A Reinforcement Learning Approach to Price Cloud Resources With Provable Convergence Guarantees

Hong Xie, *Member, IEEE*, and John C. S. Lui, *Fellow, IEEE*

***Abstract*— How to generate more revenues is crucial to cloud providers. Evidences from the Amazon cloud system indicate that "dynamic pricing" would be more profitable than "static pricing." The challenges are: *How to set the price in real-time so to maximize revenues? How to estimate the price dependent demand so to optimize the pricing decision?* We first design a discrete-time based dynamic pricing scheme and formulate a Markov decision process to characterize the evolving dynamics of the price-dependent demand. We formulate a revenue maximization framework to determine the optimal price and theoretically characterize the "structure" of the optimal revenue and optimal price. We apply the Q-learning to infer the optimal price from historical transaction data and derive sufficient conditions on the model to guarantee its convergence to the optimal price, but it converges slowly. To speed up the convergence, we incorporate the structure of the optimal revenue that we obtained earlier, leading to the VpQ-learning (Q-learning with value projection) algorithm. We derive sufficient conditions, under which the VpQ-learning algorithm converges to the optimal policy. Experiments on a real-world dataset show that the VpQ-learning algorithm outperforms a variety of baselines, i.e., improves the revenue by as high as 50% over the Q-learning, speedy Q-learning, and adaptive real-time dynamic programming (ARTDP), and by as high as 20% over the fixed pricing scheme.**

***Index Terms*— Cloud resources pricing, reinforcement learning (RL), value projection.**

I. Introduction

OVER the past decade, we have witnessed the rapid growth of cloud computing services such as Amazon Web Services, Google Cloud, and Microsoft Azure. Cloud computing services have generated tremendous economic values for various companies. It was reported that the annualized revenue of Microsoft Azure is \$18.9 billion [1], and that of Amazon Web Services is \$29.13 billion [2]. What is more promising is that the cloud computing service market is still growing [3] and a variety of companies such as Google are investing to expand their cloud computing services [4]. In general, cloud computing serves as a new computing paradigm, where cloud providers supply flexible and on-demand computing resources to cloud users.

How to generate more revenues from the cloud computing platform is crucial to cloud providers. This raises a fundamental question for cloud providers: *How to price their cloud resources?* What makes this question interesting is that each cloud provider only has a finite number of cloud resources. Typically, major cloud providers such as Amazon Web Services, Google Cloud, and Microsoft Azure, practice a fixed pricing scheme, i.e., each unit of resource is charged at a fixed price. However, evidence from the Amazon EC2 Spot Instances [5] indicate that dynamic pricing would be more profitable. In general, a dynamic pricing scheme means that each unit of resource may be charged at different prices at different time. In particular, the price may depend on the demand of cloud users or the utilization of resources.

Designing a dynamic pricing scheme has the following challenges. The first one is: *How to set the price of cloud resources in real-time so as to maximize revenues?* Note that the demand depends on the price and the supply depends on the resource usage level. Intuitively, if the demand is large or the resource usage level is high, the cloud provider may set a high price. Also, the current pricing decisions may influence the future demand due to the network externality effect in cloud computing applications [6]. Under this externality effect, the price not only influences the immediate revenue but also influences the future revenue. This unique feature of the externality effect makes previous auction-based dynamic pricing schemes [7]–[12] and online algorithm based dynamic pricing schemes [13] cannot be applied, because they did not consider the impact of price on future revenue. The cloud provider has full knowledge on her resource usage level, while the price-dependent demand is usually unknown. This raises the second challenge: *How to estimate the price dependent demand so to optimize the pricing decision?* One natural approach to address this challenge is via reinforcement learning (RL) [14], [15]. However, previous RL-based pricing schemes [14], [15] are not satisfactory. In particular, Du *et al.* [14] did not provide convergence guarantees to the optimal price. Lacking convergence guarantees is a serious flaw for cloud resource pricing problems, because it may cause great revenue loss to the cloud provider. The work [15] did not consider the resource constraint, which is unrealistic. We aim to design RL algorithms to price cloud resources dynamically, providing convergence guarantees and achieving fast learning speed. Our contributions are as follows.

Manuscript received 22 August 2020; revised 22 February 2021; accepted 18 May 2021. Date of publication 10 June 2021; date of current version 1 December 2022. The work of J. C. S. Lui was supported in part by the General Research Fund (GRF) under Grant 14200420. *(Corresponding author: Hong Xie.)*

Hong Xie is with the College of Computer Science, Chongqing University, Chongqing 400044, China (e-mail: xiehong2018@cqu.edu.cn).

John C. S. Lui is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong (e-mail: cslui@cse.cuhk.edu.hk).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TNNLS.2021.3085088.

Digital Object Identifier 10.1109/TNNLS.2021.3085088

1) We formulate a mathematical model to quantify the network externality effect in cloud computing applications [6] and characterize the evolving dynamics of the price-dependent demand.
2) We design a discrete-time-based dynamic pricing scheme. We formulate a revenue maximization framework via a Markov decision process (MDP) to determine the optimal price. We theoretically characterize the monotonicity of the optimal revenue and optimal price.
3) We apply the Q-learning to infer the optimal price from historical transaction data and derive sufficient conditions on the model to guarantee its convergence to the optimal price, but it converges slowly. To speed up the convergence, we add a value projection component into the Q-learning, which preserves the monotonicity of the optimal revenue that we obtained earlier, leading to the VpQ-learning algorithm. We derive sufficient conditions, under which the VpQ-learning algorithm converges to the optimal policy. These conditions serve as guidance to set parameters for the VpQ-learning algorithm.
4) We conduct experiments on a real-world cloud trace dataset and compare the VpQ-learning algorithm with a variety of baselines. Experimental results show that the VpQ-learning algorithm improves the revenue over the Q-learning, Speedy Q-learning and ARTDP by as high as 50%, and improves the revenue over the fixed pricing scheme by as high as 20%.

The remainder of this article is organized as follows. Section II presents the related work. Section III presents our dynamic pricing framework. Section IV theoretically characterizes the optimal revenue and price. Section V presents algorithms to infer the optimal price from historical transaction data. Section VI presents the experimental results. Section VII concludes.

II. Related Work

A variety of dynamic pricing schemes have been proposed. Auction based dynamic pricing schemes have been studied extensively [7]–[12]. This approach has explored various aspects and settings such as social welfare and profit maximization [10], soft deadline aware pricing [11], virtual cluster pricing [12], etc. In general, this approach models the demand as bids from users and applies the auction framework to characterize the interaction between the cloud operator and users. The objectives of this approach include achieving some desired properties such as truthfulness of users in reporting bids, competitive ratio analysis, etc. We consider a different model, where users do not need to bid. In particular, we use a Markov decision process to model the price-dependent demand as well as the interaction between the cloud operator and users. Our objective is different from the auction approach in that we focus on characterizing the structure of the optimal revenue and optimal price, as well as inferring the optimal price from historical transaction data. Xu and Li [16] designed a dynamic pricing scheme via the stochastic dynamic programming approach. In particular, they considered a continuous cloud resource model and applied the optimal control framework to optimize the price. However, we consider a discrete cloud resource model, to which their scheme cannot be applied. Our model captures the positive network externality effect, which is not captured in their model. Furthermore, their work focused on the scenario that the cloud provider has full knowledge of the model. However, one of our focuses is to address the challenge that some parameters are unknown to the cloud provider. Specifically, we design RL algorithms to infer the optimal price from historical transaction data. Zhang *et al.* [13] applied the online algorithm approach to study the optimal posted pricing schemes. They derived a variety of competitive ratios for their schemes. Their design objective is to maximize the social welfare of a cloud system, while our design objective is to maximize the revenue of a cloud provider. Note that it is technically nontrivial to extend their framework to maximize the revenue.

A variety of works applied the RL framework [17] to improve the system performance of cloud systems. Tesauro *et al.* [18] showed that the RL framework can effectively reduce both transients and switching delays in resource allocations. Dutreilh *et al.* [19] applied the RL framework to address the resource dimensioning issue in cloud computing. They implemented the RL algorithm into cloud controllers. Xu *et al.* [20] applied the RL framework to automate the virtual machine configuration process. Barrett *et al.* [21] applied RL framework to address the application scaling problem in cloud systems. Salahuddin *et al.* [22] showed that the RL framework can efficiently minimize the overhead of resource provisioning in vehicular clouds. Arabnejad *et al.* [23] conducted a comparison study of various RL algorithms for the fuzzy cloud auto-scaling problems. Different from their works, we apply the RL framework to "price" cloud resources. A few works applied RL to price cloud resources. Xu *et al.* [24] studied the pricing problem under the competition among multiple cloud operators. They considered one leading cloud provider and several follow-ups cloud providers. They used Q-learning to derive an optimal policy for the leading operator to set the price. To make the problem tractable, they significantly simplified the interactions among the cloud operator and the users, which is the focus of [14] and [15] and our work. In particular, Du *et al.* [14] applied deep RL to price cloud resources. They formulated a model with a large state space and used a neural network to parametrize the optimal pricing policy. They learned a policy without any theoretical guarantee via training the neural network. Their scheme is appropriate for the warm-start scenario where a vast amount of training data is available. Our work focuses on the cold-start scenario where no training data is available in advance, and the algorithm learns the optimal policy through interacting with users. Furthermore, our work has theoretical guarantees on the convergence to the optimal policy. Shi *et al.* [15] also applied the Q-learning to price the cloud resource. Their model does not consider resource constraint. Our work captures resource constraints.

