Customer Perceived Value- and Risk-Aware Multiserver Configuration for Profit Maximization

Tian Wang, *Student Member, IEEE*, Junlong Zhou, *Member, IEEE*, Gongxuan Zhang, *Senior Member, IEEE*, Tongquan Wei, *Senior Member, IEEE*, and Shiyan Hu, *Senior Member, IEEE*

Abstract—Along with the wide deployment of infrastructures and the rapid development of virtualization techniques in cloud computing, more and more enterprises begin to adopt cloud services, inspiring the emergence of various cloud service providers. The goal of cloud service providers is to pursue profit maximization. To achieve this goal, cloud service providers need to have a good understanding of the economics of cloud computing. However, the existing pricing strategies rarely consider the interaction between user requests for services and the cloud service provider and hence cannot accurately reflect the supply and demand law of the cloud service market. In addition, few previous pricing strategies take into account the risk involved in the pricing contract. In this article, we first propose a dynamic pricing strategy that is developed based on the customer perceived value (CPV) and is able to accurately capture the real situation of supply and demand in marketing. The strategy is utilized to estimate the user's demand for cloud services. We then design a profit maximization scheme that is developed based on the CPV-aware dynamic pricing strategy and considers the risk in the pricing contract. The scheme is utilized to derive the optimal multiserver configuration for maximizing the profit. Extensive simulations are carried out to verify the proposed customer perceived value and risk-aware profit maximization scheme. As compared to two state of the art benchmarking methods, the proposed scheme gains 31.6 and 30.8 percent more profit on average, respectively.

Index Terms—Cloud computing, customer perceived value, dynamic pricing, multiserver configuration, profit maximization, risk

1 Introduction

Cloud computing, as a new and effective commercial model that turns the delivery of storage, computing, and communication resources into ordinary commodities in a pay-as-you-go manner, is being paid more and more attention by academia and industry [1]. With the rapid development of virtualization techniques and the wide deployment of infrastructures in cloud computing, the number and variety of cloud service providers (e.g., Amazon EC2 [2] and Microsoft Azure [3]) have dramatically increased. The goal of these cloud service providers is to pursue profit maximization. Therefore, for these service providers, how to determine their cloud service prices and configure their cloud servers for obtaining the maximum profit are of utmost importance.

Like all the traditional businesses, the profit of a cloud service provider is the difference between the revenue gained by selling cloud services to customers and the monetary cost of rental charge as well as electricity bill of servers owned by the infrastructure providers. Either increasing the revenue or reducing the cost can improve the profit. Thus, cloud service providers develop a series of approaches [4], [5], [6], [7] to raise the profit in two aspects: one is to raise the revenue by setting a high selling price for cloud services while attracting a large number of customers to buy services and the other is to reduce operating cost for providing services. Note that a service's price as well as purchase amount interplay and thus cannot be optimized simultaneously [8]. In this paper, we propose a dynamic pricing strategy which takes into account the interaction between a service's price and purchase amount.

Unlike static pricing strategies (e.g., tiered pricing, subscription based pricing, and pay-as-you-go used in real-world cloud service providers Amazon Web Services [9] and Google App Engine [10]) that fix the price of a service request in advance as well as do not change the price with market conditions [5], [7], [11], dynamic pricing strategies adjust the service price according to customer demands. Since dynamic pricing strategies can better cope with unpredictable customer demands, they have been adopted by more and more cloud service providers. Amazon has introduced a spot pricing strategy [12] to adjust the price for a virtual instance at runtime. Based on an empirical study of Amazon's spot price history, Xu and Li [13] proposed a market-driven dynamic pricing mechanism to have a better understanding of the current market demand. A genetic algorithm based dynamic pricing strategy is developed by Macias *et al.* [14] to find the optimal price for maximizing profit. Ren *et al.* [15] presented a dynamic scheduling and pricing strategy for delay-tolerant

- T. Wang, J. Zhou, and G. Zhang are with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China. E-mail: {eclipse_wt, jlzhou, gongxuan}@njust.edu.cn.
- T. Wei is with the School of Computer Science and Technology, East China Normal University, Shanghai 200062, China. E-mail: tqwei@cs.ecnu.edu.cn.
- S. Hu is with the School of Electronics and Computer Science, University of Southampton, SO17 1BJ, Southampton, United Kingdom. E-mail: S.Hu@soton.ac.uk.

Manuscript received 3 Apr. 2019; revised 10 Nov. 2019; accepted 11 Nov. 2019. Date of publication 16 Dec. 2019; date of current version 20 Jan. 2020. (Corresponding authors: Junlong Zhou and Gongxuan Zhang.) Recommended for acceptance by J. Zhai. Digital Object Identifier no. 10.1109/TPDS.2019.2960024

batch services to maximize long-term profit for cloud service providers. A dynamic and optimal pricing scheme is especially designed for provisioning Sensors-as-a-Service [16]. However, all the above-mentioned dynamic pricing strategies do not consider customer perceived value, which is the fundamental of marketing activities.

The concept of customer perceived value (CPV) is first introduced by Zaithaml *et al.* [17] and has been widely used in the modeling of market activities [18], [19], [20]. CPV is described as the customer's overall assessment of the utility of a product or service based on perceptions of what is received and what is given [17]. It reflects the worth of a product in the customer's mind, and increases when customers believe the benefits outweigh the costs. Since customers are generally unaware of the true cost of production for the product they buy, they will only pay for the product when their CPV for the product is higher than the product's selling price. In such way, CPV impacts the purchase amount of services and ultimately influences the cloud service provider's profit. In this paper, we design a CPV-aware dynamic pricing strategy for service providers.

A pricing contract in cloud computing includes many considerations, such as the user demand for a service, the configuration of a multiserver system, the service-level agreement, the instructions of tasks to complete the service, the task waiting time, and etc. Due to the inherent uncertainty in task executions, the estimated time and monetary cost may be different from the actual ones, leading to a decreased profit or even a loss. Therefore, cloud service providers should consider the risk induced by the uncertainty in the pricing contract. In this paper, we study the problem of maximizing the profit of cloud service providers. To solve the problem, we present a CPV-based dynamic pricing strategy which conforms to the supply and demand law in economics and a risk-aware profit maximization scheme that takes into account the risk in the pricing contract. The major contributions of this paper are summarized below.

- We propose a CPV-based dynamic pricing strategy which takes into account the inter-relationship between the customers and providers of cloud services. Based on the concept of CPV, the strategy develops a calculation formula and a standard bisection based heuristic to derive customers' demand for cloud services. The correctness of the proposed pricing strategy and calculation formula are also validated.
- We design a risk-aware profit maximization scheme that optimizes the cloud service provider's profit by configuring the multiserver system. The scheme is developed based on our dynamic pricing strategy and considers the risk in the pricing contract. It also provides a simulated-annealing based heuristic to find the numerical optimal solution to the profit maximization problem.
- We conduct extensive simulation experiments to validate the proposed scheme by i) observing the changing trend of profit in different conditions as well as analyzing the factors affecting the profit, and ii) comparing the performance of the proposed scheme with two benchmarking approaches in increasing profit.

The results demonstrate that the proposed scheme not only follows the market supply and demand law but also outperforms the state of the art benchmarking approaches. Specifically, the proposed scheme can achieve 31.6 and 30.8 percent more profit on average compared to two benchmarking approaches, respectively.

The rest of the paper is organized as follows. Section 2 introduces the system architecture and models. Section 3 presents the proposed dynamic pricing strategy and Section 4 describes the proposed profit optimization scheme. The proposed scheme is validated in Section 5 and concluding remarks are given in Section 6.

2 System Architecture and Model

This section describes the cloud service provision structure and related models used in the paper.

2.1 Cloud System Architecture

Consider a three-tier cloud service provision structure consisting of three typical entities, i.e., cloud infrastructure providers, cloud service providers, and cloud customers. The three-tier structure has been widely adopted in the literature [1], [4], [6]. In the market formed by the three entities, the cloud infrastructure provider owns the hardware and software facilities, and charges the cloud service provider for renting the infrastructure resources. The cloud service provider utilizes these resources to prepare services in the form of virtual machines (VMs), and charges customers for processing their service requests. The cloud customer submits service requests to the service provider and pays for the services based on the services' amount and quality.

2.2 Cloud Service Provider Model

Cloud service providers offer many services to their customers, e.g., infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS), and software-as-a-service (SaaS). We consider IaaS cloud service providers in this paper. Examples of IaaS providers include Google Compute Engine [21], Microsoft Azure [3], and Amazon CloudFormation [22]. In IaaS, computing resources are offered as VMs to support the end users operations and the cloud service provider is responsible for running and maintaining the services. Owing to the cloud virtualization techniques, the cloud service providers could focus on their own business without spending energy on the construction and maintenance of hardware platforms. We assume that an IaaS cloud service provider solves customers' service requests by renting a multiserver system, which is constructed and maintained by an infrastructure vendor and has quite flexible architecture details for deploying pricing strategy [5]. Such a multiserver system can be in the multiple forms of infrastructures, e.g., blade centers where each server is a server blade [23], clusters of traditional servers where each server is an ordinary processor [24], and multicore server processors where each server is a single core [25]. These blades/processors/cores are simply called servers in this paper for easy presentation. The serving process operates as follows. Customers submit their service requests to the service provider, who then runs tasks/applications on the multiserver system to provide services for serving these requests.

Consider a multiserver system composed of M homogeneous servers running at the same speed of s. The homogeneous multiserver model is simple yet effective and has been widely used in the literature [1], [4], [5], [6], [20], [25]. The multiserver is typically modeled as an M/M/M queuing system in which service requests arrive the multiserver system with a rate of λ, wait in a first-come-first-served (FCFS) queue with an infinite capacity when all the M servers are busy, and get processed with a speed s when the servers are available. The arrival of service requests is governed by a Poisson process and the M servers process these requests in parallel. Note that our homogeneous multiserver model can be extended to heterogeneous multiserver systems. Like the method proposed in [26], we can configure a heterogeneous cloud computing platform as multiple homogeneous multiserver systems. Each multiserver system is deployed with special software and is devoted to serve one type of service requests and applications. In a multiserver system, the servers are homogenous and execute at the same speed. Each multiserver system can be treated as an M/M/M queuing system which handles the requests following the FCFS discipline. Using the configuration method [26], our cloud service provider model for homogeneous servers can be readily applied to heterogeneous cloud computing systems. Note that the configuration method is suitable for the current real-world cloud service providers such as Amazon EC2 [2], Microsoft Azure [3], and Aliyun [27] since they mainly focus on the multiserver system's heterogeneity from few aspects (e.g., CPU and memory). But eventually the heterogeneity could come from various aspects such as CPU cores/frequency, memory type/capacity, disk type/capacity, network type/topology, software configurations, and etc. The configuration method then becomes practically infeasible in this case since it may need infinite combinations of homogeneous multiserver systems. Thus, the scalability of the extension method should be improved for coping with all the possible scenarios. We leave the detailed discussion of this aspect to future work due to page limit.

For a customer's request in the M/M/M queuing system,[1] the service execution time is an exponential random variable, represented by $x = r/s$ with mean $\overline{x} = \overline{r}/s$, where r is the number of instructions to be executed for completing the service request. Let μ denote the service rate of customer requests, which is in fact the average number of service requests completed by a server with speed s in one unit of time. The rate μ is formulated as $\mu = 1/\overline{x} = s/\overline{r}$. Let ρ be the utilization of a server, which is defined as the average percentage of time that the server is busy and is computed as

$$\rho = \frac{\lambda}{M\mu} = \frac{\lambda}{M(s/\overline{r})} = \frac{\lambda\overline{r}}{Ms}. \tag{1}$$

Suppose P_k is the probability that k service requests are waiting or processing in the M/M/M queuing system. According to the queuing theory [29], P_k is derived as

$$P_k = \begin{cases} P_0 \frac{(M\rho)^k}{k!}, & k \leqslant M \\ P_0 \frac{M^M \rho^k}{M!}, & k > M \end{cases}, \tag{2}$$

1. Simple examples of the queuing system can be found in [28].

where P_0 is the probability of a queue with no tasks and is expressed as

$$P_0 = \left(\sum\nolimits_{k=0}^{M-1} \frac{(M\rho)^k}{k!} + \frac{(M\rho)^M}{M!} \cdot \frac{1}{1-\rho} \right)^{-1}. \tag{3}$$

Let P_b denote the probability that a newly submitted service request waits in the queue when all the servers are busy, and it is formulated as

$$P_b = \sum_{k=M}^{\infty} P_k = \frac{P_M}{1-\rho}, \tag{4}$$

where P_M is the probability of having M service requests in the system and it can be obtained using Eq. (2).

