On the Benefits of QoS-Differentiated Posted Pricing in Cloud Computing: An Analytical Model

Xiaohu Wu, Francesco De Pellegrini

Abstract—When designing service and pricing models, an important goal is to both enable user-friendly and cost-effective access to services and achieve a high system’s utilization. Such design could attract more users and improve the revenue. QoS-differentiated pricing represents an important direction to address the above design goal as illustrated by numerous studies in other fields such as Internet. In this paper, we propose the first analytical model of QoS-differentiated posted pricing in the context of cloud computing. In this model, a cloud provider offers a stable set of SLAs and their prices are also posted to users in advance, where higher QoSs are charged higher prices. As a result, users can directly choose their preferred SLA with a certain price, in contrast to many existing dynamic pricing mechanisms with the uncertain prices and availability of cloud resource. In order to maximizing the revenue of a cloud provider, the questions that arise here include: (i) setting the prices properly to direct users to the right SLAs best fitting their need, and, (ii) given a fixed cloud capacity, determining how many servers should be assigned to each SLA and which users and how many of their jobs are admitted to be served. To this respect, we propose optimal schemes to jointly determine SLA-based prices and perform capacity planning. In spite of the usability of such a pricing model, simulations also show that it could improve the utilization of computing resource by up to eleven-fold increase, compared with the standard pay-as-you-go pricing model; furthermore, the revenue of a cloud provider could be improved by up to a five-fold increase.

Index Terms—Cloud computing; Service Design; QoS-Differentiation; Posted Prices

1 INTRODUCTION

2 INTRODUCTION

The worldwide cloud Infrastructure-as-a-Service (IaaS) market is projected to grow to $34.6 billion in 2017 from $25.3 billion in 2016 and projections are made for turnaround of $71.6 billion in 2020, according to Gartner, Inc. [1]. IaaS is used as a means of delivering value to users by facilitating user’s access to computing capacities. The celebrated pay-as-you-go pricing eliminates setup and maintenance costs and attracts customers from individuals and organizations of all sizes, without the need for cloud users to own and maintain the infrastructure. Yet as more cloud providers offer these services, the IaaS market has become increasingly competitive, with cloud providers competing to offer ever-lower prices to users.

The design of a service is the activity of planning and organizing people, infrastructure, communication and material components of a service in order to improve its quality, the system’s efficiency, and the interaction between the service provider and its users [2]. This is achieved accounting for both the needs of customers and the capabilities of service providers.

Interaction. In the context of IaaS, the pricing models adopted by a cloud provider define to a large extent the ways how the computing service is delivered to the users and how they interact with the cloud provider to obtain them. For example, in Amazon EC2, spot instances are offered to users with their price varying over time and are said to be a form of auctions. Users interacts with Amazon EC2 as follows: they bid a price and are able to utilize spot instances until the spot price exceeds their bid price; users are charged the spot prices of the period in which spot instances are consumed [3]. In contrast, in Google cloud platform, the price of computing resources is fixed and posted to users in advance: stable access to computing service is guaranteed at the posted price [4].

Indeed, general auctions are widely applicable, but can be significantly complex for cloud providers to execute and for users to participate in [5]. So far, substantial works have been done that use complex techniques such as stochastic programming to cost-optimally utilize cloud instances (e.g., see [6]). So, despite a spot market is beneficial from a revenue perspective, the operational complexity and lack of price predictability it creates at the users’ end suggest it is preferable not to run one [7]. In contrast, posted pricing exhibits a desirable property in payment [5], [8]: a buyer simply behaves as a price-taker and consumes his preferred service, being able to immediately understand the price-setting method and how to participate in cost-effectively.

System Efficiency. On the cloud provider’s side, while processing jobs, the performance and utilization of a computing system are heavily constrained by the characteristics of jobs being served (e.g., their sensitivity to latency). An appropriate design of pricing could serve as a tool to incentivize users to express their job’s characteristics in a way to promote the utilization of cloud resources. In the current pricing design of Google, users may prefer to immediately process their jobs upon arrival and are not willing to tolerate any latency. This happens even if their jobs are actually delay-tolerant. In fact, the model lacks of incentives for elasticity, e.g. discounts for tolerance on task completion delay. Indeed, both theoretical and experimental results

X. Wu and F. De Pellegrini are with Fondazione Bruno Kessler, Trento, Italy. E-mail: {xiaohu.wu, fdepellegrini}@fbk.eu.
show that latency-sensitive jobs induces a low resource utilization and it is estimated that industry-wide utilization of cloud resources lies somewhere between 6% and 12% [9]. Conversely, delay-tolerant jobs tend to lead to a much higher utilization [10], [11], [12], [13], [14], [15].

**Goals in Pricing Design.** To sum up, an ideal pricing design should achieve the following three goals, where users and cloud providers can be win-win:

i) **User-friendly access.** Users can easily perceive the cost-optimal strategy to utilize cloud resources to satisfy their own QoS requirements, with minimal operational burden to implement it.

ii) **High utilization of servers and analytical revenue management.** The provider prices and offers cloud resources to users in a way such that (i) they could express and submit their jobs preferences to the cloud system with incentives towards elasticity and so high utilization of cloud resources, and, (ii) an analytical resource management and pricing scheme able to maximize revenues without violating the jobs’ QoS requirements.

iii) **Cost-effective provisioning to users.** Users want cheap cloud computing services. By increasing the utilization of cloud resources, the unit cost of computing resources is reduced, enabling cloud providers to provide cheaper computing services without any loss of revenue.

**Results.** The main contribution of this paper is to enable an analytical model of QoS-differentiated pricing in cloud computing. The aim is to achieve the above three goals in pricing design to a reasonable extent. In particular, our contributions are as follows.

1) **Model.** We propose an analytical QoS-differentiated computing service model. In this model, a cloud provider posts to users a menu of L SLAs. The l-th SLA consists of a response time $\varphi_l$ and its price $\theta_l$ where the larger the response time the lower the price. If a user $i$ chooses the l-th SLA, the cloud provider guarantees that, upon arrival of each of its jobs $j$, the job has a maximum waiting time $\varphi_l$ and then begins being processed until the completion; here, the price of processing a unit of workload of $j$ under SLA $l$ is $\theta_l$.

2) **Capacity Planning and Optimal Prices.** Given the capacity of a cloud provider and the prices of different SLAs, we propose an optimal capacity planning scheme to achieve the revenue maximization objective. The scheme performs an optimal admission control for the rate at which jobs of each QoS level are accepted. It also determines the minimum number of servers needed to guarantee each QoS requirement. Because the prices of SLAs determine the job’s arrival rate per SLA class, an optimal pricing scheme of different SLAs is designed for revenue maximization.

3) **Performance Evaluation.** In terms of the revenue of a cloud provider, the proposed model is compared with an on-demand model where users are charged a fixed price for a unit of computing resources. Simulations show that the revenue of a cloud provider could be improved by up to a five-fold increase.

In addition, by simulations, we also illustrate how the performance of a QoS-differentiated posted pricing model is affected by the number of servers available, the size of the users population, the weights of users, their sensitivity to latency, etc.

Finally, we believe an important strength of the proposed model lies the simplicity on the users’ side: users could understand the above pricing model easily and simply select the SLAs that best satisfy their need. Actually, the task of guaranteeing job’s QoS requirement is taken care of by cloud providers. In addition, diverse users are directed to different SLAs. Due to the advantage brought about when processing delay-tolerant jobs, a higher system utilization is achieved and they will be offered a lower price. Here, simulations show that the utilization of cloud resources could be improved by up to an eleven-fold increase. So, this model is user-friendly and could be more cost-effective for users. A queuing theoretical model is leveraged to obtain an analytical framework. The formulation is made as generic as possible so as to apply to any type of job’s runtime distribution. The model is specialized for the case of Pareto or exponential distributions, which are customary in cloud computing literature [10], [14].

In the rest of this work, we first provide an overview of existing pricing models in Section 3. Then, we introduce some design choices in QoS-based pricing in Section 4 and propose the QoS-differentiated pricing model of this paper in Section 5. In Section 6, we propose the optimal capacity planning and pricing algorithms for maximizing the revenue of a cloud provider. Furthermore, the whole pricing system is evaluated in Section 7. Finally, we conclude this paper in Section 8.

### 3 Overview of the Existing Pricing Models

In this section, we introduce some typical pricing models in cloud computing and show how computing service is priced and offered to users and what relates to the system efficiency. This also gives us further motivation to consider QoS-differentiated pricing in cloud computing.

**Mechanism Design.** Mechanism design methods, e.g., auction mechanisms for cloud computing, have attracted substantial attention so far and are studied from the simple one-round towards more complex online cases [18], [19], [20], [21], [22] and from the non-committed towards committed cases [23], [24], [25]. They work as follows. An user truthfully reports its job’s value and other characteristics to the cloud provider; then the provider decides what the cloud resource price is and only the jobs whose value is no less than this price are accepted. Under such schemes, users cannot be certain whether or not they can succeed in obtaining the requested computing resource and how much the payment will be. Like [5], [27], the uncertainty and dynamics that users have to face in this type of schemes are motivating researchers to consider posted pricing.

Furthermore, in many of these schemes, the worst-case analysis is used, and, the performance of a cloud system that implements these schemes will heavily depend on the worst value of some job’s characteristics. For instances, the slackness, i.e., the ratio of the job’s deadline minus arrival time to its minimum execution time, is considered in [23].

1. Although most of these mechanisms consider the objective of maximizing social welfare, it could be related to the revenue maximization objective of this paper through some existing techniques [26].
Also, the ratio of the upper bound to the lower bound of the per-unit-resource job valuation of all users, is considered in [18], [23].