Note that our pricing model is a finite MDP. There are four notable RL algorithms with theoretical convergence guarantee to infer the optimal price: 1) Q-learning [25]; 2) Speedy Q-learning [26]; 3) Zap Q-learning [27]; and 4) ARTDP [28].

Different from Speedy Q-learning and Zap Q-learning, the VpQL algorithm accelerates the Q-learning algorithm via using the monotone property of the value function. Different from ARTDP, our algorithm is a model-free algorithm. Our value projection idea is generic and can be applied to accelerate the convergence of speedy Q-learning, Zap Q-learning, and ARTDP.

Lastly, we discuss some notable dynamic pricing schemes with externality effects that were proposed in other application domains beyond cloud computing. Several dynamic pricing schemes were proposed for wireless communication networks [29]–[31]. These works focused on understanding the benefits of dynamic schemes through both analytical and numerical studies while learning the optimal dynamic price is not considered. Furthermore, these dynamic pricing schemes were tailored to wireless communication networks, whose physical model is quite different from ours. For example, the externality effect in wireless communication networks is negative which is induced by network congestion. Several dynamic pricing schemes were proposed to handle externality effect induced by social networks [32]–[34]. Similar to our model, the externality effect in these works is positive. They quantified the externality effect at the microlevel, i.e., they modeled how friends of a customer influences this customer's purchase behavior, while we quantify the externality effect at the macro level through the total amount of demand requests. These works provided analytical characterizations on the equilibrium of user behavior, structure of optimal price such as price fluctuations, etc. They did not study how to infer the optimal price. Our focus is on the learning aspect of dynamic pricing. Lastly, we are also aware of some works on dynamic competition among platforms with network externality effect [35]. They studied the product quality competition among platforms and the equilibrium of competition.

III. System Model

In this section, we first present the design of our cloud resource pricing scheme. Then, we formulate a probabilistic model to characterize cloud users' decisions, based on which we formulate an MDP to characterize the cloud provider's decisions. Finally, we formulate a revenue maximization framework and apply the Bellman equations to characterize the optimal pricing scheme. Table I summarizes key notations in this article.

A. Pricing Scheme and Decision Spaces

We consider a cloud provider who provides a total number of $S \in \mathbb{Z}_+$ units of computing resources to a large number of cloud users. Each unit of resource can be interpreted as a single CPU, or a virtual machine, etc. For simplicity of analysis, we assume that all resources are of the same type and each unit of computing resource is indivisible. We consider the scenario that the cloud provider does not have any a-prior knowledge on the demand of cloud users. Instead, we rely on the historical transactions between the cloud provider and users, which enables us to predict (or estimate) the demand. Our objective is to develop a framework to predict the future demand and optimize the price simultaneously. In this subsection, we present the pricing scheme as well as the decision space for the cloud provider and users.

TABLE I

Main Notations

S	the total number of units of cloud resources
t	the index of time slots
a_t	the per unit price in time slot t
D_t	the duration of holding a cloud resource in time slot t
$P_{\text{r}}(a)$	the prob. of a resource holder chooses to renew
$P_{\text{o}}(a)$	the prob. of a potential orderer chooses to order
$\mathcal{S}, s$	the state space, state
N_{p}	the number of potential orderers
$F_{N_{\text{p}}}(k\|s)$	the cumulative distribution function (CDF) of N_{p}
$N_{\text{r}}, N_{\text{o}}$	the number renewal requests in a time slot
$N_{\text{r}}, N_{\text{o}}$	the number of ordering requests in a time slot
N_{d}	the demand
$B(n, \rho)$	the binomial distribution with parameter n and ρ
$P(k; n, \rho)$	the probability mass function (pmf) of $B(n, \rho)$
N_{a}	the number of allocated cloud resource
$r(s, a)$	the reward function
$p(j\|s, a)$	the state transition probability
$R(s_0)$	the revenue
π, π^*	the policy, optimal policy
$v^{\pi}(s), v^*(s)$	the revenue under π, revenue under π^*
d, d^*	the decision rule, optimal decision rule
π^*	the optimal policy
Π^{SD}	a set of all the stationary and deterministic policies
$F(k; n, \rho)$	the CDF of a binomial distribution $B(n, \rho)$
$F_{N_{\text{o}}}(k\|s, a)$	the CDF of the number of ordering requests N_{o}
$F_{N_{\text{d}}}(k\|s, a)$	the CDF of the demand N_{d}
$F_{\text{st}}(k\|s, a)$	the cumulative state transition probabilities
$Q(s, a)$	the state-action dependent total reward function
E_x^f	the elasticity of a function f with respect to x
$\widetilde{\mathcal{A}}$	the discretized action space

We consider a discrete time system. More concretely, we divide the time horizon into equal-length time slots indexed by $t \in \mathbb{N}$. The length of a time slot can be one day, one week, etc. Also, $t = 0$ indicates the initial time slot. At each time slot t, the cloud provider sets a per-unit price $a_t \in \mathcal{A} \triangleq [0, A]$ and the "duration" $D_t \in \mathbb{Z}_+$ of holding on to a resource measured in terms of the number of time slots, where $A \in \mathbb{R}_+$. For example, $(a_0, D_0) = (\$1, 10)$ at $t = 0$ means that at the initial time slot, each unit of resource is sold at \$1 for holding 10 time slots. In the next time slot (i.e., time slot 1), the cloud provider can change the price and duration to $(a_1, D_1) = (\$4, 8)$. When the holding of a cloud resource is expiring at $t = 9$, the user is allowed to renew the resource, however, at the latest price and duration parameter. To complete a transaction (i.e., a resource allocation) at time slot t, the users must make the payment in the current time slot and the allocated resource will take effect from the next time slot.

For mathematical tractability, we fix the duration to be one-time slot, i.e., $D_t = 1, \forall t$. Through this, we simplify the decision of the cloud provider from two-dimension (a_t, D_t) to one dimension a_t. It is a reasonable simplification with strong physical interpretations. Consider the original pricing scheme (a_t, D_t) (i.e., $D_t \geq 1$). One unit of resource, once allocated, the per-slot price is fixed at a_t/D_t for time slots $t + 1, \ldots, t + D_t$. However, under our simplified case, the cloud provider can set different per-slot prices (i.e., a_t in the simplified case) in time slots $t+1,\ldots,t+D_t$. This means that fixing the duration to be one-time slot helps the could provider to further exploit the advantage of dynamic pricing. The operational cost for the cloud provider and the user experience will not be affected significantly, because users are allowed to flexibly renew the cloud resource, i.e., renewal is done on a slot by slot basis instead of renewing after multiple time slots.

For the ease of presentation, each cloud user can hold at most one unit of cloud resource. This does not lose any generality, because we can treat a user holding multiple units of cloud resource as "multiple users". In each decision time slot, we classify users into two types as follows.

1) *resource holders* who hold some resources in the decision time slot. Their decision space is {renew, not renew}. We remark that the renewal will be made at the new price.
2) *potential orderers* who do not hold any resource and arrive at the system in the decision time slot. They can order some cloud resources or leave the system without any ordering, i.e., decision space is {order, not order}.

When the total number of requests (renewal requests plus the ordering requests) exceeds the resource budget S, the renewal requests will be satisfied at a higher priority, and the ordering requests will be served in a first-come-first-serve manner.

Remark: We consider a discrete resource model rather than the continuous one due to the following reasons. Many real-world cloud systems such as Amazon Web Services and Huawei Cloud price their cloud resources using a discrete resource model. Many previous works [7]–[12] also considered discrete resource models, though their models are different from ours (more details on this difference are discussed in Section II).

B. Cloud Users' Decision Model

We use a probabilistic model to characterize the collective decision making of the resource holder population and the potential orderer population respectively. We define the function $P_r : \mathcal{A} \to [0,1]$ to quantify the probability that a cloud resource holder renews the holding of a resource

$$P_r(a) \triangleq \mathbb{P}[\text{a resource holder chooses to renew}|a]$$

where a denotes the latest price set by the cloud provider. Thus the probability that a resource holder does not renew is $1-P_r(a)$. We assume that $P_r(a)$ is decreasing in a to capture that resource holders are less likely to renew under higher prices. Similarly, potential orderers make decisions based on the price as well. We define the function $P_o : \mathcal{A} \to [0,1]$ to quantify the probability that a potential orderer orders a unit of cloud resource:

$$P_o(a) \triangleq \mathbb{P}[\text{a potential orderer chooses to order}|a].$$

Thus, the probability that a potential orderer does not order is $1-P_o(a)$. We like to remark that the cloud provider does not have any *a priori* knowledge on $P_r(a)$ and $P_o(a)$.

Based on cloud users' decision model, we now formulate an MDP to characterize the cloud provider's decision. An MDP is typically characterized by five elements: *states*, *actions*, *rewards*, *state transition probabilities*, and *policies*. We next introduce each of these elements.

C. States and User Demands

We say that the cloud system is at the state $s \in \mathcal{S} \triangleq \{0,1,\ldots,S\}$ if s units of resource are held by cloud users. The state s reflects the utilization level of the cloud system, e.g., $s=S$ means all resources are fully utilized, while $s=0$ means that all resources are available. Due to the positive network externality effect (i.e., cloud users may influence their friends to use the cloud system), the state of the system (or the utilization level of the system) has a positive effect on the demand of users. Intuitively, a higher utilization level implies a larger number of resource holders, and they can influence more friends to use the cloud system. Thus, there will be more potential orderers attracted to the cloud system. Formally, let N_p denote the number of potential orderers. Due to the uncertainty of the arrival of potential orderers, $N_p \in \mathbb{N}$ is a random variable with CDF

$$F_{N_p}(k|s) \triangleq \mathbb{P}[N_p \le k|s].$$

We model the positive externality stochastically as follows.