For a newly arrived service request, let W and $f_W(t)$ be the service's waiting time and corresponding probability density function (PDF) [5], respectively. Then $f_W(t)$ is

$$f_W(t) = (1 - P_b)u(t) + M\mu P_M e^{-(1-\rho)M\mu t}, \tag{5}$$

where $u(t)$ is a unit impulse function and is expressed as

$$u_z(t) = \begin{cases} z, & 0 \leqslant t \leqslant \frac{1}{z} \\ 0, & t > \frac{1}{z} \end{cases}, \tag{6}$$

and $u(t) = \lim_{z\to\infty} u_z(t)$ holds. Note that $u_z(t)$ is in fact a PDF of an arbitrary random variable, thus can be treated as the PDF of random variable r in the following [5]. The function $u_z(t)$ satisfies two properties, i.e.,

$$\int_0^{\infty} u_z(t)\, dt = 1 \tag{7}$$

$$\int_0^{\infty} t u_z(t)\, dt = z \int_0^{1/z} t\, dt = \frac{1}{2z}. \tag{8}$$

Given these, the cumulative distribution function of W is derived as $F_W(t) = 1 - \frac{P_M}{1-\rho} e^{-M\mu(1-\rho)t}$.

2.3 Service Level Agreement Model

To ensure the customer satisfaction, we need to stipulate a service-level agreement (SLA) between service providers and customers on the price and the service quality. A widely-used SLA model [30] defining the service charge for a service request with execution requirement r and response time R is adopted in the paper. That is

$$C(r,R) = \begin{cases} ar, & \text{if} \quad 0 \leqslant R \leqslant \frac{cr}{s_0} \\ ar - \gamma(R - \frac{cr}{s_0}), & \text{if} \quad \frac{cr}{s_0} < R < (\frac{a}{\gamma} + \frac{c}{s_0})r, \\ 0, & \text{if} \quad R \geqslant (\frac{a}{\gamma} + \frac{c}{s_0})r \end{cases} \tag{9}$$

where a is the service charge per unit amount of service as well as γ is a coefficient indicating the compensation strength because of low service quality. c is a constant related to the SLA and s_0 is the expected service processing speed of customers. From Eq. (7), it is clear that the service is charged based on the service response time R. Specifically, the service charge function is defined as follows. (i) If the response time R is shorter than $\frac{cr}{s_0}$, indicating a high quality of service is provided since the service request is processed at the expected speed of customers, then the service charge to a customer is

ar which is linearly proportional to the task execution requirement. (ii) If the response time R is between the interval $[cr/s_0, (a/\gamma + c/s_0)r]$, indicating a low quality of service is provided since the processing time of the service request exceeds the time at the expected speed, then the charge decreases as the response time R increases. (iii) If R is longer than $(\frac{a}{\gamma} + \frac{c}{s_0})r$, then the service is provided as free since the request is waiting too long in the queue.

The response time R of a service request is calculated as the sum of the waiting time W and the processing time r/s, that is, $R = W + r/s$. Substituting $R = W + r/s$ into Eq. (7), the service charge function can be rewritten as

$$C(r, W) = \begin{cases} ar, & \text{if} \quad 0 \leqslant W \leqslant (\frac{c}{s_0} - \frac{1}{s})r \\ (a + \frac{c\gamma}{s_0} - \frac{\gamma}{s})r - \gamma W, & \text{if} \quad (\frac{c}{s_0} - \frac{1}{s})r < W \\ & \qquad \leqslant (\frac{a}{\gamma} + \frac{c}{s_0} - \frac{1}{s})r \\ 0, & \text{if} \quad W > (\frac{a}{\gamma} + \frac{c}{s_0} - \frac{1}{s})r \end{cases}. \tag{10}$$

Since the PDF of r and W are given in Eqs. (5) and (6), the expectation of the service charge can be readily derived. As shown in Eq. (8), the charge to a service request is decided by parameters a, γ, c, r, W, s. In this paper, we focus on homogeneous multiserver systems and thus do not consider the server heterogeneity in the pricing model (i.e., charge function). However, our model can be extended to heterogeneous multiserver systems using the method [26]. The architecture of heterogeneous multiserver systems presented in [26] indicates that the parameters considered in our pricing model are sufficient but need different settings to capture the server heterogeneity. For example, our pricing model should allow the instruction related parameters of a service request to be changed with the service type and the server speed and size to be changed with the server type.

The user satisfaction of a service is evaluated from two aspects: quality of service (QoS) and price of service (PoS). QoS is a subjective concept that is introduced to describe the discrepancy between users' expectations on how the service request should be served and users' perceptions on how the service is actually performed. The expectation is built on the established price. Obviously, the higher the PoS is, the higher the expectation would be. For a given established price, if the perception (actual) performance surpasses the expectation performance, the QoS is then deemed as high, and vice verse. Given above, the user's QoS satisfaction for a service, U_{QoS}, is formulated as [31]

$$U_{\text{QoS}} = \begin{cases} 1, & \text{if} \quad P_{\text{act}} \geqslant P_{\exp} \\ e^{-\left|(P_{\text{act}} - P_{\exp})/P_{\exp}\right|}, & \text{if} \quad P_{\text{act}} < P_{\exp} \end{cases}, \tag{11}$$

where P_{act} is the actual performance and holds for $P_{\text{act}} = W$, and $P_{\exp}$ is the expectation performance and holds for $P_{\exp} = (\frac{c}{s_0} - \frac{1}{s})r$. Substituting $P_{\text{act}} = W$ and $P_{\exp} = (\frac{c}{s_0} - \frac{1}{s})r$ into Eq. (9), U_{QoS} is then re-written as

$$U_{\text{QoS}}(r, W) = \begin{cases} 1, & \text{if} \quad 0 \leqslant W \leqslant (\frac{c}{s_0} - \frac{1}{s})r \\ e^{\frac{(c/s_0 - 1/s)r - W}{(c/s_0 - 1/s)r}}, & \text{if} \quad W > (\frac{c}{s_0} - \frac{1}{s})r \end{cases}. \tag{12}$$

The PoS satisfaction of a user for a service, U_{PoS}, defined as the comparison between the expected price and the actual price, is formulated as [31]

$$U_{\text{PoS}} = e^{(C_{\exp} - C_{\text{act}})/C_{\exp}}, \tag{13}$$

where $C_{\exp}$ and C_{act} represent user's expected price and service's actual price, respectively. Clearly, if the actual price C_{act} is equal to the expected price $C_{\exp}$, the default PoS satisfaction U_{PoS} is set to 1, indicating that the price doesnot impact the total satisfaction. If C_{act} is higher than $C_{\exp}$, the user would be disappointed by the high price, thus U_{PoS} is less than 1 and decreases with the increasing actual price. If C_{act} is lower than $C_{\exp}$, the user would be delighted by the low price, thus U_{PoS} is greater than 1 and increases as the actual price decreases. The PoS (i.e., the charge of service) is determined by W and r. As the three modes of PoS defined in Eq. (8), the PoS satisfaction U_{PoS} needs to consider the three cases. (i) $C_{\exp} = C_{\text{act}} = ar$, (ii) $C_{\text{act}} = (a + \frac{c\gamma}{s_0} - \frac{\gamma}{s})r - \gamma W$ and $C_{\exp} = ar$, and (iii) $C_{\text{act}} = 0$ and $C_{\exp} = ar$. Substituting the values of C_{act} and $C_{\exp}$ into Eq. (11), U_{PoS} is then formulated as

$$U_{\text{PoS}}(r, W) = \begin{cases} 1, & \text{if} \quad 0 \leqslant W \leqslant \left(\frac{c}{s_0} - \frac{1}{s}\right)r, \\ e^{\frac{\gamma}{a}\left(\frac{1}{s} + \frac{W}{r} - \frac{c}{s_0}\right)}, & \text{if} \quad \left(\frac{c}{s_0} - \frac{1}{s}\right)r < W \\ & \qquad \leqslant \left(\frac{a}{\gamma} + \frac{c}{s_0} - \frac{1}{s}\right) \\ e, & \text{if} \quad W > \left(\frac{a}{\gamma} + \frac{c}{s_0} - \frac{1}{s}\right)r \end{cases}. \tag{14}$$

The user satisfaction U_{sat} of a service is defined as the product of QoS satisfaction and PoS satisfaction, i.e.,

$$U_{\text{sat}} = U_{\text{QoS}}(r, W) \times U_{\text{PoS}}(r, W) =$$
$$\begin{cases} 1, & \text{if} \quad 0 \leqslant W \leqslant \left(\frac{c}{s_0} - \frac{1}{s}\right)r \\ e^{1 + \frac{\gamma}{a}\left(\frac{1}{s} - \frac{c}{s_0}\right) + \left(\frac{\gamma}{a} - \frac{1}{c/s_0 - 1/s}\right)\frac{W}{r}}, & \text{if} \quad \left(\frac{c}{s_0} - \frac{1}{s}\right)r < W \\ & \qquad \leqslant \left(\frac{a}{\gamma} + \frac{c}{s_0} - \frac{1}{s}\right) \\ e^{2 - \frac{W}{\left(\frac{c}{s_0} - \frac{1}{s}\right)r}}, & \text{if} \quad W > \left(\frac{a}{\gamma} + \frac{c}{s_0} - \frac{1}{s}\right)r \end{cases}, \tag{15}$$

where $U_{\text{QoS}}(r, W)$ and $U_{\text{PoS}}(r, W)$ are given in Eqs. (9) and (12), respectively.

3 Dynamic Pricing Strategy

This section introduces the proposed dynamic pricing strategy for cloud service providers in detail. Since the strategy is designed based on CPV, in this section we first describe how to use the SERVQUAL score to model the CPV and hence estimate the user's demand for services, followed by a validation of the correctness of the SERVQUAL model and the proposed user demand function. Considering that using the analytical method to directly compute user service demand is challenging, we also provide a heuristic algorithm to obtain user demand based on CPV.

