Future Generation Computer Systems 77 (2017) 97–111

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Capacity planning for IaaS cloud providers offering multiple service classes

Marcus Carvalho [a,b,*], Daniel A. Menascé [c], Francisco Brasileiro [a]

[a] Computer and Systems Department, Universidade Federal de Campina Grande, Campina Grande, PB, Brazil
[b] Department of Exact Sciences, Universidade Federal da Paraíba, Rio Tinto, PB, Brazil
[c] Department of Computer Science, George Mason University, Fairfax, VA, United States

HIGHLIGHTS

- A capacity planning method for IaaS providers with multiple classes.
- Queuing models to estimate admission control behavior with high accuracy.
- Models of demand variance allow meeting most availability and admissibility SLOs.
- The multiclass method combined with admission control can increase utilization.

ARTICLE INFO

Article history:
Received 29 December 2016
Received in revised form 26 April 2017
Accepted 6 July 2017
Available online 26 July 2017

Keywords:
Cloud computing
Capacity planning
Resource management
Performance model
Quality of service

ABSTRACT

Infrastructure as a Service (IaaS) cloud providers typically offer multiple service classes to satisfy users with different requirements and budgets. Cloud providers are faced with the challenge of estimating the minimum resource capacity required to meet Service Level Objectives (SLOs) defined for all service classes. This paper proposes a capacity planning method that is combined with an admission control mechanism to address this challenge. The capacity planning method uses analytical models to estimate the output of a quota-based admission control mechanism and find the minimum capacity required to meet availability SLOs and admission rate targets for all classes. An evaluation using trace-driven simulations shows that our method estimates the best cloud capacity with a mean relative error of 2.5% with respect to the simulation, compared to a 36% relative error achieved by a single-class baseline method that does not consider admission control mechanisms. Moreover, our method exhibited a high SLO fulfillment for both availability and admission rates, and obtained mean CPU utilization over 91%, while the single-class baseline method had values not greater than 78%.

© 2017 Elsevier B.V. All rights reserved.

1. Introduction

Cloud computing is attracting a growing number of users with different budgets and Quality of Service (QoS) requirements for their applications. One of the most popular cloud models is the Infrastructure as a Service (IaaS), which enables users to access computational resources when needed, typically bundled as virtual machines (VMs). In this model, users pay for what they use, instead of acquiring and maintaining their own infrastructure.

Unfortunately, IaaS cloud providers currently offer very limited Service Level Agreements (SLAs), comprised of generic Service Level Objectives (SLOs). For example, Amazon EC2 [1] and Google Compute Engine [2] SLAs currently define a service uptime of 99.95% or higher as the only SLO, with a penalty for the provider (as a financial credit for affected users) if this SLO is violated. As the cloud market becomes more competitive, providers differentiate themselves by offering a wider range of service classes (referred just as "classes" hereafter), with combinations of pricing and SLOs that can attract more users and increase profits [1,3].

However, offering services to satisfy different user requirements while maintaining costs down is challenging, due to non-trivial trade-offs associated with service quality and infrastructure costs. We suggest that part of these trade-offs can be represented in a cloud resource management triangle (see Fig. 1), similarly to the time–cost-performance triangle of project management [4]. In this schema, providers can only choose two of *elasticity*, *capacity*, and *performance* constraints when making decisions for a cloud service, sacrificing the third one. Elasticity allows users to grow and shrink the amount of resources used on-demand; the drawback for

* Correspondence to: Universidade Federal de Campina Grande, Laboratório de Sistemas Distribuídos, Av. Aprígio Veloso, s/n, Bloco CO 58.429-900, Campina Grande, PB, Brazil
E-mail addresses: marcuswac@dcx.ufpb.br (M. Carvalho), menasce@gmu.edu (D.A. Menascé), fubica@dsc.ufcg.edu.br (F. Brasileiro).

http://dx.doi.org/10.1016/j.future.2017.07.019
0167-739X/© 2017 Elsevier B.V. All rights reserved.

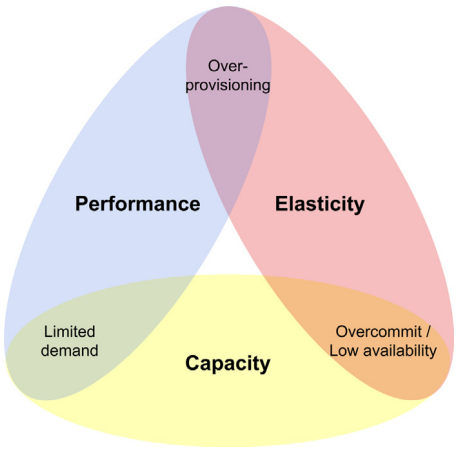


Fig. 1. Cloud resource management triangle. Providers can prioritize only two of these constraints when making decisions for a service.

the provider is the high workload variation it generates. The cloud capacity is the total amount of resources owned and maintained by the provider; it requires efficient planning because it affects the infrastructure costs and the ability to accommodate the demand. The performance of a cloud service is related to the QoS experienced by users, indicating how SLOs for metrics such as availability and response time are defined and fulfilled.

In order to deliver high-quality performance SLOs and also satisfy a flexible demand with high elasticity, providers need to overprovision resources by planning for the peak load, which results in a low average utilization and drives up their infrastructure costs. Providers that prioritize high capacity utilization to reduce costs may have to offer services with low-quality performance SLOs – e.g., lower availability by keeping requests waiting or higher response times caused by overcommitment – and/or limit elasticity by rejecting incoming requests at peak periods in order to fulfill performance SLOs of previously admitted requests.

We believe that IaaS providers can match a variety of user needs in a profitable way by offering multiple classes with different service levels on the elasticity–capacity-performance spectrum. For instance, a provider could offer production class VMs with a high-quality availability SLO and high elasticity, which would result in a low capacity utilization. The excess capacity from this class could be reclaimed by a second batch class that tolerates medium-quality availability SLO and elasticity for lower costs [5]. A third opportunistic class could also be offered to benefit from the spare capacity, but with a low-quality availability SLO and limited elasticity (e.g., Amazon EC2 *spot* [1] and Google Cloud *preemptible* [6] instances). Hence, providers can attract more users and increase their profits by offering multiple classes with different QoS and pricing.

In this context, an important problem for IaaS providers is how to make efficient resource management decisions in order to meet SLOs for different classes with low costs. The inherently high variation of cloud workloads and machine availability (e.g., due to failures and maintenance) makes such decisions very challenging [7][8, Chapter 7]. If resources are not overprovisioned, there may not be enough capacity available to allocate all VM requests at some periods; this would cause VM requests to wait, resulting in lower availability and potential SLO violations. Another challenge is posed when offering multiple classes, as resource management decisions may have different consequences depending on which classes are affected.

In order to reduce availability SLO violations, we advocate that the provider uses a quota-based admission control mechanism that may reject incoming VM requests during peak demand periods. It is assumed that rejecting new requests is less harmful than compromising the performance SLOs for requests already admitted. However, excessive rejections may limit elasticity for users and reduce providers' revenue. To deal with this, we propose that providers can define admission rate targets for different classes and use capacity planning methods combined with the admission control mechanism to find a minimum cloud capacity required to meet these targets. In summary, we argue that capacity planning and admission control decisions should be tackled in an integrated way.

In a previous work [9], we proposed a quota-based admission control mechanism that uses forecasting techniques to predict the future capacity available for each class, and dynamically defines quotas to limit the amount of resources that can be requested for each class. This paper focuses on the capacity planning problem faced by an IaaS cloud provider. It combines our prior results on admission control with a novel capacity planning method for long-term resource management. The paper shows how to make capacity planning decisions, considering the existence of an admission control mechanism, to simultaneously fulfill VM availability SLOs and admission rate targets for multiple classes, aiming to minimize capacity costs. Queuing-theoretic analytical models are used to estimate admission rates and find out the minimum capacity required to meet VM availability SLOs and admission rate targets for each class.

The main contributions of this paper are: (1) a formal definition of the cloud capacity planning problem for multiple classes; (2) a capacity planning method based on analytical models to find the minimum capacity required to meet VM availability and admission rate SLOs for multiple classes; (3) a description of how to use queuing-theoretical analytical models to estimate admission rates obtained by a quota-based admission control mechanism, and answer what-if questions for capacity planning; (4) the evaluation of the proposed method through trace-driven simulations, using data from a large-scale production cloud system, and comparisons with baseline methods.

Our capacity planning method was shown to be very useful in finding the minimum capacity required to meet VM availability and admission rate SLOs for multiple service classes. The results showed that our method: (1) estimated the best cloud capacity with a mean relative error of 2.5% with respect to simulation results, while a single-class baseline method that does not consider admission control mechanisms exhibited a 36% relative mean error; (2) met admission rate targets for all classes; (3) obtained a high CPU utilization with mean values over 91%, compared to a mean value not greater than 73%, obtained by a single-class baseline model without admission control; and (4) achieved high VM availability SLO fulfillment values, close to 100% for the highest priority classes and over 94% for the lowest priority class, while a multiclass baseline method without admission control presented an availability SLO fulfillment as low as 42% for the lowest priority class.

This paper is organized as follows. Section 2 discusses related work. Section 3 presents the IaaS cloud system model and formalizes the problem. Section 4 describes the capacity planning method based on queuing theoretic admission control models. Section 5 describes how we evaluated our capacity planning method through trace-driven simulations and instantiated the admission control models. Section 6 shows the evaluation results of our capacity planning method in comparison to simulation results and baseline methods. Finally, Section 7 presents our concluding remarks and future work.

2. Related work

Although cloud computing resource management has been vastly studied in previous work [10], the problem has been mostly studied from the cloud user's point of view [11–15]. On the contrary, this paper addresses the capacity planning problem from the perspective of an IaaS cloud provider. From this perspective, many strategies have been proposed for the allocation and migration of VMs, aiming to improve cloud utilization and reduce SLO violations [16–19]. Our paper has similar goals but tackles a different problem. We propose a capacity planning model that complements VM allocation and migration methods, and can be combined with these methods to develop a complete cloud resource management solution.

Carvalho et al. [5] proposed a way to reclaim unused cloud resources to offer a new *Economy* class with long-term availability SLOs, by predicting a resource quota to be available for this class in 6-month periods. Subsequently, Carvalho et al. [9] proposed a more general admission control mechanism that handles short-term demand variations to fulfill VM availability SLOs for multiple classes. However, having an inadequate cloud capacity can cause either unnecessary high costs with resource overprovisioning, or a low VM admission rate, which could harm the reputation and profit of cloud providers. In this paper we propose a novel capacity planning method, that takes into account the presence of an admission control mechanism, and allows providers to find out the minimum resource capacity required to meet both availability and admission rate SLOs for multiple classes.

Marshall et al. [20] consider an opportunistic VM class (i.e., with no SLOs) to be combined with a high-quality VM class and improve cloud utilization. Similar best-effort services are offered by public cloud providers, such as Amazon EC2 spot [1] and Google Cloud preemptible [6] instances. Our work not only combines multiple classes to obtain a high resource utilization, but also defines VM availability and admission rate SLOs for each class, showing how to achieve high SLO fulfillment for them.