Assumption 1: The CDF $F_{N_p}(k|s)$ is decreasing in s, i.e., $F_{N_p}(k|s+1) \le F_{N_p}(k|s)$.

Assumption 1 states that when the state is larger (i.e., the number of resource holders is higher), the number of potential orderers is more likely to be larger. Its physical meaning is a positive externality effect in demand, i.e., probabilistically a larger number of resource holders can influence more of their friends to use the cloud system resulting in a larger volume of demand. Essentially, one will see later that the analytical results as well as the VpQ-learning algorithm are tailored to this positive externality effect. If Assumption 1 does not hold, i.e., there is no such positive externality effect, one needs to significantly modify the analytical results as well as the VpQ-learning algorithm. A direct consequence of this assumption is that the expected number potential orderers increases in s, i.e., $\mathbb{E}[N_p|s+1] \ge \mathbb{E}[N_p|s]$ (it follows that $\mathbb{E}[N_p|s] = \sum_{j=1}^{S}[1-F_{N_p}(j|s)]$). Our analysis relies on this mild assumption on the CDF of N_p without assuming any specific instances of $F_{N_p}(j|s)]$. Furthermore, the cloud provider does not have any a priori knowledge on $F_{N_p}(j|s)]$.

We define the demand as the total number of requests (i.e., the number of renewal requests plus the number of ordering requests) from users. Let N_r and N_o denote the number of renewal requests and the number of ordering requests in a time slot respectively. Let N_d denote the demand, formally $N_d \triangleq N_r + N_o$. We define the following notation to simplify the presentation.

Definition 1: Suppose X is a random variable and follows a binomial distribution $B(n,\rho)$. We denote its pmf as

$$P(k;n,\rho) \triangleq \mathbb{P}[X=k] = \binom{n}{k}\rho^k(1-\rho)^{n-k}.$$

With some basic probability argument, we can derive the pmf of the demand N_d as

$$\mathbb{P}[N_d = j|s,a] = \sum_{k=0}^{j} \mathbb{P}[N_r = k|s,a]\mathbb{P}[N_o = j-k|s,a]$$

where $\mathbb{P}[N_r = k|s,a]$ and $\mathbb{P}[N_o = k|s,a]$ are:

$$\begin{aligned}\mathbb{P}[N_r = k|s,a] &= P(k;s,P_r(a))\\ \mathbb{P}[N_o = k|s,a] &= \sum_{i\geq k}(F_{N_p}(i+1|s) - F_{N_p}(i|s))P(k;i,P_o(a)).\end{aligned}$$

The demand function $\mathbb{P}[N_d = j|s,a]$ is unknown to the cloud provider because $P_r(a)$, $P_o(a)$ and $F_{N_p}(i|s)$ are all unknown to her.

D. Actions, Rewards and Station Transitions

Recall that the duration is fixed to be one-time slot. Thus, in each time slot, the action for the cloud provider is to set the price, i.e., set $a \in \mathcal{A}$. The action a impacts two elements of the MDP: 1) the reward (measured in terms of revenue) in the current time slot and 2) the state transition probabilities, which influences the reward in the subsequent time slots. Let us define them individually.

As shown in the last subsection, the action a of the cloud provider influences the demand of users, i.e., N_d. Due to resource budget S, the number of allocated cloud resource (denoted by N_a) for each given demand N_a is

$$N_a = \begin{cases} N_d, & \text{if } N_d < S\\ S, & \text{if } N_d \geq S.\end{cases}$$

At each time slot, the reward to the cloud provider is defined as the expected revenue, i.e., the total expected payments collected from cloud users

$$r(s,a) \triangleq a\mathbb{E}[N_a|s,a] = a\sum_{j\in\mathcal{S}} j\mathbb{P}[N_a = j|s,a]$$

where we derive $\mathbb{P}[N_a = j|s,a]$ as

$$\mathbb{P}[N_a = j|s,a] = \begin{cases}\mathbb{P}[N_d = j|s,a], & \text{if } j < S\\ \mathbb{P}[N_d \geq S|s,a], & \text{if } j = S.\end{cases}$$

We will see later that $\mathbb{P}[N_a = j|s,a]$ is actually the state transition probabilities.

Let $p(j|s,a)$ denote the probability that the system will be at state j at the next time slot, given that the system is at state s in the current time slot and the cloud provider selects an action $a \in \mathcal{A}$. In fact, the system will be at state j at the next time slot if and only if the cloud provider allocates j units of resources in current time slot, namely,

$$p(j|s,a) = \mathbb{P}[N_a = j|s,a].$$

We like to remark that the reward $r(s,a)$ and state transition probability $p(j|s,a)$ are unknown to the cloud provider because the nested demand function $\mathbb{P}[N_d = j|s,a]$ is unknown.

E. Optimal Policies to Maximize Revenue

We consider expected infinite-horizon discounted revenue denoted by $R(s_0)$ for the cloud provider, which is defined as

$$R(s_0) \triangleq \mathbb{E}\left[\sum_{t=0}^{\infty}\delta^t r(s_t,a_t)|s_0\right]$$

where s_0 denotes the initial state and $\delta \in (0,1)$ denotes the discounting factor. Here we take expectation because the states s_t is a random variable due to uncertainty in state transition. We will call it revenue $R(s_0)$ for simplicity. The infinite horizon captures that the cloud provider runs their business for a long time and the discounting factor can be interpreted as inflation in economics. Considering $R(s_0)$ makes sense because, for the cloud provider, the marginal cost of allocating a unit of resource is very small as compared to the cost of maintaining the computing resource in data centers. Cloud providers may be allowed to be at different initial states. For example, a startup cloud provider may have an initial state $s_0 = 0$, meaning that she has not attracted any users yet. A mature cloud provider may have a large initial state, meaning that she has attracted a large number of users already. This article aims to select the optimal actions to maximize the revenue for all possible initial states so that it can be applied to different types of cloud providers.

We apply the conventional notation of policies to characterize the optimal pricing decision. We focus on a special class of policies, i.e., the stationary and deterministic (SD) policies, because they can attain the maximum revenue [36] and they are easy to locate, analyze and implement. An SD policy means that at each time slot the cloud provider applies the same Markovian deterministic decision rule to select the price. Formally, an SD policy π is defined as $\pi \triangleq (d)^\infty$, where the Markovian deterministic decision rule $d : \mathcal{S} \to \mathcal{A}$ prescribes a price for each state. Under a SD policy the action is determined by the current state only, i.e., $a_t = d(s_t)$. Given an initial state $s_0 = s$, we define the revenue of a policy π as

$$v^\pi(s) \triangleq \mathbb{E}[R(s)|\pi].$$

Our objective is to locate the optimal policy denoted by π^*

$$\pi^* \in \arg\max_{\pi\in\Pi^{\mathrm{SD}}} v^\pi(s)$$

where Π^{SD} denotes a set of all the SD policies.

F. Optimality Conditions

We characterize the optimal policy via the Bellman equations [36]. Let $\pi^* = (d^*)^\infty$ denote the optimal policy and let $v^*(s) \triangleq v^{\pi^*}(s)$ denote the optimal revenue function. Then, the $v^*(s)$ is a unique solution of the Bellman equations [36]

$$v^*(s) = \max_{a\in\mathcal{A}}\left\{r(s,a) + \delta\sum_{j\in\mathcal{S}} p(j|s,a)v^*(j)\right\}$$

where $r(s,a)$ is the expected revenue earned in the current time slot, and $\delta\sum_{j\in\mathcal{S}} p(j|s,a)v^*(j)$ is the future revenue that the cloud provider will earn in the subsequent time slots. The physical meaning of the optimal revenue is a balance between the current revenue and the future revenue. Once we have the optimal value function $v^*(s)$, the optimal action satisfies [36]

$$d^*(s) \in \arg\max_{a\in\mathcal{A}} \left\{ r(s,a) + \delta \sum_{j\in\mathcal{S}} p(j|s,a) v^*(j) \right\}.$$

These two equations serve as building blocks to: 1) analyze and locate the optimal policy when model parameters, i.e., $P_r(a)$, $P_o(a)$ and $F_{N_p}(i|s)$ are given and 2) infer the optimal policy when these model parameters are unknown. We next state a scaling property, which will simplify our discussion.

Theorem 1: Suppose $d^*(s)$ and $v^*(s)$ are the optimal price and revenue function respectively. Consider a scaled system such that $\tilde{a} \in [0, \xi A]$, $\tilde{P}_r(\tilde{a}) = P_r(\tilde{a}/\xi)$, and $\tilde{P}_o(\tilde{a}) = P_o(\tilde{a}/\xi)$, where $\xi > 0$. The optimal price and revenue function for the scaled system are $\xi d^*(s)$ and $\xi v^*(s)$ respectively.

Theorem 1 states that as we scale the maximum price linearly, we scale the optimal price and optimal revenue function linearly at the same rate. This implies that we can normalize the action space such that $\mathcal{A} = [0, 1]$. *We present all proofs to lemmas and theorems in our supplementary file.*

Summary: The demands are determined by model parameters $P_r(a)$, $P_o(a)$ and $F_{N_p}(i|s)$, which are all unknown in practice. In the following, we first analyze the optimal policy assuming these model parameters are given. Through this, we gain insights to design RL algorithms to estimate the demand and optimize the pricing decisions simultaneously.

IV. Optimal Pricing & Monotonicity

In this section, we study the case that all model parameters are given. We first characterize the properties of the model. Then we characterize the monotonicity of the optimal revenue and optimal policy.