3.1 SERVQUAL Score to Model CPV

SERVQUAL [32] is a research instrument designed for capturing consumer's expectations and perceptions of a service from multiple dimensions that represent service quality. The fundamental of SERVQUAL is the gap model [33] that service quality is determined by the difference between the level of service perceived by users and the level of service expected by users. Based on the gap model, SERVQUAL score (represented by SQS) is utilized to model the CPV for a service, and is expressed as

$$\mathrm{SQS} = \sum_{\sigma=1}^{n} w_\sigma(\mathrm{Pec}_\sigma - \mathrm{Exp}_\sigma), \tag{16}$$

$$\begin{aligned}
&\mathrm{SQS}_{\mathrm{price}}(r) = \overline{\mathrm{SQS}_{\mathrm{price}}(r,W)} = \overline{\mathrm{Pec}_{\mathrm{price}} - \mathrm{Exp}_{\mathrm{price}}}\\
&= \overline{C(r,W) - ar} = \int_0^{\left(\frac{c}{s_0}-\frac{1}{s}\right)} (ar - ar)\, dt\\
&\quad + \int_{\left(\frac{c}{s_0}-\frac{1}{s}\right)}^{\left(\frac{a}{\gamma}+\frac{c}{s_0}-\frac{1}{s}\right)} f_W(t)\left(\left(\frac{c\gamma}{s_0} - \frac{\gamma}{s}\right)r - \gamma t\right) dt + \int_{\left(\frac{a}{\gamma}+\frac{c}{s_0}-\frac{1}{s}\right)}^{\infty} -ar f_W(t)\, dt\\
&= \int_{\left(\frac{c}{s_0}-\frac{1}{s}\right)}^{\left(\frac{a}{\gamma}+\frac{c}{s_0}-\frac{1}{s}\right)} \left(\frac{c\gamma}{s_0} - \frac{\gamma}{s}\right) r M\mu P_M e^{-(1-\rho)M\mu t}\, dt\\
&\quad - \int_{\left(\frac{c}{s_0}-\frac{1}{s}\right)}^{\left(\frac{a}{\gamma}+\frac{c}{s_0}-\frac{1}{s}\right)} M\mu P_M \gamma t e^{-(1-\rho)M\mu t}\, dt\\
&\quad - \int_{\left(\frac{a}{\gamma}+\frac{c}{s_0}-\frac{1}{s}\right)}^{\infty} ar M\mu P_M e^{-(1-\rho)M\mu t}\, dt\\
&= \frac{P_M r}{\rho - 1}\left(\frac{c\gamma}{s_0} - \frac{\gamma}{s}\right)\left(e^{-(1-\rho)M\mu(\frac{a}{\gamma}+\frac{c}{s_0}-\frac{1}{s})r} - e^{-(1-\rho)M\mu(\frac{c}{s_0}-\frac{1}{s})r}\right)\\
&\quad - \frac{P_M\gamma}{\rho - 1}\cdot\left(\left(1 + (1-\rho)M\mu\left(\frac{a}{\gamma} + \frac{c}{s_0} - \frac{1}{s}\right)r\right)e^{-(1-\rho)M\mu\left(\frac{a}{\gamma}+\frac{c}{s_0}-\frac{1}{s}\right)r}\right.\\
&\quad \left. - \left(1 + (1-\rho)M\mu\left(\frac{c}{s_0} - \frac{1}{s}\right)r\right)e^{-(1-\rho)M\mu\left(\frac{c}{s_0}-\frac{1}{s}\right)r}\right)\\
&\quad + \frac{arP_M}{\rho - 1} e^{-(1-\rho)M\mu\left(\frac{a}{\gamma}+\frac{c}{s_0}-\frac{1}{s}\right)r}.
\end{aligned} \tag{17}$$

$$\begin{aligned}
&\mathrm{SQS}_{\mathrm{price}} = \overline{\mathrm{SQS}_{\mathrm{price}}(r)} = \int_0^\infty \frac{1}{\overline{r}} e^{-\frac{z}{\overline{r}}} \mathrm{SQS}_{\mathrm{price}}(z) dz\\
&= \frac{P_M}{(\rho-1)\overline{r}}\left(\frac{1}{(\overline{r} + (1-\rho)M\mu(\frac{a}{\gamma} + \frac{c}{s_0} - \frac{1}{s}))^2}\right.\\
&\quad \left. - \frac{1}{(\overline{r} + (1-\rho)M\mu(\frac{c}{s_0} - \frac{1}{s}))^2}\right)\cdot\left(\frac{c\gamma}{s_0} - \frac{\gamma}{s}\right)\\
&\quad - \frac{P_M\gamma}{(\rho-1)\overline{r}}\left(\overline{r} + (1-\rho)M\mu\left(\frac{a}{\gamma} + \frac{c}{s_0} - \frac{1}{s}\right)\right.\\
&\quad \cdot \frac{1}{\left(\frac{1}{\overline{r}} + (1-\rho)M\mu\left(\frac{a}{\gamma} + \frac{c}{s_0} - \frac{1}{s}\right)\right)^2} - (\overline{r} + (1-\rho)M\mu\left(\frac{c}{s_0} - \frac{1}{s}\right)\\
&\quad \left. \cdot \frac{1}{(\frac{1}{\overline{r}} + (1-\rho)M\mu(\frac{c}{s_0} - \frac{1}{s}))^2}\right)\Bigg) + \frac{aP_M}{(\rho-1)\overline{r}}\\
&\quad \cdot \frac{1}{(\frac{1}{\overline{r}} + (1-\rho)M\mu\left(\frac{a}{\gamma} + \frac{c}{s_0} - \frac{1}{s})\right)^2}.
\end{aligned} \tag{18}$$

$$\begin{aligned}
&\mathrm{SQS}_{\mathrm{waiting-time}}(r) = \overline{\mathrm{SQS}_{\mathrm{waiting-time}}(r,W)}\\
&= \overline{\mathrm{Pec}_{\mathrm{waiting-time}} - \mathrm{Exp}_{\mathrm{waiting-time}}} = \overline{W - \left(\frac{c}{s_0} - \frac{1}{s}\right)r}\\
&= \int_0^\infty \left(t - \left(\frac{c}{s_0} - \frac{1}{s}\right)r\right) f_W(t)\, dt\\
&= \int_0^\infty tM\mu P_M e^{-(1-\rho)M\mu t}\, dt - \left(\frac{c}{s_0} - \frac{1}{s}\right)r \cdot \int_0^\infty f_W(t)\, dt\\
&= \frac{P_M}{(1-\rho)^2 M\mu} - \frac{P_M}{1-\rho} - (1 - P_b)\left(\frac{c}{s_0} - \frac{1}{s}\right)r.
\end{aligned} \tag{19}$$

$$\begin{aligned}
\mathrm{SQS}_{\mathrm{waiting-time}} &= \overline{\mathrm{SQS}_{\mathrm{waiting-time}}(r)}\\
&= \int_0^\infty \frac{1}{\overline{r}} e^{-\frac{z}{\overline{r}}} \mathrm{SQS}_{\mathrm{waiting-time}}(z)\, dz\\
&= \frac{P_M}{(1-\rho)^2 M\mu} - \frac{P_M}{1-\rho} - (1 - P_b)\left(\frac{c}{s_0} - \frac{1}{s}\right)\overline{r}.
\end{aligned} \tag{20}$$

where n is the number of the service's attributes, w_σ is the weight of the service's σth attribute, Pec_σ is the perception for the attribute, and Exp_σ is the expectation for the attribute.

In this paper, we consider two most important attributes (i.e., price and waiting-time) of a service and treat their importance as equal. In such a case, we have $n = 2$, $w_1 = w_2 = 0.5$, $\mathrm{Pec}_1 = \mathrm{Pec}_{\mathrm{price}}$, $\mathrm{Exp}_1 = \mathrm{Exp}_{\mathrm{price}}$, $\mathrm{Pec}_2 = \mathrm{Pec}_{\mathrm{waiting-time}}$, and $\mathrm{Exp}_2 = \mathrm{Exp}_{\mathrm{waiting-time}}$. For the price attribute, the perception $\mathrm{Pec}_{\mathrm{price}}$ is the service charge $C(r,W)$ given in Eq. (8) and the expectation $\mathrm{Exp}_{\mathrm{price}}$ is ar while for the waiting time attribute, the perception $\mathrm{Pec}_{\mathrm{waiting-time}}$ is W and the expectation $\mathrm{Exp}_{\mathrm{waiting-time}}$ is $(\frac{c}{s_0} - \frac{1}{s})r$. Using the gap model, we can derive the SQS with respect to price and waiting time attributes (represented by $\mathrm{SQS}_{\mathrm{price}}$ and $\mathrm{SQS}_{\mathrm{waiting-time}}$) as in Eqs. (3.1)-(14). (See the top of this page.) Note that the derived $\mathrm{SQS}_{\mathrm{price}}$ and $\mathrm{SQS}_{\mathrm{waiting-time}}$ are with different units. To consider the two attributes together, we need to normalize the value of the two variables. Let $\mathrm{SQS}'_{\mathrm{price}}$ and $\mathrm{SQS}'_{\mathrm{waiting-time}}$ represent the normalized $\mathrm{SQS}_{\mathrm{price}}$ and $\mathrm{SQS}_{\mathrm{waiting-time}}$ obtained by the maximum-minimum normalization method [34], respectively. Then, the SERVQUAL score SQS can be calculated as

$$\mathrm{SQS} = 0.5 \times \mathrm{SQS}'_{\mathrm{price}} + 0.5 \times \mathrm{SQS}'_{\mathrm{waiting-time}}. \tag{21}$$

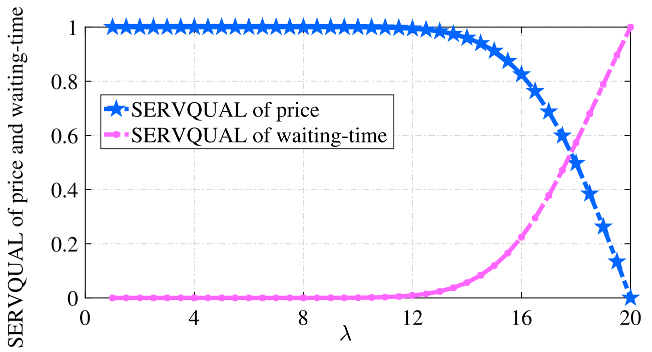
We adopt a user demand function [35] which indicates that the user's demand for services is linear with the CPV (quantified by SQS). That is

$$D_{\mathrm{CPV}}(\mathrm{SQS}) = \psi + \phi \times \mathrm{SQS}, \tag{22}$$

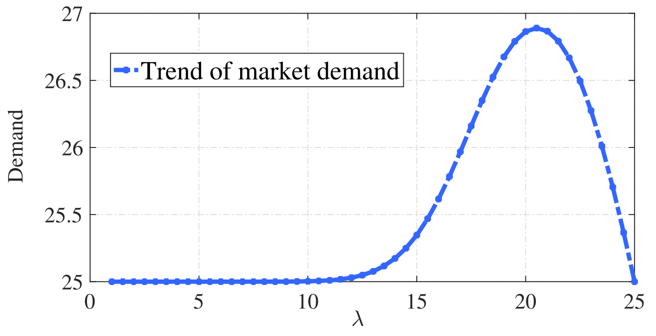
where ψ and ϕ $(\psi, \phi > 0)$ are constants representing the basic demand and potential market demand. Note that the calculation of SQS is independent of the user demand function used here.

3.2 Validate SERVQUAL Model and Demand Function

To validate the SERVQUAL model and user demand function, we carry out several simulations to evaluate $\mathrm{SQS}_{\mathrm{price}}$,



(a) The normalized changing trend of $\text{SQS}_{\text{price}}$ and SQS.



(b) The changing trend of user demand.

Fig. 1. $\text{SQS}_{\text{price}}$, $\text{SQS}_{\text{waiting-time}}$, and D_{CPV} for different service arrival rates (λ).

$\text{SQS}_{\text{waiting-time}}$, and D_{CPV} for different service arrival rates (λ). In the simulations, ψ is set to 15 and ϕ is set to 20 as in [35]. The normalized results of $\text{SQS}_{\text{waiting-time}}$ and $\text{SQS}_{\text{price}}$ are summarized in Fig. 1a. From the figure, we can clearly observe that with the increase in the number of arrival services, $\text{SQS}_{\text{waiting-time}}$ and $\text{SQS}_{\text{price}}$ both remain the same when the service number is less than 11 and then $\text{SQS}_{\text{waiting-time}}/\text{SQS}_{\text{price}}$ significantly increases/decreases when the service number is larger than 11. This observation is in line with the SLA model that when users' waiting time exceeds a threshold, the service provider will reduce the service charge/price to compensate for the latency. The results of D_{CPV} are summarized in Fig. 1b. It can be easily observed from the figure that with the increase in the number of arrival services, D_{CPV} remains the same when the service number is less than 11. When the service number is larger than 11, D_{CPV} increases first and then decreases. This observation is in line with the phenomenon in the cloud service market that when users' waiting time exceeds a threshold, the user service demand will increase first due to the compensation and then decrease due to the intolerable waiting time. The two observations discussed above show the correctness of the SERVQUAL model and user demand function.