Unuvar et al. [21] present a stochastic admission control model with resource overcommitment. They fit observed VM usage data to a Beta distribution and reject requests based on the probability of exceeding an utilization threshold. Cherkasova and Phaal [22] propose a predictive admission control for e-commerce applications that adjusts an admission threshold based on the average observed load. Similarly, we also consider a predictive admission control mechanism that defines quotas based on the estimated probability of violating SLOs. However, differently from these studies, our model considers multiple classes and different SLOs. Moreover, we propose a capacity planning method that can be combined with admission control mechanisms so that admission rate targets can also be met in addition to availability SLOs.

Khazaei et al. [23,24] propose an analytical model based on the *M/G/m/m+r* queue to analyze the performance of IaaS clouds. Ghosh et al. [25,26] propose an optimization model based on stochastic machine availability and performance sub-models to plan the capacity of IaaS providers. Their models use Markov chain techniques to approximate system performance metrics. Gmach et al. [27] propose a capacity planning method based on workload demand prediction techniques. In order to find the required capacity to accommodate the workload, they identify demand seasonality and trends by using periodogram and auto-correlation functions, and group similar patterns using *k-means* clustering. However, these models consider a fixed number of identical machines, a single service class and require previous simulations to perform what-if analysis. Moreover, some of them assume Poisson arrival processes and formulas that require factorial calculations of the number of servers that may not be feasible to apply directly for large scale systems. Although our capacity planning method is based on approximations of queuing-theoretical models, we adapted these models to consider heterogeneous physical machines with varying availability, multiple classes, and scenarios with large numbers of VMs and machines, which is essential for large-scale cloud environments. We also consider heterogeneous VM requests and combine our method with a predictive admission control model, which was highlighted by Ghosh et al. [26] as an important area of future research.

Ranaldo and Zimeo [28] perform capacity planning to offer PaaS services in the cloud with the goal of maximizing satisfaction of cloud providers and customers, considering application performance SLAs, available capacity and costs. Their model predicts the utility that can be achieved with new SLAs based on queuing-theoretical performance models. However, their capacity planning method assumes previous knowledge of the workload, which can be infeasible in practice. Moreover, planning is performed for the number of virtual machines allocated to specific customer contracts, and not for the entire cloud. Zhang et al. [29] also propose a method to plan the number of virtual machines needed to fulfill performance requirements for specific cloud applications. Their method is based on workload prediction techniques, queuing-based performance models, and binary search to find the best capacity. In this paper we tackle a different problem, which is the capacity planning of infrastructure cloud providers in long-term periods, for all applications in a cloud. Our capacity planning method considers multiple service classes, and does not require detailed knowledge of future workload, but only general statistics to feed the proposed analytical queuing models.

3. Problem statement

This section describes the cloud model with the notation used in the paper, and formalizes the capacity planning problem for multiple classes.

3.1. System model

This work addresses the resource management of an IaaS cloud *provider* that owns a set of physical *machines*. Each machine has a certain *capacity* for each type of *resource*, such as CPU, memory, and disk. The *nominal cloud capacity* is the total amount of resources aggregated over all machines owned by the provider. The *available cloud capacity* accounts only for machines that are available for users and may vary over time (e.g., due to machine failures and maintenance). Different *instance types* are offered, which represent a combination of the capacities to be allocated for each resource type (e.g., 2 CPU-cores, 8 GB memory, and 32 GB disk), typically bundled as VMs.

Cloud users *request* VMs and their associated instance types to the provider. Fig. 2 shows the possible states of a VM request and the transitions between them. New requests are either *admitted* or *rejected* in the admission control phase. An admitted request moves to the scheduling phase, where it can be either in the *running* or in the *pending* state. A pending request changes to the *running* state when its associated VM is *allocated* to an available machine; this can only happen if the requested VM capacity does not exceed the available capacity of the machine where the VM is allocated. A VM whose associated request is in the running state can be *interrupted* due to machine failures or preemption by higher-priority requests, taking the request back to the pending state. A request moves to the *released* state when the required service time of its associated VM is satisfied.

For simplicity's sake, we only consider CPU capacity (in CPU-cores) when allocating VMs to available machines, because CPU was shown to be the bottleneck resource in cloud workload analysis [5,7]. Nevertheless, other resource types (e.g., memory and disk)

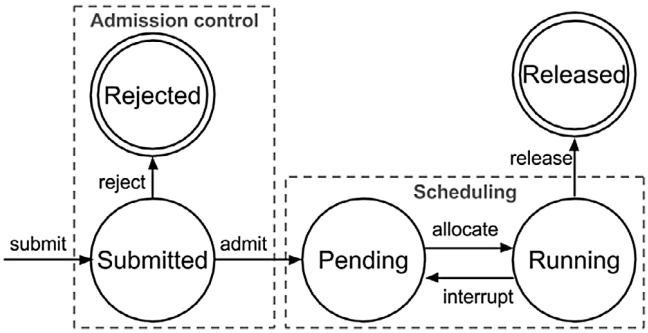


Fig. 2. VM request state diagram in our IaaS cloud system model.

could be considered by allocating a VM only if there is enough capacity for all of its required resource types. We note that networking capacity is important for VM allocation and migration [30]. However, these management activities are not within the scope of this paper. Moreover, planning network fabric can be tackled separately from planning compute and storage server hardware [8, Chapter 3]; this is the case because scaling the network is not only a matter of adding more servers, but planning the entire network topology [31].

A cloud provider can offer multiple *classes*, with different pricing and expected QoS. Each class has an SLA, which is a contract that defines SLOs the provider promises to meet, and penalties the provider pays in case an SLO is violated (i.e., the promise is not fulfilled). We consider the following SLOs for each class:

- *VM Availability SLO*: the minimum VM availability accepted for requests admitted in a class; availability values below the SLO result in SLO violations. We define *VM availability* as the percentage of time a VM is in the running state since its request submission until its release.
- *VM Admissibility SLO*: the minimum VM admission rate accepted for requests submitted in a class; admissibility values below the SLO result in SLO violations. We define *VM admission rate* as the percentage of VM requests admitted by the admission control mechanism.

We adopt a preemptive priority scheduler that allocates VMs of higher-priority classes first, similarly to Google's Borg system [32]. The cloud provider needs to define a priority order for the offered classes. Typically, a class that requires a higher QoS with more rigid SLOs will have a higher-priority assigned. In this model, lower-priority VMs can be preempted if this will free enough resources to allocate higher-priority requests. Therefore, the resource capacity available for a specific class depends not only on the total cloud capacity available, but also on the demand from higher-priority classes—the highest-priority class is the only one for which the available capacity depends solely on the available machines. This way, higher-priority classes will have more capacity available to allocate their VMs.

We consider a quota-based admission control mechanism, in which the provider dynamically defines a *quota* to limit the amount of resources that can be allocated to a class. A new incoming request for a class is rejected if the total capacity already allocated to this class plus the capacity requested exceeds the quota of the class. Similar strategies include the quotas used by Borg [32] and the limits imposed by the Amazon EC2 cloud [33]. Resource quotas are assigned for each class based on the estimated resource capacity to be available for this class and its VM availability SLO. We use predictive quota assignment methods previously proposed, which aim to achieve high request admission rates for all classes while meeting their availability SLOs [9].

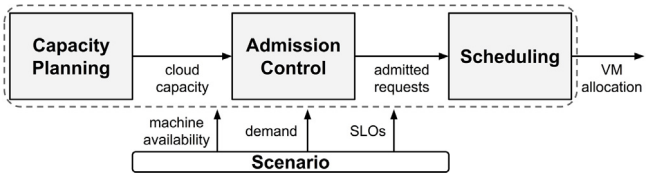


Fig. 3. Part of the cloud resource management process.

The assessment of cloud resource management decisions is made during an *observation period*, which is divided into time slots of finite duration called *epochs* [34]. The beginning of a new epoch occurs when at least one of these events happens: (1) the arrival of new VM requests; (2) the release of running requests; or (3) the change on the available cloud capacity. Thus, epochs can have different durations. Let $E_1, \ldots, E_i, \ldots, E_N$ denote the epochs in an observation period, where N is the number of epochs in that period. Let $b(E_i)$ and $e(E_i)$ be the beginning and end times of epoch i, respectively. Therefore, we define the observation period as the time interval $[b(E_1), e(E_N)]$. Let $\Delta_i = e(E_i) - b(E_i)$ be the duration of epoch i, and $\Delta = e(E_N) - b(E_1)$ be the duration of the observation period in time units.

We use the following notation throughout the paper:

- H: number of physical machines owned by the provider. Let h denote the machine index ($h \in \mathbb{N} : 1 \leq h \leq H$).
- R: number of classes offered by the provider. Let r denote the class index ($r \in \mathbb{N} : 1 \leq r \leq R$), where $r = 1$ represents the highest priority class and $r = R$ the lowest priority class.
- $W_r = \{V_{1r}, \ldots, V_{jr}, \ldots, V_{|W_r|r}\}$: workload stream for class r during the observation period, where V_{jr} denotes the jth VM request for class r.
- D_{jr}: requested resource capacity (i.e., VM size) in CPU-cores for the jth VM request of class r.
- S_{jr}: service time (i.e., required execution time) in time units for the jth VM request of class r.
- C_h: capacity of machine h in CPU-cores. Let $\mathcal{C} = \sum_{h=1}^{H} C_h$ be the nominal cloud capacity.
- $A_{ih} \in \{0, 1\}$: availability of machine h at epoch i; indicates whether the machine is available (1) or unavailable (0). Let $A_h = \sum_{i=1}^{N} A_{ih} \cdot \Delta_i / \Delta$ be the mean availability of machine h.
- $x_{jr} \in \{0, 1\}$: admission of the jth VM request of class r; indicates whether the request is admitted (1) or rejected (0) by the admission control mechanism.
- $y_{ijrh} \in \{0, 1\}$: allocation of the jth VM request of class r in machine h at epoch i; indicates whether the request is allocated in this machine (1) or not (0).
- γ_{jr}: VM availability observed for the jth request of class r. Let γ_r^{min} be the VM availability SLO (i.e., the minimum value accepted) for class r.
- θ_r: VM admission rate observed for class r. Let θ_r^{min} be the admissibility SLO (i.e., the minimum admission rate accepted) for class r.

3.2. Problem formulation

This paper addresses the capacity planning problem, which is part of the resource management process of IaaS cloud providers, as shown in Fig. 3.

In the capacity planning phase, the provider decides the cloud capacity required to execute an expected workload such that the VM admissibility SLO for each class is met. In the admission control phase, the provider decides which requests to reject in order to meet the VM availability SLOs of admitted requests. Finally, in

the scheduling phase, the provider chooses the VMs that will be running at each epoch, and in which machines they will be allocated [35]. Typically, capacity planning is performed offline and for long-term periods (e.g., months), while admission control and scheduling are performed online and very frequently (e.g., every second).

We define resource management decisions as an optimization problem, focusing on this work on the capacity planning phase. The objective is to minimize the nominal cloud capacity, subject to VM availability and admissibility SLOs to be fulfilled for all classes. The decision variables for each phase are the following:

- *Capacity planning*: the set of H machines in the cloud, where each machine h ($1 \leq h \leq H$) has capacity C_h and mean availability A_h.
- *Admission control*: the admission variables x_{jr} for each jth VM request of class r.
- *Scheduling*: the allocation variables y_{ijrh} for each jth VM request of class r in machine h at epoch i.

