# Revenue Maximization with Optimal Capacity Control in Infrastructure as a Service Cloud Markets

Adel Nadjaran Toosi, *Member, IEEE*, Kurt Vanmechelen, *Member, IEEE*, Kotagiri Ramamohanarao, and Rajkumar Buyya, *Fellow, IEEE*

**Abstract**—Infrastructure-as-a-Service cloud providers offer diverse purchasing options and pricing plans, namely on-demand, reservation, and spot market plans. This allows them to efficiently target a variety of customer groups with distinct preferences and to generate more revenue accordingly. An important consequence of this diversification however, is that it introduces a non-trivial optimization problem related to the allocation of the provider's available data center capacity to each pricing plan. The complexity of the problem follows from the different levels of revenue generated per unit of capacity sold, and the different commitments consumers and providers make when resources are allocated under a given plan. In this work, we address a novel problem of maximizing revenue through an optimization of capacity allocation to each pricing plan by means of admission control for reservation contracts, in a setting where aforementioned plans are jointly offered to customers. We devise both an optimal algorithm based on a stochastic dynamic programming formulation and two heuristics that trade-off optimality and computational complexity. Our evaluation, which relies on an adaptation of a large-scale real-world workload trace of Google, shows that our algorithms can significantly increase revenue compared to an allocation without capacity control given that sufficient resource contention is present in the system. In addition, we show that our heuristics effectively allow for online decision making and quantify the revenue loss caused by the assumptions made to render the optimization problem tractable.

## 1 Introduction

In the infrastructure as a service (IaaS) model of cloud computing, customers purchase units of computing time on virtual machine (VM) instances in a flexible *pay-as-you-go* manner through the Internet [1]. Cloud providers maintain large-scale data centers to offer these computational resources on-demand and at a relatively low cost thanks to the associated economies of scale. Yet, to ensure business success, they need to obtain the highest possible revenue from selling available resource capacity. Methods such as adopting differentiated pricing plans, market segmentation [2], and demand forecasting [3] can be used so that the maximal amount of capacity is sold at the highest possible price.

As the computational resources involved constitute a non-storable or perishable commodity [4], cloud providers benefit from maximizing resource utilization to maximize revenue. Consequently, many IaaS providers offer various pricing plans (or markets) such as *reservation (subscription)* and *spot market*-based plans, in addition to an *on-demand pay-as-you-go* pricing plan. In the reservation plan, users pay an upfront reservation fee to reserve resources for a specific period of time (e.g., one year). In exchange, they receive a significant discount on the hourly resource usage price. The spot market allows users to submit the maximum price they are willing to pay to an auction-like mechanism as a bid. Users gain access to the acquired resources as long as their bid exceeds the provider's last computed market clearing price, which also determines the resource's uniform usage price until the next market clearing.

The segmentation of demand through different pricing plans is attractive to the provider for a number of reasons. For instance, risk-free income from reservations, on the one hand, provides guaranteed cash flow through long-term commitments. As such, it can compensate for the demand uncertainty associated with the on-demand pay-as-you-go pricing plan [2]. The spot market, on the other hand, can attract price-sensitive users that can tolerate service interruptions, allowing the provider to generate additional revenue from spare capacity without exposure to the risks of overbooking capacity.

The use of multiple pricing plans introduces a number of non-trivial trade-offs to IaaS providers with respect to revenue maximization. Although on-demand pay-as-you-go requests often generate the highest revenue per hour of usage sold, they suffer from future demand uncertainty. The upfront fee of the reservation plan is beneficial from a cash flow perspective, but in the long-run, the total revenue generated is lower than the one obtained by selling equivalent usage hours under an on-demand plan. Moreover, the provider is liable to offer guaranteed availability for reserved requests to honor the associated service level agreement (SLA). This might be costly when customers do not fully utilize their reserved capacity in the reservation's lifetime [4].