A. Properties of the Model

To simplify notations, we define the following:

$$F(k; n, \rho) \triangleq \mathbb{P}[X \le k] = \sum_{i=0}^{k} \binom{n}{i} \rho^i (1-\rho)^{n-i}$$

which denotes the CDF of a binomial distribution with parameter $n \in \mathbb{N}_+$, $\rho \in [0, 1]$ and $k \in \{0, 1, \ldots, n\}$. Let

$$F_{N_r}(k|s,a) \triangleq \mathbb{P}[N_r \le k|s,a]$$

denote the CDF of the number of renewal requests N_r. In the following lemma, we derive the closed-form $F_{N_r}(k|s,a)$ and characterize its monotonicity.

Lemma 1: The CDF of N_r can be expressed as $F_{N_r}(k|s,a) = F(k; s, P_r(a))$. For any $k < s$ and $P_r(a) \in (0,1)$, it holds that $F_{N_r}(k|s+1,a) < F_{N_r}(k|s,a)$ and $\partial F_{N_r}(k|s,a)/\partial a > 0$.

Lemma 1 states that the number of renewal requests increases if the number of resource holders (i.e., state s) increases or the cloud provider decreases the price.

Denote the CDF of the number of ordering requests N_o as

$$F_{N_o}(k|s,a) \triangleq \mathbb{P}[N_o \le k|s,a].$$

In the following lemma, we derive its closed-form expression and characterize its monotonicity.

Lemma 2: The CDF of N_o can be expressed as

$$F_{N_o}(k|s,a) = \sum_{i=0}^{\infty} [F_{N_p}(i+1|s) - F_{N_p}(i|s)] F(k; i, P_o(a)).$$

For any $P_o(a) \in (0,1)$ we have $F_{N_o}(k|s+1,a) \le F_{N_o}(k|s,a)$, and $\partial F_{N_o}(k|s,a)/\partial a > 0$ holds.

Lemma 2 states that the number of ordering requests increases if the number of resource holders increases. This is due to the positive network externality effect [6]. Furthermore, the number of ordering requests increases if the cloud provider decreases the price.

Denote the CDF of the demand N_d as

$$F_{N_d}(k|s,a) \triangleq \mathbb{P}[N_d \le k|s,a].$$

In the following lemma, we derive its closed-form expression and characterize its monotonicity.

Lemma 3: The CDF of N_d can be expressed as

$$F_{N_d}(k|s,a) = \sum_{i=0}^{s} [F_{N_r}(i+1|s,a) - F_{N_r}(i|s,a)] F_{N_o}(k-i|s,a).$$

For any $P_r(a) \in (0,1)$ and $P_o(a) \in (0,1)$, it holds that $F_{N_d}(k|s+1,a) < F_{N_d}(k|s,a)$, and $\partial F_{N_d}(k|s,a)/\partial a > 0$.

Lemma 3 states that the demand increases when the number of resource holders increases or the price decreases. Based on Lemma 3, we next derive the closed-form reward $r(s,a)$ and characterize its monotonicity.

Lemma 4: The reward $r(s,a)$ can be derived as

$$r(s,a) = a \sum_{k=0}^{S-1} [1 - F_{N_d}(k|s,a)]$$

and satisfies $r(s+1,a) > r(s,a)$.

Lemma 4 states that given the same price, the reward (earned in each time slot) increases as the number of resource holder increases. Let $F_{\text{st}}(k|s,a) \triangleq \sum_{j=0}^{k} p(j|s,a)$ denote the cumulative state transition probabilities. We next derive the closed form $F_{\text{st}}(k|s,a)$ and characterize its monotonicity.

Lemma 5: The $F_{\text{st}}(k|s,a)$ can be expressed as

$$F_{\text{st}}(k|s,a) = \begin{cases} F_{N_d}(k|s,a), & \text{if } k < S \\ 1, & \text{if } k = S. \end{cases}$$

For any $k < S$, $P_r(a) \in (0,1)$ and $P_o(a) \in (0,1)$, it holds that $F_{\text{st}}(k|s+1,a) < F_{\text{st}}(k|s,a)$ and $\partial F_{\text{st}}(k|s,a)/\partial a > 0$.

Lemma 5 states that the system is more likely to transits to a large state (i.e., high resources usage) if the cloud provider decreases the price.

B. Optimal Revenue and Optimal Policy

Based on the optimality conditions derived in Section III-F, and the model properties derived in Section IV-A, we first characterize the monotonicity of the optimal revenue function.

Theorem 2: The optimal revenue function $v^*(s)$ is increasing in the state s, i.e., $v^*(s+1) > v^*(s)$.

Theorem 2 states that the optimal revenue for the cloud provider increases in the state s (i.e., cloud resource usage level). This implies that a cloud provider wants to maintain the resource usage at a high level. This is intuitive because a higher usage of the resource means a larger number of users leading to a larger demand (Lemma 3). Note that the proof is nontrivial because we do not have a closed-form optimal revenue function. Sustaining a high level of resource usage is nontrivial. On the one hand, a new cloud provider is initialized with state $s = 0$, i.e., no users in the cloud system. On the other hand, setting a small price can increase the resource usage quickly (Lemma 5), however, it may cause some revenue loss and add some financial burden to the cloud provider in the short term.

Characterizing the optimal policy is challenging because we need to know the future reward (determined by the optimal revenue function). However, we only have the monotone properties of the optimal revenue function (Theorem 2). We, therefore, aim to establish conditions under which the optimal policy exhibits some nice properties. Through these nice properties, we uncover insights on the optimal policy.

We employ convex optimization theory [37] to characterize the optimal policy. For the ease of presentation, we define a function to quantify the state-action dependent total reward (i.e., the reward earned in the current time slot plus that earned in subsequent time slots). Formally, let $Q(s, a)$ denote the state-action dependent total reward function

$$Q(s,a) \triangleq r(s,a) + \delta \sum_{j \in \mathcal{S}} p(j|s,a) v^*(j).$$

Note that the function $Q(s, a)$ is nonlinear in a, and the $v^*(j)$ can be treated as constants (even though their exact values are unknown). Then, for each given s, finding the optimal price $d^*(s)$ is equivalent to solve the following nonlinear optimization problem with a compact domain.

Problem 1: Given the state s, find the optimal price

$$\begin{aligned} &\max_a \; Q(s,a) \\ &\text{s.t. } a \in \mathcal{A}. \end{aligned}$$

Thus, locating the optimal policy is equivalent to solving a sequence of Problem 1 with under different stats $s = 0, 1, \ldots, S$. In the following lemma, we apply convex optimization theory [37] to characterize the optimal solution of Problem 1 as well as the optimal policy.

Lemma 6: Suppose for all $s \in \mathcal{S}$, the function $Q(s, a)$ is strictly concave with respect to a, that is,

$$\frac{\partial^2 Q(s,a)}{\partial a^2} < 0. \tag{1}$$

Problem 1 has a unique solution under each $s \in \mathcal{S}$, implying that there exists a unique optimal policy π^*. Furthermore, if $Q(s, a)$ satisfies the following extra condition:

$$\frac{\partial Q(s+1,a)}{\partial a} \geq \frac{\partial Q(s,a)}{\partial a} \tag{2}$$

the unique optimal solution of Problem 1 is nondecreasing in the state s, implying $d^*(s+1) \geq d^*(s)$.

Lemma 6 states sufficient conditions to guarantee the uniqueness and monotonicity of the optimal policy. It states that conceptually the cloud provider would set a larger price when the state is larger (i.e., a higher resource usage level). This is because the demand will be larger when the state increases (as stated in Lemma 3). Lemma 6 is a direct consequence of the convex optimization theory, and Conditions (1) and (2) are quite general in the sense that they are imposed on the total reward $Q(s, a)$, which does not rely on any specific structure of the MDP model. These two conditions lay the foundation for us to characterize the optimal policy. However, the generality of Conditions (1) and (2) makes it difficult to interpret them. We next refine Conditions (1) and (2) via exploring the specific structure of our MDP model for cloud applications, in order to obtain more interpretable conditions.

Theorem 3: If $F_{\text{st}}(j|s, a)$ is convex with respect to a, then Condition (1) holds, implying a unique optimal policy π^*.

Theorem 3 states a sufficient condition to guarantee that Condition (1) holds, implying the uniqueness of the optimal policy. This condition means that there is an increasing return effect in transferring to a small state (i.e., low the resource usage level) when the cloud provider increases the price. In other words, when the price is small, increasing the price only increases the probability of transferring to a small state slightly. When the price is large, increasing the price increases that probability significantly. Refining Condition (2) is a little bit more subtle. For ease of presentation, we define the elasticity of a function.

Definition 2: The elasticity of a function f with respect to x is defined as $E_x^f \triangleq (\partial f/f)/(\partial x/x)$.

The elasticity E_x^f is a standard notion in economic literature works, where the function f is usually interpreted as demand and x is usually interpreted as a price. An economic interpretation of the elasticity is the ratio of the relative change in demand (i.e., f) with respect to the relative change in price (i.e., x). In the following theorem, we present a sufficient condition (in terms of elasticity) to guarantee that Condition (2) holds.

Theorem 4: If $F_{\text{st}}(j|s, a)$ is strictly convex with respect to a and the elasticity of the probability decrement function $\Delta F_{\text{st}}(j|s, a) \triangleq F_{\text{st}}(j|s, a) - F_{\text{st}}(j|s-1, a)$ with respect to a is bounded below, that is,

$$E_a^{\Delta F_{\text{st}}} \geq -1 \Big/ \left(1 + \frac{\delta S A}{a(1-\delta)}\right)$$

then, Condition (2) holds, i.e., $(\partial Q(s, a)/\partial a)$ is increasing in s.