A *decision scenario* represents the set of input data used to assess a cloud resource management solution, given as:

$$\Psi = \langle N, R, W_r, D_{jr}, S_{jr}, \gamma_r^{min}, \theta_r^{min} \rangle.$$

Therefore, the optimization problem can be formulated as:
Given Ψ,
minimize

$$\mathcal{C} = \sum_{h=1}^{H} C_h, \tag{1}$$

s.t.

$$\sum_{r=1}^{R} \sum_{j=1}^{|W_r|} y_{ijrh} \cdot D_{jr} \leq C_h \cdot A_{ih}, \quad \forall i, h \tag{2}$$

$$y_{ijrh} \leq x_{jr}, \quad \forall i, j, r, h \tag{3}$$

$$\gamma_{jr} \geq \gamma_r^{min}, \quad \forall j, r \tag{4}$$

$$\theta_r \geq \theta_r^{min}, \quad \forall r. \tag{5}$$

The objective function (Eq. (1)) to be minimized is the nominal cloud capacity, which is the sum of the capacities of all machines defined in the capacity planning phase. The capacity constraint (Eq. (2)) states that the capacity allocated for all VMs in a machine must not exceed the machine's capacity, and that unavailable machines must not have any VM allocated. The admission constraint (Eq. (3)) means that a VM request can be allocated if, and only if, it has been admitted in the admission control phase. Finally, the SLA constraints state that VM availability (Eq. (4)) and admission rates (Eq. (5)) must not be lower than the minimum accepted values (i.e., the SLOs) defined for each class.

Unfortunately, solving this optimization problem would require detailed knowledge of future demand and machine availability values by the provider, which is unrealistic. Therefore, in this work we propose a capacity planning method based on queuing-theory approximations that only needs high level workload statistics as input. The capacity planning method is combined with a quota-based admission control mechanism aiming to fulfill both VM availability and admissibility SLOs for all classes offered by the provider.

4. Multiclass capacity planning

This section describes our capacity planning method, that uses analytical models to find the best cloud capacity aiming to meet availability and admissibility SLOs for multiple classes. It also shows how we adapted two queuing-theoretical models to be used by our multiclass capacity planning method.

4.1. Capacity planning method

The proposed capacity planning method is based on a queuing-theoretical model to estimate the admission rates resulted from an admission control mechanism [36]. We use a multi-server finite buffer queue to model admission rates, where customers are rejected when the buffer is full, similarly to a quota-based admission control mechanism that rejects VM requests when the quota is exceeded.

The mapping between the queuing model and our cloud model is the following: (1) the number of servers in the queuing model corresponds to the total number of CPU cores on the cloud machines; (2) the number of customers in the queuing model corresponds to the total number of CPU cores from admitted VM requests either running or pending, and (3) the finite buffer size corresponds to the admission control quota—i.e., the maximum number of CPU cores allowed from admitted VMs either running or pending.

Because the capacity available for lower-priority classes depends on the demand of higher-priority classes, the model cannot be solved independently for each class. Thus, we propose a hierarchical approach that solves the model iteratively from the highest to the lowest priority class.

Our method uses the queuing theoretical analytical model to answer what-if questions, by estimating the admission rate for each class given a nominal cloud capacity, the mean machine availability, the expected demand and the VM availability SLOs for each class. Then, based on this model, the capacity planning method estimates the minimum capacity required to meet both VM availability and admissibility SLOs for all classes.

The basic queuing model input parameters for the what-if analysis are:

- $\mathcal{C}$: nominal cloud capacity in CPU-cores.
- $\hat{A}$: estimated mean availability of physical machines.
- $\hat{\lambda}_r$: estimated mean arrival rate for class r in requested CPU-cores per unit time. Let $\vec{\lambda} = (\hat{\lambda}_1, \ldots, \hat{\lambda}_r, \ldots, \hat{\lambda}_R)$ be the vector of estimated mean arrival rates of all classes.
- $\hat{\mu}_r$: estimated mean service rate for class r in CPU-cores per unit time. Let $\vec{\mu} = (\hat{\mu}_1, \ldots, \hat{\mu}_r, \ldots, \hat{\mu}_R)$ be the vector of estimated mean service rates of all classes.
- $\hat{v}_{a,r}$: squared coefficient of variation of the inter-arrival time for class r. Let $\vec{v}_a = (\hat{v}_{a,1}, \ldots, \hat{v}_{a,r}, \ldots, \hat{v}_{a,R})$ be the vector of estimated coefficients of variation of the request inter-arrival time of all classes.
- $\hat{v}_{s,r}$: squared coefficient of variation of the service time for class r. Let $\vec{v}_s = (\hat{v}_{s,1}, \ldots, \hat{v}_{s,r}, \ldots, \hat{v}_{s,R})$ be the vector of estimated coefficients of variation of the service time of all classes.
- γ_r^{min}: minimum VM availability (i.e., the SLO) defined for class r. Let $\vec{\gamma}^{\,min} = (\gamma_1^{min}, \ldots, \gamma_r^{min}, \ldots, \gamma_R^{min})$ be the vector of VM availability SLOs of all classes.
- θ_r^{min}: minimum VM admissibility (i.e., the SLO) defined for class r. Let $\vec{\theta}^{\,min} = (\theta_1^{min}, \ldots, \theta_r^{min}, \ldots, \theta_R^{min})$ be the vector of VM admissibility SLOs of all classes.

For a given input, the model output is:

- $\hat{\theta}_r$: estimated admission rate for class r when using a quota-based admission control. Let $\vec{\theta} = (\hat{\theta}_1, \ldots, \hat{\theta}_r, \ldots, \hat{\theta}_R)$ be the vector of estimated admission rates of all classes.

Queuing models typically consider a constant number of servers c as input. However, the available cloud capacity may change over time, which yields a changing number of servers. Moreover, the cloud machines and VM sizes are typically heterogeneous, so the mapping of customers to servers is not straightforward. We

simplify this by considering the number of servers c in the queuing system as the mean number of CPU-cores available in the cloud, given by

$$c = \hat{A} \cdot \mathcal{C}. \tag{6}$$

According to the preemptive priority scheduling policy described in Section 3, each class r can have a different class capacity c_r available to allocate its VMs. The *mean available class capacity* for class r is calculated as the mean remaining cloud capacity available after allocating VM requests from higher-priority classes, defined as

$$c_r = c - \sum_{k=1}^{r-1} n_k \tag{7}$$

where n_k is the mean number of k-class customers in the system, defined as the mean capacity requested by k-class VMs in CPU-cores ($n_k \geq 0, \forall k$). Note that the capacity available to a class is never greater than the capacity available to classes of higher priorities ($c_r \leq c_k, \forall r > k$).

We estimate the traffic intensity for class r using Little's law [35] as

$$\rho_r = \frac{\lambda_r}{\mu_r \cdot c_r}, \tag{8}$$

and the mean number of r-class customers in the system as

$$n_r = \frac{\lambda_r}{\mu_r} = c_r \cdot \rho_r. \tag{9}$$

We consider the maximum number of customers allowed in a finite buffer queue as the admission control quota for class r. From our quota-based admission control model [9], we define the maximum number of customers in the system for class r as

$$K_r = \frac{c_r}{\gamma_r^{min}}, \tag{10}$$

where γ_r^{min} is the VM availability SLO for class r.

Based on these input parameters, we estimate the admission rate for requests of class r as the probability of not blocking (rejecting) a request, according to a finite buffer queuing model.

Algorithm 1 shows our capacity planning method. The function is implemented as a simple hill-climbing method to find the minimum capacity required to meet the VM availability and admissibility SLOs for each class, described as follows: starting from a capacity of size zero, it iteratively increments the cloud capacity by ε units at each step and estimates the admission rates for each class in this scenario, until the admission rates are above the target for all classes. Thus, the function returns the minimum capacity with precision ε for which the admissibility SLO of each class is met.

This hill-climbing method was used because, in addition to being simple to implement and understand, it always finds the optimal capacity value with precision ε. This method is especially effective because the admission rate is a monotonically increasing function of the cloud capacity. Moreover, capacity planning is typically performed offline and for long-term periods (e.g., months), thus the performance of the combinatorial search method used is not an important factor. Nevertheless, this method was able to quickly find (in less than 1 min) the optimal value even for the large-scale configurations tested, because the analytical model is solved very fast.

The admission control model to estimate admission rates (line 5 of Algorithm 1) can be implemented using different models. The following sections describe two such queuing models that we adapted to our capacity planning method: a *G/GI/c/K* diffusion approximation model and a *M/M/c/K* model.

Algorithm 1 Capacity planning method.

```
1: function FindBestCapacity(ε, λ⃗, μ⃗, v⃗_a, v⃗_s, γ_r^min, θ_r^min)
2:     c ← 0
3:     repeat
4:         c ← c + ε
5:         θ⃗ ← EstimateAdmissionRates(c, λ⃗, μ⃗, v⃗_a, v⃗_s, γ_r^min)
6:     until θ̂_r ≥ θ_r^min,  ∀ r
7:     return c
8: end function
```

4.2. G/GI/c/K diffusion approximation model

The *G/GI/c/K* queue has a general arrival process, independent and identically distributed service times with a general distribution, multiple (i.e., c) servers, and a finite buffer that allows a maximum number K of customers in the system. Although there is no exact solution for a *G/GI/c/K* queuing system, there are useful approximations for this model. We used the diffusion approximation for the *G/GI/c/K* queue proposed by Whitt [37]. This approximation is based on heavy-traffic limits, which are intended for a large number of servers and high traffic intensity. These assumptions fit well with our problem, since IaaS clouds typically involve a large number of servers and typically exhibit a high system utilization. We summarize here the main formulas of the diffusion approximation used in our method and describe how we adapted the model to our multiclass problem. For more details on this model, please refer to the work of Whitt [37].

The diffusion approximation for the *G/GI/c/K* queue is based on the heavy-traffic limit β given by

$$\beta = \sqrt{c} \cdot (1 - \rho) \quad \text{for } 0 < \beta < \infty, \tag{11}$$

where ρ is the traffic intensity and c the number of servers.

For finite buffer queues, an additional heavy-traffic limit κ is incorporated by letting $K \to \infty$ as $c \to \infty$ so that

$$\kappa = \frac{K - c}{\sqrt{c}} \quad \text{for } 0 < \kappa \leq \infty, \tag{12}$$

where K is the maximum number of customers in the system.

This approximation uses the *asymptotic peakedness* to quantify demand variability. The *peakedness* is the variance divided by the mean of the steady-state number of customers in the system. The peakedness approaches the asymptotic peakedness as the arrival rate increases in the heavy-traffic regime. The asymptotic peakedness z is approximated by Whitt based on a H_2^* service time distribution, given by

$$z = \frac{p \cdot (v_a + v_s)}{2}, \tag{13}$$

where v_a is the squared coefficient of variation (SCV, i.e., variance divided by the square of the mean) of the inter-arrival time; v_s is the SCV of the service time; and the service time distribution H_2^* is a mixture of an exponential distribution with probability p and a unit point mass at 0 with probability $1 - p$.

The average coefficient of variation parameter v is also used in the approximations and is defined as

$$v = \frac{z}{p} = \frac{v_a + v_s}{2}. \tag{14}$$

For simplicity and without loss of generality, we assume that H_2^* follows an exponential distribution, i.e., $p = 1$. Hence, from Eqs. (13) and (14) we have

$$v = z = \frac{v_a + v_s}{2}. \tag{15}$$

The delay probability is the steady-state probability that all servers are busy and an arriving customer must wait in the waiting

line. The asymptotic-delay-probability α is a function of the heavy-traffic limits β and κ obtained from Eqs. (11) and (12), given by

$$\alpha(\beta, \kappa) = [1 + \beta \cdot \Phi(\beta)/(\phi(\beta) \cdot (1 - e^{-\kappa\beta}))]^{-1}, \tag{16}$$

where Φ is the cumulative distribution function (CDF), and ϕ is the probability density function (PDF) of a standard normal random variable.

