{k}{k-1}ln(k\varpi) = \frac{k}{ln(k\varpi)}ln(k\varpi) = k.$$

If we consider the case that competition for resource is intense, the σ is close to 1. When D_r/F_r is *2*, the competitive ratio is close to 2.85, as illustrated in Fig. 3. □
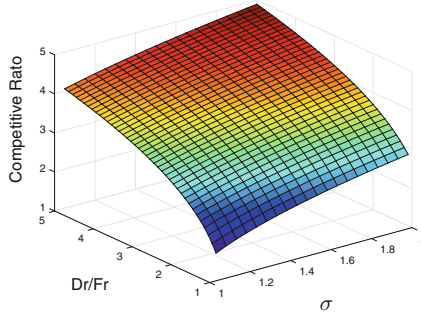

Fig. 3. Theoretical competitive ratio.

5.4   Setting the Batch Interval θ

In our batch auction, the more jobs we handle in a batch, the more information we have for social welfare maximization. Nonetheless, we can't over-extend the length of a batch given that cloud jobs have deadlines to meet. Precise optimization of the job interval length is left as future research, and we provide here a brief discussion only. Let W_i be the time required to execute a job i, and $\bar{\rho}$ be the expected number of user arrivals per slot. In general, an appropriate length of a batch round depends on values of W_i, deadline d_i and arrival time t_i of user

$i, i \in [I]$. We can set a target threshold on the job loss rate (*e.g.*, 10%), the ratio of jobs who cannot meet their deadlines due to delayed bid admission.

Assume that job processing time and $d_i - t_i$ are normally distributed, by $N(a_1, b_1^2)$ and $N(a_2, b_2^2)$, respectively. The max waiting time for each user equals $d_i - t_i - W_i$, and is also normally distributed as $N(a_1 - a_2, b_1^2 + b_2^2)$. If user i's maximum waiting time $\theta_i < \theta$, we will lose this job. Thus the length of batch interval θ can be set by (for $\leq 10\%$ job loss):

$$\{\max \theta, s.t. \sum_{t=1}^{\theta} F(\theta) \leq 0.1, \theta \in \{1, 2, 3, 4, ...\}.\}$$

where $F(\theta)$ is the Normal cumulative distribution function of θ.

6   Performance Evaluation

We evaluate our batch auction algorithm A_{batch} and its sub-algorithms by trace-driven simulation studies. We leverage Google cluster data [5], which captures rich information on user jobs, including start time, resource demand (CPU, RAM and Disk), and duration. We translate cloud job requests into bids, arriving in a one month time window. We assume that each sub-task consumes [1, 10] slots, and each time slot is one hour. Job deadlines are set randomly between the arrival time and system end time. The demand of resources (CPU, RAM and Disk) is set randomly between [0, 1], with the resource capacity set to 50. We use *user density* to express the number of users in one batch interval, arriving as a Poisson process.

A. Comparison with Classic Online Auctions
We compare our batch auction with a traditional online auction in terms of social welfare, as shown in Fig. 4. Under the same simulation settings, we compare the two algorithms in 10 different sets of simulation studies. Our batch auction achieves a higher social welfare in all of them. Intuitively, the online auction processes bids in a FCFS fashion, while the batch auction considers most attractive bids first in each batch. Figure 5 shows another set of comparisons. The superiority of batch auction remains clear, with different number of time slots and user density. Social welfare fluctuates with the increase of the number of users and user density. The batch auction performs better with higher user density. The influence of different batch interval θ for the batch performance is illustrated in Fig. 6. As θ grows, the cloud social welfare initially grows as well. However, when θ is too large so that more bids are lost due to delays, as we can see in Fig. 6, a gradual decrease in the percentage of winners leads to a decreasing trend in social welfare. Recall that in the analysis of θ in the previous section, a too large θ is not suitable for our batch auction.
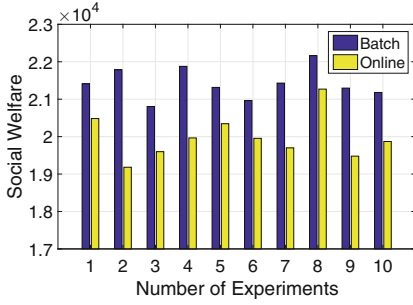
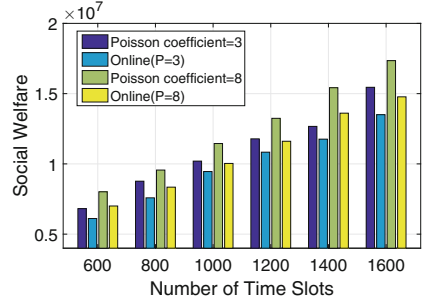**Fig. 4.** Social welfare, batch vs. online auctions.