3.3 Derive User Demand Based on CPV

Using Eqs. (21)-(22) to compute $D_{\text{CPV}}(\text{SQS})$ directly is challenging due to the complexity of $\text{SQS}_{\text{price}}$ and $\text{SQS}_{\text{waiting-time}}$. However, by using the standard bisection (SB) method, we can easily find a numerical solution of $D_{\text{CPV}}(\text{SQS})$. Therefore, we develop a SB-based algorithm to obtain the user demand $D_{\text{CPV}}(\text{SQS})$. The details of our algorithm are given in Algorithm 1. The algorithm works as follows. It first defines a function $Y = \psi + \phi \times \text{SQS} - \lambda$ and finds a monotone interval $[\lambda_{\text{low}}, \lambda_{\text{high}}]$ such that $Y(\lambda_{\text{low}}) > 0$ and $Y(\lambda_{\text{high}}) < 0$ hold (line 1). It then iteratively divides the interval in two by computing the mid point $\lambda_{\text{mid}} = (\lambda_{\text{low}} + \lambda_{\text{high}})/2$ of the interval and the value of function Y at that point if $Y(\lambda_{\text{low}}) - Y(\lambda_{\text{high}}) > \varepsilon$ holds for an arbitrarily small positive number ε (lines 2-9). In each round of iteration, the searching interval $[\lambda_{\text{low}}, \lambda_{\text{high}}]$ and the mid point λ_{mid} are updated. When the iteration stops, the algorithm selects the updated mid point as the $D_{\text{CPV}}(\text{SQS})$ (lines 10-11).

Algorithm 1. The SB-Based Method to Derive User Demand

```
1: define a function $Y = \psi + \phi \times \text{SQS} - \lambda$ and find a
   monotone interval $[\lambda_{\text{low}}, \lambda_{\text{high}}]$ such that $Y(\lambda_{\text{low}}) > 0$
   and $Y(\lambda_{\text{high}}) < 0$;
2: **while** $Y(\lambda_{\text{low}}) - Y(\lambda_{\text{high}}) > \varepsilon$ **do**
3:   $\lambda_{\text{mid}} = (\lambda_{\text{low}} + \lambda_{\text{high}})/2$;
4:   **if** $Y(\lambda_{\text{mid}}) < 0$ **then**
5:     $\lambda_{\text{high}} = \lambda_{\text{mid}}$;
6:   **else**
7:     $\lambda_{\text{low}} = \lambda_{\text{mid}}$;
8:     **break**;
9:   calculate $Y(\lambda_{\text{low}})$ and $Y(\lambda_{\text{high}})$;
10: $\lambda_{\text{mid}} = (\lambda_{\text{low}} + \lambda_{\text{high}})/2$;
11: $D_{\text{CPV}}(\text{SQS}) = \lambda_{\text{mid}}$;
```

4 Profit Optimization Scheme

In this section, we describe the proposed profit optimization scheme in detail. Specifically, we first show the method to calculate the cloud service provider's profit, then discuss the optimal multiserver configuration for maximizing the profit, and finally present the heuristic algorithm to obtain the maximum profit considering the risk in the pricing contract.

4.1 Calculate the Cloud Service Provider's Profit

The profit of a cloud service provider is generated from the gap between the revenue earned by purchasing services to customers and the monetary cost of processing customers' service requests. Below, we describe the methods to estimate the revenue, cost, and profit in detail.

Revenue. To estimate the revenue $\mathcal{R}$ of a cloud service provider, we first need to know the expected charge C_{exp} to a service request. The C_{exp} is calculated as follows [4]:

$$C(r) = \overline{C(r,W)} = \int_{-\infty}^{\infty} f_W(t)C(r,t)\,dt \tag{23}$$

$$C_{\text{exp}} = \overline{C(r)} = \int_{0}^{\infty} f_r(z)C(z)dz, \tag{24}$$

where $f_W(t)$ is the PDF of waiting time W given in Eq. (5), $f_r(z)$ is the impulse function given in Eq. (6), and $C(r,W)/C(r,t)$ is given in Eq. (8). Using Eq. (2.3) and the optimal setting of parameter γ (i.e., $\gamma = a/(c/s_0 - 1/s)$ [4]), C_{exp} is derived as

$$C_{\text{exp}} = -\frac{P_b a\bar{r}}{(2(Ms - \lambda\bar{r})(\frac{c}{s_0} - \frac{1}{s}) + 1)((Ms - \lambda\bar{r})(\frac{c}{s_0} - \frac{1}{s}) + 1)} + a\bar{r} = C(M,s). \tag{25}$$

Clearly, C_{exp} can be described as a function of the number and speed of servers (i.e., M and s).

The total revenue of a cloud service provider is the product of the expected charge to a service request and the user demand for this service, i.e.,

$$Revenue = C(M, s) \times D_{\text{CPV}}, \tag{26}$$

where D_{CPV} is the user demand for this service based on customer perceived value.

Cost. The cost of a cloud service provider contains two parts: the money paid to rent cloud computing infrastructure and the electricity expense to maintain the operation of the computing infrastructure. The cost model is a commonly accepted model that has been widely used in the literature [1], [4], [5], [6], [20], [26]. Suppose δ is the fee paid by the provider to rent a server during a sale period T. For renting a multiserver system composed of M servers during the period T, the provider then needs to pay

$$Rent = M\delta \cdot T. \tag{27}$$

Electricity fee is a significant expense for today's data centers and can be calculated as the product of servers' energy consumption and electricity price. The energy consumption of a server can be modeled at different levels of abstraction. In this paper, we consider server's energy consumption at the abstraction level of digital CMOS circuit. Let E^T represent the energy consumption of the M servers during the sale period T. Then, using the CMOS-level power model introduced in [5], E^T can be calculated as

$$E^T = M \cdot (Pow_{\text{dyn}} \cdot \rho + Pow_{\text{sta}}) \cdot T, \tag{28}$$

where ρ is the server utilization, Pow_{dyn} is the dynamic power dissipation, and Pow_{sta} is the static power dissipation. Assuming C^T is the price of the energy consumed by servers during the period T, the electricity bill is derived as

$$Bill = E^T \cdot C^T = M \cdot (Pow_{\text{dyn}} \cdot \rho + Pow_{\text{sta}}) \cdot T \cdot C^T. \tag{29}$$

Increasing server utilization is helpful to serve more requests and hence bring more revenue, but it also leads to a higher electricity bill. The service provider obtains more profit only when the extra revenue is greater than the additional electricity bill.

Profit. The cloud service provider's profit during a sale period T is derived as the revenue minus the expenses including the rental cost and electricity cost, i.e.,

$$Profit = Revenue - Rent - Bill, \tag{30}$$

where $Revenue$, $Rent$, and $Bill$ are given in Eqs. (23), (24), and (26), respectively. Substituting these into Eq. (27), we can rewrite $Profit$ as a function, represented by $G(M, s)$, of the multiserver configuration M and s. That is

$$\begin{aligned} Profit = {} & C(M, s) \cdot D_{\text{CPV}} - M\delta \cdot T \\ & - M(Pow_{\text{dyn}} \cdot \rho + Pow_{\text{sta}}) \cdot C^T \cdot T = G(M, s). \end{aligned} \tag{31}$$

4.2 Derive the Optimal Setup to Maximize the Profit

As introduced in Section 2.2, P_M is the probability of having exact M service requests in the system and can be derived using Eq. (2) as

$$P_M = P_0 \frac{(M\rho)^M}{M!}.$$

By applying Taylor series expansions $\sum_{k=0}^{M-1} (M\rho)^k / k! \approx e^{M\rho}$ and $M! \approx \sqrt{2\pi M}(\frac{M}{e})^M$, P_M is then approximately expressed as

$$P_M \approx \frac{1-\rho}{\sqrt{2\pi M}(1-\rho)(\frac{e^{\rho-1}}{\rho})^M + 1}.$$

Substituting the approximate P_M and the P_b given in Eq. (4) into $C(M, s)$, we can derive

$$\begin{aligned} C(M, s) \approx a\bar{r}\Big(1 - & \frac{1}{\big(2(Ms - \lambda\bar{r})(\frac{c}{s_0} - \frac{1}{s}) + 1\big)} \\ \cdot & \frac{1}{\big((Ms - \lambda\bar{r})(\frac{c}{s_0} - \frac{1}{s}) + 1\big)} \cdot \frac{1}{\big(\sqrt{2\pi M}(1-\rho)(\frac{e^\rho}{e\rho})^M + 1\big)}\Big). \end{aligned}$$

For the sake of easy representation, we let

$$f_1 = \sqrt{2\pi M}(1-\rho)(e^\rho / e\rho)^M + 1,$$

$$f_2 = 2(Ms - \lambda\bar{r})\left(\frac{c}{s_0} - \frac{1}{s}\right) + 1,$$

$$f_3 = (Ms - \lambda\bar{r})\left(\frac{c}{s_0} - \frac{1}{s}\right) + 1.$$

$C(M, s)$ is then rewritten as

$$C(M, s) = a\bar{r}\left(1 - \frac{1}{f_1 + f_2 + f_3}\right).$$

From Eq. (1) we can deduce that $M\rho = \frac{D_{\text{CPV}}\bar{r}}{s}$. Substituting it into $C(M, s)$ and $G(M, s)$, then we can find the optimal M for maximizing profit by calculating the partial derivative of $G(M, s)$ with respect to M. That is

$$\frac{\partial G}{\partial M} = D_{\text{CPV}} \frac{\partial C}{\partial M} - (\delta + P_{\text{sta}} \cdot C^T) \cdot T.$$

The partial derivatives used for calculating $\frac{\partial G}{\partial M}$ are

$$\frac{\partial C}{\partial M} = \frac{a\bar{r}}{(f_1 \cdot f_2 \cdot f_3)^2}\left(\frac{\partial f_1}{\partial M} \cdot f_2 \cdot f_3 + \frac{\partial f_2}{\partial M} \cdot f_1 \cdot f_3 + \frac{\partial f_3}{\partial M} \cdot f_1 \cdot f_2\right),$$

$$\frac{\partial f_1}{\partial M} = \sqrt{2\pi}\left(\frac{1}{2\sqrt{M}}(1-\rho)\Psi + \frac{\rho}{\sqrt{M}}\Psi + \sqrt{M}(1-\rho)^2\Psi\right),$$

$$\frac{\partial f_2}{\partial M} = 2s\left(\frac{c}{s_0} - \frac{1}{s}\right), \frac{\partial f_3}{\partial M} = s\left(\frac{c}{s_0} - \frac{1}{s}\right), \Psi = (e^\rho / e\rho)^M.$$

Similarly, we can find the optimal s for maximizing profit by calculating the partial derivative of $G(M, s)$ with respect to s. That is

$$\frac{\partial G}{\partial s} = D_{\text{CPV}} \frac{\partial C}{\partial s} - D_{\text{CPV}}\bar{r}\xi(\alpha - 1)s^{\alpha-2} \cdot C^T \cdot T.$$

The partial derivatives used for calculating $\frac{\partial G}{\partial s}$ are

$$\frac{\partial C}{\partial s}=\frac{a\overline{r}}{(f_1f_2f_3)^2}\left(\frac{\partial f_1}{\partial s}f_2f_3+\frac{\partial f_2}{\partial s}f_1f_3+\frac{\partial f_3}{\partial s}f_1f_2\right),$$

$$\frac{\partial f_1}{\partial s}=\sqrt{2\pi M}\big(\rho+M(1-\rho)^2\big)\frac{\partial \Psi}{\partial s},\frac{\partial \Psi}{\partial s}=\frac{M}{s}(1-\rho)\Psi,$$

$$\frac{\partial f_2}{\partial s}=2\left(\frac{Mc}{s_0}-\frac{D_{\mathrm{CPV}}\overline{r}}{s^2}\right),\frac{\partial f_3}{\partial s}=\frac{Mc}{s_0}-\frac{D_{\mathrm{CPV}}\overline{r}}{s^2}.$$

Once the partial derivatives of $G(M,s)$ given in Eq. (4.1) with regard to M and s are obtained, we can compute and derive the optimal solutions to maximizing profit by setting these partial derivatives equal to 0.