For the *G/GI/c/K* queue, the delay-probability approximation is given by

$$\tau = \alpha(\beta/\sqrt{z}, p \cdot \kappa/\sqrt{z}), \tag{17}$$

where $\alpha(.)$ is the asymptotic-delay-probability function in Eq. (16).

The blocking probability (i.e., the probability that an arriving request is rejected because it finds K requests in the system) for the *G/GI/c/K* queuing system is approximated by the function

$$\pi(\kappa, \tau, \beta, v, \rho, c) = \frac{f(\kappa, \tau, \beta, v) \cdot v}{\rho \cdot \sqrt{c}}, \tag{18}$$

where $f(.)$ is the PDF of the steady-state queue length for the diffusion process evaluated at the upper boundary κ, defined as

$$f(\kappa, \tau, \beta, v) = \frac{\tau \cdot \beta \cdot e^{-\kappa\beta/v}}{v \cdot (1 - e^{-\kappa\beta/v})}. \tag{19}$$

Finally, based on the probability of not blocking (rejecting) a request in the *G/GI/c/K* queue, we estimate the request admission rate for class r as

$$\hat{\theta}_r = 1 - \pi(\kappa_r, \tau_r, \beta_r, v_r, \rho_r, c_r), \tag{20}$$

where $\pi(.)$ is the blocking probability approximation function for the *G/GI/c/K* queue in Eq. (18).

Algorithm 2 presents the function that implements the *G/GI/c/K* model, which estimates the admission rates $\vec{\theta}$ for each class r when using a quota-based admission control. It iterates over each class r, from the highest priority ($r = 1$) to the lowest priority ($r = R$) class, to estimate the admission rates of each class for a cloud capacity size given as input. This algorithm can be used by our capacity planning method to estimate admission rates and find the best cloud capacity to meet admissibility SLOs.

Algorithm 2 *G/GI/c/K* model to estimate admission rates for each class.

function EstimateAdmissionRates($c, \vec{\lambda}, \vec{\mu}, \vec{v}_a, \vec{v}_s, \gamma_r^{min}$)
  $p \leftarrow 1$
  $c_1 \leftarrow c$
  for $r \leftarrow 1, R$ **do**
    $\rho_r \leftarrow \lambda_r/(\mu_r \times c_r)$ ▷ Eq. 8
    $n_r \leftarrow c_r \times \rho_r$ ▷ Eq. 9
    $K_r \leftarrow c_r/\gamma_r^{min}$ ▷ Eq. 10
    $\beta_r \leftarrow \sqrt{c_r}\,(1 - \rho_r)$ ▷ Eq. 11
    $\kappa_r \leftarrow (K_r - c_r)/\sqrt{c_r}$ ▷ Eq. 12
    $v_r \leftarrow (\hat{v}_{a,r} + \hat{v}_{s,r})/2$ ▷ Eq. 14
    $z_r \leftarrow p \times v_r$ ▷ Eq. 13
    $\tau_r \leftarrow \alpha(\beta_r/\sqrt{z_r}, p * \kappa_r/\sqrt{z_r})$ ▷ Eq. 17
    $\hat{\theta}_r \leftarrow 1 - \pi(\kappa_r, \tau_r, \beta_r, v_r, \rho_r, c_r)$ ▷ Eq. 20
    $c_{r+1} \leftarrow c_r - n_r$ ▷ Eq. 7
  end for
  return $\vec{\theta}$
end function

4.3. M/M/c/K model

The *M/M/c/K* queue assumes that the inter-arrival times and the service times are exponentially distributed; there are multiple (c) servers to process requests; and there is a finite buffer with size K. This queuing model can be solved as a Markov process, as described by Kleinrock [36] and adapted to our multiclass problem as follows.

Based on the traffic intensity ρ calculated for each class r as shown in Eq. (8), we can calculate the probability of having n requests in the system as

$$P_n(\rho, c, K) = \begin{cases} \dfrac{(c\rho)^n}{n!} P_0(\rho, c, K) & \text{if } 1 \le n < c \\ \dfrac{\rho^n c^c}{c!} P_0(\rho, c, K) & \text{if } c \le n \le K, \end{cases} \tag{21}$$

and the probability of having no requests in the system can be obtained by observing that $\sum_{n=0}^{K} P_n = 1$:

$$P_0(\rho, c, K) = \left[1 + \frac{(1 - \rho^{K-c+1})(c\rho)^c}{c!(1 - \rho)} + \sum_{n=1}^{c-1} \frac{(c\rho)^n}{n!} \right]^{-1}. \tag{22}$$

The blocking probability for the *M/M/c/K* model is calculated as the probability that all servers are busy—i.e., considering $n = K$ in Eq. (21). Thus, by using as input the number of servers c_r available for class r (Eq. (7)), the traffic intensity ρ_r for class r (Eq. (8)), and the buffer size (i.e., the admission control quota) K_r for class r (Eq. (10)), we can estimate the admission rate for each class r as

$$\hat{\theta}_r = 1 - P_K(\rho_r, c_r, K_r). \tag{23}$$

Algorithm 3 shows the function that implements the *M/M/c/K* model that estimates the admission rates $\vec{\theta}$ for each class r when using a quota-based admission control. It also iterates over each class, from the highest to the lowest priority class, to estimate the admission rates of each class for a cloud capacity size given as input.

Algorithm 3 *M/M/c/K* model to estimate admission rates for each class.

function EstimateAdmissionRates($c, \vec{\lambda}, \vec{\mu}, \vec{v}_a, \vec{v}_s, \gamma_r^{min}$)
  $c_1 \leftarrow c$
  for $r \leftarrow 1, R$ **do**
    $\rho_r \leftarrow \lambda_r/(\mu_r \times c_r)$ ▷ Eq. 8
    $n_r \leftarrow c_r \times \rho_r$ ▷ Eq. 9
    $K_r \leftarrow c_r/\gamma_r^{min}$ ▷ Eq. 10
    $\hat{\theta}_r \leftarrow 1 - P_K(\rho_r, c_r, K_r)$ ▷ Eq. 23
    $c_{r+1} \leftarrow c_r - n_r$ ▷ Eq. 7
  end for
  return $\vec{\theta}$
end function

5. Evaluation methodology

This section describes how we set up the simulation environment and scenarios to evaluate our capacity planning method using the *G/GI/c/K* and the *M/M/c/K* queuing models.

5.1. Simulation environment

We evaluate our capacity planning method by comparing its output with simulated results. We developed a cloud resource management simulator that implements the methods described in this paper. More details on the simulator and its validation are presented in Appendix.

We use trace-driven simulations fed by workload traces from production systems. In the absence of newer, publicly available data, we use traces from a Google's cluster that have been made publicly available a few years ago [38]. These traces span 29 days in May 2011 from a cluster with over 12 thousand machines, and comprise over 25 million task submissions (see the works of Reiss et al. [7] and Abdul-Rahman and Aida [39] for analyses of these

traces). We see no reason for the characteristics of Google's workload to have substantially changed since then. Thus, given the large number, diversity and maturity of applications running in Google's clusters, we believe these workload traces are still relevant and representative of cloud environments [40].

The resource capacity data in the trace is normalized by the size of the largest machine in the cluster, which was not released; although we do not have absolute resource capacity values, we are still able to calculate relative values of requested capacity over the machines' capacities, as they are normalized by the same constant factor. This relative resource capacity information is sufficient for our evaluation.

We consider each task in the trace as a VM request. This is how we map attributes from the traces to simulation data:

- *VM request submission time* ($b(V_{jr})$) $\rightarrow$ submission time of each task.
- *VM class* (r) $\rightarrow$ priority range associated to each task. Based on the trace description [38], we assign three priority groups: "prod" to tasks with highest priorities ($9 \leq priority \leq 11$); "batch" to tasks with middle priorities ($2 \leq priority \leq 8$); and "free" to tasks with lowest priorities ($0 \leq priority \leq 1$).
- *VM requested capacity* (D_{jr}) $\rightarrow$ requested CPU capacity for each task. For tasks that update this value over time, we consider the maximum value observed during the task's lifetime.
- *VM service time* (S_{jr}) $\rightarrow$ total time each task was in the *running* state in time units.
- *Machine capacity* (C_h) $\rightarrow$ CPU capacity of each physical machine in CPU-cores.
- *Machine availability* (A_{ih}) $\rightarrow$ actual time periods that each physical machine was available.

Because task and machine availability events occur very frequently in the traces, we aggregate events in fixed 5-min intervals. This was done to reduce simulation time and make it feasible to evaluate the complete trace in many different scenarios. Thus, admission control and scheduling decisions are made at this time granularity, which makes every epoch i to have the same size ($\Delta_i = 5$ min, $\forall i$). If in the traces a machine had become unavailable at any time within a 5-min epoch period, we consider this machine was unavailable during the whole epoch. VM request submission events within an epoch are anticipated to the beginning of the same epoch, and VM request release events are delayed to the beginning of the following epoch.

We use a preemptive priority scheduling policy, as described in Section 3.1. The total capacity allocated to VMs cannot be greater than the available cloud capacity—i.e., there is no resource overbooking. The scheduler allocates VM requests based on their priorities. Requests within the same priority are allocated in a *First-Come First-Served* basis. We also use an aggressive backfilling approach [41], on which requests are skipped in the scheduling queue if they do not fit in the current class available capacity, so smaller requests with lower priorities can be allocated instead. Although skipped requests become pending (and unavailable), they are reconsidered by the scheduler at every next epoch, being able to preempt lower-priority requests to be allocated when possible.

We claim in this work that capacity planning and admission control methods should be performed in an integrated way. At each new epoch, the admission control mechanism can adjust the quota defined for each class and reject incoming requests if the corresponding quota is exceeded. As we focus on a capacity planning method based on the output of a quota-based admission control mechanism, the VM placement phase is simplified in the simulations (i.e., VMs are not assigned to particular machines). Therefore, we consider the cloud provider has a single large machine (i.e., $H = 1$), so C_h is the total cloud capacity defined in the capacity planning phase and A_{i1} is the fraction of the total cloud capacity available at epoch i. This simplification hides the resource fragmentation that can happen when scheduling in multiple machines. The model can be improved in future work to consider a fragmentation factor to compensate wasted capacity due to fragmentation.

As admission control mechanism we used our *pred-ets* predictive heuristic, which according to our previous work exhibited the best prediction results [9]. Any admission control heuristic can be used; however, capacity planning requires an analytical model to estimate the admission control behavior, as we did in this work. The *pred-ets* heuristic uses the Exponential Smoothing (ETS) forecasting method to assign a quota to each class. The ETS method combines models based on the weighted averages of input values, with the weights exponentially decaying for older values. It has three components: Error correction (E), Trend (T), and Seasonal (S). Each component can be of a certain type: None, Additive, Multiplicative, and other variations. The different combinations of component types result in different ETS methods, where each method has a set of parameters to be estimated [42, chap. 7]. Similarly to our previous work [9], we used the ETS implementation from the R *forecasting* package [43]. We used a confidence level of 95% to calculate prediction confidence intervals when defining quotas—this can be adjusted by providers for more/less conservative predictions [5].