- *A. N. Toosi, K. Ramamohanarao, and R. Buyya are with Department of Computing and Information System, University of Melbourne, Parkville Campus, Melbourne, VIC 3010, Australia. E-mail: adeln@pgrad.unimelb.edu.au, {kotagiri, rbuyya}@unimelb.edu.au.*
- *K. Vanmechelen is with Department of Mathematics and Computer Science, University of Antwerp, Belgium. E-mail: kurt.vanmechelen@ua.ac.be.*

*Manuscript received 26 May 2014; revised 28 Oct. 2014; accepted 25 Nov. 2014. Date of publication 17 Dec. 2014; date of current version 4 Sept. 2015. Recommended for acceptance by B. He and B. Veeravalli.*
*For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.*
*Digital Object Identifier no. 10.1109/TCC.2014.2382119*

Allocating this underutilized reserved capacity to on-demand requests (i.e., overbooking resources), creates the risk of SLA violations. Spot instances on the other hand, can be terminated by the provider whenever their resources are required to honor commitments made with respect to other pricing plans. The provider therefore has the freedom to accommodate spot instances in the underutilized reserved capacity of the data center. Consequently, the benefits of this flexibility from a revenue management perspective must be considered when admitting new reservation contracts.

We address the problem of maximizing revenue when these three pricing models are jointly applied. Our main research question is the following: with limited resources available, and considering the dynamic and stochastic nature of customers' demand, how can expected revenue be maximized through an optimal allocation of capacity to each pricing plan?

We frame our algorithmic contributions within a *revenue management* framework that supports the three presented pricing plans and that incorporates an admission control system for requests of the reservation plan. To the best of our knowledge, we are the first to consider a joint offering of on-demand pay-as-you-go, reservation, and spot markets in a revenue maximization problem for IaaS cloud providers.

In summary, our *main contributions* are:

- We formulate the optimal capacity control problem that results in the maximization of revenue as a finite horizon *Markov decision process (MDP)* [5]. We present a stochastic dynamic programming technique to compute the maximum number of reservation contracts the provider is to accept from the arriving demand in order to maximize revenue.
- For a provider with large capacity, the use of the stochastic dynamic programming technique is computationally prohibitive. We, therefore, present two algorithms to increase the scalability of our solution. The first increases the spatial and temporal granularity of the problem in order to solve it in a time suitable for practical online decision making. The second sacrifices accuracy to an acceptable extent through a number of simplifying assumptions on the utilization of reserved capacity and the lifetime of on-demand requests.
- We evaluate our proposed framework through large-scale simulations, driven by cluster-usage traces provided by Google. We propose a scheduling algorithm that generates VM requests based on the demand captured in these traces. Using pricing conditions that are aligned with those of Amazon EC2[1], we demonstrate that our admission control algorithms substantially increase revenue for the provider.

This paper is organized as follows: after reviewing the related work in Section 2, we introduce the system model in Section 3. Therein we discuss the cloud pricing models used in this paper, the optimal revenue management problem, and a stochastic dynamic programming technique to tackle the problem. We propose two admission control algorithms namely *pseudo optimal* and *heuristic* in Section 4. Section 5 focuses on the revenue management framework and its high-level architecture. The performance evaluation of the framework and a comparison between the admission control algorithms is presented in Section 6. Our conclusions and future work are presented in Section 7.

1. http://aws.amazon.com/ec2/pricing

## 2 Related Work

Revenue management (*also known as yield management*) is the process of maximizing revenue from a fixed capacity for perishable resources using market segmentation and demand management techniques. During the last few decades, revenue management has witnessed significant scientific and practical advances especially in the airline and hotel industries. As the literature on the topic is vast, we focus on its relevant applications to cloud computing. Interested readers can find a detailed overview of revenue management in [6].

An early attempt to incorporate revenue management into cloud computing by Püschel and Neumann [7] investigates techniques such as client classification and dynamic pricing in a policy-based admission control model. Similar work has been done by Meinl et al. [8] who applies derivative markets and yield management techniques for revenue maximization.

Macías et al. [9] investigate dynamic pricing, over-provisioning, and selective SLA violation to maximize cloud provider revenue. Recently, Kashef et al. [10] propose a system architecture for cloud service providers that combines demand-based pricing with resource provisioning. They compare two revenue management techniques for cloud computing. The first sets the timing for offering price discounts, whereas the second determines the number of VMs that should be offered at full price.

Anandasivam et al. [11] utilize a *bid-price control* technique that originates from the revenue management literature for capacity management which accepts or denies incoming requests for service in order to increase revenue. Bid-price control is an accepted and efficient method in airline revenue management in which threshold values, also called bid prices, are set for each leg of an itinerary and a ticket is sold if its fare exceeds the sum of the bid prices along the path. Their model considers multiple resources such as CPU, memory, storage, and bandwidth, while our model comprises bundles of resources, i.e., VM instances.