4.3 Risk-Aware Profit Maximization

Although finding an optimal configuration of M and s is helpful to maximize the profit, the pricing contract still involves some risk for the cloud service provider. Specifically, due to the inherent uncertainty in the probability distribution of waiting times and the number of task instructions, the estimated distributions of time and cost may be different from the actual ones, leading to a decreased profit or even a loss. To make the studied problem more realistic, we formally define the risk based on loss and take into account the risk by adding it as a part of the optimization objective. The definitions of LOSS, RISK, and Risk-aware Profit are described as below.

- Definition 1: LOSS. For the ith running of a specific service, let M_i^* and s_i^* represent the actual number of servers and the actual operating speed of servers, respectively. When the cloud service provider generates a contract with his/her predicted configuration M and s, the loss of profit is then derived as

$$LOSS_i=G(M_i^*,s_i^*)-G(M,s), \tag{32}$$

  where G is the function to calculate the profit.
- Definition 2: RISK. For the ith running of a specific service, the risk of a service provider is defined as the average of profit loss over the previous $i-1$ runs of this service. That is

$$RISK_i=\frac{\sum_{j=1}^{i-1}LOSS_j}{i-1}. \tag{33}$$

- Definition 3: Risk-aware Profit. Considering the risk, the profit gained by providing a service is derived as

$$Profit_i^{\mathrm{risk}}=G(M,s)+RISK_i. \tag{34}$$

We summarize the basic steps needed to compute the risk-aware profit in Algorithm 2. After introducing the risk-related concepts and how to estimate the risk-aware profit, we then present the risk-aware profit maximization problem, followed by our approach to the profit maximization problem.

In this paper, we aim at maximizing the profit of the cloud service provider with the consideration of the risk in pricing contract by determining the optimal number of servers, operating speed of servers, and price of services. We assume that the cloud service provider optimizes his/her decisions at the beginning of each scheduling interval. The problem is formulated as

$$\mathrm{Max}:\ Profit_i^{\mathrm{risk}}=G(M,s)+RISK_i. \tag{35}$$

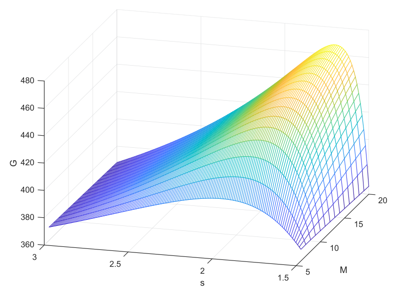
From Eqs. (28)-(31) we can see that the key of solving the maximization problem is to find the optimal point of $G(M,s)$. Solving the optimization function $G(M,s)$ analytically requires that M and s are continuous variables and needs to introduce a closed-form expression for approximating the function [4], [5], [6]. However, considering that M and s of actual systems are discrete variables and the error caused by closed-form approximation is non-negligible, analytically solving the function is not feasible. Although we cannot solve the function analytically, we still can carry out extensive experiments to plot out $G(M,s)$ under varying configurations for checking the existence of optimal solution to $G(M,s)$ as in [4], [6]. Fig. 2 presents the values of $G(M,s)$ under six different configurations. All the figures of $G(M,s)$ show that there must be an optimal point where the profit is maximized. Note that the experiment results here are not meant to be comprehensive but just to demonstrate that the optimal point exists. Based on this observation, we then use a heuristic algorithm to find a numerical optimal solution.

Algorithm 2. Calculate the Risk-Aware Profit

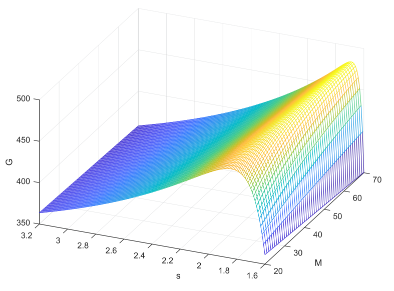
```
1: if $i=1$ then
2:     $LOSS_i=0$;
3:     $RISK_i=0$;
4:     calculate $Profit_i^{\mathrm{risk}}$ using Eq. (30);
5: else
6:     $j=1$;
7:     $LOSS_{\mathrm{total}}=0$;
8:     while $j\leqslant i-1$ do
9:         $LOSS_j=G(M_j^*,s_j^*)-G(M,s)$;
10:        $LOSS_{\mathrm{total}}=LOSS_{\mathrm{total}}+LOSS_j$;
11:        $j++$;
12:    calculate $RISK_j$ using Eq. (29);
13:    calculate $Profit_i^{\mathrm{risk}}$ using Eq. (30);
```

To find the numerical optimal solution, we first need to discretize the solution space. Under the discretized solution space, the simplest way to find the optimal solution is using brute force search, which is however not suitable because of its high time complexity. To overcome this shortcoming, we propose a discrete simulated annealing (SA)-based heuristic algorithm. The SA [36] is motivated by an analogy to physical annealing in solids. It is a probabilistic technique for approximating the global optimum of a given function. It can avoid being trapped at local optima by introducing a diversification mechanism (also called restart strategy). The details of our heuristic algorithm are given in Algorithm 3.

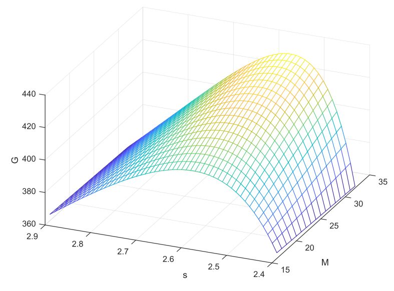
The algorithm begins with an initial solution obtained by assuming the multiserver configuration with the maximum capacity is utilized (lines 1-5). Note that the effectiveness of the algorithm is independent of the initial solution selection. Any arbitrary valid initial solution can be also used here. It then initializes the "temperature" of this solution as a high value, sets the Boltzmann constant k, the cooling rate CR, the iteration number $iterator_{\mathrm{num}}$, and the changing steps ($step_M$ and $step_s$) of multiserver size and speed, and generates a



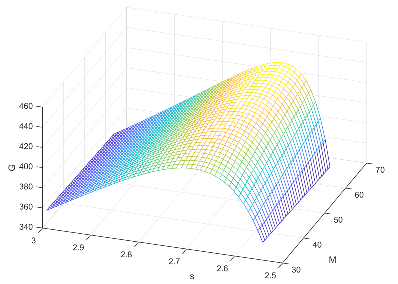
(a) $D_{\text{CPV}} = 4.8532$, $s = 1.5 \rightarrow 3.0$, $M = 6 \rightarrow 20$.



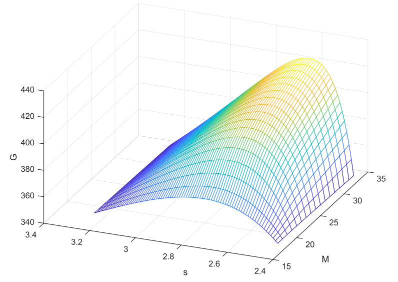
(b) $D_{\text{CPV}} = 9.7188$, $s = 1.6 \rightarrow 3.2$, $M = 20 \rightarrow 70$.



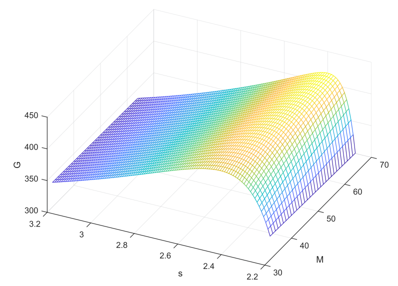
(c) $D_{\text{CPV}} = 14.5625$, $s = 2.4 \rightarrow 2.9$, $M = 16 \rightarrow 32$.



(d) $D_{\text{CPV}} = 19.7031$, $s = 2.55 \rightarrow 3$, $M = 32 \rightarrow 64$.



(e) $D_{\text{CPV}} = 24.625$, $s = 2.4 \rightarrow 3.2$, $M = 16 \rightarrow 32$.



(f) $D_{\text{CPV}} = 34.7344$, $s = 2.2 \rightarrow 3.2$, $M = 32 \rightarrow 64$.

Fig. 2. The values of $G(M, s)$ under varying configurations.

random constant ϖ_{thresh} from the range of (0,1] (lines 6-10). The "temperature" will gradually decrease during the simulated annealing process. At each temperature, the algorithm performs a certain amount of iterations and considers the neighbor solutions. Once a new solution is obtained (lines 13-18), its profit (represented by $Profit_{\text{next}}$) is calculated using Eq. (4.1) and compared to that (represented by $Profit_{\text{cur}}$) of the current solution (lines 19-20). If $\Delta Profit = Profit_{\text{next}} - Profit_{\text{opt}} > 0$, indicating the new solution can bring a higher profit, the new solution is accepted (lines 21-25). Otherwise, the new solution is a worse one and the algorithm then chooses to either keep the current solution (lines 31-33) or accept the new one with a probability (lines 26-30), in order to avoid getting stuck in local optima. During the simulated annealing process (i.e., the optimal solution searching process), the "temperature" is reduced using a cooling coefficient (line 34). After a large amount of iterations for searching the optimal solution, if the profits of neighbor solutions become very close

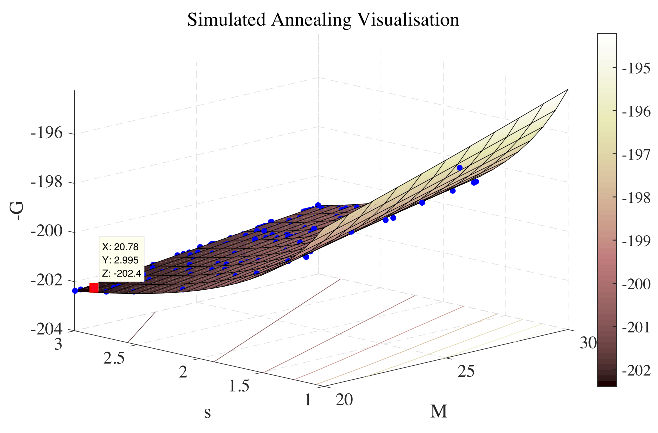


Fig. 3. An illustration of the process of searching the optimal solution using Algorithm 3.

(i.e., $|Profit_{\text{next}} - Profit_{\text{cur}}| < \varepsilon$ where ε is an arbitrarily small positive number), the outer while-loop is terminated and the global optimal solution is found (lines 35-36).

Fig. 3 shows the searching process of the optimal solution using Algorithm 3. The blue points in the figure are the local optimal solutions obtained during the search and the red point ($M = 20.78, s = 2.995, Profit = 202.4$) is the final solution found by Algorithm 3. From the figure we can deduce that the algorithm can not only avoid falling into local optima but also find a solution that is very close to the global optimal solution.