5.2. Metrics and scenarios

The capacity planning method is assessed for the following metrics:

- *Admission rate*: percentage of requests of class r admitted by the admission control mechanism, given by

$$\theta_r = \frac{\sum_{j=1}^{|W_r|} x_{jr}}{|W_r|} \cdot 100\%. \tag{24}$$

- *Best cloud capacity*: minimum resource capacity required to fulfill admissibility SLOs for all classes. A smaller capacity would violate admissibility SLOs, while greater capacities would increase infrastructure costs unnecessarily.
- *Availability SLO fulfillment*: percentage of admitted requests of class r for which the observed VM availability was equal or greater than the VM availability SLO defined for the class.
- *Mean cloud utilization*: mean percentage of the available cloud capacity actually allocated for VMs.
- *Absolute error*: absolute difference between observed and estimated value for a metric, calculated as

$$|x - \hat{x}| \tag{25}$$

  where x is the observed (simulated) value and $\hat{x}$ is the value estimated by the model.
- *Relative error*: absolute error divided by the observed value, calculated as

$$\frac{|x - \hat{x}|}{x}. \tag{26}$$

We evaluate different cloud scenarios by changing the following input parameters:

- *Capacity size factor*: a multiplicative factor applied to the capacity of each machine extracted from the original traces—e.g., a capacity size factor of 1.1 means that the cloud has 10% more total capacity than the original value in the traces.

Table 1
Input parameters' values and ranges for the simulation scenarios.

Input parameter	Values
Capacity size factor	[0.6, 1.3]
Load factor	[0.5, 2]
VM availability SLO	$\langle prod = 100\%, batch = 90\%, free = 50\% \rangle$
Admissibility SLO	$\langle prod = 99.9\%, batch = 99\%, free = 0\% \rangle$

- *Load factor*: a multiplicative factor applied to the requested resource capacity of each VM request extracted from the original traces—e.g., a load factor of 1.1 means that each submitted VM request requests 10% more capacity than the original value in the traces, increasing the overall offered load by 10%.

We evaluate the capacity planning methods for different combinations of capacity size factors and load factors. The availability and admissibility SLOs for each class are fixed in all scenarios. Table 1 shows all input parameter values and ranges used in the scenarios evaluated.

5.3. Instantiation of analytical admission control models

Our capacity planning method described in Section 4 depends on an admission control model to estimate the admission rate for each class in a given scenario. Thus, we instantiate the *G/GI/c/K* and *M/M/c/K* admission control models by implementing Algorithms 2 and 3, respectively. The input values required for these models are obtained as follows.

The base mean cloud capacity c used as input is obtained from the traces, and a capacity size factor is applied to this base value to explore different capacity scenarios. For planning the cloud capacity, the parameter c is the method output instead of input, as shown in Algorithm 1, and denotes the best capacity estimated by the model for a given precision value ε.

The mean arrival rate λ_r for each class r is obtained from the traces by averaging for each class the number of CPU cores requested at each 5-min interval. We explore different demand scenarios by applying *load factors* to the base arrival rates obtained from the traces.

The mean VM service times μ_r are not easily obtained from the traces because many VMs were either submitted before the beginning of the trace or released after its end. This happens mostly to the *prod* class (26% of requests), which makes the exact service time mean statistic for these VMs unavailable. Because discarding these VMs would result in underestimated service times, we use Little's law [35] to estimate the mean service time for class r based on the mean number of customers and arrival rates, given as

$$\hat{\mu}_r = \frac{n_r^{\infty}}{\lambda_r}, \tag{27}$$

where n_r^{∞} is the mean number of r-class customers observed in the system. This value is obtained from simulations that consider an infinite number of servers; this way, there is no waiting time at the queue and the mean response time is equal to the mean service time.

The squared coefficient of variation for the inter-arrival times ($v_{a,r}2$) and service times ($v_{s,r}$) for each class r, required as input to the *G/GI/c/K* model, are also obtained from the traces by dividing the sample variation by the squared mean of each metric [37].

The VM availability SLOs (γ_r^{min}) and admissibility SLOs (θ_r^{min}) for each class r used in the capacity planning evaluation are defined in Table 1. We do not define an admissibility SLO for the *free* class when planning the cloud capacity because we consider this an opportunistic class.

As mentioned above, some VM requests are submitted before the beginning of the trace and are not released until after its end; we call them *permanent VMs*. The total capacity requested by permanent VMs is significant for the *prod* class (24% of the total capacity requested), but not for *batch* and *free* classes (<1%). Permanent VMs are assumed to be known, so we consider that the capacity available to arriving non-permanent VM requests is the unused capacity by the permanent VMs. The admission rate metric is estimated for non-permanent VMs only. When planning the cloud capacity, we first discard permanent VMs to estimate the best capacity for non-permanent VMs only, then we add to this value the capacity requested by permanent VMs to obtain an estimate for all VMs.

Although we are dealing with thousands of servers, the *G/GI/c/K* diffusion approximation model is well suited for such large scale systems and can be applied directly even with big numbers. On the other hand, the *M/M/c/K* model is unfeasible to be applied directly in large scale systems, because it requires factorial calculations over the number of servers c, as shown in Eq. (21). To deal with this, we normalize both the cloud capacity and the VM requested capacity by a factor before applying this model, in a way that the maximum cloud capacity is 100 and we are able to calculate its factorial value to plan the cloud capacity. Rounding the number of servers and customers in the system from thousands to values up to one hundred can affect the accuracy of the model, but the results were shown satisfactory even with the simplification for this model.

5.4. Baseline capacity planning methods

We compare our multiclass capacity planning method instantiated with the *G/GI/c/K* and *M/M/c/K* admission control models with other capacity planning methods, described here and used as baseline, that do not use admission control.

5.4.1. M/M/c model without admission control

This model considers inter-arrival times and service times exponentially distributed, multiple c servers, and an infinite waiting queue—i.e., it does not consider an admission control mechanism [36, pp. 102–105]. We estimate the mean VM availability as the probability of having the number of customers in the system n less than or equal to the number of servers c (i.e., $P[n \leq c]$). Thus, we apply a hill climbing method to find the lowest cloud capacity c such that there is no requests waiting in the queue (i.e., $P[n \leq c] = 100\%$).

5.4.2. M/M/c multiclass model without admission control

This model is similar to the *M/M/c* model (i.e., it does not consider admission control) but was adapted to deal with multiple classes and their different VM availability SLOs. Similarly to the *G/GI/c/K* and *M/M/c/K* models, we also apply this method hierarchically from the highest to the lowest priority class. We estimate the mean VM availability for class r as the probability of having the number of customers in the system n_r less than or equal to the number of servers c_r (i.e., $\alpha_r = P[n_r \leq c_r]$). Thus, we apply a hill climbing method to find the lowest cloud capacity c such that the VM availability SLOs are met for all classes r (i.e., $\forall r : P[n_r \leq c_r] \geq \alpha_r^{min}$).

6. Results

This section presents the evaluation results by comparing metrics from the trace-driven simulations with the values estimated by our method.

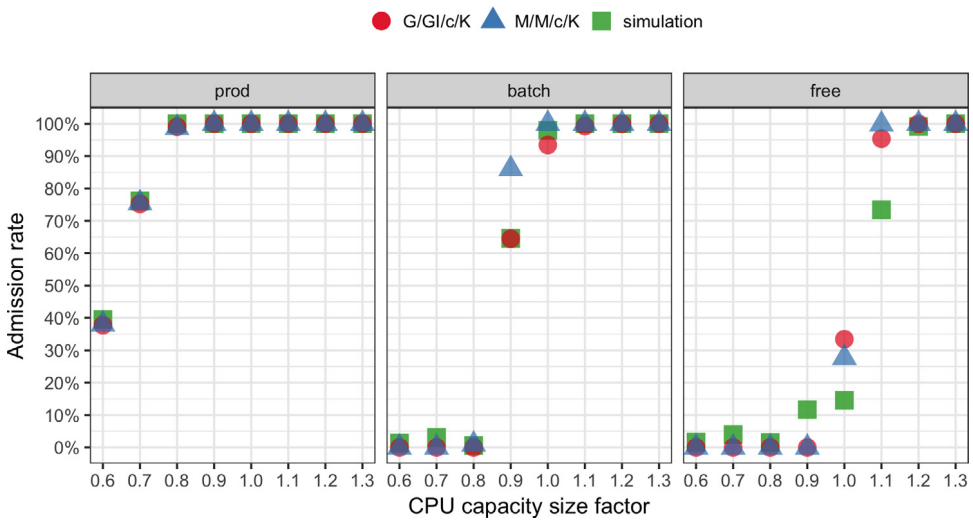


Fig. 4. Admission rates obtained from simulations compared with values estimated by the capacity planning model for different cloud capacity sizes and classes.

6.1. Validation of analytical models

We first perform a what-if analysis to validate the analytical queuing models, showing how well they estimate admission rates for each class when varying the cloud capacity and offered load.

Fig. 4 compares the admission rates obtained from simulations with the values estimated by our capacity planning method using the *G/GI/c/K* and *M/M/c/K* models, for different cloud capacity sizes and classes. The load factor is fixed at 1. Both models presented good accuracy for the *prod* class, but the maximum absolute error was slightly higher for the *G/GI/c/K* model (1.8%) than for the *M/M/c/K* model (1.5%). For the lower-priority classes, the *G/GI/c/K* model presented more accurate results than the *M/M/c/K* model, with significant differences on the admission rate estimation errors. For the *batch* class, the *G/GI/c/K* and *M/M/c/K* models had maximum absolute errors of 4.5% and 21.4%, respectively, and for the *free* class the maximum absolute errors were 21.9% and 26.4%, respectively.

Note that model estimation absolute errors tends to be higher for lower-priority classes. This happens because the capacity available for lower-priority classes also depends on the demand for higher-priority classes, adding more levels of uncertainty and variation as the priority decreases, which makes it harder to capture in the model. We believe this is not a critical issue because violating SLOs for lower-priority classes usually has a lower impact on a provider's revenue. Nevertheless, we suggest using a safety margin when planning the capacity for lower-priority classes for reducing the risk of SLO violations for them.

Fig. 5 compares the admission rates obtained from simulations with the values estimated by our capacity planning method using the *G/GI/c/K* and *M/M/c/K* models for different offered loads and classes. The capacity size factor is fixed at 1. Similarly to the what-if analysis for the capacity size, both models had a good accuracy for the *prod* class, but the *G/GI/c/K* presented smaller errors for lower-priority classes. Note that the model estimation errors also tend to be higher for lower-priority classes. Moreover, we can see for lower-priority classes that the admission rate values have a sharper decrease when increasing the offered load. This happens because a higher load not only increases the demand for these classes but also decreases the capacity available for them, as the demand for higher-priority classes also increases, causing a higher impact on their admission rates.

6.2. Capacity planning method evaluation

This subsection presents the evaluation results for our capacity planning method. We instantiate the *G/GI/c/K* and *M/M/c/K* models to apply our capacity planning method, estimating the best capacity required to fulfill VM availability and admissibility SLOs for all classes in different demand scenarios. We compare our capacity planning method results with results from baseline methods and the actual values found through simulations.