Fig. 5. Social welfare achieved by A_{batch}.
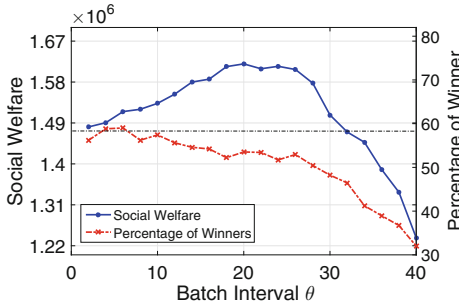

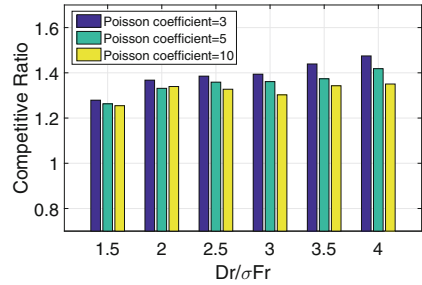**Fig. 6.** Social welfare and percentage of winners, varying batch length.

Fig. 7. Competitive ratio of auction algorithm A_{batch}.

B. Competitive Ratio of the Batch auction

Next we study the competitive ratio achieved by our batch auction. As we proved in Theorem 6, the competitive ratio depends on $D_r/\sigma F_r$. Figure 7 shows that the competitive ratio grows as $D_r/\sigma F_r$ increases. The observed competitive ratio is much better than the theoretical bound and remains smaller than 2; this can be partly explained by the fact that the theoretical bound is a pessimistic worst case scenario uncommon in practice. The ratio fluctuates with user population and sightly decreases with as $D_r/\sigma F_r$ decreases. The batch auction favors intensive user arrivals.

C. Performance of A_{batch}: The Role of System Parameters

We next examine the resource occupation ratio σ (defined in Sect. 3) of our batch auction. As we can see in Fig. 8, under different numbers of time slots and user density, the resource occupation ratio of the batch auction mechanism is constantly beyond 90% and often close to 1. Figure 9 demonstrates the variation of social welfare with different number of users. The social welfare grows mildly but steadily as the number of users and the number of time slots grow.
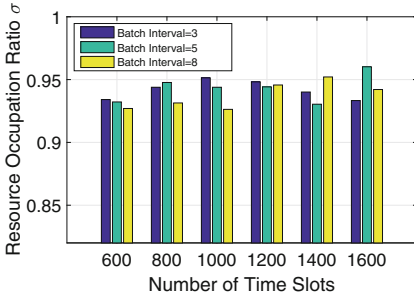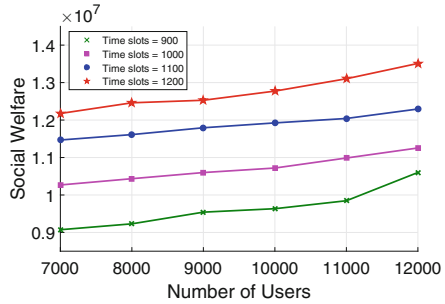
Fig. 8. Resource occupation ratio σ.

Fig. 9. Social welfare of A_{batch}, varying user population.

7   Conclusion

This work is the first in the cloud computing literature that studies efficient auction algorithm design for container services. It is also the first that designs batch online auctions, aiming at more informed decision making through exploiting the elastic nature of cloud jobs. We combined techniques from compact exponential optimization, posted price mechanisms, and primal-dual algorithms for designing a cloud container auction that is incentive compatible, computationally efficient, and economically efficient. As future directions, it will be interesting to study (i) cloud jobs that cannot be suspended and resumed; (ii) pre-processing of cloud jobs with tight deadlines to choose between immediate acceptance or delayed processing of their bids; and (iii) cloud container auctions that make revocable decisions, where a partially executed cloud job may or may not contribute towards social welfare of the cloud.