Theorem 4 states that if $F_{\text{st}}(j|s, a)$ is strictly convex with respect to a and the elasticity of $\Delta F_{\text{st}}(j|s, a)$ with respect to a is bounded bellow, then Condition (2) holds. We like to point out that besides Theorem 3 and Theorem 4, there are also other ways to refine Conditions (1) and (2). We leave it as future work for further exploration.

Our results thus far assume all model parameters are given. In practice, $P_r(a)$, $P_o(a)$ and $F_{N_p}(i|s)$ are unknown. We next address this challenge.

V. Inferring Optimal Pricing Decisions

We apply the Q-learning to infer the optimal price from historical transaction data and derive sufficient conditions on the model to guarantee its convergence to the optimal price, but it converges slowly. To speed up the convergence, we incorporate the monotonicity of the optimal revenue (Theorem 2), leading to the VpQ-learning algorithm. We also prove the convergence of the VpQ-learning algorithm. We will show the efficiency of the VpQ-learning algorithm via experiments in Section VI.

A. Inferring the Optimal Price via Q-Learning

1) Overview: Note that $P_r(a), P_o(a)$ and $F_{N_p}(i|s)$ are unknown. In practice, the cloud provider has full knowledge to her historical transaction data, i.e., the states, the actions and the per-slot revenue earned in previous time slots, etc. Formally, we denote the transaction data obtained in time slot t as

$$\boldsymbol{h}_t \triangleq (a_t, N_d(t), N_a(t))$$

where a_t, $N_d(t)$ and $N_a(t)$ denote the action (or price), demand requests and number of units of cloud resources allocated in time slot t. The historical transaction data up to time slot t is $\{\boldsymbol{h}_0, \boldsymbol{h}_1, \dots, \boldsymbol{h}_t\}$. Our objective is to infer the optimal price at each time slot based on the transaction data up obtains before that time slot. Recall that the action space is continuous, which makes it difficult to infer the optimal price [17]. In practice, cloud providers can discretize the action (or price) space $\mathcal{A}$. For example, Amazon Web Services uses a discrete price space. For the remainder of this article, let us consider a discrete action space

$$\widetilde{\mathcal{A}} \triangleq \{\tilde{a}_m | m \in \{1, \dots, M\}, a_m \in \mathcal{A}\}$$

where $M \in \mathbb{N}_+$. Recall that the cloud provider's decision model is a discounted infinite horizon MDP. There are two representative types of RL algorithms [17]: 1) model-free algorithms, e.g., Q-learning [25] and 2) model-based algorithms, e.g., ARTDP [28]. Note that there are no regret bounds for learning discounted MDPs [17] and convergence analysis is essential for such learning tasks. We apply the model-free algorithm, i.e., Q-learning, to infer the optimal price. The reason is as follows. We do not assume parametric forms on the model, i.e., the reward and state transition probabilities. For our model, the reward can be calculated from state transition probabilities. Thus, for model-based algorithms, one needs to estimate the state transition probabilities. Each state-action pair (s, a) is associated with a state transition distribution, whose support is $\mathcal{S}$. Roughly, there are in total $|\mathcal{S}| \times |\widetilde{\mathcal{A}}| \times |\mathcal{S}| = |\mathcal{S}|^2|\widetilde{\mathcal{A}}|$ parameters to learn. However, for Q-learning, one only needs to estimate the Q function, which has $|\mathcal{S}||\widetilde{\mathcal{A}}|$ elements, i.e, $|\mathcal{S}||\widetilde{\mathcal{A}}|$ parameters to estimate.

2) Inferring Optimal Price via Q-Learning: The core idea of the Q-learning algorithm is that in each time slot it infers the optimal price from the estimate of $Q(s, a)$ and then using the transaction data generated in this time slot to update the estimate of $Q(s, a)$. We fit the Q-learning algorithm to our model and describe it in Algorithm 1. Step 2 states that the cloud provider with probability ϵ_t selects a price uniformly at random from $\widetilde{\mathcal{A}}$. The purpose is to explore potential promising prices (this is the exploration step) due to that the estimate of $Q(s, a)$ may contain errors. With probability $1 - \epsilon_t$, the cloud provider selects the price a_t, which maximizes the estimate of $Q(s, a)$, i.e., $a_t \in \arg\max_{a \in \widetilde{\mathcal{A}}} Q(s_t, a)$ (this is the exploitation step). Step 3 corresponds to the observation of demand, i.e., after the cloud provider sets a price a_t, she receives or observes $N_d(t)$ requests at time slot t. Note that the $N_d(t)$ is a sample drawn according the CDF $F_{N_d}(k|s_t, a_t)$ and the $F_{N_d}(k|s_t, a_t)$ is unknown to the cloud provider. Namely, $F_{N_d}(k|s_t, a_t)$ models the unknown environment. Step 4 simulates the resource allocation that the cloud provider allocates $N_a(t)$ units of resource being aware of the resource constraint. Step 5 states that the cloud provider earns a per-slot reward of $\hat{r}(s_t, a_t) = a_t N_a(t)$. Step 6 simulates the state transition, i.e., the states in the next time slot will be $s_{t+1} = N_a(t)$. Based on the reward and state transition in time slot t, the new estimate of $Q(s_t, a_t)$ is $\hat{r}(s_t, a_t) + \delta \max_{a \in \widetilde{\mathcal{A}}} Q(s_{t+1}, a)$. Then, in step 7, we update the estimate of $Q(s_t, a_t)$ as a balance between the old estimate $Q(s_t, a_t)$ and the new estimate $[\hat{r}(s_t, a_t) + \delta \max_{a \in \widetilde{\mathcal{A}}} Q(s_{t+1}, a)]$, where $\gamma_t \in (0, 1)$ is the balancing factor (called the learning rate). In summary, the historical transaction data is used to update the estimate of Q function incrementally. More specifically, the transaction data $\boldsymbol{h}_t$ obtained in time slot t is used to update the least estimated of Q function, which is calculated from $\{\boldsymbol{h}_0, \boldsymbol{h}_1, \dots, \boldsymbol{h}_{t-1}\}$.

Algorithm 1 Optimal Price Inference Using Q-Learning

```
Require: Discounting factor $\delta$, learning rate $\gamma_t \in (0, 1)$, exploration probability $\epsilon_t \in (0, 1)$, initialization of the $Q(s, a)$.
1: for $t = 0$ to $\infty$ do
2:     With probability $\epsilon_t$, $a_t \sim$ UniformRandom($\widetilde{\mathcal{A}}$), with probability $1 - \epsilon_t$, $a_t \in \arg\max_{a \in \widetilde{\mathcal{A}}} Q(s_t, a)$.
3:     Receive $N_d(t)$ requests.
4:     Allocate $N_a(t)$ units of cloud resources such that $N_a(t) = S$ if $N_d(t) > S$, otherwise $N_a(t) = N_d(t)$.
5:     Earn a per-slot revenue (or reward) $\hat{r}(s_t, a_t) = a_t N_a(t)$.
6:     State transition $s_{t+1} = N_a(t)$.
7:     Update the $Q$ function: $Q(s_t, a_t) \leftarrow (1 - \gamma_t) Q(s_t, a_t) + \gamma_t \big[\hat{r}(s_t, a_t) + \delta \max_{a \in \widetilde{\mathcal{A}}} Q(s_{t+1}, a)\big]$.
8: end for
```

Remark: Algorithm 1 adopts the classical Q-learning to our pricing problem. Note that under some ill-conditioned MDPs, Q-learning does not converge to the optimal policy [17]. Lacking convergence guarantees is a serious problem for cloud resource pricing, because it may lead to great revenue loss to the cloud provider. In the next lemma, we tailor the Q-learning to our pricing problem in that we establish sufficient conditions on the model to guarantee that the Q-learning converges to the optimal policy.

Lemma 7: Suppose $0 < P_r(a) < 1, 0 < P_o(a) < 1, \forall a \in \mathcal{A}$, and $F_{N_p}(0|0) < 1$. There exists γ_t and ϵ_t such that

$$\tilde{\epsilon}_t(s, a) \to 0, \quad \sum_{t=0}^{\infty} \tilde{\epsilon}_t(s, a) = \infty \tag{3}$$

$$\sum_{t=0}^{\infty} \tilde{\gamma}_t(s, a) = \infty, \quad \sum_{t=0}^{\infty} \tilde{\gamma}_t^2(s, a) < \infty \tag{4}$$

where $\tilde{\gamma}_t(s,a)$ and $\tilde{\epsilon}_t(s,a)$ are defined as

$$[\tilde{\gamma}_t(s,a), \tilde{\epsilon}_t(s,a)] \triangleq \begin{cases} [\gamma_t, \epsilon_t], & \text{if } (s_t,a_t) = (s,a) \\ [0,0], & \text{otherwise.} \end{cases}$$

For each selection of γ_t, ϵ_t satisfying Condition (3) and (4), Algorithm 1 converges to an optimal policy.

Lemma 7 is a simple extension of the classical results on Q-learning [38]. We omit its proof due to the page limit. Lemma 7 states sufficient conditions on the model parameters, under which Algorithm 1 is asymptotically accurate in inferring the optimal price. Note that these conditions are mild and can be broadly satisfied in practice. In the following lemma, we analyze the computational complexity of Algorithm 1.

Lemma 8: In each round t, the computational complexity of Algorithm 1 is $O(|\widetilde{\mathcal{A}}|)$.