We simulated different combinations of load factors and capacity sizes to find the actual best capacity for each demand scenario—i.e., the minimum capacity required to fulfill availability and admissibility SLOs according to simulations. For each load scenario, we vary the capacity size factor starting from 0 with increments of 0.1 until the SLOs are met for all classes. A smaller increment would result in a better precision, but it would require more simulations to find the best capacity and would become very time consuming.

We also applied our capacity planning method to estimate the best cloud capacity for the different load factors and compare with the simulation results. We used the same increment size of 0.1 in our hill-climbing approach, in order to match the same precision as the simulations. We also applied the baseline methods for comparison. The closer the best capacities estimated by a method are from the simulation results, the better this method is considered.

Fig. 6 compares the best capacity sizes obtained from simulations with the values estimated by the different capacity planning methods when varying the cloud offered load. We see that the best capacity size found in simulations grows linearly as the offered load increases. Our multiclass capacity planning method based on admission control, using both *G/GI/c/K* and *M/M/c/K* models, is able to capture well this linear behavior for the scenarios evaluated. The *M/M/c/K* model tends to plan capacities smaller than the best ones, having a maximum absolute error of 0.2 and a mean relative error of 13%; this underprovisioning happens because this model assumes exponential distributions for interarrival and service times while the actual distributions exhibits higher variances. Because the *G/GI/c/K* model assumes a generic distributions for interarrival and service times and incorporates the coefficients of variation in the calculations, this model exhibited the best results having a maximum absolute error of 0.1 on the capacity factor (i.e., the minimum precision in our evaluation) and a mean relative error of 2.5%. The *M/M/c* model plans capacities for the cloud much higher than the best ones, having a maximum absolute error of 0.9 and a mean relative error of 36%; this highlights the fact

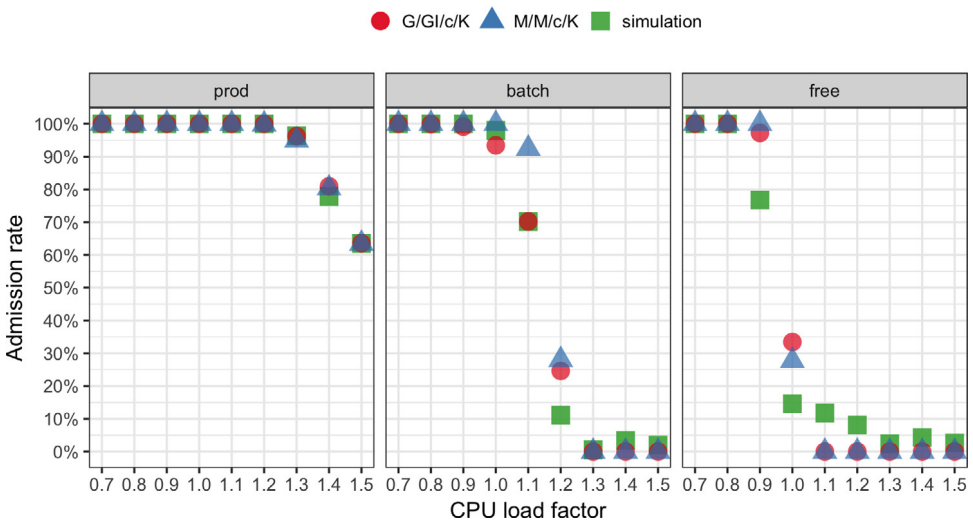


Fig. 5. Admission rates obtained from simulation compared with values estimated by the capacity planning model for different offered loads and classes.

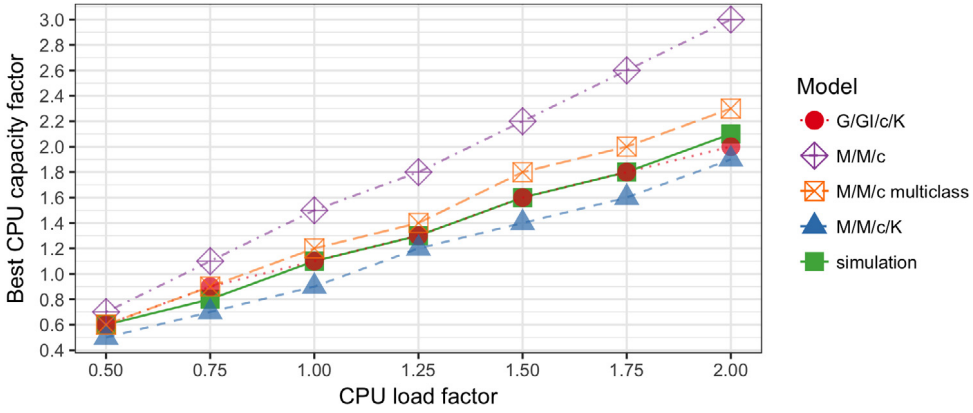


Fig. 6. Minimum capacity required to fulfill SLOs obtained from simulations compared to the values estimated by the capacity planning model.

that providers can significantly reduce infrastructure costs when considering multiple classes and using a quota-based admission control mechanism. By adding multiclass capabilities, the *M/M/c multiclass* model improved the results of the basic *M/M/c* model; however, it still does unnecessary overprovisioning because it does not use an admission control mechanism, resulting in a maximum absolute error of 0.2 in the capacity factor and a mean relative error of 8.9%. Notice that for small load factors, the capacities planned for all methods do not seem significantly different; this happens because the increment of 0.1 when searching the best capacity in the simulation and analytical methods is large when compared to a smaller offered load, giving a low precision for these scenarios.

Fig. 7 presents the admission rates obtained by our capacity planning method using the models that use a quota-based admission control mechanism. The horizontal dashed lines represent the admission rate targets for each class. For the *G/GI/c/K* model, the admissibility SLOs were fulfilled in all scenarios except for the *batch* class when the CPU load factor was 2.0; in this case, the *batch* admission rate was 98.3% while the admission rate target for this class was 99%. For the *M/M/c/K* model, the admissibility SLOs were violated for the *batch* class in all scenarios, exhibiting in the worst case an admission rate of 68.6% while the admission rate target for this class was 99%. Note that for the *G/GI/c/K* model, the admission rate for the *free* class tends to decrease when increasing the CPU load factor. This happens because when the CPU load factor increases, the mean CPU arrival rate also increases but the coefficient of variation decreases. With a lower coefficient of variation, the *G/GI/c/K* model uses a smaller safety margin to meet *prod* and *batch* SLOs, leaving less spare space to admit opportunistic requests from the *free* class. Because different coefficients of variation do not affect the *M/M/c/K* model, the admission rate for the *free* class does not change significantly for this model when varying the CPU load factor. Even though both models may exhibit low admission rates for the *free* class in some scenarios, it does not harm admissibility SLOs because *free* is a best effort class and does not have an admission rate target.

Fig. 8 shows the mean CPU utilization observed when using different capacity planning methods and varying the CPU load. The *M/M/c/K* model achieved the highest mean CPU utilization, having values of ≈98% in all scenarios; although a high utilization is good for reducing costs, in this case it generated side effects such as the admissibility SLO violations discussed previously. The *G/GI/c/K* model had mean CPU utilization values ranging from 91% to 98%; differently from the *M/M/c/K* model, the *G/GI/c/K* model could achieve high CPU utilizations while meeting admissibility SLOs in most scenarios. Note that for the *G/GI/c/K* model, the utilization tends to increase for higher load factors; this also happens because the coefficient of variation decreases and a smaller safety margin is needed to accommodate the requests and meet SLOs. The *M/M/c multiclass* model also exhibited a high mean CPU utilization ranging from 91% to 98%, because it does not use an admission control mechanism and tends to accept more requests. The *M/M/c* model exhibited the lowest mean CPU utilization, with values ranging

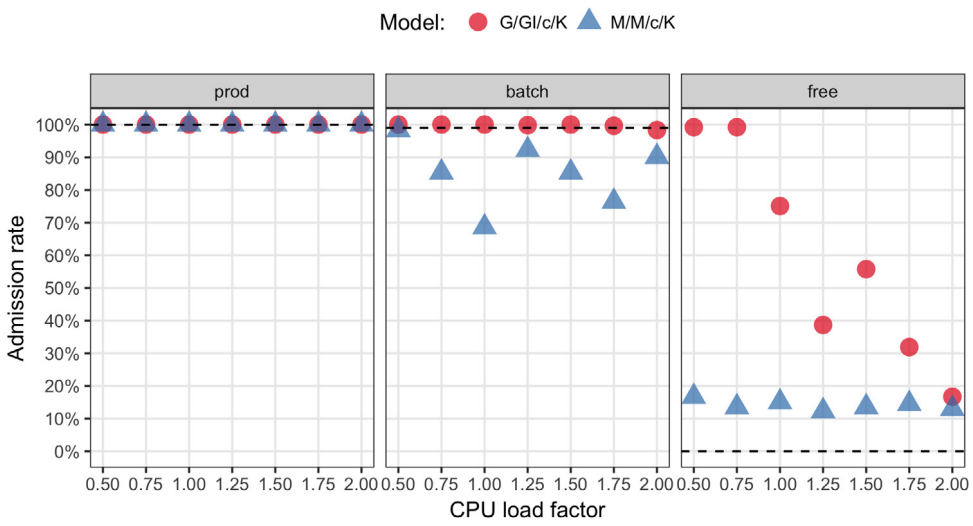


Fig. 7. Admission rates obtained by different capacity planning methods for different CPU loads factors and classes.

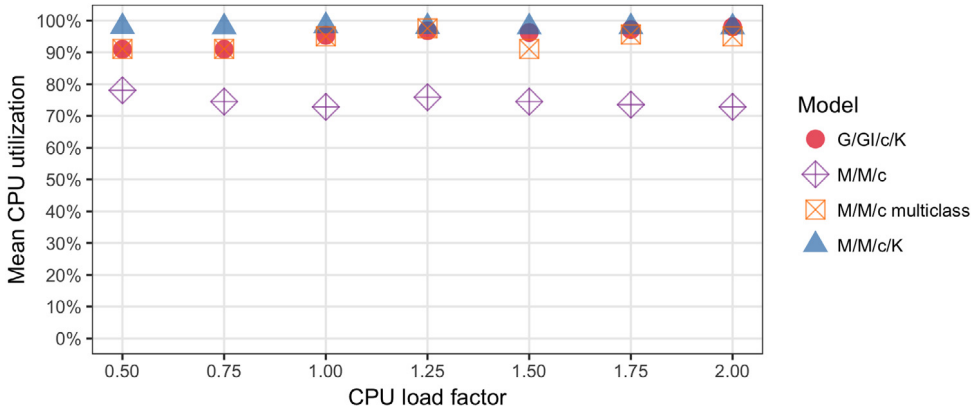


Fig. 8. Mean CPU utilization exhibited by different capacity planning methods for different CPU loads.

from 73% to 78%; this happens because this method does not differ the multiple classes and aims at a 100% availability SLO for all classes, thus unnecessarily overprovisioning resources and driving up infrastructure costs.