Such choices may lead to the system’s inefficiency and does not well fit the second and third goals we wish to fulfill in our pricing design. The reason for this is explained as follows

In practice, both latency-critical and delay-tolerant jobs coexist in the cloud. Thus, better overall performance could be achieved by partitioning the whole cloud system into isolated subsystems, each of them providing different QoS guarantees. In our scheme, we assume that jobs with different QoS requirements/jobs characteristics (e.g., slackness) are separately submitted to different subsystems for processing.

For example, when the schemes in [23], [24], [25] are adopted, the subsystems that process the jobs with larger slackness will have better performances, further improving overall system performance. However, available analytical frameworks are based on worst-case analysis, making it difficult to perform optimal capacity planning. It hence becomes difficult to determine how many servers are allocated to each type of jobs to satisfy their QoS requirement. This will be further explained in Section 4.

**Spot Pricing.** Another line of works aims at evaluating the effectiveness of a hybrid market in Amazon EC2 via queuing theory [28]. There, spot and on-demand instances are offered to users. As stated in Section 2, spot instances are also a kind of auctions and on-demand instances are charged a fixed price per unit of time when utilizing a virtual machine.

In particular, Abhishek et al. model the hybrid market of Amazon as a queuing system that is further analyzed by auction theory [29]. The value of a job is an decreasing function of its completion time and, with the assumption that the cloud provider has access to an infinite pool of resources. The value of a job is an decreasing function of its completion time, they showed that the introduction of spot instances could not increase the revenue of a cloud provider. The reason for this is that, in a hybrid market, there is no way to prevent high-value low-waiting-time-cost jobs from choosing the spot market when they would have been willing to pay a higher on-demand price. More recently, L. Dierks, and S. Seuken consider a more real assumption where the pool of on-demand instances is finite and the provider incurs a cost for every instance and showed that a hybrid market can indeed lead to a higher revenue than a single on-demand market [30].

**Posted Pricing.** In contrast to Amazon EC, Google manages to remove the complexity in cloud pricing and advocates to use an opposite pricing philosophy in order to be customer-friendly [36]. It only provides per-minute billing at a fixed price [4]. Recently, Li et al. also laid some theoretical foundation for posted pricing in cloud computing [27]. In particular, they designed a type of pricing functions based on the resource utilization ratios and, when using the pricing functions to determine the price of cloud resources for every arriving job, they proved the competitive ratio, i.e., the worst-case ratio of the social welfare achieved by this pricing function to the social welfare of an optimal solution.

Here, the competitive ratio also depends on the worst value of some job’s characteristics, i.e., the slackness.

**QoS-Differentiated Pricing.** QoS-differentiated pricing is a special form of posted pricing. In fact, as a generic technique, QoS differentiation has been successfully applied to other fields such as modern IP networks for managing and classifying traffics, improving the network utilization, and providing QoS guarantee; the related pricing schemes have been studied very heavily [37], [38], [39], [40]. Here, services could be differentiated to provide low-latency to network traffics such as voice or streaming media while providing simple best-effort service to web traffic or file transfers.

Recently, the effectiveness of QoS-differentiated pricing has been validated in the field of cloud computing by a preliminary experimental work [16]: whenever jobs of different users are submitted, the cloud provider specifies two completion times for each job based its current load and compute the corresponding prices; the later a job is completed, the cheaper the price is. The experimental results show that such a pricing system can yields 40% improvement over the standard fixed pricing scheme. However, the pricing scheme in [16] is more cloud-centric and it is not designed to satisfy per user QoS requirements. In other words, if there are not enough servers available, the cloud provider has to announce large job completion times to all users, despite of the fact that they may be even latency-critical with the need to be completed immediately.

Finally, it appears preferable to offer users a pre-defined set of QoS requirements, including a wide range of completion times. Then, a user can choose a QoS requirement that fits best her needs, and the cloud provider, in turn, has to ensure that the user’s jobs are completed satisfying her QoS requirements.

### 4 Design Choice in QoS-Based Pricing

We recall that our objective is the pricing design goal introduced in Section 2. To this aim, before describing the model, we discuss available mathematical tools able to provide the intended analytical resource management framework. Also, we identify the job characteristics to be priced so as to achieve as high resources utilization as possible, aiming at revenue gains for cloud providers.

**Choice of Analysis Methods.** Many existing pricing or scheduling schemes show that the worst-case performance (e.g., utilization, social welfare) of a large scale parallel computing system depends on the worst value of a certain characteristic of all coming jobs [15], [23], [24], [25]. This motivates us to differentiate jobs by the value of this job’s characteristic. However, most such results are based on worst case analysis to derive performance guarantees. Typically, such guarantees are represented by the ratio of the performance of an algorithm to the performance of an offline optimal algorithm; in the offline algorithm, the characteristics of jobs incoming over time are assumed known in advance.

Such analysis is done where the optimal algorithm is impossible to obtain and the optimal performance is usually estimated by its upper bound. Here, We do not know the actual resource utilization that these algorithms achieve and they also don’t require the knowledge of job’s arrival rate
and runtime, that determines the utilization of a system. As a result, when jobs with different QoS requirements arrive, using the worst-case analysis, we cannot derive the utilizations of different levels of QoS requirement and further the minimum number of servers needed to process the arriving jobs. Although the pricing model of this paper is motivated by the progress in [15], [23], [24], [25], we will use the theoretical tool of queuing theory to propose an analytical resource management in cloud computing.

Choice of the Runtime Distribution. So far, in the cloud computing field, Pareto distribution is often used for characterizing the job’s runtime of a published Google’s workload [10], [14], while exponential distribution is applied [33], [41], [42], e.g., to characterize a Microsoft’s production workload [14], [43]. The focus of this paper is not on the theoretical development of various queuing models under different runtime assumptions; in contrast, our focus is on developing an analytical QoS-based pricing model that could work under any type of runtime distribution by using some generic assumptions.

Choice of Job Characteristics to be Priced. The measure on what factors to be priced is the efficiency of resource utilization in cloud computing, and, the job’s characteristics of users could be an indicator of the cloud system’s utilization and performance. Accordingly, one should choose such job’s characteristics (e.g., their waiting time, the slackness [15], [25] etc.) to be priced. Then, by setting the price of job’s characteristic properly, each user will behave in a desired way such that the resource efficiency are maximized when an optimal resource management scheme inside the cloud is also established, increasing the revenue of a cloud provider.

As shown in the related literature [10], [11], [12], [13], [17], under various queuing models (e.g., M/M/1, M/Pareto/1, M/G/1, G/G/1), irrespective of the jobs runtime distribution, if they could tolerate a larger waiting time, a larger job arrival rate will be allowed at each server and more jobs could be processed with their QoS requirement guaranteed, thus achieving a higher utilization. Hence, in this paper, the factor that we would price is the waiting time of jobs. The specific queuing results when the runtime of jobs follows an exponential or a Pareto distribution are also presented in the following.

4.1 Exemplified Queuing Results

Exponential Case. When the runtime of jobs follows an exponential distribution, an important measure used to guarantee QoS is completing all jobs of a user at some percentile [45], [46]. For example, a 95th percentile of some time $\varphi$ requires that 95% of jobs complete with a maximum waiting time $\varphi$. Such QoS guarantee ensures that a large enough portion of processed jobs will be completed within their response time requirement. In the existing literature [11], in the case where the runtime of jobs follows an exponential distribution, the proportion of jobs whose waiting time does not exceed $\varphi$ is minimized when jobs have a deterministic waiting time, no matter whether the first-come-first-served or earliest deadline first discipline is used. In other words, given the percentile requirement, when all jobs have a common maximum acceptable waiting time, a server could accept the most jobs and achieve the highest utilization.

Let $\lambda$ and $\mu$ denote the job arrival rate and service rate of a server, and, the utilization of this server is

$$\rho = \frac{\lambda}{\mu}. \quad (1)$$

In the following, we show the relation between the utilization $\rho$ and the waiting time $\varphi$, and, the related analysis is derived from the results of Ali Movaghar in [11].

Let $\rho_0$ denote the probability that there is no jobs in the queue, and

$$\frac{1}{\rho_0} = 1 + \rho + \frac{\rho^2}{\rho - 1} \left( e^{(\rho - 1)\cdot \mu - \varphi} - 1 \right), \text{ if } \rho \neq 1.$$ 

In the case where all jobs have a maximum acceptable waiting time $\varphi$, the proportion of jobs whose waiting time exceeds $\varphi$ is

$$\alpha_d = \rho_0 \cdot e^{(\rho - 1)\cdot \mu \cdot \varphi}. \quad (2)$$ 

When $\alpha_d = 0.05$, given the system parameter $\mu$ and the maximum acceptable waiting time $\varphi$, we could derive the maximum utilization $\rho_{\text{max}}$ of a server to meet the QoS requirement from (2). Obviously, if a server accept jobs at a rate smaller than $\rho_{\text{max}} \cdot \mu$, this server could still guarantee the QoS requirement of these jobs; if the rate become larger than $\rho_{\text{max}} \cdot \mu$, the QoS requirement will be violated and the proportion of jobs whose waiting time is larger than $\varphi$ will exceed 5%. Correspondingly, we call $\rho_{\text{max}} \cdot \mu$ to be the maximum job arrival rate to meet the QoS requirement, denoted by $\lambda_{\text{max}}$.

In Fig. 1, we illustrate how the utilization of a server varies with the waiting time where $\mu = 1$ and $\alpha_d = 0.05$. We can see that, $\rho_{\text{max}}$ increases with the value of $\varphi$. Furthermore, $\lambda_{\text{max}}$ increases with $\rho_{\text{max}}$ and also $\varphi$.