Lemma 8 states that in time slot t, the computational complexity of Algo. 1 is linear in the action space size $|\mathcal{A}|$.

B. Speeding Up the Convergence

Algorithm 1 may converge slowly. It updates the estimate of $Q(s,a)$ only when a state-action pair (s,a) is visited by a transaction data. Under large state space size or the large action space size, it may take a very long time to estimate the Q function across all state-action pairs accurately. Speeding up the convergence of Algorithm 1 can lead to higher revenue. This is because we consider the online setting, where the cloud provider estimate the optimal price and optimize the pricing decisions simultaneously. The revenue during the learning period, i.e., each time slot is counted. In other words, a suboptimal action in the learning period can lead to revenue loss to the cloud operator. Improving the convergence speed means that the operator can estimate the optimal policy accurately using less rounds, which can help the cloud operator avoid suboptimal actions or revenue loss. We now incorporate the monotonicity of the optimal revenue (Theorem 2) to speed up the convergence. Theorem 2 implies the following monotone property of $Q(s,a)$.

Corollary 1: Under the action space $\mathcal{A}$, the function $Q(s,a)$ is nondecreasing in s, i.e., $Q(s,a) \le Q(s+1,a)$.

Corollary 1 states that the total reward for the cloud provider is nondecreasing in s. It is trivial to show that under the action space $\widetilde{\mathcal{A}}$, Theorem 2 and Corollary 1 still hold. We next use the monotonicity of $Q(s,a)$ to improve the accuracy of estimating $Q(s,a)$. Our ideas is to preserve the monotonicity of the estimate of $Q(s,a)$ after each update. More concretely, each time when we update one element of the Q function, e.g., $Q(s,a)$, the sequence $Q(0,a), Q(1,a), \ldots, Q(S,a)$ may not be monotone. Thus, we project them such that the monotonicity is preserved. Formally, we describe this idea in Algorithm 2, which we call the VpQ-learning algorithm. The input of Algorithm 2 and 1 are the same except for one difference: We require that the initialized value $Q(s,a)$ must be monotone in s for each fixed a. Then, in each time slot t, we apply the step 2 to 7 of Algorithm 1 to infer the optimal price and update the estimated $Q(s_t,a_t)$. After the update, suppose $Q(0,a_t), Q(1,a_t), \ldots, Q(S,a_t)$ are not monotone. Then, in step 3 to 5, we propagate the value of $Q(s_t,a_t)$ upward to make the sequence $Q(s_t,a_t), Q(s_t+1,a_t), \ldots, Q(S,a_t)$ being monotone. In step 6–8 we propagate the value of $Q(s_t,a_t)$ downward to make the sequence $Q(0,a_t), Q(1,a_t), \ldots, Q(s_t,a_t)$ being monotone. In the following theorem, we show that the VpQ-learning algorithm converges to the optimal policy.

Algorithm 2 VpQ-Learning

```
Require: $\delta$, $\gamma_t$, $\epsilon_t$, $Q(s,a)$ (must be monotone in $s$);
1:  for $t = 0$ to $\infty$ do
2:    Execute step 2–7 of Algorithm 1.
3:    for $j = s_t$ to $S-1$ do
4:      if $Q(j+1,a_t) < Q(j,a_t)$ then
5:        $Q(j+1,a_t) \leftarrow Q(j,a_t)$
6:      end if
7:    end for
8:    for $j = s_t$ decreases to 1 do
9:      if $Q(j-1,a_t) > Q(j,a_t)$ then
10:       $Q(j-1,a_t) \leftarrow Q(j,a_t)$
11:     end if
12:   end for
13: end for
```

Remark: The VpQ-learning improves the Q-learning to tailor the problem. More specifically, the projection of Q function in VpQ-learning is valid only when Corollary 1 holds. Note that in general Corollary 1 does not hold. For the pricing problem considered in this article, Corollary 1 is proven to hold.

The value projection component in VpQ-learning is proposed by us. The major role of this projection component is utilizing Corollary 1 to speed up the convergence. More specifically, Corollary 1 states that for each given action a, the $Q(s,a)$ function is monotone with respect to state s. This projection component utilizes this monotone property to update multiple elements of the estimate of Q function [even when an (s,a) pair is not visited] with one transaction data. Note that Algorithm 1 updates only one element of the estimate of Q function with one transaction data, i.e., update the estimate of $Q(s,a)$ when a transaction data visits a (s,a) pair. Hence, this projection component can make the estimate of the Q function converge faster to its groundtruth value.

Theorem 5: Under the same assumptions and conditions in theorem 7, Algorithm 2 converges to an optimal policy.

Theorem 5 states sufficient conditions, under which Algorithm 2 converges to an optimal policy. Note that the conditions are the same as Theorem 7. With value projection, the convergence is guaranteed without adding extra conditions. In the following lemma, we analyze the computational complexity of Algorithm 2.

Lemma 9: In each round t, the computational complexity of Algorithm 2 is $O(|\widetilde{\mathcal{A}}| + |\mathcal{S}|)$.

Lemma 9 states that in each round t, the computational complexity of Algorithm 2 is linear in the size of actions $|\mathcal{A}|$ and linear in the size of state space $|\mathcal{S}|$. Compared with

Lemma 8, in each round t, Algorithm 2 uses $|\mathcal{S}|$ operations than Algorithm 1.

VI. Experiments On Real-World Data

We conduct experiments on a real-world cloud dataset and compare the VpQ-learning algorithm with a variety of baselines, e.g., Q-learning, Speedy Q-learning, etc. Experiment results show that the VpQ-learning algorithm improves the revenue over the Q-learning, Speedy Q-learning and ARTDP by as high as 50%. It can also achieve at least 80% revenue of the asynchronous value iteration algorithm (i.e., Q-learning with model parameters). Lastly, it improves the revenue over the fixed pricing scheme by as high as 20%.

A. Experiment Settings

1) The Dataset: We use the cloud trace dataset published by Wolski and Brevik [39], [40]. The dataset contains the traces of five clouds implemented using Eucalyptus. From them, we select the traces collected from the one that has the largest number of resources. The selected cloud has 31 nodes, and each node has 32 cores. In total, a number of 16912 trace data items were collected in one month. These data items are of two types: 1) starting item, which contains the starting time stamp and the instance ID; and 2) stopping item, which contains the stopping time stamp and the instance ID.

2) Parameter Settings: We interpret each core as one unit of resource. In total we have $31 \times 32 = 992$ units of resources. Namely, the resource constraint is $S = 992$. We set the length of each time slot to be one day. One can also set the length to be one week, ten days, etc, which will scale the demands. We use a parameterized model to infer the distribution of the number of potential users. In particular, we assume that the arrival of potential users follow a Poisson distribution with a rate $\lambda(s)$ for each given state s. To capture the positive network externality [6], the rate $\lambda(s)$ is increasing in s. Furthermore, the rate $\lambda(s)$ is concave in s to capture diminishing return in the positive network effect. In this article we choose $\lambda(s) = \lambda(0)(1 + \ln(1 + s))$, which is a natural choice to satisfy the monotone and concave properties. From the dataset, we only have the ordering requests, from which we infer $\lambda(s)$. The dataset does not contain price information, we thus consider the cloud provider uses a fixed price scheme and we infer the probability of a potential orderer ordering a resource as 0.5. Hence the arrival of ordering request also follows a Poisson distribution with rate $0.5\lambda(s)$. The average rate across all states is $(0.5/S)\sum_{s=0}^{S}\lambda(0)(1+\ln(1+s))$. From the dataset we compute the empirical average rate as

$$\bar{\lambda} = \frac{\text{total number of requests}}{\text{total number of days}} = \frac{8448}{33} = 256.$$

Then we have $(0.5/S)\sum_{s=0}^{S}\lambda(0)(1+\ln(1+s)) = \bar{\lambda}$, implying $\lambda(0) = 2\bar{\lambda}S/\sum_{s=0}^{S}(1 + \ln(1 + s))$. Given $S = 992$ and $\bar{\lambda} = 256$, we can calculate $\lambda(0) = 74.074$. Therefore, we have $\lambda(s) = 74.074(1 + \ln(1 + s))$.

The model parameters $P_r(a)$ and $P_o(a)$ cannot be inferred from the dataset because the dataset does not contain price information. To the best of our knowledge, there is no public dataset that contains price information. We, therefore, synthesize them in a systematic manner, in particular, capturing the price sensitivity to ordering or renewing behavior. We consider the following form of $P_r(a)$ and $P_o(a)$

$$P_o(a) = 1 - a^{\alpha}, \quad P_r(a) = 1 - a^{\beta}$$

where $\alpha \in \mathbb{R}_+$ and $\beta \in \mathbb{R}_+$ model the potential orderers' and resource holders' sensitivity to price respectively. Large value of α (or β) implies they are less sensitive to the price. Note that we consider this form of $P_r(a)$ and $P_o(a)$, because the price sensitivity is an important factor in affecting the revenue of a pricing mechanism.

Note that these parameters are unknown to the cloud provider and they will be used to simulate the model only. In the simulation, we set $\widetilde{\mathcal{A}} = \{0.1, 0.2, \ldots, 0.9\}$, $s_0 = 0$, $\delta = 0.999$, $Q(s, a) = 1$, $\epsilon_t = 0.1/(\tilde{N}_t(s_t) + 1)$, and $\gamma_t = 0.1/(\tilde{N}_t(s_t, a_t) + 1)$, where $\tilde{N}_t(s_t)$ and $\tilde{N}_t(s_t, a_t)$ denote the number of visits to state s_t and (s_t, a_t) pair up to the t-th time slot.