5 Numerical Results

In this paper, we consider the scenario that a cloud service provider rents a multiserver system to serve the requests submitted by cloud users. Such a multiserver system could be in the multiple forms of infrastructures, e.g., blade centers [23], clusters of traditional servers [24], and multicore server processors [25]. The cloud service provider charges users for processing their service requests based on the services amount and quality. To represent this scenario, we need to simulate a multiserver system and estimate the service demand of users. In the experiment, we assume that the multiserver system is in the form of multicore server processors, and thus we use several up-to-date multiprocessors (i.e., Intel Xeon Skylake-Platinum 8153 [37], Intel KNL-Xeon Phi 7250 [38], NVIDIA-V100 [39], IBM Power7-BladeCenter PS704 [40]) to simulate the multiserver system. The parameters used to estimate the electricity cost and renting cost of this multiserver system as well as the parameters related to service requests are listed in Table 1. The values of these parameters (such as the multiserver size M, the multiserver speed s, and the user service demand D_{CPV}) adopted in the experiments are either extracted from actual system (i.e.,

TABLE 1
Definition of Main Notations Used in the Experiment

Notation	Definition	Value
s_0	the customer's expected service processing speed	1 billion instructions per second [4]
c	a constant coefficient	3 [4]
$\overline{r}$	the expected number of instructions to complete a service	1 billion instructions [4]
a	the service charge per unit amount of service	15 cents per billion instructions [4]
δ	the fee of renting a server during a sale period	1.5 cents per second [4]
ξ	the processor dependent coefficient	9.4192 [4]
α	a constant coefficient	2 [4]
Pow_{sta}	the static power dissipation	2 $Watts$ per second [5]
C^T	the price of the energy consumed by the servers during a sale period	0.1 cents per $Watt \times second$ [5]

Intel Xeon Skylake-Platinum 8153, Intel KNL-Xeon Phi 7250, NVIDIA-V100, IBM Power7-BladeCenter PS704) or widely accepted in the literature (e.g., [4], [5]). The values of M and s used in the experiments also satisfy the constraint of server utilization, i.e., $0 < \rho = \frac{\lambda}{M\mu} = \frac{\lambda}{M(s/\overline{r})} = \frac{\lambda\overline{r}}{Ms} < 1$. Thus, our parameter settings should be realistic and acceptable.

In this section, three sets of simulation experiments are conducted to validate the proposed scheme from different perspectives. In the first set of experiments, we observe the changing trends of user demand and profit under varying multiserver configurations. This experiment is to show how the user demand and profit are affected by multiserver configurations. In the second set of experiments, we find the optimal configuration of multiserver speed and size as well as the corresponding profit under different user demands [41]. This experiment is to show how the optimal multiserver configuration and profit are affected by user demands. In the third set of experiments, we compare the proposed profit maximization scheme with two benchmarking methods. This experiment is to show the effectiveness of the proposed scheme in increasing profit. All the simulation experiments are implemented on a laptop equipped with 2.2 GHz Intel i7 six-core processor and 16 GB DDR4 memory, and running a Unix version of Matlab x64.

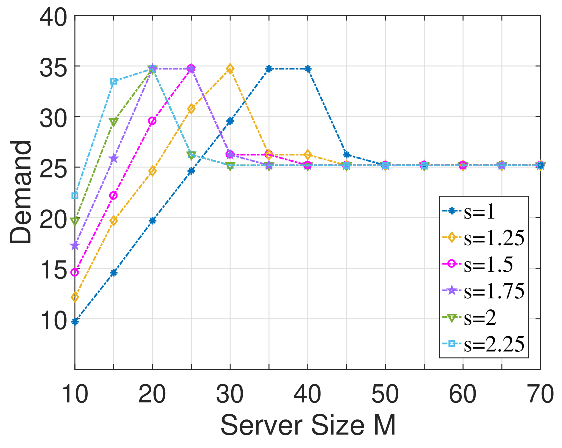
5.1 User Demand and Profit

In this set of experiments, we conduct a series of numerical calculations to show the user demand and profit under the varying configurations of the server size M and the server speed s, which take the value from $\{10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70\}$ and $\{1, 1.25, 1.5, 1.75, 2.0, 2.25\}$, respectively. To derive the user demand and profit, we use the parameters s_0, $\overline{r}$, a, and c as presented in Table 1.

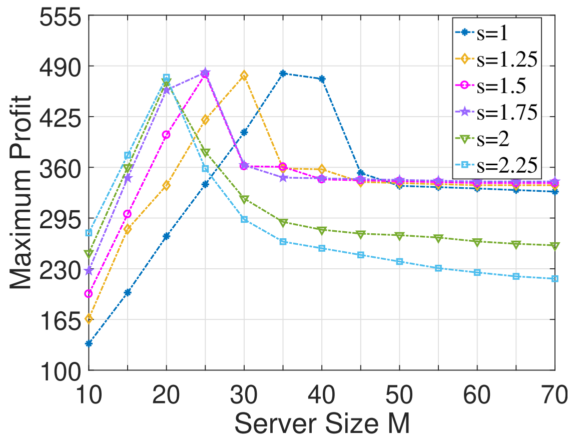
Fig. 4a shows that no matter which processing speed s is adopted by the multiserver system, the user demand for services first quickly increases to a peak, then gradually decreases, and finally keeps constant, as the multiserver size M grows from 10 to 70. The reason why the user demand quickly increases first is that adding more servers can attract more users by providing a higher computing performance. However, when the user demand becomes high, the multiserver system may not be able to timely process these service requests. In this case, users have to wait and their CPVs become low, which in turn decreases the demand of users for service. Note that the user demand would eventually remain a constant. This is because there is always a certain service demand in the market that is stable and not affected by external factors.

Algorithm 3. The Discrete SA-Based Heuristic to Maximize the Profit

```
Input: $D_{\mathrm{CPV}}(\mathrm{SQS})$, $\overline{r}$, $[M_{\min}, M_{\max}]$, $[s_{\min}, s_{\max}]$;
Output: optimal server size $M_{\mathrm{opt}}$, optimal server speed $s_{\mathrm{opt}}$,
and maximal profit $Profit_{\mathrm{opt}}$;
1:  discretize $[M_{\min}, M_{\max}]$ and $[s_{\min}, s_{\max}]$;
2:  select node $(M_{\max}, s_{\max})$ as start node $(M, s)$;
3:  $M_{\mathrm{cur}} = M_{\max}$, $s_{\mathrm{cur}} = s_{\max}$;
      // the initial point;
4:  calculate $Profit_{\mathrm{cur}} = G(M_{\mathrm{cur}}, s_{\mathrm{cur}})$ using Eq. (4.1);
5:  initialize $M_{\mathrm{opt}} = 0$, $s_{\mathrm{opt}} = 0$, $Profit_{\mathrm{opt}} = 0$;
6:  set $temperature = 2000$;
      // the initial temperature;
7:  set $k = Boltzmann\ constant$;
8:  set $CR = 0.99$; // the cooling rate;
9:  set $iterator_{\mathrm{num}} = 10000$, $step_M = 1$, $step_s = 0.1$;
10: generate a random constant from the range of (0,1] using
      $\varpi_{\mathrm{thresh}} = \mathrm{Rand}(0, 1)$;
11: while true do
12:   for $i = 0; i < iterator_{\mathrm{num}}; i++$ do
13:     while true do
14:       derive two variables by $\varpi_M = \mathrm{Rand(int)}$ and
          $\varpi_s = \mathrm{Rand(double)}$;
15:       $M_{\mathrm{next}} = M_{\mathrm{cur}} + step_M \times M_{\mathrm{cur}} \times \varpi_M$;
16:       $s_{\mathrm{next}} = s_{\mathrm{cur}} + step_s \times s_{\mathrm{cur}} \times \varpi_s$;
17:       if($M_{\mathrm{next}} \in [M_{\min}, M_{\max}]$&& $s_{\mathrm{next}} \in [s_{\min}, s_{\max}]$) then
18:         Break;
19:     derive $Profit_{\mathrm{next}} = G(M_{\mathrm{next}}, s_{\mathrm{next}})$ by Eq. (4.1);
20:     $\Delta Profit = Profit_{\mathrm{next}} - Profit_{\mathrm{cur}}$;
21:     if $\Delta Profit > 0$ then
22:       $M_{\mathrm{opt}} = M_{\mathrm{next}}, s_{\mathrm{opt}} = s_{\mathrm{next}}$;
23:       $Profit_{\mathrm{opt}} = Profit_{\mathrm{next}}$;
          //keep the new solution;
24:       $M_{\mathrm{cur}} = M_{\mathrm{next}}, s_{\mathrm{cur}} = s_{\mathrm{next}}$;
          //the new point;
25:       $Profit_{\mathrm{cur}} = Profit_{\mathrm{next}}$;
26:     else if $\exp(\frac{\Delta Profit}{temperature \times k}) > \varpi_{\mathrm{thresh}}$ then
27:       $M_{\mathrm{cur}} = M_{\mathrm{next}}, s_{\mathrm{cur}} = s_{\mathrm{next}}$;
28:       $Profit_{\mathrm{cur}} = Profit_{\mathrm{next}}$;
          //accept new point with a probability;
29:       $M_{\mathrm{opt}} = M_{\mathrm{next}}, s_{\mathrm{opt}} = s_{\mathrm{next}}$;
30:       $Profit_{\mathrm{opt}} = Profit_{\mathrm{next}}$;
31:       else
32:         $M_{\mathrm{opt}} = M_{\mathrm{cur}}, s_{\mathrm{opt}} = s_{\mathrm{cur}}$;
33:         $Profit_{\mathrm{opt}} = Profit_{\mathrm{cur}}$;
        //keep the current solution;
34:   $temperature = temperature \times CR$;
35:   if $|Profit_{\mathrm{next}} - Profit_{\mathrm{cur}}| < \varepsilon$ then
36:     Break;
```

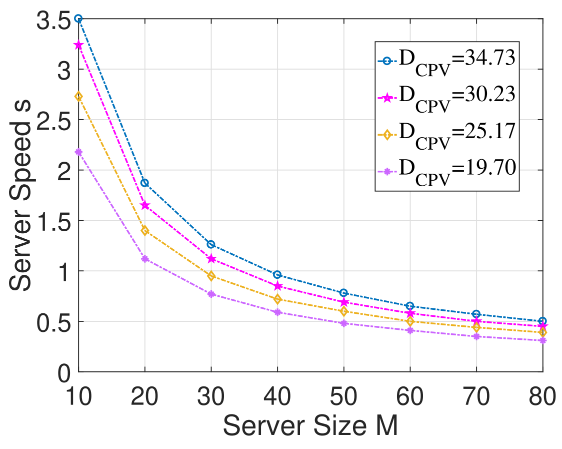


(a) User demand in different multiserver configurations.

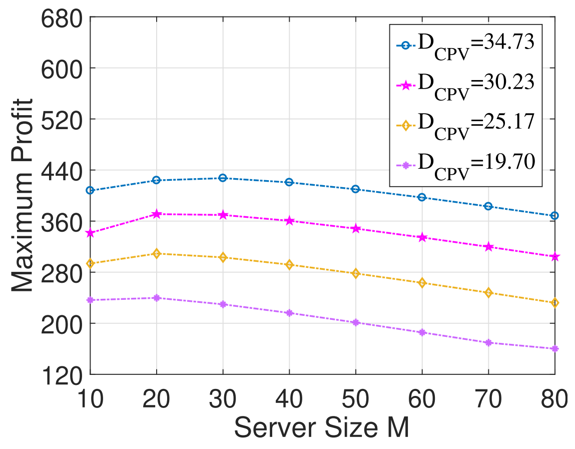


(b) Profit in different multiserver configurations.

Fig. 4. User demand and profit.



(a) Optimal server speed vs. multiserver size.



(b) Optimal profit vs. multiserver size.

Fig. 5. Optimal speed and profit under varying multiserver sizes and user demands.

Similarly, we can observe from Fig. 4b that no matter which processing speed s is adopted by the multiserver system, the cloud service provider's profit also first rapidly increases to a peak, then gradually decreases, and finally keeps stable, as the multiserver size M grows from 10 to 70. The initial increase of profit is due to the extra revenue gained by the increased demand, which is much higher than the additional cost of using the newly added severs. However, with the continuous increase of server size, the extra revenue may not be able to afford the additional cost and the user demand will decrease, leading to a reduced profit. Finally, the profit tends to be stable when the user demand becomes a constant.