Pareto Case. Assume that the job’s runtime follows a Pareto distribution with a parameter $\alpha > 1$ and a minimum runtime $\tau$; then the expected runtime of jobs is $\frac{\tau}{\alpha - 1}$. Another common way to guarantee QoS is completing all jobs within some expected waiting time, and, the related results are from the work of Zheng et al. [10], presented as follows. When $\frac{1}{\lambda} - \frac{\alpha}{\alpha - 1} \tau > 0$, all jobs have a finite expected waiting time

$$\varphi = \frac{1}{\lambda} \log \left( \frac{\alpha (\lambda \tau)^{\alpha - 1} \Gamma(-\alpha, \lambda \tau)}{1 - \frac{\alpha}{\alpha - 1} \lambda} \right), \quad (3)$$

where $\Gamma(s, z) = \int_z^\infty x^{s-1} e^{-x} dx$ is the upper incomplete gamma function. The expected utilization of a server is

$$\rho = \frac{1}{1 + \alpha (\lambda \tau)^{\alpha} \Gamma(-\alpha - 1, \lambda \tau)} \quad (4)$$
The utilization under different waiting times is illustrated in Figure 1 where \( \alpha = 1.4 \) and \( \tau = 1/6 \). Similar to the exponential case, the maximum acceptable waiting time \( \varphi \) determines the maximum utilization \( \rho_{\text{max}} \) of a server and the maximum acceptable job arrival rate \( \lambda_{\text{max}} \), in order to meet the QoS requirement.

5 AN ANALYTICAL CLOUD PRICING MODEL

In this section, we propose a QoS-differentiated cloud pricing system and the corresponding internal resource management model.

5.1 The Service and Pricing Model

Goal of the Cloud. Assume that the cloud provider holds a fixed computing capacity of \( m \) servers. The objective of our pricing system is to incentivize users - through the offer of lower prices - to provide desired job characteristics so as to achieve a high system’s utilization and get more jobs completed. Aim is to satisfy the job’s QoS requirement while maximize the cloud provider’s revenue.

QoS-Differentiated Pricing. There are \( L \) service level agreements (SLAs), specified by the values of some job’s characteristic \( \varphi_1, \varphi_2, \ldots, \varphi_L \). In this paper, the considered job’s characteristic is job’s waiting time, and, when a user \( i \) chooses the \( l \)-th SLA, each of its jobs \( j \) is expected to be completed by the deadline \( a_j + s_j + \varphi_l \), where \( a_j \) and \( s_j \) is the arrival time and runtime of \( j \).

The price of the \( l \)-th SLA is denoted by \( \theta_l \). Here, the parameters \( \varphi_l \) and \( \theta_l \) are such that

\[
0 \leq \varphi_1 < \varphi_2 < \ldots < \varphi_L < \infty
\]

(5)

\[
p = \theta_1 > \theta_2 > \ldots > \theta_L > 0
\]

(6)

The first SLA specified by \( \{ \varphi_1, \theta_1 \} \) corresponds to the standard pricing model in the cloud market (e.g., on-demand instances in Amazon EC2, the pricing in Google Cloud Platform) where users have no willingness to tolerate any delay in the completion of their jobs, and \( \theta_1 \) is the price of utilizing a unit of computing resources in a unit of time. For the other SLAs, their prices turn lower than \( p \) at the expense of delaying the completion of \( j \) to a time larger than \( a_j + s_j \); the more delay a user can tolerate to complete its jobs, the lower the price is.

This is the pricing and scheduling framework which we refer to in the rest of the paper, as illustrated in Figure 2.

User’s Choice for Surplus Maximization. We assume that there are a total of \( K \) users. Although users may have diverse requirements of QoS, the design of cloud and other network pricing and resource management systems works widely with a type of concave logarithmic or exponential utility functions \( \varphi \). These model a situation where both the utility of a user and its marginal utility decreases as its waiting time increases. The cloud providers can further set the weights of applications or users to differentiate among them. Finally, the form of utility functions that we use is as follows:

Definition 5.1. When the jobs of a user \( i \) is completed with a maximum waiting time \( \varphi \), we assume generally that the utility per unit of workload of the user is of the following form when completing a unit of its workload:

\[
U_i(\varphi) = \alpha_i \cdot P(\varphi), \ \varphi \geq 0
\]

(7)

where \( \alpha_1, \alpha_2, \ldots, \alpha_K \) are the weights of users and \( P(\varphi) \) is a concave decreasing function.

As a result, the utility of a user \( i \) when completing job \( j \) under the \( l \)-th SLA is \( s_j \cdot U_i(\varphi_l) \). Here, \( P(\varphi) \) reflects the sensitivity to latency of the whole population of users. Since different users have different weights, from one to another SLAs, the amounts of user’s utilities will have different changes.

When applying the utility functions appearing, e.g., \( [10], [31], [32] \) in the context of this paper, function \( P(\varphi) \) can be set as follows:

\[
(1 - \beta)^{-1} (1/(1 + \varphi))^{1-\beta}, \ \beta \in (0, 1),
\]

(8)

or,

\[
\log (1 + (\epsilon + \varphi)^{-1}), \ \epsilon > 0.
\]

(9)

The objective of each user \( i \) is to maximize its unit utility \( U_i(\varphi_l) \) minus the cost \( \theta_l \) of completing its jobs by purchasing resources from the cloud, i.e., user \( i \) aims to select the \( l^*_i \)-th SLA such that

\[
l^*_i = \arg \max_{1 \leq l \leq L} \{ U_i(\varphi_l) - \theta_l \}.
\]

(10)

By setting the prices of SLAs properly, the above pricing system provides incentives for users to select appropriate characteristics, e.g., larger waiting times, which naturally align with increased system’s utilization and cloud provider’s revenue.

5.2 Internal Resource Management

We assume that each user \( i \) submits jobs to the cloud system continuously over time. Under the above service model and given the prices of different SLAs, a user will select some SLA to process its jobs and pay for the computing
service it obtains. The job arrival process of a tagged user \( i \) is assumed to follows a poisson distribution with a mean \( h_i \). Also, throughout the paper, we allow the job’s runtime to follow an arbitrary type of distribution.

**Capacity Division.** Assume that a cloud provider holds \( m \) servers. The whole capacity of a cloud is divided into \( L \) processing units, and, for all \( 1 \leq l \leq L \), the \( l \)-th processing unit is assigned \( m_l \) servers where \( \sum_{l=1}^{L} m_l = m \). The \( l \)-th processing unit is used to exclusively process all jobs under the \( l \)-th SLA. Assume that the total job arrival rate of users at the \( l \)-th processing unit is \( \Lambda_l \), and, the mean service rate of a server and job’s runtime are assumed to be \( \mu \) and \( \omega \) where \( \mu \cdot \omega = 1 \). As a result, a capacity planning scheme is needed inside the cloud to determine the number of servers assigned to each processing unit, ensuring that the QoS requirement of the accepted jobs of users is satisfied.

**Dispatching Jobs to Servers.** Since the servers are considered to be homogeneous computing units, each server has the same probability of being available at any given time and of being assigned a task, whenever a user’s job arrives at some processing unit [10] [35]. The probability that a server will be chosen for processing the job is 1/\( m_l \). Thus, using a similar argument to [10], we conclude that

**Lemma 5.1.** If job arrivals at the \( l \)-th processing unit follow a poisson distribution with a mean \( \Lambda_l \) and there are \( m_l \) servers at the \( l \)-th processing unit, the job arrivals at each server follow a poisson distribution with a mean \( \lambda_l = \frac{\Lambda_l}{m_l} \).

At a server, the first-come-first-served discipline is applied to process jobs, and, each server is used to process the jobs with the same QoS requirement.

In general, under various queuing models (e.g., M/M/1, M/Pareto/1 M/G/1, G/G/1), when the jobs to be processed on a server could tolerate a larger waiting time, a larger job arrival rate is allowed at this server, without violating the QoS requirement [10], [11], [12], [13]. Correspondingly, we define the following function to model such situation.

**Definition 5.2.** With the QoS requirement defined by \( \varphi \) satisfied, the maximum job arrival rate \( \lambda_{\text{max}} \) at this server is defined by an increasing function:

\[
\lambda_{\text{max}} = Q_1(\varphi). \tag{11}
\]

**Definition 5.3.** Given the arrival rate \( \lambda \) of jobs at a server, the utilization of this server \( \rho \) is defined by an increasing function:

\[
\rho = Q_2(\lambda). \tag{12}
\]

The function \( Q_1 \) and \( Q_2 \) could be specified when the type of job’s runtime distribution is determined. In the previous section, we have actually presented two sample cases where such definitions can be rendered in closed form. For example, when job’s runtime follows an exponential distribution, \( Q_1 \) (resp. \( Q_2 \)) could be specified by \[ \text{Table 1: Key Notation} \]

| Notation | Explanation |
|----------|-------------|
| \( L \) | the number of SLAs |
| \( l \) | the waiting time of the \( l \)-th SLA |
| \( \eta_l \) (resp. \( \theta_l \)) | the price (resp. the optimal price) of the \( l \)-th SLA |
| \( m \) | the total number of servers possessed by a cloud provider |
| \( m_l \) (resp. \( m_l' \)) | the number of servers (resp. the optimal number of servers) assigned to the \( l \)-th processing unit, as illustrated in Figure 2 |
| \( \Lambda_l \) (resp. \( \Lambda_l' \)) | the total (resp. optimal accepted) job arrival rate at the \( l \)-th processing unit |
| \( \lambda_l \) | the job arrival rate at each server of the \( l \)-th processing unit |
| \( \lambda_{l_{\text{max}}} \) | under the \( l \)-th SLA, the maximum acceptable job arrival rate at each server of the \( l \)-th processing unit |
| \( \lambda_{l_{\text{max}}} \) | the minimum number of servers needed to fulfill the \( l \)-th SLA when the arrival rate of jobs is \( \Lambda_l \), i.e., \( \lfloor \Lambda_l / \lambda_{l_{\text{max}}} \rfloor \) |
| \( \Lambda_l \) | the maximum acceptable arrival rate of jobs under the \( l \)-th SLA when there are \( m' \) servers, i.e., \( \Lambda_l(m') = \lambda_{l_{\text{max}}} m' \) |
| \( Q_l(\Lambda) \) | a queue under the \( l \)-th SLA whose job arrival rate is \( \Lambda \) |
| \( Q_l(\Lambda) \) | the maximum unit revenue that could be achieved by a server |
| \( Q_l(\Lambda) \) | two virtual (sub-)queues split from \( Q_l(\Lambda) \); the first with a job arrival rate \( \lambda_{l_{\text{max}}}^{(1)}(\Lambda) = \Lambda_l(m') \) where \( m' = \lfloor \Lambda / \lambda_{l_{\text{max}}}^{(1)}(\Lambda) \rfloor \), and, the second with a job arrival rate \( \lambda_{l_{\text{max}}}^{(2)}(\Lambda) = \Lambda - \lambda_{l_{\text{max}}}^{(1)}(\Lambda) \) where \( 0 \leq \lambda_{l_{\text{max}}}^{(2)}(\Lambda) < \lambda_{l_{\text{max}}}^{(1)}(\Lambda) \) |