3) Baselines and Metrics: We compare the VpQ-learning algorithm with: **1) Q-learning** [25]; **2) speedy Q-learning** [26]; **3) ARTDP** [28]; and **4) Asynchronous value iteration**. Note that the asynchronous value iteration is used to understand the limits of RL algorithms [28]. It is the Q-learning with model parameters, i.e., in each iteration of the Algorithm 1, the model parameters are given to compute $\mathbb{E}[Q(s_t, a_t)]$. We simulate the model and run these algorithms to set the price in each time slot. For comparison studies, we define the profit improvement of the VpQ-learning over the Q-learning as

$$\text{ImpOverQL} \triangleq \frac{v(s_0|\text{VpQL}) - v(s_0|\text{QL})}{v(s_0|\text{QL})}$$

where $v(s_0|\text{QL})$ and $v(s_0|\text{VpQL})$ denote the long term profit when the operator sets the price using Q-learning and VpQ-learning respectively. Similarly, we define the improvement over speedy Q-learning, ARTDP, and asynchronous value iteration as ImpOverSpeedy, ImpOverARTDP and ImpOverAsy respectively. Note that we do not compare with the Zap Q-learning [27] due to its high computation complexity, making it not practical to infer the optimal price in real-time.

4) Remark on Evaluating Learning Algorithms: Note that we evaluate learning algorithm in an online setting. All the above inferred parameters are used to simulate the online learning environment. Each algorithm does not have any *a priori* knowledge on the model parameters in inferring the optimal price.

B. Comparision With Learning Algorithms

Fig. 1 presents the revenue and revenue improvement when α varies from 0.1 to 1. Fig. 1(a) shows that the revenues for the VpQ-learning algorithm and these four baseline algorithms increase as the value of α increases. This implies that the cloud provider can earn more revenue when potential orderers are less sensitive to the price. This is because the cloud provider can set higher prices while attracting the same number of orderers. The revenue curve of the VpQ-learning algorithm lies

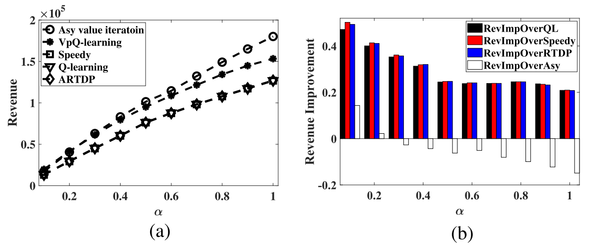

Fig. 1. Impact of α on revenue and revenue improvement $\beta = 0.5$. (a) Revenue. (b) Revenue improvement.

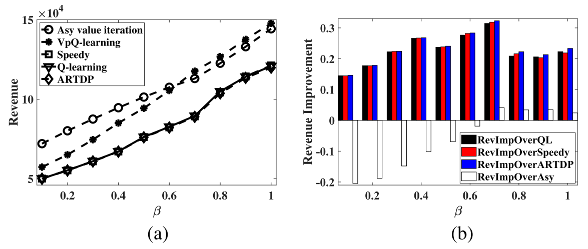

Fig. 2. Impact of β on revenue and revenue improvement $\alpha = 0.5$. (a) Revenue. (b) Revenue Improvement.

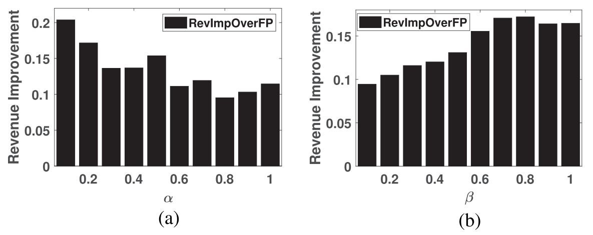

Fig. 3. Revenue improvement over the fixed pricing scheme. (a) $\beta = 0.5$. (b) $\alpha = 0.5$.

above that of the Q-learning, speedy Q-learning, and ARTDP algorithm. This means that using the VpQ-learning Algorithm, the cloud provider can earn more revenues. Fig. 1(b) shows that the revenue improvement is significant, i.e., as high as 50% and at least 20%. These results imply that the VpQ-learning algorithm converges faster than the Q-learning, speedy Q-learning, and ARTDP algorithm in estimating the optimal price. The revenue improvement decreases in α. Namely, the VpQ-learning algorithm can speed up the convergence more, when potential orderers are more sensitive to price. Fig. 1(b) shows that when α is no larger than 0.2, the VpQ-learning algorithm can improve the revenue over the asynchronous value iteration by as high as 10%. But, when α is larger than 0.2, the VpQ-learning algorithm can achieve at least 80% revenue of the asynchronous value iteration. Recall that the asynchronous value iteration needs to input model parameters. Hence, the VpQ-learning algorithm is highly efficient in inferring the optimal price.

Fig. 2 presents the revenue and the revenue improvement when β varies from 0.1 to 1. Fig. 2(a) shows that the revenues for each algorithm increases as the value of β increases. This implies that the cloud provider can earn more revenues, when resource holders are less sensitive to price. This is because the cloud provider can set higher prices while attracting the same amount of renewal requests. The revenue curve of the VpQ-learning algorithm lies above that of the Q-learning, speedy Q-learning, and ARTDP algorithm. This means that using the VpQ-learning Algorithm, the cloud provider can earn more revenues. Fig. 1(b) shows that the revenue improvement is significant, i.e., as high as more than 30%. These results imply that the VpQ-learning algorithm converges faster than the Q-learning, speedy Q-learning, and ARTDP algorithm in estimating the optimal price. Fig. 1(b) shows that when β is no less than 0.7, the VpQ-learning algorithm can improve the revenue over the asynchronous value iteration by as high as 5%. But, when β is less than 0.7, the VpQ-learning algorithm can achieve at least 80% revenue of the asynchronous value iteration. Recall that the asynchronous value iteration needs to input model parameters. Hence, the VpQ-learning algorithm is highly efficient in inferring the optimal price.

Lessons Learned: The cloud provider earns more revenues when potential orderers (or resource holders) become less sensitive to price. The VpQ-learning algorithm can improve the revenue over the Q-learning, Speedy Q-learning and ARTDP by as high as 50%. It can also achieve at least 80% revenue of the asynchronous value iteration algorithm.

C. Comparison With Fixed Pricing Scheme

Here, we compare the VpQ-learning algorithm with a fixed pricing scheme. Note that we do not compare to existing dynamic pricing schemes, because they either apply to different cloud models, e.g., auction model, or have different design objectives, e.g., welfare maximization. Please refer to Section II for more justifications on this point. A cloud provider can fix the price at any price in $\widetilde{\mathcal{A}}$. The cloud provider does not have any *a priori* knowledge on which price can lead to a large revenue. We thus study the average revenue improvement of the VpQ-learning algorithm over the fixed pricing scheme

$$\text{RevImpOverFP} \triangleq \frac{v(s_0|\text{QLVP}) - v(s_0|\text{Fixed})}{v(s_0|\text{Fixed})}$$

where $v(s_0|\text{Fixed}) \triangleq \sum_{a\in\widetilde{\mathcal{A}}} v(s_0|a)/|\widetilde{\mathcal{A}}|$ is defined as the average revenue for the fixed pricing scheme and $v(s_0|a)$ denotes the revenue when the price is fixed at a. Fig. 3 presents the revenue improvement RevImpFP when we vary α and β from 0.1 to 1. From Fig. 3(a) we can observe that the revenue improvement is as high as 20% and it decreases in α (i.e., potential orderers become less sensitive to price). It implies that the optimal price is more difficult to infer when potential orderers become less sensitive to price. This is because the price effect becomes less distinguishable as α increases. From Fig. 3(b) we can observe that the revenue improvement is as high as 15%, and it increases as β increases (i.e., resource holders become less sensitive to price). It implies that the optimal price is easier to infer when resource holders become less sensitive to price. This is because in each time slot, the number of resource holders is known to the cloud provider, and the cloud provider can predict the demand more accurate as resource holders become less sensitive to price.

Lessons Learned: The revenue improvement of the VpQ-learning algorithm over the fixed pricing scheme is as high as 20%. This revenue improvement increases as potential orderers become more sensitive to price or resource holders become less sensitive to price.

VII. Conclusion

This article develops an RL framework to price cloud resources with positive network externality effect. We formulate a probabilistic model to quantify the positive network externality effect in cloud applications and we formulate a revenue maximization framework via an MDP to determine the optimal price. We analytically show that: 1) under the positive network externality effect, the value function (i.e., optimal long-term revenue) increases in the state (i.e., resource usage level) of the MDP; and 2) under certain regularity conditions on the transition probability (i.e., resource demand from cloud users) of the MDP, the optimal price is unique and monotone in the state of the MDP. We apply the classical Q-learning algorithm to infer the optimal price and derive sufficient conditions on the model to guarantee the convergence to optimal price. However, the Q-learning may converge slowly because each transaction data is used to update one element of the Q function. To speed up the convergence, we incorporate the monotonicity of the Q function to update the estimate of the Q function, leading to the VpQ-learning algorithm. The VpQ-learning updates at most $|\mathcal{S}|$ elements of the estimate of Q function with one transaction data. We derive sufficient conditions, under which the VpQ-learning algorithm converges to the optimal policy. These conditions serve as guidance to set parameters for the VpQ-learning algorithm. We conduct experiments on a real-world dataset to compare the VpQ-learning algorithm with a variety of baselines. Experimental results show that the cloud provider earns more revenues when potential orderers (or resource holders) become less sensitive to price. The VpQ-learning algorithm can improve the revenue over the Q-learning, Speedy Q-learning and ARTDP by 50%. It can also achieve at least 80% revenue of the asynchronous value iteration algorithm (i.e., Q-learning with model parameters). This revenue improvement ratio increases as potential orderers become more sensitive to price or resource holders become less sensitive to price. Lastly, it can improve the revenue over a fixed pricing scheme by 20%. This revenue improvement ratio increases as potential orderers become more sensitive to price or resource holders become less sensitive to price.