The main difference between these works and ours is that none of them considers the joint adoption of the multiple different pricing plans presented in this paper. As a result they are not applicable for many cloud providers that are currently offering different pricing plans.

In our model, the provider is faced with stochastic and dynamic arrivals and departures of customer requests and must decide on whether to admit an incoming reservation contract or to reject it. Similarly, Nair and Bapna [12] introduced a revenue management technique based on the admission control for the application domain of an Internet service provider (ISP). They formulate the problem as a continuous time Markov decision process over an infinite planning horizon to dedicate ISP capacity to customers at any instant of time. Despite these similarities, their application domain differs from ours and their approach cannot be directly applied in the cloud context. Interested readers can find an approximate analytical model for performance analysis of cloud data centers in [13].

Mazzucco and Dumas [14] examine the problem of allocating servers to two classes of customers, *premium* and *basic*, in a revenue maximizing way. The authors rely on a queuing model to tackle the optimization problem. Their work differs from ours since they target platform as a service providers; therefore their assumptions, pricing plans, and experimental settings differ.

There is a large body of research devoted to minimizing cost for cloud consumers when multiple pricing models are offered, see for example [15], [16], [17], [18]. However, limited investigation has been done on resource allocation and capacity planning techniques to maximize provider revenue. The problem of dynamically allocating resources to different spot markets for revenue maximization has been investigated by Zhang et al. [3]. Supply adjustment and dynamic pricing are used as a means to maximize revenue and meet customer demand. They model the problem as a constrained discrete-time and finite-horizon optimal control problem and adopt *model predictive control* (MPC) techniques to design the dynamic algorithm solution. MPC is a widely used industrial technique for advanced multivariable control in the presence of nonlinearities and uncertainties. The study does not consider the coexistence of multiple markets, focusing solely on the spot market. Deciding on the optimal capacity segmentation for on-demand and spot market requests has been formulated as a Markov decision process by Wang et al. [2]. As a part of their work, they propose an optimal mechanism for a spot market based on a uniform price auction. In their model, they only consider on-demand pay-as-you-go and spot market requests and assume that reservation contracts are always fulfilled.

Xu and Li [4] present an infinite horizon stochastic dynamic program to maximize revenue under stochastic demand arrivals and departures. They focus on the spot market and do not consider the joint offering of multiple pricing plans. Similarly, Truong-Huu and Tham [19] formulate the competition among cloud providers as a non-cooperative stochastic game which is modeled as a Markov decision process to maximize revenue. At each step of the game, providers dynamically propose optimal price policies with regard to the current policies of their competitors. According to providers' price policies, customers will decide on which provider to submit their requests. Authors also introduce a novel approach for the cooperation among providers to enhance revenue and acquire the needed resources at any given time. Both studies rely on dynamic pricing as the main technique to maximize revenue, whereas our work focuses on capacity management and admission control without imposing any particular dynamic pricing policies.

## 3 System Model

In this section, we review common cloud pricing plans, and formulate the optimal capacity control technique with a revenue maximization objective. Fig. 1 schematically illustrates our system model.

### 3.1 Cloud Pricing Plans

#### 3.1.1 On-Demand Pay-As-You-Go Plan

This plan charges customers for compute capacity based on actual usage, without requiring any contractual long-term commitments. The service is charged for at a fixed rate $p$ per billing cycle (e.g., hourly) from the time the VM instance is launched until it is terminated. Customers can retain an instance for an indefinite time. A request for a new on-demand instance can be denied if the provider has insufficient resources available. Note that $p$ is fixed at most IaaS providers for a long period of time (i.e., months to years), and can therefore be viewed as a constant value. Moreover, the one-hour billing cycle, selected based on the Amazon EC2 billing period, can be replaced with any other billing period, e.g., per-minute or per-day billing cycle without any specific change to our model.