**Lemma 5.1.** If job arrivals at the \( l \)-th processing unit follow a poisson distribution with a mean \( \Lambda_l \) and there are \( m_l \) servers at the \( l \)-th processing unit, the job arrivals at each server follow a poisson distribution with a mean \( \lambda_l = \frac{\Lambda_l}{m_l} \) and are assumed to be \( \rho_l \). Hence, given the prices of different SLAs, the unit revenue obtained from a server in a unit of time is

\[
v_l = \rho_l \cdot \theta_l. \tag{13}
\]

Hence, on average, the whole cloud ecosystem can obtain an expected revenue in a unit of time

\[
v(m, \theta) = \sum_{l=1}^{L} m_l \cdot v_l, \tag{14}
\]

where \( m = [m_1, \ldots, m_L] \) and \( \theta = [\theta_1, \ldots, \theta_L] \).

5.3 System Parameters to be Optimized

Now, we introduce what remains to be done to achieve the best performance of the above cloud pricing system. In order to maximize the revenue of a cloud provider (i.e., (14)), we need to address the following two optimization problems.

**Problem 1:** Capacity Planning and Admission Control. Given the prices of different SLAs, every user will choose a SLA to maximize its surplus and, as a result, the arrival rate \( \Lambda_l \) of jobs at every processing unit \( l \) is determined, for all \( 1 \leq l \leq L \). The cloud system needs to determine the optimal admission rate \( \lambda_l^* \) (\( \leq \Lambda_l \)) of jobs at which each processing unit \( l \) will accept jobs and the optimal number \( m_l^* \) of servers needed to guarantee the QoS requirement, with the objective of maximizing (14), where \( \sum_{l=1}^{L} m_l^* = m \).

To help readers grasp the relations among parameters in the subsequent optimization process, the above problem
could also be simply denoted by a function with its output as the decision variables here:
\[
(m^*, \Lambda^*) = \mathcal{F}(m, \Lambda, \theta, \varphi),
\]
where \( m^* = [m_1^*, \ldots, m_L^*], \Lambda^* = [\Lambda_1^*, \ldots, \Lambda_L^*], \varphi = [\varphi_1, \ldots, \varphi_L], \) and \( \Lambda = [\Lambda_1, \ldots, \Lambda_L]. \)

Let \( h = [h_1, \ldots, h_K] \) and \( U = [U_1(\varphi), \ldots, U_K(\varphi)]. \) In \([15]\), \( U, \varphi, \) and \( \theta \) determine the users’ choice of SLAs, and, together with \( h, \) we could derive the job’s arrival rate \( \Lambda \) at different processing units. We would in Section 6.1 propose an optimal admission control and capacity planning scheme to determine \( m^* \) and \( \Lambda^* \) for the proposed pricing system.

**Problem 2: Optimal Prices.** User’s behavior and choice for SLAs is influenced by the prices of SLAs and we would set appropriate prices to direct each user to the right SLA for revenue maximization. So, the second problem is how to determine the optimal prices of SLAs \( \theta^* = [\theta_1^*, \ldots, \theta_L^*]. \)

Built on \([14], [15], \) and \([13]\), this problem could be finally denoted by the following function:
\[
\theta^* = \arg\max_{\theta} v(m^*, \theta) = \mathcal{G}(m, h, \varphi, U).
\]

With our solution to the first problem, we would propose a procedure to determine the optimal prices of different SLAs in Section 6.2.

In this paper, all omitted proofs are given in the Appendix. The main notation of this paper is listed in Table 1.

### 6 Revenue Maximization

In this section, we maximize the revenue of a cloud provider given its capacity.

#### 6.1 Capacity Planning and Admission Control

As stated in Section 5.3, given the prices of different SLAs, the arrival rate of jobs at each processing unit is fixed. In this subsection, we propose an optimal capacity planning scheme in the cloud under the resource allocation model in Section 5.2.

Given the QoS requirement (i.e., \( \varphi_i \)) that has to be satisfied at each processing unit, the maximum acceptable job arrival rate of a server, denoted by \( \lambda_{max}^{(l)} \), and, the maximum utilization of a server, denoted by \( \rho_{max} \), could be derived from \([11]\) and \([12]\). The maximum unit revenue in \([13]\), denoted by \( v_{max} \), is achieved when the accepted arrival rate of jobs at a server is the maximum acceptable rate.

Under the dispatching policy in Section 5.3, we define the following parameters by Lemma 5.1 to help us identify some conceptual states for optimal capacity planning:

- \( m(L) \) denotes the minimum number of servers needed to fulfill the \( l \)-th SLA when the arrival rate of jobs is \( \Lambda \), i.e.,
  \[
  m_L(L) = \lfloor \Lambda/\lambda_{max} \rfloor.
  \]

- \( \Lambda_l(m') \) denotes the maximum acceptable arrival rate of jobs under the \( l \)-th SLA when there are \( m' \) servers, i.e.,
  \[
  \Lambda_l(m') = \lambda_{max}\cdot m'.
  \]

In the case where \( \sum_{l=1}^L \lfloor \Lambda_l/\lambda_{max} \rfloor \leq m \), the optimal capacity planning scheme is to accept all jobs that arrive at each processing unit and assign \( \lfloor \Lambda_l/\lambda_{max} \rfloor \) servers to the \( l \)-th SLA for all \( l \in [1, L] \). Hence, in the rest of this subsection, we only study the opposite case where \( \sum_{l=1}^L \lambda_{max} > m \). As stated in Section 6.2, each server is used to execute the jobs with the same QoS requirement. To maximize the total revenue of all servers, we should make each server generate the highest possible unit revenue when facing the jobs of all users.

Let \( Q_l(\Lambda) \) denote a queue whose job arrival rate is \( \Lambda \) and whose QoS requirement is completing all jobs with a maximum waiting time \( \varphi_l \). In particular,

- \( Q_l(\Lambda_l) \) characterizes all jobs that arrive at the \( l \)-th processing unit;
- \( Q_l(\Lambda^*_l) \) will be used to characterize the accepted jobs from \( Q_l(\Lambda_l) \) in an optimal solution to \( [15] \).

The algorithmic framework to be proposed for capacity planning is illustrated in Figure 3 according to the relations among parameters, we separate each queue \( Q_l(\Lambda_l) \) into two virtual (sub-)queues \( Q_l^{(1)}(\Lambda_l) \) and \( Q_l^{(2)}(\Lambda_l) \); under the capacity constraint, the queues that generate the highest unit values are selected from \( Q_l^{(1)}(\Lambda_l), Q_l^{(2)}(\Lambda_l), \ldots, Q_L^{(2)}(\Lambda_L) \), which could be further transformed into an optimal solution to our original problem \([15]\). In the following, we elaborate this framework and prove its optimality.

**Virtual Queues.** We first define two virtual queues.

**Definition 6.1.** The queue \( Q_l(\Lambda) \) is separated into two virtual (sub-)queues under the l-th SLA that are defined as follows:

- \( Q_l^{(1)}(\Lambda) \) with a job arrival rate \( \lambda_l^{(1)}(\Lambda) = \Lambda(m') \) where \( m' = \lfloor \Lambda/\lambda_{max} \rfloor \), and,
- \( Q_l^{(2)}(\Lambda) \) with a job arrival rate \( \lambda_l^{(2)}(\Lambda) = \Lambda - \lambda_l^{(1)}(\Lambda) \) where \( 0 \leq \lambda_l^{(2)}(\Lambda) < \lambda_{max} \).

**Problem Transformation.** As far as the queue \( Q_l(\Lambda_l) \) is concerned, to gain a unified representation, we also denote the virtual queues \( Q_l^{(1)}(\Lambda_1), Q_l^{(2)}(\Lambda_1), \ldots, Q_l^{(2)}(\Lambda_L) \) by \( Q_1, Q_2, \ldots, Q_{2L} \) with their arrival rates being \( \Lambda_1, \Lambda_2, \ldots, \Lambda_{2L} \). In the following, we define a new and easier problem whose optimal solution could be transformed into an optimal solution to the original problem \([15]\):

- maximize the total unit revenue when dispatching all or partial jobs of queues \( Q_1, Q_2, \ldots, Q_{2L} \) to \( m \) servers.

Here, the accepted jobs from two different queues will be viewed to be processed in two different processing units although they may have the same QoS requirement. As a result, a cloud system of \( m \) servers is divided into 2\( L \) processing units.

Formally, we denote by \( \bar{\varphi} = [\bar{\varphi}_1, \ldots, \bar{\varphi}_{2L}] \) and \( \bar{\theta} = [\bar{\theta}_1, \ldots, \bar{\theta}_{2L}] \) the QoS requirements of \( Q_1, \ldots, Q_{2L} \) and their prices. Let \( \Lambda = [\Lambda_1, \ldots, \Lambda_{2L}] \), and, similar to \([15]\), the new optimization problem could be expressed by a function with its output as the decision variables to maximize the total unit revenue:
\[
(\tilde{m}^*, \tilde{\Lambda}^*) = \mathcal{H}(m, \Lambda, \bar{\theta}, \bar{\varphi}).
\]
Here, $\tilde{m}^*$ denotes the number of assigned servers and $\tilde{\Lambda}^* = [\tilde{\Lambda}^*_1, \ldots, \tilde{\Lambda}^*_L]$. Each $\tilde{m}^*_i$ denotes the number of assigned servers and each $\tilde{\Lambda}^*_i$ denotes the accepted job arrival rate at the $i$-th processing unit where $1 \leq i \leq 2L$ and $\sum_{i=1}^{2L} \tilde{m}^*_i \leq m$.