One limitation of this work is that only one cloud provider is considered. In practice, there can be multiple cloud providers in the market. If these cloud providers are willing to cooperate, i.e., aggregate their resource as a whole to set price, they can be treated as a super cloud provider and our method can directly be applied. If these cloud providers compete against each other, our method cannot be directly applied, but we believe that our method can serve as a building block to study the pricing problem under competition. We like to remark that extending our method to study price competition is highly nontrivial and we leave it as our future work.

References

[1] T. Team, *Microsoft's Cloud Services Boost Revenues*. Washington Boulevard Jersey City, NY, USA: Forbes, 2017.

[2] N. Wingfield, *Amazon's Cloud Business Lifts its Profit to a Record*. New York, NY, USA: The New York Times, 2017.

[3] Q. Hardy, *Why the Computing Cloud Will Keep Growing and Growing*. Washington Boulevard Jersey City, NY, USA: Forbes, 2017.

[4] Q. Hardy, *Google Races to Catch Up in Cloud Computing*. Washington Boulevard Jersey City, NY, USA: Forbes, 2017.

[5] *Amazon EC2 Spot Instances Pricing. Amazon EC2 Spot Instances Pricing*, Amazon, Seattle, WA, USA, 2014.

[6] B. Williams, *The Economics of Cloud Computing: An Overview for Decision Makers*, 1st ed. Indianapolis, IN, USA: Cisco Press, 2012.

[7] W.-Y. Lin, G.-Y. Lin, and H.-Y. Wei, "Dynamic auction mechanism for cloud resource allocation," in *Proc. 10th IEEE/ACM Int. Conf. Cluster, Cloud Grid Comput.*, May 2010, pp. 591–592.

[8] W. Wang, B. Liang, and B. Li, "Revenue maximization with dynamic auctions in IaaS cloud markets," in *Proc. IEEE/ACM 21st Int. Symp. Qual. Service (IWQoS)*, Jun. 2013, pp. 1–6.

[9] W. Shi, L. Zhang, C. Wu, Z. Li, and F. C. M. Lau, "An online auction framework for dynamic resource provisioning in cloud computing," in *Proc. ACM Int. Conf. Meas. Model. Comput. Syst. (SIGMETRICS)*, 2014, pp. 71–83.

[10] X. Zhang, Z. Huang, C. Wu, Z. Li, and F. C. Lau, "Online auctions in IaaS clouds: Welfare and profit maximization with server costs," in *Proc. ACM SIGMETRICS*, 2015, pp. 3–15.

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

[12] W. Shi, C. Wu, and Z. Li, "An online mechanism for dynamic virtual cluster provisioning in geo-distributed clouds," in *Proc. IEEE INFOCOM 35th Annu. IEEE Int. Conf. Comput. Commun.*, Apr. 2016, pp. 1–9.

[13] Z. Zhang, Z. Li, and C. Wu, "Optimal posted prices for online cloud resource allocation," in *Proc. ACM SIGMETRICS / Int. Conf. Meas. Modeling Comput. Syst.*, Jun. 2017, pp. 1–26.

[14] B. Du, C. Wu, and Z. Huang, "Learning resource allocation and pricing for cloud profit maximization," in *Proc. 33rd AAAI Conf. Artif. Intell. (AAAI)*, 2019, pp. 7570–7577.

[15] B. Shi, H. Zhu, H. Yuan, R. Shi, and J. Wang, "Pricing cloud resource based on reinforcement learning in the competing environment," in *Proc. Int. Conf. Cloud Comput.* Cham, Switzerland: Springer, 2018, pp. 158–171.

[16] H. Xu and B. Li, "Dynamic cloud pricing for revenue maximization," *IEEE Trans. Cloud Comput.*, vol. 1, no. 2, pp. 158–171, Nov. 2013.

[17] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, vol. 1, no. 1. Cambridge, MA, USA: MIT Press, 1998.

[18] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani, "A hybrid reinforcement learning approach to autonomic resource allocation," in *Proc. IEEE Int. Conf. Autonomic Comput.*, Jun. 2006, pp. 65–73.

[19] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck, "Using reinforcement learning for autonomic resource allocation in clouds: Towards a fully automated workflow," in *Proc. ICAS*, 2011, pp. 67–74.

[20] C.-Z. Xu, J. Rao, and X. Bu, "URL: A unified reinforcement learning approach for autonomic cloud management," *J. Parallel Distrib. Comput.*, vol. 72, no. 2, pp. 95–105, Feb. 2012.

[21] E. Barrett, E. Howley, and J. Duggan, "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud," *Concurrency Comput., Pract. Exper.*, vol. 25, no. 12, pp. 1656–1674, Aug. 2013.

[22] M. A. Salahuddin, A. Al-Fuqaha, and M. Guizani, "Reinforcement learning for resource provisioning in the vehicular cloud," *IEEE Wireless Commun.*, vol. 23, no. 4, pp. 128–135, Aug. 2016.

[23] H. Arabnejad, C. Pahl, P. Jamshidi, and G. Estrada, "A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling," in *Proc. 17th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2017, pp. 64–73.

[24] B. Xu, T. Qin, G. Qiu, and T.-Y. Liu, "Optimal pricing for the competitive and evolutionary cloud market," in *Proc. 24th Int. Joint Conf. Artif. Intell.*, 2015, pp. 139–145.

[25] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, Cambridge, U.K., 1989.

[26] M. G. Azar, R. Munos, M. Ghavamzadeh, and H. Kappen, "Speedy Q-learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2011, pp. 2411–2419.

[27] A. M. Devraj and S. Meyn, "Zap Q-learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 2235–2244.

[28] A. G. Barto, S. J. Bradtke, and S. P. Singh, "Learning to act using real-time dynamic programming," *Artif. Intell.*, vol. 72, nos. 1–2, pp. 81–138, Jan. 1995.

[29] L. Cabral, "Dynamic price competition with network effects," *Rev. Econ. Stud.*, vol. 78, no. 1, pp. 83–111, Jan. 2011.

[30] J. Huang and L. Gao, "Wireless network pricing," *Synth. Lectures Commun. Netw.*, vol. 6, no. 2, pp. 1–176, 2013.

[31] Z. Xiong, D. Niyato, P. Wang, Z. Han, and Y. Zhang, "Dynamic pricing for revenue maximization in mobile social data market with network effects," *IEEE Trans. Wireless Commun.*, vol. 19, no. 3, pp. 1722–1737, Mar. 2020.

[32] A. Ajorlou, A. Jadbabaie, and A. Kakhbod, "Dynamic pricing in social networks: The word-of-mouth effect," *Manage. Sci.*, vol. 64, no. 2, pp. 971–979, Feb. 2018.

[33] O. Candogan, K. Bimpikis, and A. Ozdaglar, "Optimal pricing in networks with externalities," *Oper. Res.*, vol. 60, no. 4, pp. 883–905, Aug. 2012.

[34] A. Makhdoumi, A. Malekian, and A. E. Ozdaglar, "Strategic dynamic pricing with network effects," in *Proc. Rotman School Manage. Working Paper*, 2017, Art. no. 2980109.

[35] H. Halaburda, B. Jullien, and Y. Yehezkel, "Dynamic competition with network externalities: How history matters," *RAND J. Econ.*, vol. 51, no. 1, pp. 3–31, Mar. 2020.

[36] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ, USA: Wiley, 1994.

[37] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[38] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, 1st ed. Nashua, NH, USA: Athena Scientific, 1996.

[39] R. Wolski and J. Brevik, "Using parametric models to represent private cloud workloads," *IEEE Trans. Services Comput.*, vol. 7, no. 4, pp. 714–725, Oct. 2014.

[40] R. Wolski. (2014). *The Dataset*. [Online]. Available: https://www.cs.ucsb.edu/~rich/workload/

Hong Xie (Member, IEEE) received the B.Eng. degree from the School of Computer Science and Technology, University of Science and Technology of China, Hefei, China, in 2010, and the Ph.D. degree from the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, in 2015, under the supervision of Prof. J. C. S. Lui.

He is currently a Research Professor with the College of Computer Science, Chongqing University, Chongqing, China. His research interests include online learning, networking and data science.

Prof. Xie is a member of ACM.

John C. S. Lui (Fellow, IEEE) received the Ph.D. degree in computer science from the University of California at Los Angeles, CA, USA, in 1992.

He is currently the Choh-Ming Li Chair Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong. His current research interests include machine learning, online learning (e.g., multiarmed bandit, reinforcement learning), network science, future internet architectures and protocols, network economics, network/system security, and large scale storage systems.

Dr. Lui is an Elected Member of the IFIP WG 7.3, a fellow of ACM, a fellow of IEEE, a Senior Research Fellow of the Croucher Foundation and was the past Chair of the ACM SIGMETRICS (2011–2015). He received various departmental teaching awards and the CUHK Vice-Chancellor's Exemplary Teaching Award. He is a co-recipient of the Best Paper Award in the IFIP WG 7.3 Performance 2005, IEEE/IFIP NOMS 2006, SIMPLEX 2013, and ACM RecSys 2017.