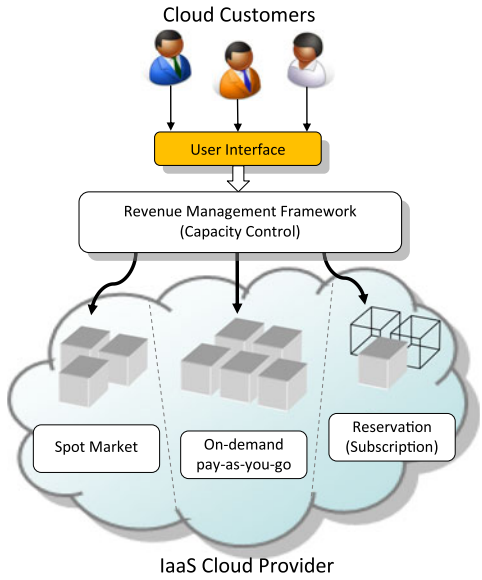


Fig. 1. Schematic system model for the capacity control problem.

#### 3.1.2 Reservation Plan

This plan allows customers to reserve an instance for a certain *reservation period* (e.g., months or years) and assures that the reserved capacity is available whenever it is required in that period. During the reservation period, the reservation is said to be *live*.

The customer pays an upfront reservation fee of $\varphi$, after which the usage is either free (e.g., as in GoGrid[2]) or heavily discounted (e.g., Amazon EC2). The one-time fee must be paid irrespective of how much the instance is used during the reservation period. The total amount of instance hours consumed by a single customer account are aggregated per billing cycle and then automatically matched to any reserved capacity contracts the customer has in its portfolio.

Let $\alpha \in [0, 1]$ be the discount factor on the on-demand plan's rate $p$ that is obtained when reserving a given instance type. A total of $h$ hours of usage in the reservation period then costs $\varphi + \alpha ph$. For example, in Amazon EC2, a premium of \$61 reserves an `m1.small` instance (Linux, US East, Light Utilization) for 1 year, resulting in a \$0.034 per hour usage price within the reservation period compared to the on-demand hourly rate of \$0.060 ($\alpha = 0.57$), see also Table 1. Partial utilization of the reserved capacity can still

2. GoGrid, http://www.gogrid.com/.

TABLE 1
Pricing of the On-Demand, Reserved and Spot Standard Small Instances (`m1.small`, Linux, us-east) in Amazon EC2

| Pricing Plan | Upfront | Hourly rate |
|---|---|---|
| **On-demand** | \$0 | \$0.060 |
| **1-year Reserved (light Utilization)** | \$61 | \$0.034 |
| **1-year Reserved (Medium Utilization)** | \$139 | \$0.021 |
| **1-year Reserved (Heavy Utilization)** | \$169 | \$0.014 |
| **Spot** | \$0 | Spot Market Price |

*Amazon EC2 pricing as of March. 30, 2014, http://aws.amazon.com/ec2/pricing/.*

lead to cost benefits for customers. For example, for the `m1.small` reserved instance, a cost reduction is obtained if the instance runs for more than 2,347 hours (or roughly 98 days), that is, $61 + 2347 \times 0.034 \simeq 2347 \times 0.060$. Therefore, the *break-even point* for acquiring a reservation is 98 days.

Some cloud providers offer multiple reservation plans with different reservation periods and expected utilization levels. For example, Amazon offers one or three-year terms contract for light, medium, and heavy levels of utilization. For the sake of simplicity, we limit the discussion to one type of reservation within a given reservation period ($\tau$). Our model can be extended to include more than one type of reservations.

#### 3.1.3 Spot Market

In this plan, customers submit their bids for acquiring instances while the provider reports a market-wide clearing price at which allocated instances are charged. The instance can be terminated by the provider as soon as the spot market's clearing price rises above the customer's bid. The customer therefore does not have full control over the instance's lifetime. A variety of market mechanisms can be used, e.g., variants on auction mechanisms, that determine the market's allocation and pricing rules. Likewise, the frequency of the mechanism's clearing can vary (e.g., upon each bid arrival, instance termination, every hour). At present, providers offer limited transparency w.r.t. the actual mechanisms used.

Consequently, we do not consider any specific spot pricing mechanism and situate the fine-grained computation of spot price dynamics outside the scope of this work. Instead, we model the spot instance price by a constant factor $\beta \in [0, 1]$ that determines the average discount rate relative to the on-demand price. We therefore assume that on average, the spot market price lies below the on-demand rate, which is reasonable given the lower quality of service (QoS) provided. According to Amazon EC2[3], recent spot prices are typically $86$ percent lower on average compared to on-demand pay-as-you-go instances, i.e., $\beta = 0.14$. We assume that a provider always retains the capability of terminating spot instances in favor of more profitable requests as a tool to increase revenue.