In the following, we explain the relation between the optimal solutions to (15) and (18). As we will show, the problem (18) exhibits a nice greedy-choice property \[47\]: a globally optimal solution could be assembled by making locally optimal (greedy) choices.

**Lemma 6.1.** In an optimal solution to (15), the optimal number of servers assigned to each processing unit is the minimum number of servers needed to fulfill the $l$-th SLA, i.e., $m_l(\Lambda^*_l)$, for all $1 \leq l \leq L$.

**Lemma 6.2.** In an optimal solution to (18), we have that

- for all $1 \leq l \leq 2L$, the optimal number of servers assigned to each processing unit is the minimum number of servers needed to fulfill the SLA;
- for all $1 \leq l \leq L$, the arrival rate of the accepted jobs of $Q^{(2)}_l(\Lambda^*_l)$ is such that $\tilde{\Lambda}^*_l / \lambda^{(2)}_{\max}$ is an integer, while the arrival rate of the accepted jobs of $Q^{(2)}_l(\Lambda^*_l)$ is either zero or $\lambda^{(2)}_{\max}$.

For each integer $l \in [1, L]$, let $i_l$ and $j_l$ be such that $\tilde{Q}_{i_l}$ and $\tilde{Q}_{j_l}$ are two queues under the $l$-th SLA where $1 \leq i_l, j_l \leq 2L$; so, we assume without loss of generality that $\tilde{Q}_{i_l}$ and $\tilde{Q}_{j_l}$ are respectively $Q^{(1)}_l(\Lambda_l)$ and $Q^{(2)}_l(\Lambda_l)$.

**Theorem 6.1.** An optimal solution to (18) corresponds to an optimal solution to (15) such that, at the $l$-th processing unit,

1. the accepted job arrival rate $\Lambda^*_l$ equals the sum of $\tilde{\Lambda}^*_l$ and $\tilde{\Lambda}^*_{j_l}$;
2. the number of assigned servers $m^*_l$ equals the sum of $\tilde{m}^*_i$ and $\tilde{m}^*_j$.

**Algorithm.** In the following, built on Theorem 6.1, we will give an optimal solution to (15) by proposing an optimal solution to (18). We define the unit revenue of queue $Q^{(1)}_l(\Lambda_l)$ (or $\tilde{Q}_{i_l}$) to be

- the maximum unit revenue $v^{(l)}_{\max}$ under the $l$-th SLA, i.e., when it is assigned $\lambda^{(1)}_l(\Lambda_l)/\lambda^{(l)}_{\max}$ servers, and define the unit revenue of $Q^{(2)}_l(\Lambda_l)$ (or $\tilde{Q}_{j_l}$) to be
- the unit revenue when it is assigned $[\lambda^{(2)}_l(\Lambda_l)/\lambda^{(l)}_{\max}]$.

In the latter case, the unit revenue is either zero or some value less than $v^{(l)}_{\max}$ since $0 \leq \lambda^{(2)}_l(\Lambda_l) < \lambda^{(l)}_{\max}$ by Definition 6.1. We also denote the unit revenues of $Q_1, \tilde{Q}_2, \ldots, Q_{2L}$ by $v_1, v_2, \ldots, v_{2L}$; in terms of their unit revenues, we assume without loss of generality that $v_1 \geq v_2 \geq \ldots \geq v_{2L}$.

Finally, we propose a procedure, presented as Algorithm 1, to determine the optimal $m^*$ and $\Lambda^*$ of the problem (15) and the optimal $m^*$ and $\Lambda^*$ of the problem (18). In Algorithm 1, the queues $\tilde{Q}_1, \tilde{Q}_2, \ldots$ that generate the highest revenue are selected successively with the capacity constraint, which is also elaborated in Figure 3. As implied by the line 5 of Algorithm 1, the accepted jobs of both $Q^{(1)}_l(\Lambda_l)$ and $Q^{(2)}_l(\Lambda_l)$ under the same SLA are finally merged together as the total accepted jobs at the $l$-th processing unit of the problem (15), forming a single queue with an arrival rate of jobs $\tilde{\Lambda}^*_l$.

**Theorem 6.2.** Given the prices of SLAs, Algorithm 1 gives an optimal solution to (15) such that the total unit revenue of all servers is maximized.

**Proof.** It suffices to show that, Algorithm 1 gives an optimal solution to (18). Then, by Theorem 6.1, an optimal solution to (15) is given by the operations in line 6 of Algorithm 1.

Firstly, we show two properties in an optimal solution to (18). These two properties jointly determine an optimal solution to (15). Each server is used to process the jobs with the same QoS requirement and, by Lemma 6.2 in an optimal solution, the first property is that: there exists an integer $l \in [1, L]$ for each server such that it has a job arrival rate of either $\lambda^{(1)}_l$ or $\lambda^{(2)}_l(\Lambda_l)$. As a result, it either achieves the maximum unit revenue under the $l$-th SLA (i.e., $v^{(l)}_{\max}$) by processing jobs of $Q^{(1)}_l(\Lambda_l)$, or, the maximum possible
by processing all jobs of $Q_i^{(2)}(\Lambda_l)$. Further, we show that the problem exhibits the greedy-choice property [47] by contradiction. Suppose that there exist two queues $Q_i$ and $Q_j$ such that (1) $\bar{v}_i > \bar{v}_j$ and (2) jobs of $Q_j$ are included in an optimal solution while (a part of) jobs of $Q_i$ are excluded. Here, either $Q_i$ is from some $Q_i^{(2)}(\Lambda_l)$ and all its jobs are excluded, or, $Q_l$ is from some $Q_l^{(1)}(\Lambda_l)$ and the remaining jobs of $Q_l^{(1)}(\Lambda_l)$ have an arrival rate that is a multiple of $\lambda_{\text{max}}$. Replacing the existing jobs of $Q_j$ on some server with the jobs of $Q_i$ with an arrival rate $\lambda_{\text{max}}$ in the latter case, or, with all jobs of $Q_l$ in the former case, could achieve a higher unit revenue. This contradicts that the original solution is optimal. Hence, in an optimal solution, the second property is that, the jobs of the queues with the highest unit revenues are considered one by one to be accepted with the capacity constraint.

Secondly, we prove that, Algorithm [3] gives a solution that satisfies the two properties above. It considers the queues in the non-increasing order of their unit revenues and accepts as many queues as possible with the highest unit revenues. Let $l'$ denote the last queue whose jobs are accepted by Algorithm [3] after $Q_l$ is considered, the number of unassigned servers becomes zero. For all $1 \leq l_0 \leq l'$, if $Q_{l_0}$ is from $Q_i^{(2)}(\Lambda_l)$ from some $l \in [1, L]$, all jobs of $Q_{l_0}$ are accepted and $[\lambda_i^{(2)}(\Lambda_l)/\lambda_{\text{max}}] = 1$ server is assigned to $Q_{l_0}$ where $0 < \lambda_i^{(2)}(\Lambda_l)/\lambda_{\text{max}} < 1$, since the condition in the line 4 of Algorithm [4] is true and by the line 5 of Algorithm [3] and Definition 6.1. For all $1 \leq l_0 \leq l' - 1$, if $Q_{l_0}$ is from $Q_i^{(1)}(\Lambda_l)$ from some $l$, all jobs of $Q_{l_0}$ are accepted and $\lambda_i^{(1)}(\Lambda_l)/\lambda_{\text{max}}$ server(s) are or are assigned to $Q_{l_0}$ where $\lambda_i^{(1)}(\Lambda_l)/\lambda_{\text{max}}$ is an integer, due to Definition 6.1 and since the number of available servers is still greater than zero after some servers are assigned to $Q_{l_0}$. If $Q_l$ is from some $Q_l^{(1)}$, constrained by the number of currently unassigned (i.e., $m'$ in Algorithm [3]) just before $Q_l$ is considered, the accepted job arrival rate of $Q_l$ is $\Lambda_l(m')$ and the number of assigned servers is $m'$.

Combining the two points above, the theorem holds. □

6.2 Prices of Different SLAs

In this subsection, we determine the optimal prices of different SLAs in the pricing system.

Problem Reduction. There are $K$ users and we assume without loss of generality that $\alpha_1 > \alpha_2 > \cdots > \alpha_K$ for the coefficients in [5.1]. As a result, the utility functions of all users satisfy the following relation:

$$U_l(\varphi) > U_{l+1}(\varphi) > \cdots > U_K(\varphi)$$

for an arbitrary $\varphi \in \mathcal{R}$.

Let $\Phi = \{1, 2, \cdots, K\}$. Given the prices of $L$ SLAs, assume that the $l$-th SLA will be chosen by a subset of users $\Phi_l \subseteq \Phi$. $\Phi_l$ is either empty or non-empty. In the case where $\Phi_l$ is empty, the cloud provider will not provide the $l$-th SLA to users and its price could be viewed to be $\infty$. In an optimal solution, the users also behave as above and there exists an integer $L'$ and a sequence $l_1 < l_2 < \cdots < l_{L'}$, where $L' \leq L$, such that only the $l_1$-th, $\cdots$, $l_{L'}$-th SLAs are offered to users.

Hence, the following two steps could be used to determine the optimal prices of these SLAs:

(i) for every combination of $\{1, 2, \cdots, L\}$, use it to determine which SLAs to be offered, and, find an algorithm to compute the optimal prices of these chosen SLAs to generate the highest unit revenue in [47].