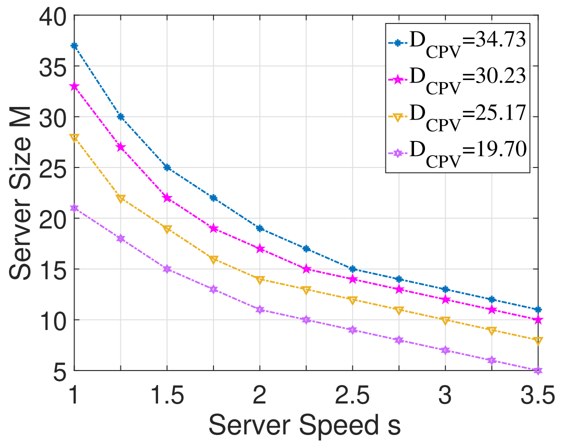
From Figs. 4a and 4b, we can also find that when the multiserver size is small (i.e., $10 \leq M \leq 20$), increasing service processing speed is effective in attracting more user demands and hence obtaining a higher profit. But when the multiserver size becomes larger (i.e., $M > 20$), increasing service processing speed cannot ensure more user demands and larger profits. This is because that high server speed and large server size both lead to a high cost of operating the multiserver system.

5.2 Optimal Multiserver Configuration and Profit

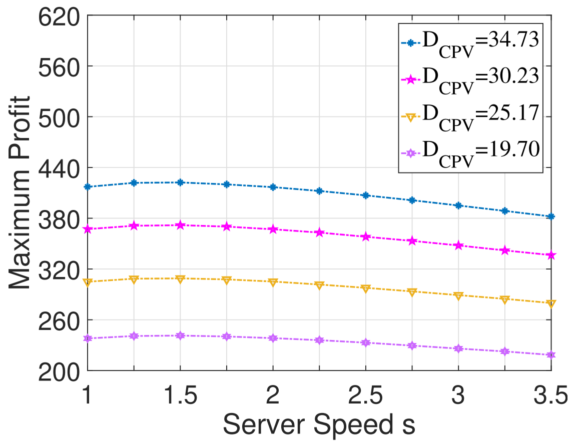
In this set of experiments, we carry out a series of numerical computations in order to first find the optimal server speed and the corresponding maximum profit for varying user demands and multiserver sizes, then find the optimal multiserver size and the corresponding maximum profit for varying user demands and server speeds, and finally find the optimal multiserver size and server speed under the given user demands.

Fig. 5 shows the optimal server speed and the corresponding maximum profit when varying the multiserver size and user service demand. The server size takes the value from $\{10, 20, 30, 40, 50, 60, 70, 80\}$[2] and the user service demand takes the value from $\{34.73, 30.23, 25.17, 19.7\}$. The values of parameters s_0, $\overline{r}$, a, c, ζ, α, P_{sta}, δ, C^T are consistent with those in Table 1. As can be seen from Fig. 5a, for a given service demand, the optimal server speed decreases with the increase in the size of the multiserver system. This is because when the required computing capability for completing a given amount of service requests is fixed, increasing the

2. This setup could cover most of parameters of the aforementioned up-to-date multiprocessors. However, it doesnot include the case of NVIDIA-V100 (i.e., 5,120 cores) since such a large server size is not common in real-world cloud service providers. For example, the server size used by Amazon EC2 [2], Microsoft Azure [3], Aliyun [27], and Huawei Cloud [42] are in the rang of [1, 96], [1, 64], [1, 104], and [1, 64], respectively.



(a) Optimal multiserver size vs. server speed.



(b) Optimal profit vs. server speed.

Fig. 6. Optimal multiserver size and profit under varying server speeds and user demands.

TABLE 2
The Maximum Profit and the Corresponding Multiserver Configuration

Demand (D_{CPV})	9	11	13	15	17	19	21	23	25	27
Optimal size	12	13	14	15	16	17	18	19	20	21
Optimal speed	1	1.1	1.2	1.3	1.35	1.38	1.43	1.44	1.45	1.46
Maximum profit	106.20	135.60	156.56	181.17	206.22	231.45	255.97	281.56	306.92	332.25

multiserver size necessitates the decreased server speed. Fig. 5b presents the corresponding maximum profit when increasing the multiserver size and user service demand. Clearly, larger user demands can bring higher profit for cloud service providers. From the figure we can also observe that the profit is affected by the multiserver size under a certain user demand. Specifically, when the multiserver size is small and the server speed is high, renting more servers is able to increase the profit. When the multiserver size becomes greater further and exceeds a threshold, the profit decreases. This is because that to maintain a steady computing capability for the given user demand, the server speed necessarily decreases with the increase of multiserver size. In the case of small multiserver size and high server speed, the saved energy cost derived by lowering the server speed is greater than the extra cost of renting more servers. However, when the multiserver size becomes large and the server speed drops to a certain level, the increased renting cost surpasses the saved energy cost, leading to a decreased profit.

Fig. 6 presents the optimal multiserver size and the corresponding maximum profit when varying the server speed and user service demand. The server speed takes the value from $\{1, 1.25, 1.5, 1.75, 2, 2.25, 2.5, 2.75, 3.0, 3.25, 3.5\}$ and the user service demand takes the value from $\{19.7, 25.17, 30.23, 34.73\}$. To derive the results, we use the parameters as same in Table 1. From Figs. 6a and 6b, we can observe that with the increase of server speed, i) the optimal multiserver size becomes smaller, which is due to that increasing the server speed necessitates the smaller multiserver size under a certain user demand, and ii) the profit gradually decreases, which is because that the high energy cost of using fast server speed leads to the decrease of profit.

Table 2 presents the optimal multiserver size, optimal server speed, and maximum profit under varying user demands. The user demand varies from 9 to 27 in the step of 2. As can be seen from the table, the optimal multiserver size M is monotonically increasing with the increase of user demands D_{CPV}. The reason is that higher user demand needs a greater computing capacity.

5.3 Performance Comparison

In this set of experiments, we compare our CPV-based and risk-aware profit maximization scheme with two state of the art benchmarking methods COMCPM [4] and OMCPM [5] under the same experimental settings. The two benchmarking methods both consider the service-level agreement as well as customer satisfaction, and derive an optimal multiserver configuration and service price for maximizing profit. Compared with COMCPM and OMCPM, our scheme is based on the CPV that reflects users willingness to purchase cloud services and takes into account the risk involving in the pricing contract. In the experiments, we vary the user demands and find the optimal multiserver configuration (M and s) and the corresponding profit for each user demand. We use the same values of parameters such as s_0, $\overline{r}$, a, c, ζ, α, P_{sta}, δ, C^T as in Table 1. The two benchmarking methods COMCPM and OMCPM are briefly described below.

- COMCOM [4] is a profit maximization scheme that specially considers the customer satisfaction in cloud. The scheme operates as follows. It first defines the concept of customer satisfaction leveraged from economics and develops a formula to measure the customer satisfaction in cloud. Based on the affection of customer satisfaction on server's workload, it then analyzes the interaction between the market demand and the customer satisfaction as well as calculates the service arrival rate. Finally, it solves the profit

TABLE 3
Profit Comparison Between COMCPM and the Proposed Scheme

COMCPM				Proposed				Improvement
Demand	optM	optS	Profit	Demand	optM	optS	Profit	$\frac{\text{Proposed}-\text{COMCPM}}{\text{COMCPM}}$
4.85	6	1.13	42.33	4.97	6	1.1	57.10	34.9%
9.79	12	1.04	88.68	9.96	10	1.19	118.18	33.3%
14.72	17	1.05	135.57	14.97	15	1.15	178.55	31.7%
19.68	22	1.06	182.71	19.96	20	1.12	239.46	31.1%
24.63	27	1.06	230.06	24.95	25	1.11	300.25	30.5%
29.59	32	1.06	277.55	29.97	30	1.09	361.21	30.1%
34.55	37	1.06	325.14	34.97	35	1.09	422.28	29.9%

TABLE 4
Profit Comparison Between OMCPM and the Proposed Scheme

OMCPM				Proposed				Improvement
Demand	optM	optS	Profit	Demand	optM	optS	Profit	$\frac{\text{Proposed}-\text{OMCPM}}{\text{OMCPM}}$
4.78	6.05	1.05	42.04	4.94	6	1.08	56.18	33.7%
9.63	11.67	1.01	87.65	9.95	10	1.17	116.10	32.5%
14.55	17.23	0.99	134.75	14.93	15	1.14	176.37	30.9%
19.44	22.76	0.98	181.44	19.95	20	1.11	236.29	30.2%
24.43	28.27	0.98	229.49	24.92	25	1.10	297.71	29.7%
29.30	33.78	0.97	276.38	29.95	30	1.08	357.50	29.4%
34.31	39.27	0.97	324.65	34.95	35	1.08	419.07	29.1%

maximization problem for cloud service providers by finding out the optimal configuration of servers. In COMCOM, the optimal configuration of servers is obtained using a discrete hill climbing algorithm.

- OMCPM [5] is a profit maximization scheme that treats a multiserver system as an M/M/M queuing model as well as considers two server speed and power consumption models. The scheme works as follows. It first derives the probability density function (PDF) of the waiting time of a newly arrived service request and calculates the expected service charge to a service request. Based on the PDF and expected service charge of service requests, it then gets the expected net business gain (i.e., profit) in one unit of time and obtain the optimal configuration of server size and server speed numerically.

Table 3 compares the profit achieved by the proposed scheme with that of COMCPM [4]. It is clear that the proposed scheme outperforms COMCPM in terms of increasing the cloud service provider's profit. The improvement of profit achieved by the proposed scheme over COMCPM is 31.6 percent on average, and can be up to 34.9 percent. We can also derive a similar conclusion from Table 4 that the profit of the proposed scheme is greater than that of OMCPM [5]. Specifically, the profit improvement achieved by the proposed scheme over OMCPM is 30.8 percent on average, and can be up to 33.7 percent. The higher profit achieved by the proposed scheme benefits from not only the optimal configuration of servers (i.e., the server size M and the server speed s) but also the more user service demands captured by the scheme. In addition, the proposed scheme also considers the risk involving in the pricing contract to avoid the loss caused by the risk. This is helpful for the cloud service provider to gain a higher profit.

6 Conclusion

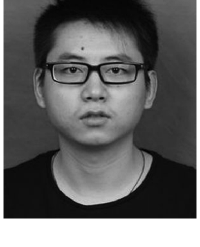
In this paper, we consider customer perceptive value and risk in solving the optimal multiserver configuration problem for maximizing the cloud service provider's profit. Specifically, we propose a method to model the customer perceptive value and hence estimate the user demand for cloud services. Based on the estimated user demand, we study the optimal multiserver configuration problem for profit maximization and show how to derive the analytical optimal solution. Furthermore, we develop a simulated annealing based algorithm to find the numerical optimal solution when considering the risk in the pricing contract. To validate the effectiveness of the proposed scheme, we analyze the changing trend of profit derived by the proposed scheme under different configurations and compare the proposed scheme with two benchmarking approaches. The results reveal that the proposed scheme not only follows the supply and demand law of the cloud service market but also increases the profit by up to 34.9 percent when compared to the two benchmarking approaches.

Acknowledgments

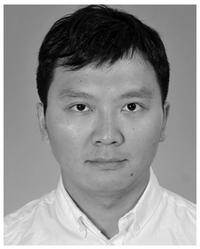
The authors would like to thank the editors and the anonymous reviewers for their constructive comments and valuable suggestions. In addition, the authors would like to express their sincere gratitude to their collaborator, professor Keqin Li in the State University of New York at New Paltz, for his helpful advice on this work. This work was supported in part by the National Natural Science Foundation of China (Grant Nos. 61802185 and 61272420), in part by Natural Science Foundation of Jiangsu Province (Grant No. BK20180470), in part by the Fundamental Research Funds for the Central Universities (Grant No. 30919011233), in part by Open Research Project of The Hubei Key Laboratory of Intelligent Geo-Information Processing (Grant No. KLIGIP-2018A04), and in part by Shanghai Municipal Natural Science Foundation under Grant 16ZR1409000.