Fig. 9 shows the availability SLO fulfillment obtained by the different capacity planning methods for different classes when varying the CPU load. For the *prod* and *batch* classes, all methods exhibited SLO fulfillment greater than 99%, except the *M/M/c/K* model that fulfilled SLOs for 96% of *batch* requests in the worst case scenario. The *M/M/c multiclass* method had a low SLO fulfillment of 42% for the *free* class in the worst case, showing that this model that tries to meet availability SLOs only through capacity planning and not using admission control mechanisms, is not efficient in meeting VM availability SLOs for all classes. The basic *M/M/c* model could meet availability SLOs for all classes even without performing admission control; on the other hand, this model increases the infrastructure costs by overprovisioning resources, as shown previously in the utilization analysis. The *G/GI/c/K* and *M/M/c/K* models applied to our capacity planning method exhibited for the *free* class a worst-case availability SLO fulfillment of 94% and 96%, respectively; although we believe these SLO fulfillment values are good enough for an opportunistic class, a provider can improve them by defining a higher prediction confidence level and increasing the admission control safety margins, having as consequence a lower admission rate for this class [5].

Although the proposed capacity planning method is based on queuing-theory models that consider assumptions which may not necessarily hold in practice, we see that the method is quite robust and useful to perform what-if analysis, estimate the admission rates for different cloud scenarios, and plan the capacity required to meet VM availability and admissibility SLOs for multiple classes. Similar observations about the robustness of queuing models were made by Jeff Buzen in the 1970's when he developed the theory of operational analysis and derived the same queuing results obtained through stochastic analysis but under far less strict assumptions [44].

7. Concluding remarks and future work

IaaS cloud providers can offer a wide range of service classes to increase their profits, by attracting users with different QoS requirements and budgets. Therefore, an important problem for providers is how to minimize their capacity costs while meeting SLOs defined for the different classes. Admission control mechanisms were shown efficient on meeting availability SLOs for multiple classes [9]. However, such mechanisms alone can result either in low admission rates or resource overprovisioning, which can reduce providers' profits. This paper showed how cloud providers can make capacity planning decisions to address this problem, so that they can also meet admission rate targets for multiple classes with an adequate resource provisioning. We proposed a capacity planning method based on queuing-theory models that estimates the behavior of a quota-based admission control mechanism to

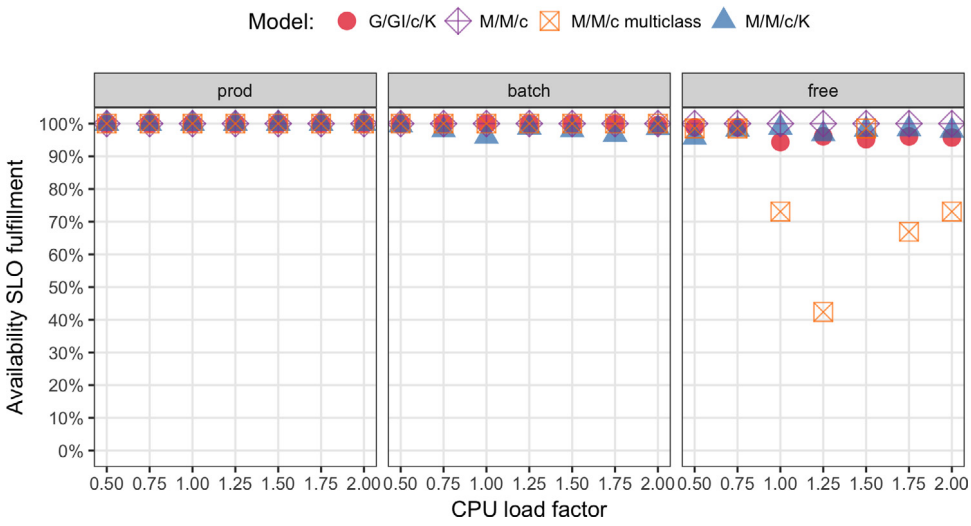


Fig. 9. VM availability SLO fulfillment achieved by different capacity planning methods for different CPU loads factors and classes.

find the minimum capacity required to meet both VM availability and admissibility SLOs for multiple classes.

Our analytical capacity planning method was shown to be very useful at finding the minimum capacity required to meet SLOs by capturing well the admission control mechanism behavior observed in simulations for different cloud scenarios. When applying our capacity planning method with a *G/GI/c/K* queuing-theoretical admission control model, we were able to: (1) estimate the best cloud capacity with a mean relative error of 2.5% with respect to simulation results, while a *M/M/c* baseline method exhibited a 36% mean relative error; (2) meet the admission rate targets defined for all classes, while a *M/M/c/K* model violated admissibility SLOs in all scenarios; (3) obtain a high CPU utilization with mean values over 91% while a *M/M/c* baseline model exhibited mean values up to 73%; (4) achieve a high VM availability SLO fulfillment with values close to 100% for the *prod* and *batch* classes and 94% for the lowest priority *free* class, while the *M/M/c multiclass* model exhibited SLO fulfillment as low as 42% for the *free* class. These results highlight that combining the capacity planning method with a quota-based admission control and considering multiple classes can significantly reduce the cloud capacity when compared to baseline methods that do not consider admission control, thus increasing utilization and reducing infrastructure costs while meeting VM availability and admissibility SLOs.

For future work, we plan to: evaluate the capacity planning and admission control methods for multiple resource dimensions (e.g., considering memory and disk in addition to CPU); improve the capacity planning model to deal with multiple heterogeneous physical machines and consider fragmentation in the VM scheduling; and allow resource overbooking when making resource management decisions by examining resource usage over time.

Acknowledgments

Marcus Carvalho thanks the support from CNPq/Brazil, the Computer Science Department at George Mason University (GMU) and its C4I center for hosting him as a visiting scholar. Francisco Brasileiro is a CNPq/Brazil researcher (grant 311297/2014-5). The work of Daniel Menascé is partially supported by the AFOSR grant FA9550-16-1-0030. The experiments were run on ARGO, a research computing cluster provided by the Office of Research Computing at GMU (http://orc.gmu.edu).

Appendix. Simulator validation

In our evaluation, we used a cloud resource management simulator developed by our research group, namely *cloudrm-sim*. We implemented the different admission control and scheduling techniques as described in our evaluation methodology (Section 5). The simulator is an open-source project publicly available.[1]

To validate the simulator, we compared the following metrics obtained with the simulator, with results obtained by a well-known analytical queuing model.

- *Mean admission rate*: the percentage of requests admitted for a class.
- *Mean number of customers in the system*: the mean number of VM requests for a class, over time, that are either pending or running in the system.

The system behavior is estimated using the M/M/c/K analytical queuing model presented in Section 4.3, which considers Poisson arrival rates, exponential service times, multiple servers, and a limited buffer size (i.e., a resource quota). The admission rate metric is estimated for each class based on the M/M/c/K queuing model using Eq. (23). The mean number of customers in the system for each class r is estimated based on the probability distribution function of this metric for the M/M/c/K queuing model, given in Eq. (21).

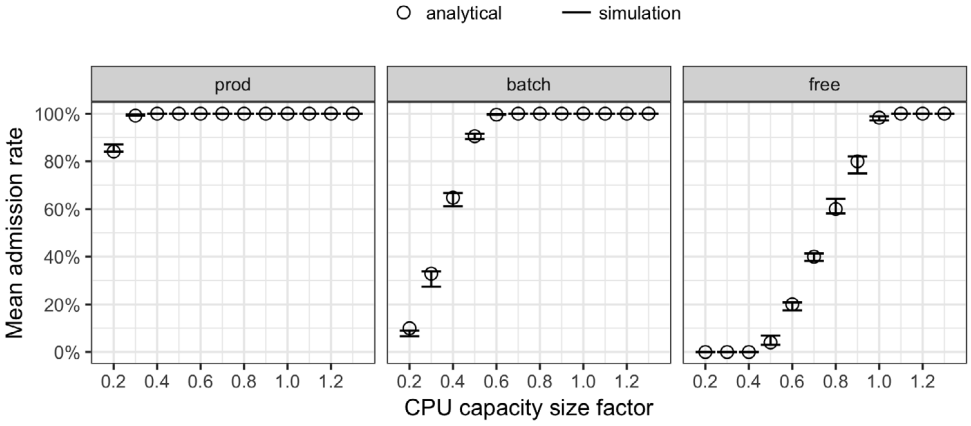
The simulator is fed with synthetic workloads generated with the same Markovian assumptions used by the M/M/c/K queuing model to estimate the system behavior. This synthetic workload generation assumes Poisson arrival rates, exponential service times, and a constant CPU requested capacity of 1 CPU-core for all VM requests ($D_{jr} = 1,\ \forall j, r$). We consider the same three priority classes presented in Section 5; each class has specific arrival rate and service time distributions. We also consider a fixed quota K_r for each class r, which is calculated iteratively from the highest to the lowest priority class, using Eq. (10). Table A.1 shows the values and ranges for the input parameters used in the simulator validation scenarios.

We explore different capacity planning decisions to validate the simulation behavior in different resource contention scenarios. For that, we apply a CPU capacity factor to a base CPU capacity, performing a parameter sweep for this factor parameter. The

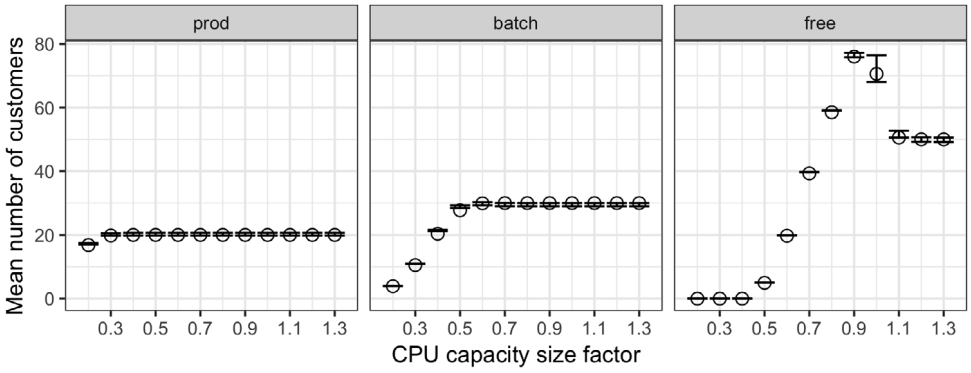
[1] https://github.com/marcuswac/cloudrm-sim

Table A.1
Input parameters' values and ranges for the simulator validation scenarios.

Input parameter	Values
Base CPU capacity	100
CPU capacity factor	[0.2, 1.3]
VM availability SLO	$\langle prod = 100\%, batch = 90\%, free = 50\% \rangle$
Mean arrival rate (requests per minute)	$\langle prod = 20, batch = 50, free = 25 \rangle$
Mean service time (minutes)	$\langle prod = 50, batch = 15, free = 50 \rangle$



(a) Mean admission rate.



(b) Mean number of customers.

Fig. A.1. Results estimated by the analytical queuing model compared with confidence intervals calculated from simulation results.

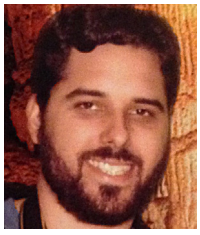
simulation for each scenario is executed for 30 simulation days, considering that each epoch has a fixed size of 5 min ($\Delta_i = 5$ min, $\forall i$), similarly to Section 5. For each simulation day, we calculate the mean values for the output metrics used in the validation, and confidence intervals for the mean at a 95% confidence level.