(ii) select the combination that achieves the maximum revenue.

Denote by $l'_1 < l'_2 < \cdots < l'_L$, the combination selected in the above 2nd step, where $L' < L$. This combination specifies that the $l'_1$-th, $\cdots$, $l'_L$-th SLAs are to be offered in an optimal solution and the prices that achieve the maximum revenue is also the optimal prices that should be set for these SLAs.

Our original problem in this subsection is to determine which SLAs are to be offered to users in an optimal solution and the optimal prices of these SLAs. Now, it reduces to a simpler problem as follows. Given an arbitrary subset of SLAs, when each SLA of this subset will be chosen by some users, we need to find an algorithm to determine the optimal prices of these SLAs in this subset. Since the number $L$ of SLAs offered to users is finitely bounded, the number of all the combinations of $\{1, 2, \cdots, L\}$ is a constant. Only if the algorithm for the subset of SLAs has a polynomial time complexity, the algorithm that we obtain for our original problem also has a polynomial time complexity.

To avoid introducing too much notation, in the following, without loss of generality, we propose an algorithm to determine the optimal prices of SLAs $\theta'_1, \theta'_2, \cdots, \theta'_L$ when all $L$ SLAs will be chosen by users.

Algorithm. Each SLA will be chosen by a subset of users and a higher QoS requirement will have a higher price. We first identify some structural information in an optimal solution.
Lemma 6.3. Given the optimal prices $\theta^1_i, \ldots, \theta^L_i$ of the $L$ SLAs, there exist integers $i^1_1, \ldots, i^L_1$ such that all $1 \leq i^1_1 < \cdots < i^L_1 \leq K$ such that, for all $l \in [1, L]$, the $l$-th SLA is to be chosen by users $i^l_{n-1} + 1, \ldots, i^l_n$, where $i^l_0 = 0$.

Proof. Let $1 \leq j_1 < j_2 < L$ and $i_1$ and $i_2$ denote two users who will respectively choose the $j_1$-th and $j_2$-th SLAs; then we have $i_1 < i_2$. We prove this observation by contradiction. Otherwise, we have $i_1 > i_2$, and the, users’ choice implies the following relations

$$U_{i_1}(\phi_{j_1}) - \theta^1_{j_1} > U_{i_2}(\phi_{j_2}) - \theta^1_{j_2},$$
$$U_{i_2}(\phi_{j_1}) - \theta^1_{j_1} < U_{i_2}(\phi_{j_2}) - \theta^2_{j_2}. \tag{19}$$

Multiplying (19) by $-1$ and then adding it to (19), we derive that $U_{i_1}(\phi_{j_1}) - U_{i_1}(\phi_{j_2}) > U_{i_2}(\phi_{j_1}) - U_{i_2}(\phi_{j_2})$. According to Definition 5.1, this contradicts with the fact that $i_1 > i_2$. Each SLA will be chosen by a subset of users and, using the above observation, we conclude that the indexes of users who choose a higher QoS requirement will be smaller than the indexes of users who choose a lower QoS requirement. As a result, there exists a sequence $i^1_1, \ldots, i^L_1$ where $i^1_1 < \cdots < i^L_1 \leq K$ such that Lemma 6.3 holds.

Since the capacity of a cloud provider is assumed to be limited, the optimal price $\theta^L_i$ of the $L$-th SLA might not be a value larger than $U_K(\phi_L)$ such that not all users would accept to choose some SLA (with their surpluses larger than zero). In other words, the $i^L_2$ in Lemma 6.3 is not necessarily equivalent to $K$.

For all $l \in [1, L]$, let $U_{i_1}^l(\phi_l) = U_{i_1}(\phi_l) - U_{i_1}(\phi_{l+1})$ and $U_{i_1}^{l+1}(\phi_l) = U_{i_1}(\phi_l) - U_{i_1}(\phi_{l+1} + 1)$.

Definition 6.2. We define the parameter $\theta^L_i$ to be such that,

(i) for all $1 \leq l \leq L - 1$, it is the maximum $\theta^l_i$ that satisfies the following relation:

$$U_{i_1}^{l+1} + \theta_{l+1} > \theta_{l} < U_{i_1}^{l} + \theta_{l+1},$$

(ii) $\theta^L_i$ is set to a value smaller than but close enough to $U_{i_1}(\phi_i)$.

Theorem 6.3. There exists a sequence $1 \leq i^1_1 < i^2_1 < \cdots < i^L_1 \leq K$ such that, for all $1 \leq l \leq L$, the optimal price of the $l$-th SLA is $\theta^l_i$, as specified in Definition 6.2.

Proof. Given the optimal prices of different SLAs, there exist $i^1_1, \ldots, i^L_1$ that satisfy the properties described in Lemma 6.3. Suppose that the $i^1_1, i^2_1, \ldots, i^L_1$ are known in advance. For all $l \in [1, L - 1]$, the $l$-th SLA will be chosen by user $i^l_1$ but not by user $i^l_1 + 1$. This means that the optimal price of the $l$-th SLA needs to satisfy the following relations

$$U_{i^l_1}(\phi_l) - \theta^l_i > U_{i^l_1}(\phi_{l+1}) - \theta_{l+1}, \tag{21}$$
$$U_{i^l_1}(\phi_l) - \theta^l_i < U_{i^l_1}(\phi_{l+1}) - \theta_{l+1}. \tag{22}$$

To maximize the revenue of a cloud provider (i.e., $\Pi$), for all $l \in [1, L - 1]$, the optimal price of the $l$-th SLA will be $\theta^l_i$ as specified in Definition 6.2. For the last SLA, to ensure that the user $i^L_1$ will choose the $L$-th SLA, we need the relation that $U_{i^L_1}(\phi_l) - \theta^L_i > 0$, as a result, the optimal price of the $L$-th SLA is the $\theta^L_i$ specified in Definition 6.2. Finally, the theorem holds.

Algorithm 2: Optimal Prices of Different SLAs

1. $A' \leftarrow \{(i_1, \ldots, i_L) | 1 \leq i_1 < \cdots < i_L \leq K\}; \quad // A'$ is a $L$-combination of $\{1, 2, \ldots, K\}$.
2. $A \leftarrow \{(\theta^1_i, \theta^2_i, \ldots, \theta^L_i) | (i_1, \ldots, i_L) \in A'\}; \quad // A$ is the set whose element is a tuple of the optimal prices of SLAs in the case where $i^1_1 = i_1, i^2_1 = i_2, \ldots, i^L_1 = i_L$, as defined in Definition 6.2.
3. $\theta^* \leftarrow \arg\max_{\theta \in \Theta} \{v(m^* \theta)\}; \quad // \theta^*$ is the optimal prices of SLAs are returned, achieving the highest revenue.

Fig. 4: Three curves of $\mathcal{P}(\varphi)/\mathcal{P}(0)$ with $\beta = 0.75, 0.45, 0.25$ from top to bottom, implying how much the utility of a population of users is decreased by as the waiting time increases.

7 PERFORMANCE EVALUATION

In this section, we will evaluate the performance of the proposed analytical QoS-differentiated pricing model. In our evaluations, a main performance metric is how much the unit revenue of a cloud provider is improved by, when the proposed pricing model is compared with the standard on-demand model. In the on-demand model, users are charged a fixed price whenever they utilize a server for a unit of time and it corresponds to the first SLA in our model as explained in Section 5.

7.1 Simulation Setup

Assume that there are $K = 50$ users. For simplicity, the job arrival rate $h_i$ of each user $i$ is set to 20. We use the utility function in Definition 5.1 and is further specified by (9).

The performance of a QoS-differentiated pricing model and user’s choices of SLAs are jointly affected by (i) the number of servers possessed by a cloud provider, (ii) the
user’s sensitivities to latency, and, (iii) the weights of utility functions, etc. We therefore consider three cases where users have high, medium, and low sensitivities to latency; correspondingly, we set $\beta$ to 0.75, 0.45, and 0.25 respectively. The effect of $\beta$ on the utility of a population of users is also illustrated in Figure 4. In terms of the weights of utility functions, we consider two cases where the weight distribution is compact and loose respectively. In the former, the weights of the 1st, 2nd, $\cdots$, 50th users are set to 100, 99, $\cdots$, 51; in the latter, the ratio of the largest weight of users to the smallest one is set to be larger, e.g., about 20, and, for all $1 \leq i \leq 50$, the $(51 - i)$-th user’s weight is set to $1 + (i - 1) \cdot 0.4$.

In the following, the parameter $r_{i,j}^{(k)}$ is used to denote the ratio (of the total unit revenue of our model to the total unit revenue when only the first SLA is provided) minus 1, under some simulation setup. In particular, under the same $k$ and $j$, $r_{1,j}^{(k)}$, $r_{2,j}^{(k)}$, and $r_{3,j}^{(k)}$ denote the values of this parameter in the cases where $\beta$ equals 0.25, 0.45, and 0.75 respectively, $r_{1,1}^{(k)}$ and $r_{1,2}^{(k)}$ denote the parameter values in the compact and loose cases for the weight distribution, and, $r_{1,1}^{(1)}$, $r_{1,2}^{(2)}$, and $r_{1,2}^{(3)}$ denote the parameter values when a cloud provider respectively possesses 800, 1600, and 2400 servers. For example, $r_{1,2}^{(2)}$ denotes the parameter value in the case where $\beta = 0.75$, the weight distribution is loose, and a cloud provider possesses 1600 servers.

Performance Metrics. In this paper, the main performance metric is the revenue improvement of our model, i.e.,

$$r_{i,j}^{(k)},$$

representing how much the revenue of a cloud provider is improved by when the proposed pricing model of this paper is compared with the standard on-demand model.