References

1. Aliyun Container Engine. http://cn.aliyun.com/product/contain-erservice
2. Amazon ECS. https://aws.amazon.com/cn/ecs/
3. Azure Container. https://azure.microsoft.com/en-us/services/container-service/
4. Batch Applications. https://github.com/Azure/azure-content/blob/master/articl es/batch/batch-hpc-solutions.md
5. Google Cluster Data. https://code.google.com/p/googlecl-usterdata
6. Google Container Engine. http://cloud.google.com/container-engine/
7. Bitton, S., Emek, Y., Kutten, S.: Efficient dispatching of job batches in emerging clouds. In: Proceedings of IEEE INFOCOM (2018)
8. Chen, B., Deng, X., Zang, W.: On-line scheduling a batch processing system to minimize total weighted job completion time. J. Comb. Optim. **8**(1), 85–95 (2004)
9. Dove, A.: Life science technologies: biology watches the cloud. Science **340**(6138), 1350–1352 (2013)
10. Etzion, H., Moor, S.: Simulation of online selling with posted-price and auctions: comparison of dual channel's performance under different auction mechanisms. In: Proceedings of HICSS (2008)

11. Golin, M.J., Rote, G.: A dynamic programming algorithm for constructing optimal prefix-free codes with unequal letter costs. IEEE Trans. Inf. Theor. **44**(5), 1770–1781 (1998)

12. Gopinathan, A., Li, Z.: Strategyproof auctions for balancing social welfare and fairness in secondary spectrum markets. In: Proceedings of the IEEE INFOCOM (2011)

13. Gu, S., Li, Z., Wu, C., Huang, C.: An efficient auction mechanism for service chains in the NFV market. In: Proceedings of IEEE INFOCOM (2016)

14. He, S., Guo, L., Guo, Y., Wu, C.: Elastic application container: a lightweight approach for cloud resource provisioning. In: Proceedings of IEEE International Conference on Advanced Information Networking and Applications (2012)

15. Huang, Z., Kim, A.: Welfare maximization with production costs: a primal dual approach. In: Proceedings of ACM-SIAM SODA (2015)

16. Kumar, D., Shae, Z., Jamjoom, H.: Scheduling batch and heterogeneous jobs with runtime elasticity in a parallel processing environment. In: Proceedings of IEEE IPDPSW (2012)

17. Mohamed, N.M., Lin, H., Feng, W.: Accelerating data-intensive genome analysis in the cloud. In: Proceedings of BICoB (2013)

18. Myerson, R.B.: Optimal auction design. Math. Oper. Res. **6**(1), 58–73 (1981)

19. RightScale: Social Gaming in the Cloud: A Technical White Paper (2013)

20. Shi, W., Wu, C., Li, Z.: RSMOA: a revenue and social welfare maximizing online for dynamic cloud resource provisioning. In: Proceedings of IEEE IWQoS (2014)

21. Shi, W., Zhang, L., Wu, C., Li, Z., Lau, F.: An online auction framework for dynamic resource provisioning in cloud computing. In: Proceedings of ACM SIG-METRICS (2014)

22. Tosatto, A., Ruiu, P., Attanasio, A.: Container-based orchestration in cloud: state of the art and challenges. In: Proceedings of Ninth International Conference on Complex, Intelligent, and Software Intensive Systems (2015)

23. Waibel, P., Yeshchenko, A., Schulte, S., Mendling, J.: Optimized container-based process execution in the cloud. In: Panetto, H., Debruyne, C., Proper, H., Ardagna, C., Roman, D., Meersman, R. (eds.) On the Move to Meaningful Internet Systems, OTM 2018 Conferences, OTM 2018. Lecture Notes in Computer Science, vol. 11230, pp. 3–21. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-02671-4_1

24. Williamson, D.P.: The primal-dual method for approximation algorithms. Math. Program. **91**(3), 447–478 (2002)

25. Xu, X., Yu, H., Pei, X.: A novel resource scheduling approach in container based clouds. In: Proceedings of IEEE ICCS (2014)

26. Zhang, E., Zhuo, Y.Q.: Online advertising channel choice - posted price vs. auction (2011)

27. Zhang, H., Jiang, H., Li, B., Liu, F., Vasilakos, A.V., Liu, J.: A framework for truthful online auctions in cloud computing with heterogeneous user demands. IEEE Trans. Comput. **65**(3), 805–818 (2016)

28. Zhang, L., Li, Z., Wu, C.: Dynamic resource provisioning in cloud computing: a randomized auction approach. In: Proceedings of IEEE INFOCOM (2014)

29. Zhang, X., Huang, Z., Wu, C., Li, Z., Lau, F.: Online auctions in IaaS clouds: welfare and profit maximization with server costs. In: Proceedings of ACM SIG-METRICS (2015)