Fig. A.1 compares analytical and simulation results. The analytical model estimates that the mean admission rates for all classes must increase as the CPU capacity increases, until reaching 100% of VM requests admitted (see Fig. A.1a). The confidence intervals for the mean admission rates generated by simulations closely match the expected analytical results, which validates the correct behavior of the multi-class admission control simulation. Moreover, the analytical model estimates that, in most scenarios, the mean number of customers in the system must increase as the CPU capacity increases, until it stabilizes when there is enough capacity to admit all requests, and there are no requests waiting in the queue (see Fig. A.1b); this increase on the mean number of customers is related to the increase on admission rates discussed previously. However, the number of customers in the system for the *free* class presents a different behavior as the CPU capacity is increased; this happens because the *free* class has a lower availability SLO, allowing more requests to be waiting in the queue for longer periods. Therefore, as the CPU capacity is increased, the mean number of customers for the *free* class first increases, because more requests are being admitted and kept in the waiting queue, then it decreases, when there is enough capacity to admit all requests, but some requests are still waiting in the queue; and finally, it stabilizes, when all requests are admitted and do not wait to be allocated. We observe that the confidence intervals generated by simulations are also matching the expected analytical results for the mean number of customers in the system, which provides evidences of the correctness of the simulator.

References

[1] Amazon EC2 - Instance Purchasing Options, 2016. https://aws.amazon.com/ec2/pricing.
[2] Google Compute Engine - Service Level Agreement (SLA), 2015. https://cloud.google.com/compute/sla.
[3] M. Al-Roomi, S. Al-Ebrahim, S. Buqrais, I. Ahmad, Cloud computing pricing models: a survey, Int. J. Grid Distrib. Comput. 6 (5) (2013) 93–106.
[4] H. Kerzner, Project management: A systems approach to planning, scheduling, and controlling, eleventh ed, Wiley, 2013.
[5] M. Carvalho, W. Cirne, F. Brasileiro, J. Wilkes, Long-term SLOs for reclaimed cloud computing resources, in: ACM Symposium on Cloud Computing, SoCC, 2014, pp. 20:1–20:13.

[6] Google Compute Engine - Preemptible Instances, 2015. https://cloud.google.com/compute/docs/instances/preemptible.
[7] C. Reiss, A. Tumanov, G.R. Ganger, R.H. Katz, M.A. Kozuch, Heterogeneity and dynamicity of clouds at scale: Google trace analysis, in: ACM Symposium on Cloud Computing, SoCC, 2012, pp. 7:1–7:13.
[8] L.A. Barroso, J. Clidaras, U. Hölzle, The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, second ed, in: Synthesis Lectures on Computer Architecture, Morgan & Claypool Publishers, 2013.
[9] M. Carvalho, D. Menascé, F. Brasileiro, Prediction-based admission control for IaaS clouds with multiple service classes, in: IEEE International Conference on Cloud Computing Technology and Science, CloudCom, 2015, pp. 82–90.
[10] B. Jennings, R. Stadler, Resource management in clouds: Survey and research challenges, J. Netw. Syst. Manag. (JNSM) 23 (3) (2015) 567–619.
[11] M.D. de Assunção, C.H. Cardonha, M.A. Netto, R.L. Cunha, Impact of user patience on auto-scaling resource capacity for cloud services, Future Gener. Comput. Syst. (FGCS) 55 (2016) 41–50.
[12] J. Espadas, A. Molina, G. Jiménez, M. Molina, R. Ramírez, D. Concha, A tenant-based resource allocation model for scaling Software-as-a-Service applications over cloud computing infrastructures, Future Gener. Comput. Syst. (FGCS) 29 (1) (2013) 273–286.
[13] P.D. Maciel, F. Brasileiro, R.A. Santos, D. Candeia, R. Lopes, M. Carvalho, R. Miceli, N. Andrade, M. Mowbray, Business-driven short-term management of a hybrid IT infrastructure, J. Parallel Distrib. Comput. (JPDC) 72 (2) (2012) 106–119.
[14] Z. Shen, S. Subbiah, X. Gu, J. Wilkes, Cloudscale: elastic resource scaling for multi-tenant cloud systems, in: ACM Symposium on Cloud Computing, SoCC, 2011, pp. 5:1–5:14.
[15] Z. Gong, X. Gu, J. Wilkes, Press: Predictive elastic resource scaling for cloud systems, in: IEEE International Conference on Network and Service Management, CNSM, 2010, pp. 9–16.
[16] A. Aldhalaan, D. Menascé, Autonomic allocation of communicating virtual machines in hierarchical cloud data centers, in: International Conference on Cloud and Autonomic Computing, ICCAC, 2014, pp. 161–171.
[17] M. Kesavan, I. Ahmad, O. Krieger, R. Soundararajan, A. Gavrilovska, K. Schwan, Practical compute capacity management for virtualized datacenters,, IEEE Trans. Cloud Comput. (TCC) (2013).
[18] E. Casalicchio, D.A. Menascé, A. Aldhalaan, Autonomic resource provisioning in cloud systems with availability goals, in: ACM Cloud and Autonomic Computing Conference, CAC, 2013, pp. 1:1–1:10.
[19] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, D. Pendarakis, Efficient resource provisioning in compute clouds via VM multiplexing, in: IEEE/ACM International Conference on Autonomic Computing and Communications, ICAC, 2010, pp. 11–20.
[20] P. Marshall, K. Keahey, T. Freeman, Improving utilization of infrastructure clouds, in: IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2011, pp. 205–214.
[21] M. Unuvar, Y. Doganata, A. Tantawi, M. Steinder, Cloud overbooking through stochastic admission controller, in: International Conference on Network and Service Management, CNSM, 2014, pp. 320–323.
[22] L. Cherkasova, P. Phaal, Session-based admission control: A mechanism for peak load management of commercial web sites, IEEE Trans. Comput. (TC) 51 (6) (2002) 669–685.
[23] H. Khazaei, J. Misic, V. Misic, Performance of cloud centers with high degree of virtualization under batch task arrivals, IEEE Trans. Parallel Distrib. Syst. 24 (12) (2013) 2429–2438.
[24] H. Khazaei, J. Misic, V.B. Misic, Performance analysis of cloud computing centers using M/G/m/m+r queuing systems, IEEE Trans. Parallel Distrib. Syst. (TPDS) 23 (5) (2012) 936–943.
[25] R. Ghosh, F. Longo, R. Xia, V.K. Naik, K.S. Trivedi, Stochastic model driven capacity planning for an infrastructure-as-a-service cloud, IEEE Trans. Serv. Comput. (TSC) 7 (4) (2014) 667–680.
[26] R. Ghosh, F. Longo, V.K. Naik, K.S. Trivedi, Modeling and performance analysis of large scale iaas clouds, Future Gener. Comput. Syst. (FGCS) 29 (5) (2013) 1216–1234.
[27] D. Gmach, J. Rolia, L. Cherkasova, A. Kemper, Capacity management and demand prediction for next generation data centers, in: IEEE International Conference on Web Services, ICWS, 2007, pp. 43–50.
[28] N. Ranaldo, E. Zimeo, Capacity-driven utility model for service level agreement negotiation of cloud services, Future Gener. Comput. Syst. (FCGS) 55 (2016) 186–199.
[29] Q. Zhang, H. Chen, Y. Shen, S. Ma, H. Lu, Optimization of virtual resource management for cloud applications to cope with traffic burst, Future Gener. Comput. Syst. (FGCS) 58 (2016) 42–55.
[30] M.A. Sharkh, M. Jammal, A. Shami, A.H. Ouda, Resource allocation in a network-based cloud computing environment: design challenges, IEEE Commun. Mag. 51 (11) (2013) 46–52.
[31] A. Vahdat, M. Al-Fares, N. Farrington, R.N. Mysore, G. Porter, S. Radhakrishnan, Scale-out networking in the data center, IEEE Micro 30 (4) (2010) 29–41.
[32] A. Verma, L. Pedrosa, M.R. Korupolu, D. Oppenheimer, E. Tune, J. Wilkes, Large-scale cluster management at Google with Borg, in: European Conference on Computer Systems, EuroSys, 2015, pp. 18:1–18:17.
[33] Amazon EC2 FAQ, 2016. http://aws.amazon.com/ec2/faqs.
[34] D. Menascé, S. Bardhan, Epochs: Trace-Driven Analytical Modeling of Job Execution Times, Tech. Rep., George Mason University, 2014.
[35] D. Menasce, L. Dowdy, V.A. Almeida, Performance by Design: Computer Capacity Planning By Example, Prentice Hall, 2004.
[36] L. Kleinrock, Queueing Systems: Theory, John Wiley & Sons, 1975.
[37] W. Whitt, A diffusion approximation for the G/GI/n/m queue, Oper. Res. 52 (2004) 922–941.
[38] C. Reiss, J. Wilkes, J.L. Hellerstein, Google cluster-usage traces: format + schema, Google Inc., 2011. Posted at URL http://code.google.com/p/googleclusterdata/wiki/TraceVersion2.
[39] O.A. Abdul-Rahman, K. Aida, Towards understanding the usage behavior of Google cloud users: the mice and elephants phenomenon, in: IEEE International Conference on Cloud Computing Technology and Science, CloudCom, 2014, pp. 272–277.
[40] W. Cirne, E. Frachtenberg, Web-scale job scheduling, in: Workshop on Job Scheduling Strategies for Parallel Processing, Springer, 2012, pp. 1–15.
[41] D. Feitelson, A. Weil, Utilization and predictability in scheduling the IBM SP2 with backfilling, in: Merged Intl. Parallel Processing Symp. and Symp. Parallel and Distributed Processing, 1998, pp. 542–546.
[42] R.J. Hyndman, G. Athanasopoulos, Forecast: Principles and Practice, OTexts.com, 2014.
[43] R. Hyndman, Y. Khandakar, Automatic time series forecasting: The forecast Package for R, J. Stat. Softw. 27 (3) (2008) 1–22.
[44] J. Buzen, Fundamental operational laws of computer system performance, Acta Inform. 7 (2) (1976) 167–182.



Marcus Carvalho is an Assistant Professor at the Federal University of Paraíba, Brazil. He received his D.Sc., M.Sc. and B.S. degrees in Computer Science at the Federal University of Campina Grande, Brazil, in 2016, 2011 and 2008, respectively. He was a visiting scholar at the University of British Columbia, Canada, in 2011, a software engineering intern at Google, USA, in 2013, and a visiting scholar at George Mason University, USA, in 2015. His research interests include cloud computing, computer systems performance analysis and modeling, and data analytics.

Daniel A. Menascé is a University Professor of Computer Science and was the Senior Associate Dean of the Volgenau School of Engineering at George Mason University, Fairfax, VA, USA, for seven years. He received a Ph.D. in CS from the University of California at Los Angeles. He is a Fellow of the ACM and of the IEEE and a recipient of the 2001 A.A. Michelson Award from the Computer Measurement Group. He published over 250 papers and five books and his research interests include autonomic computing, cloud computing, analytic modeling and analysis of computer systems, and software performance engineering.

Francisco Brasileiro is a Full Professor at the Federal University of Campina Grande, Brazil. He received B.S. (1988) and M.Sc. (1989) degrees in Computer Science from the Federal University of Paraíba, Brazil, and a Ph.D. degree (1995) in Computer Science from the University of Newcastle upon Tyne, UK. His research interests are in distributed systems in general, with focus on federated and cooperative systems. He has been the Principal Investigator of over 20 projects, funded by both governmental agencies, and industry. He is a member of the Brazilian Computer Society, the ACM, and the IEEE Computer Society.