As discussed in Section 3, two common distributions used to model the job’s runtime are exponential and Pareto, and, Figure 4 illustrates the change of utilization with the waiting time under these distributions. In either case, the function $\rho = Q_2(\lambda)$ is increasing. Suppose that a cloud provider would provide $L = 5$ SLAs, and, in the following, we exemplify the effect of job’s runtime distribution (i.e., the curves in Figure 4) on the revenue improvement.

In the exponential and Pareto cases, we respectively set the waiting time in the 1st SLA to $\varphi_1 = 0$ and $\varphi_1' = 0.05$; correspondingly, the utilizations are 5.26% and 0.3168%. In the latter case, setting $\varphi_1'$ to a value larger than but close to 0 guarantees a non-zero utilization. After the values of $\varphi_2$, $\varphi_3$, $\varphi_4$, and $\varphi_5$ are set for the exponential case, there exists a solution for the Pareto case where the revenue improvement is better. In particular, we could set the waiting times of the 2nd, 3rd, 4th, and 5th SLAs $\varphi_2$, $\varphi_2'$, $\varphi_4$, and $\varphi_5$ in the Pareto case to be such that (i) the ratios of the utilizations under the waiting times $\varphi_2$, $\varphi_2'$, $\varphi_4$, and $\varphi_5$ to the utilization under $\varphi_1'$ is the same as the ratios of the utilizations under $\varphi_2$, $\varphi_3$, $\varphi_4$, and $\varphi_5$ to the utilization under $\varphi_1$ in the exponential case, and, (ii) $\varphi_1 > \varphi_1'$ for all $2 \leq i \leq 5$. As a result, under each of the 2nd, 3rd, 4th, and 5th SLAs, a user will have a larger utility in the Pareto case than the one in the exponential case, and, a larger total unit revenue is achieved in the Pareto case. Finally, we choose a worse case to evaluate the revenue improvement where job’s runtime distribution is exponential. We also set $\varphi_2 = 1$, $\varphi_3 = 2$, $\varphi_4 = 4$, and $\varphi_5 = 8$, where the corresponding utilizations are respectively 13.62%, 28.63%, 58.83%, and 85.34%.

7.2 Results.

The main simulation results are illustrated in Figure 5 where the values of $r_{1,1}^{(k)}$ and $r_{1,2}^{(k)}$ correspond to the blue and red ones at the location $3 \cdot (i - 1) + k$. The revenue improvement of our model over the standard on-demand model ranges from 537.5% to 72.34% in the compact case and from 339.3% to 31.78% in the loose case, which vary with the parameters $m$ and $\beta$. In all cases, the revenue improvements are either huge (e.g., up to 537.5%) or remarkable (e.g., 31.78% in the worst case). As shown in Figure 5 when a cloud provider adopts the QoS-differentiated pricing model, a significant revenue improvement could be achieved especially in the cases where the population of users is less sensitive to latency, the weights of users are more compact, and the population of users is large compared with its processing capacity (i.e., the number of possessed servers).

In the following, we take a closer look at the simulation results in some typical cases to explain the above phenomena.

Phenomenon (i). Under the same the weight distribution and $\beta$, and, given the population of users, the revenue improvement decreases with the number of servers a cloud provider possesses, as shown in Figures 5.

We consider an example in the compact case where $\beta = 0.5$. The related simulation results are listed in rows 2-4 of Table 2. Here, each item in the first column specifies the case under which the simulation is done, e.g., the item in the second row implies that the weight distribution is compact, $m = 800$, and $\beta = 0.45$. The last column of Table 2 gives the revenue improvements under different cases. Each tuple $(y_1, y_2, y_3, y_4)$ under the $l$-th SLA respectively specify the values of $\theta^*_l$, $\Lambda^*_l$, $\tau^*_l$, and $\mu^*_l$ in an optimal solution when there are $m$ servers. As a result, this tuple implies that, at the $l$-th processing unit, partial or all jobs of the $(i^*_l - 1)$-th, $\cdots$, $i^*_l$-th users will be served at a price of $\theta^*_l$, and, the cloud provider will accept jobs at a rate of $\Lambda^*_l$ with a total of $m^*_l$ utilized. The cross in Table 2 denotes that this SLA will not be offered in an optimal solution where $i^*_l$ could be viewed to equal $i^*_{l-1}$ where $i^*_0 = 0$. 

![Fig. 5: Revenue Improvements.](image-url)
The second phenomenon is that, the more compact the weight distribution of users, the larger the revenue improvement in the former case will also be larger than the one in the latter case, which is also implied by the simulation results in rows 3 and 4 of Table 2 and, the (optimal) revenue improvement in the former case will also be larger.

Phenomenon (iii). The third phenomenon is that, the more compact the weight distribution of users, the larger the revenue improvement. In the loose case, a cloud provider avoids admitting too many users; otherwise, there exists some SLAs that are chosen by a significant part of users and whose prices will be very low, which leads to a low revenue. The simulation results in the compact and loose cases are respectively given in rows 3 and 6 of Table 2, where $m = 1600$ and $\beta = 0.5$. In the former case, a total of 38 users are admitted while 24 users are admitted in the latter case.

### TABLE 2: The Optimal Solutions under Different Cases.

| SLA  | 1st SLA | 2nd SLA | 3rd SLA | 4th SLA | 5th SLA | Revenue Improvement |
|------|---------|---------|---------|---------|---------|---------------------|
| (compact, 800, 0.45) | $\times$ | $\times$ | $\times$ | (18, 57.93, 360.0, 612) | (26, 40.73, 160.0, 188) | 264.8% |
| (compact, 1600, 0.45) | $\times$ | $\times$ | $\times$ | (37, 47.47, 560.0, 952) | (38, 34.21, 19.63, 23) | 170.5% |
| (compact, 2400, 0.45) | $\times$ | $\times$ | (26, 61.60, 520.0, 1817) | (41, 43.40, 300.0, 510) | (44, 31.00, 60.00, 71) | 117.4% |
| (compact, 1600, 0.2) | $\times$ | $\times$ | (13, 43.38, 259.0, 907) | (32, 27.00, 380.0, 646) | (34, 17.19, 40.00, 47) | 106.1% |
| (loose, 1600, 0.45) | $\times$ | (3, 17.17, 59.94, 440) | (16, 12.27, 260.0, 909) | (22, 8.718, 120.0, 204) | (24, 6.190, 40.00, 47) | 89.57% |

### TABLE 3: The Optimal Solutions for the On-demand Pricing Model under Different Cases.

| SLA  | 1st SLA | 2nd SLA | 3rd SLA | 4th SLA | 5th SLA | Revenue Improvement |
|------|---------|---------|---------|---------|---------|---------------------|
| (compact, 800, 0.45) | (3, 196.0, 42.11, 800) | (5, 192.0, 84.20, 1600) | (7, 188.0, 126.3, 2400) | (6, 192.0, 84.20, 1600) | (5, 192.0, 84.20, 1600) | 89.57% |

In addition, the standard on-demand model is a special case and corresponds to the first SLA of our model; the related simulation results to be used are listed in Table 3. For example, the first column says that, under the compact case where $m = 800$ and $\beta = 0.45$, the optimal solution is specified by the tuple (3, 196.0, 42.11, 800). This tuple also implies that, partial jobs of the first three users will be served with a price of 196.0, and, the cloud provider accepts jobs at a rate of 42.11 with all 800 servers utilized.

From Tables 2 and 3 we could see that, our model could greatly improve the utilization of cloud resources. The on-demand model achieves a utilization of 5.26%, and, when $m$ equals 800, 1600 and 2400 servers, a cloud provider could respectively serve 3, 5, and 7 users, in spite of the weight distribution. In contrast, in our model, when $m$ equals 800, 1600, and 2400, our model could respectively serve 26, 38 and 44 users and achieves a resource utilization of 65.00%, 47.39%, and 55.00%. The above phenomenon arises also due to the high utilization achieved by our pricing model. With more servers held by a cloud provider, it could admit much more users than the on-demand model. To admit more users, this leads to that the prices of SLAs in our model have a larger decrement, which is more directly shown by the results of rows 3-4. This brings about the above phenomenon since the utilization of servers under each SLA is always fixed.

Phenomenon (ii). The second phenomenon is that, the smaller the value of $\beta$ is, the smaller the revenue improvement is. Intuitively, this is due to the utilities of users decrease with the waiting time more dramatically when $\beta$ is smaller. As a result, with a smaller $\beta$, the price of a lower QoS requirement will be lower and therefore the revenue improvement becomes smaller. For example, when $m = 1600$ and under the same weight distribution, we consider two cases where $\beta$ equals 0.45 and 0.25 respectively. The related simulation results are given in rows 3 and 5 of Table 2. In the latter case, assume that the $l$-th SLA is chosen by users $i^*_{l-1} + 1, \cdots, i^*_l$. In the former case, with its value of $\beta$, we could set the prices of SLAs to be such that each SLA $l$ will also be chosen by users $i^*_{l-1} + 1, \cdots, i^*_l$ (such prices could be determined by Definition 6.2). As a result, with a larger $\beta$ in the former case, the price of each SLA will be lower than the one in the latter case, which is also implied by the simulation results in rows 3 and 4 of Table 2 and, the (optimal) revenue improvement in the former case will also be larger.

### 8 Conclusion

QoS differentiation has been validated as a very effective technique in other domains than cloud computing. It fits nicely the need of a service provider to trade off SLA requirements and available resources. In particular, in modern IP networks, QoS differentiation is a key tool for resource efficiency, provision of QoS guarantees, and user-friendly access to services. Several related pricing schemes are also studied in depth in the related literature. In this paper, we propose the first analytical QoS-differentiated pricing model in the cloud computing domain, where the IaaS paradigm is the natural application field for such schemes.

In this work, optimal schemes have been proposed to well address two key, intertwined aspects of the model: (i) the pricing of different levels of QoS requirements, and, (ii) the arrival rate of jobs accepted to be processed, in connection with the number of servers assigned to each level of QoS requirement. We leverage queuing models to obtain the analytical framework used in this paper. The framework is made generic to work under any type of job’s runtime distribution.