REFERENCES

[1] P. Cong *et al.*, “Developing user perceived value based pricing models for cloud markets,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 12, pp. 2742–2756, Dec. 2018.
[2] Amazon EC2, 2019. [Online]. Available: https://aws.amazon.com/cn/ec2/pricing/on-demand
[3] Microsoft Azure, 2019. [Online]. Available: https://azure.microsoft.com/zh-cn/pricing/details/virtual-machine-scale-sets
[4] M. Jing, K. Li, and K. Li, “Customer-satisfaction-aware optimal multiserver configuration for profit maximization in cloud computing,” *IEEE Trans. Sustain. Comput.*, vol. 2, no. 1, pp. 17–29, Jan.–Mar. 2017.
[5] J. Cao, K. Huang, K. Li, and A. Y. Zomaya, “Optimal multiserver configuration for profit maximization in cloud computing,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1087–1096, Jun. 2013.
[6] M. Jing, K. Li, A. Ouyang, and K. Li, “A profit maximization scheme with guaranteed quality of service in cloud computing,” *IEEE Trans. Comput.*, vol. 64, no. 11, pp. 3064–3078, Nov. 2015.
[7] Y. Lee, C. Wang, A. Y. Zomaya, and B. Zhou, “Profit-driven scheduling for cloud services with data access awareness,” *J. Parallel Distrib. Comput.*, vol. 72, no. 4, pp. 591–602, 2012.
[8] H. Chun, “Optimal pricing and ordering policies for perishable commodities,” *Eur. J. Oper. Res.*, vol. 144, no. 1, pp. 68–82, Jan. 2003.
[9] Amazon Web Services, 2019. [Online]. Available: http://aws.amazon.com/
[10] Google App Engine, 2019. [Online]. Available: https://appengine.google.com/
[11] M. Ghamkhari and H. Mohsenian-Rad, “Energy and performance management of green data centers: A profit maximization approach,” *IEEE Trans. Smart Grid*, vol. 4, no. 2, pp. 1017–1025, Jun. 2013.
[12] Amazon EC2 spot instances, 2019. [Online]. Available: https://aws.amazon.com/cn/ec2/spot/pricing
[13] H. Xu and B. Li, “Dynamic cloud pricing for revenue maximization,” *IEEE Trans. Cloud Comput.*, vol. 1, no. 2, pp. 158–171, Jul.–Dec. 2013.
[14] M. Macias and J. Guitart, “A genetic model for pricing in cloud computing markets,” *ACM Symp. Appl. Comput.*, pp. 113–118, 2011.
[15] S. Ren and M. V. D. Schaar, “Dynamic scheduling and pricing in wireless cloud computing,” *IEEE Trans. Mobile Comput.*, vol. 13, no. 10, pp. 2283–2292, Oct. 2014.
[16] S. Chatterjee, R. Ladia, and S. Misra, “Dynamic optimal pricing for heterogeneous service-oriented architecture of sensor-cloud infrastructure,” *IEEE Trans. Services Comput.*, vol. 10, no. 2, pp. 203–216, Mar./Apr. 2017.
[17] V. A. Zeithaml, “Consumer perceptions of price, quality, and value: A means-end model and synthesis of evidence,” *J. Marketing*, vol. 52, no. 3, pp. 2–22, 1988.
[18] Z. Yang and R. T. Peterson, “Customer perceived value, satisfaction, and loyalty: The role of switching costs,” *Psychol. Marketing*, vol. 21, no. 10, pp. 799–822, 2004.
[19] H. G. M. Gordon and L. Terrence, “Customer satisfaction with services: Putting perceived value into the equation,” *J. Services Marketing*, vol. 14, no. 5, pp. 392–410, 2000.
[20] P. Cong *et al.*, “User perceived value-aware cloud pricing for profit maximization of multiserver systems,” in *Proc. IEEE Int. Conf. Parallel Distrib. Syst.*, 2017, pp. 537–544.
[21] Google Compute Engine, 2019. [Online]. Available: https://cloud.google.com/products/compute-engine
[22] Amazon CloudFormation, 2019. [Online]. Available: http://aws.amazon.com/cloudformation
[23] K. Li, “Optimal load distribution for multiple heterogeneous blade servers in a cloud computing environment,” *J. Grid Comput.*, vol. 11, no. 1, pp. 27–46, 2013.
[24] B. N. Chun and D. E. Culler, “User-centric performance analysis of market-based cluster batch schedulers,” in *Proc. IEEE/ACM Int. Symp. Cluster Comput. Grid*, 2002, pp. 30–30.
[25] K. Li, “Optimal configuration of a multicore server processor for managing the power and performance tradeoff,” *J. Supercomput.*, vol. 61, no. 1, pp. 189–214, 2012.
[26] K. Li, J. Mei, and K. Li, “A fund-constrained investment scheme for profit maximization in cloud computing,” *IEEE Trans. Services Comput.*, vol. 11, no. 6, pp. 893–907, Nov./Dec. 2018.
[27] Aliyun, 2019. [Online]. Available: https://www.aliyun.com/price/product?spm=5176.8789780.1092586.1.1da155caFpHojQ&aly_as=6NLKdXpV#/ecs/detail
[28] Example of queuing models, 2018. [Online]. Available: http://www.universalteacherpublications.com/univ/ebooks/or/Ch10/mm1ex.htm
[29] L. Kleinrock, *Queueing Systems: Computer Applications*, vol. 2. New York, NY, USA: Wiley, 1976.
[30] R. C. Lewis and B. H. Booms, “The marketing aspects of service quality,” *Emerg. Perspectives Services Marketing*, vol. 65, no. 4, pp. 99–107, 1983.
[31] M. Unuvar, S. Tosi, Y. Doganata, M. Steinder, and A. Tantawi, “Selecting optimum cloud availability zones by learning user satisfaction levels,” *IEEE Trans. Service Comput.*, vol. 8, no. 2, pp. 199–211, Mar./Apr. 2015.
[32] A. Parasuraman, V. A. Zeithaml, and L. L. Berry, “SERVQUAL: A multiple-item scale for measuring consumer perc,” *J. Retailing*, vol. 64, no. 1, 1988, Art. no. 12.
[33] L. S. V. Velsen, M. F. Steehouder, and T. D. Jong, “Evaluation of user support: Factors that affect user satisfaction with helpdesks and helplines,” *IEEE Prof. Commun.*, vol. 50, no. 3, pp. 219–231, Sep. 2007.
[34] Normalization, 2019. [Online]. Available:https://en.wikipedia.org/wiki/Normalization_(statistics)
[35] Y. Wang, A. Meliou, and G. Miklau, “A consumer-centric market for database computation in the cloud,” 2016. [Online]. Available: https://arxiv.org/abs/1609.02104
[36] S. Lyden and M. E. Haque, “A simulated annealing global maximum power point tracking approach for PV modules under partial shading conditions,” *IEEE Trans. Power Electron.*, vol, 31, no. 6, pp. 4171–4181, Jun. 2016.
[37] Intel Xeon Skylake, 2017. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/series/125191/intel-xeon-scalable-processors.html
[38] Intel KNL, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Xeon_Phi
[39] NVIDIA V100, 2019. [Online]. Available: https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units
[40] IBM Power7, 2019. [Online]. Available: https://en.wikipedia.org/wiki/POWER7
[41] Y. Hu, J. Wong, G. Iszlai, and M. Litoiu, “Resource provisioning for cloud computing,” in *Proc. Conf. Center Adv. Stud. Collabortive Res.*, 2009, pp. 101–111.
[42] Huawei Cloud, 2019. [Online]. Available: https://www.huaweicloud.com/product/ecs.html



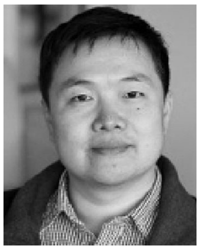
Tian Wang (S'19) is currently working toward the PhD degree in the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. His current research interests include the areas of cloud computing and cyber security. He is a student member of the IEEE.



Junlong Zhou (S'15–M'17) received the PhD degree in computer science from East China Normal University, Shanghai, China, in 2017. He was a visiting scholar with the University of Notre Dame, Notre Dame, Indiana, from 2014 to 2015. He is currently an assistant professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. His research interests include real-time embedded systems, cloud computing and IoT, and cyber physical systems. He has published 60 refereed papers, most of which are published in premium conferences and journals. He received the Reviewer Award from the *Journal of Circuits, Systems, and Computers*, in 2016. He has served as publication chairs, publicity chairs, section chairs, and TPC member for numerous conferences. He has been an associate editor of the *Journal of Circuits, Systems, and Computers*, and serves as a guest editor for several special issues of the *ACM Transactions on Cyber-Physical Systems*, *IET Cyber-Physical Systems: Theory & Applications*, and *Elsevier Journal of Systems Architecture: Embedded Software Design*. He is a member of the IEEE.

Gongxuan Zhang (SM'12) received the BS degree in electronic computer from Tianjin University, in 1983, and the MS and PhD degrees in computer application from the Nanjing University of Science and Technology, in 1991 and 2005, respectively. He was a senior visiting scholar with the Royal Melbourne Institute of Technology from 2001 to 2002 and with the University of Notre Dame from 2017 to 2017. Since 1991, he has been with the Nanjing University of Science and Technology, where he is currently a professor with the School of Computer Science and Engineering. His current research interests include multicore and parallel processing and distributed computing. He is a senior member of the IEEE.

Tongquan Wei (M'11–SM'19) received the PhD degree in electrical engineering from Michigan Technological University, Houghton, Michigan, in 2009. He is currently an associate professor with the Department of Computer Science and Technology, East China Normal University, Shanghai, China. His current research interests include real-time embedded systems, green and reliable computing, parallel and distributed systems, and cloud computing. He has published more than 70 papers in these areas, most of which are published in premium conferences and journals including IEEE/ACM DAC, IEEE/ACM ICCAD, IEEE/ACM DATE, the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *IEEE Transactions on Circuits and Systems*, *IEEE Transactions on Smart Grid*, *IEEE Transactions on Dependable and Secure Computing*, *IEEE Transactions on Aerospace and Electronic Systems*, *IEEE TR*, *IEEE Transactions on Sustainable Computing*, *IEEE Communications Surveys and Tutorials*, etc. He has been a regional editor of the *Journal of Circuits, Systems, and Computers* since 2012. He served as a guest editor for several special sections of the *IEEE Transactions on Industrial Informatics*, *ACM Transactions on Embedded Computing Systems*, and *ACM Transactions on Cyber-Physical Systems*. He is a senior member of the IEEE.



Shiyan Hu (SM'10) received the PhD degree in computer engineering from Texas A&M University, in 2008. He is currently a professor and chair in cyber-physical system security with the University of Southampton. His research interests include cyber-physical systems and cyber-physical system security, where he has published more than 100 refereed papers. He is an ACM distinguished speaker, an IEEE Systems Council distinguished lecturer, an IEEE Computer Society distinguished visitor, a recipient of the 2017 IEEE Computer Society TCSC Middle Career Researcher Award, 2014 U.S. National Science Foundation (NSF) CAREER Award, and 2009 ACM SIGDA Richard Newton DAC scholarship. His publications have received a few distinctions, which includes the 2018 IEEE Systems Journal Best Paper Award, 2017 Keynote Paper in *IEEE Transactions on Computer-Aided Design*, and the Front Cover in IEEE Transactions on Nanobioscience in March 2014. He is the chair for IEEE Technical Committee on Cyber-Physical Systems. He is the editor-in-chief of the *IET Cyber-Physical Systems: Theory & Applications*. He serves as an associate editor of the *IEEE Transactions on Computer-Aided Design*, *IEEE Transactions on Industrial Informatics*, *IEEE Transactions on Circuits and Systems*, *ACM Transactions on Design Automation for Electronic Systems*, and *ACM Transactions on Cyber-Physical Systems*. He has served as a guest editor for eight IEEE/ACM Journals such as the *Proceedings of the IEEE* and *IEEE Transactions on Computers*. He has held chair positions in many IEEE/ACM conferences. He is a fellow of the IET and British Computer Society, and a senior member of the IEEE.