Simulations show that the revenue of a cloud provider could be improved by up to a five-fold increase and the system’s utilization could be improved by an eleven-fold increase.
REFERENCES

[1] "Gartner Says Worldwide Public Cloud Services Market to Grow 18 Percent in 2017." http://www.gartner.com/newsroom/id/3616417

[2] https://en.wikipedia.org/wiki/Service_design

[3] "Amazon E2 Options." https://aws.amazon.com/ec2/purchasing-options/

[4] "Pricing Philosophy: affordable on-demand prices and a commitment to Moores Law." https://cloud.google.com/pricing/philosophy/

[5] Michal Feldman, Nick Gravin, and Brendan Lucier. "Combinatorial "Pricing Philosophy: affordable on-demand prices and a commitment to Moores Law." https://cloud.google.com/pricing/philosophy/

[6] Aviad Rubinstein. "On the Computational Complexity of Optimal Simple Mechanisms," in ACM ITCS, 2016.

[7] Ali Movaghar. "On Queueing with Customer Impatience until the Beginning of Service." Queueing Systems: Theory and Applications, 1998.

[8] John N. Daigle. "The Basic M/G/1 Queueing System." Queueing Theory with Applications to Packet Telecommunication, pp. 159-223. ISBN 0-387-22857-8. Springer, 2005.

[9] J. F. C. Kingman and M. F. Atiyah. "The Single Server Queue in Heavy Traffic," in Mathematical Proceedings of the Cambridge Philosophical Society, vol. 57, 1961.

[10] Jeff Rasley, Konstantinos Karanasos, Srikanth Kandula, Rodrigo Fonseca, Milan Vojnovic, and Sriram Rao. "Efficient Queue Management for Cluster Scheduling," in ACM EuroSys, 2016.

[11] Sanjoy K. Baruah, and Jayant R. Haritsa. "Scheduling for Overload Management for Infrastructure as a Service (IaaS) in Cloud Computing: A Survey." Journal of Network and Computer Applications, 2014.

[12] Ian A. Kash, and Peter B. Key. "Pricing the Cloud." IEEE Internet Computing, 2016.

[13] Ruiting Zhou, Zongpeng Li, Chuan Wu, and Zhiyi Huang. "An Efficient Cloud Market Mechanism for Computing Jobs with Soft Deadlines." IEEE/ACM Transactions on Networking, 2016.

[14] Hong Xu, and Baochun Li. "A Study of Pricing for Cloud Resources." ACM SIGMETRICS Performance Evaluation Review, 2013.

[15] Ali Movaghar. "On Queueing with Customer Impatience until the Beginning of Service." Queueing Systems: Theory and Applications, 1998.

[16] Xiaoxi Zhang, Zhiyi Huang, Chuan Wu, Zongpeng Li, and Francis Lau. "An Online Auction Framework for Dynamic Resource Provisioning in Cloud Computing," in ACM SIGMETRICS, 2014.

[17] Xiaoji Zhang, Zhiyi Huang, Chuan Wu, Zongpeng Li, and Francis Lau. "Online Auctions in IaaS Clouds: Welfare and Profit Maximization with Server Costs." in ACM SIGMETRICS, 2015.

[18] Junliang Chen, Chen Wang, Bing Bing Zhou, Lei Sun, Young Choon Lee, and Albert Y. Zomaya. "Tradeoffs between profit and customer satisfaction for service provisioning in the cloud," in IEEE HPDC, 2011.

[19] Xiao Liu, Xinyu Wang. "How to Bid the Cloud?" in ACM SIGCOMM, 2015.

[20] Mustafa Y. Arslan, and Sampath Rangarajan. "A Practical Traffic Engineering Approach," in ACM SIGCOMM 2013.

[21] Vineet Abhishek, Ian A. Kash, and Peter B. Key. "Effective Modeling Approach for IaaS Data Center Performance Analysis under Heterogeneous Workload." IEEE Transactions on Cloud Computing, 2016.

[22] Hadoop GridMix. http://hadoop.apache.org/docs/r1.2.2/gridmix.html.

[23] Junliang Chen, Chen Wang, Bing Bing Zhou, Lei Sun, Young Choon Lee, and Albert Y. Zomaya. "Tradeoffs between profit and customer satisfaction for service provisioning in the cloud," in IEEE HPDC, 2011.

[24] Harish Kasture, Davide B. Bartolini, Nathan Beckmann, and Daniel Sanchez. "Fast Analytical Power Management for Latency-Critical Systems." IEEE/ACM MICRO, 2015.

[25] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. "Introduction to Algorithms (2nd edition)." McGraw-Hill Higher Education, 2001.

APPENDIX A

PROOF OF LEMMA 6.1

(By contradiction.) If $Q_l(A_l)$ is assigned more than $\frac{\ell_l}{\lambda_{max}}$ servers, we could execute $Q_l(A_l)$ on $\frac{\ell_l}{\lambda_{max}}$ servers, which is feasible by Lemma [5.1] and, process more jobs on the remaining servers, achieving a higher revenue. Hence, the lemma holds.

APPENDIX B

PROOF OF LEMMA 6.2

(By contradiction.) The proof of the first point in Lemma 6.2 is similar to the proof of Lemma 6.1 and we omit it here. Now, we prove the second point. If the arrival rate of the accepted jobs of $Q_l(\ell_l)$ in an optimal solution is not a multiple of $\frac{\ell_l}{\lambda_{max}}$, this means that we still have
the opportunity to increase the arrival rate of each server to $\lambda_{\text{max}}^{(l)}$ to get a solution to (18) with a higher revenue. If the arrival rate of the accepted jobs of $Q_{(2)}^l(\Lambda_l)$ is not zero but smaller than $\lambda_{(2)}^*(\Lambda_l)$, one server will be assigned for processing the accepted jobs and this means that we also have opportunity to increase the job arrival rate at this server from $Q_{(2)}^l(\Lambda_l)$, achieving a higher revenue. Hence, the lemma holds.

**APPENDIX C**

**Proof of Theorem 6.1**

It suffices to show that, an optimal solution to (18) corresponds to a feasible solution to (15) that generates the same total unit revenue, and, vice versa. The latter shows that, the total unit revenue generated by an optimal solution to (18) is an upper bound of the maximum unit revenue of (15); the former shows that, there is a feasible solution to (15) with the same total unit revenue as the optimal solution to (18). As a result, this feasible solution derived from the optimal solution to (18) is an optimal solution to (15).

Firstly, we prove the latter. Given an optimal solution to (18), the accepted jobs at the $l$-th processing unit forms a queue $Q_{l}(\Lambda^*_l)$ with an arrival rate $\lambda^*_l$ and $m_l(\Lambda^*_l)$ assigned servers by Lemma 6.1. By Definition 6.1, $Q_l(\Lambda^*_l)$ could be split into two queues $Q_{(1)}^l(\Lambda^*_l)$ and $Q_{(2)}^l(\Lambda^*_l)$ with the job arrival rates $\lambda^*(\Lambda^*_l)$ and $\lambda^*_2(\Lambda^*_l)$. $Q_{(1)}^l(\Lambda^*_l)$ could be executed on $\lambda^*(\Lambda^*_l)/\lambda_{\text{max}}^{(l)}$ servers while $Q_{(2)}^l(\Lambda^*_l)$ could be executed on $\lfloor \lambda^*_2(\Lambda^*_l)/\lambda_{\text{max}}^{(l)} \rfloor$ server, with their QoS requirements satisfied under the dispatching policy in Section 5.2 by Lemma 5.1. Here, we have $\lambda^*(\Lambda^*_l)/\lambda_{\text{max}}^{(l)} + \lfloor \lambda^*_2(\Lambda^*_l)/\lambda_{\text{max}}^{(l)} \rfloor = m_l(\Lambda^*_l)$ by Definition 6.1. Hence, all $Q_{(1)}^1(\Lambda^*_1), Q_{(2)}^1(\Lambda^*_1), Q_{(1)}^2(\Lambda^*_2), Q_{(2)}^2(\Lambda^*_2)$ define a feasible solution to (15) and the latter is therefore proved.

Secondly, we prove the former. Given an optimal solution to (18), by Lemma 6.2, the accepted jobs of $Q_{l,i}$ could be executed on $\tilde{m}^*_l = \tilde{\lambda}^*_l/\lambda_{\text{max}}^{(l)}$ servers where $\tilde{\lambda}^*_l/\lambda_{\text{max}}^{(l)}$ is an integer; the accepted jobs of $Q_{l,j}$ has a job arrival rate of $\tilde{\lambda}^*_j \in [0, \lambda_{\text{max}}^{(l)}]$ and could be executed on $\tilde{m}^*_j = \lfloor \tilde{\lambda}^*_j/\lambda_{\text{max}}^{(l)} \rfloor$ server. The accepted jobs of both $Q_{l,i}$ and $Q_{l,j}$ could be merged to form a single queue $Q_{l}^*$ with a job arrival rate $\tilde{\lambda}^* + \tilde{\lambda}^*$. This queue could be assigned $\tilde{m}^*_l + \tilde{m}^*_j$ servers, with its QoS requirement satisfied by Lemma 5.1. All queues $Q_{1}^*, \cdots, Q_{L}^*$ define a feasible solution to (15) and the former is therefore proved.

**APPENDIX D**

**Proof of Theorem 6.4**

Algorithm 2 checks each element of a $L$-combination of $\{1, 2, \cdots, K\}$ and computes the corresponding prices of SLAs when $i_1^* = i_1, i_2^* = i_2, \cdots, i_L^* = i_L$, choosing a tuple of prices that achieves the highest revenue. The time complexity of checking each element of a $L$-combination of $\{1, 2, \cdots, K\}$ is $L! (K-L)!$. A cloud provider would provide a finite number of SLAs and $L$ could be bounded by a constant. Hence, Algorithm 2 has a time complexity polynomial in $K$. 