A disadvantage of the reservation plan is that the provider is liable to provide guaranteed availability for reserved instances while customers do not necessarily utilize their reserved capacity fully [20]. An opportunity therefore exists to make this underutilized capacity available to demands from other pricing plans. As spot instances are allowed to be terminated by the provider, we model the possibility that the provider accommodates them in the data center's underutilized reserved capacity. In principle, it is also possible to make underutilized capacity available to on-demand instance requests. This however, creates the risk of SLA violations occurring as the provider has no direct control over the lifetime of an on-demand instance. We therefore rule out such a strategy in this work.

### 3.2 The Optimal Capacity Control Problem

To maximize revenue, the cloud provider aims to optimally allocate its available capacity to requests from different pricing plans. In this section, we formally describe the problem of optimizing admission decisions on reservation contracts such that the overall revenue is maximized.

Suppose that the provider's capacity is $C$ for a specific instance type, i.e., at any given time, up to $C$ instances of that type can be hosted simultaneously. We consider the given instance type as the only one in the system. Consequently it represents our unit of capacity. However, this is not a limiting assumption as we can model other instances as multiples of the unit capacity with a limited error.

We discretize the time horizon of the admission controller into identically sized slots. The slot size is aligned with the provider's billing cycle (e.g., an hour). We assume that, given the large degree of workload multiplexing, the provider is able to predict upcoming demand for its different pricing plans for $\Gamma$ time slots.[4]

Suppose that at the current time $t = 0$, the provider predicts the number of requests in the reservation, on-demand and spot markets for a window size $\Gamma$ as $(d^r_0, \ldots, d^r_{\Gamma-1})$, $(d^o_0, \ldots, d^o_{\Gamma-1})$ and $(d^s_0, .., d^s_{\Gamma-1})$, respectively. The provider makes a decision to admit $r_t$ reservation contracts at time $t$ with $0 \leq r_t \leq d^r_t$ to maximize the revenue generated in the window. Our formulation is therefore greedy with respect to the size of the prediction window.

Let $l^r_t$ denote the total number of previously admitted reservation contracts remaining live at time $t$ (i.e., reserved capacity at time $t$ is $l^r_t + r_t$). Similarly, the total number of previously running on-demand and spot instances that remain active at that time are denoted by $l^o_t$ and $l^s_t$, respectively. Therefore at time $t$ the provider can potentially accommodate $o_t$ additional on-demand instances without overbooking the infrastructure:

$$o_t = \min\big(C - l^r_t - r_t - l^o_t, d^o_t\big). \tag{1}$$

Let $u_t \in [0, 1]$ denote the utilization of the reserved capacity at time $t$, e.g., if the total number of live reservations at time $t$ is $1{,}000$ and $600$ reserved instances are running at that time, $u_t = 0.6$. After accommodating the reservation contracts and on-demand requests, the remaining capacity can be used for spot instances, that is, $\min(C - u_t \times (l^r_t + r_t) - l^o_t - o_t, d^s_t)$.

*Problem definition*. The provider's problem is to find $r_0, r_1, \ldots, r_{\Gamma-1}$, such that the revenue within the prediction

3. Amazon EC2 Spot Instances, http://aws.amazon.com/ec2/purchasing-options/spot-instances/.

4. Note that our aim, in this paper, is not to present specific workload prediction techniques and this has previously been addressed in the literature [3], [21].

TABLE 2
Symbols and Definitions

| Symbol | Definition | Symbol | Definition |
|---|---|---|---|
| $\Gamma$ | Prediction window size | $\tau$ | Reservation period in number of time slots |
| $p$ | On-demand pay-as-you-go instance price per hour | $u_t$ | Reserved capacity utilized by reserved instances at time $t$ |
| $\varphi$ | Upfront reservation fee (premium) | $h_j$ | Lifetime of instance $j$ in number of hours |
| $\alpha$ | Reservation discount rate, the reserved instance price is $\alpha p$ per hour | $\varsigma_t$ | Data center state at stage $t$, $\varsigma_t = (l_t^r, l_t^o, i_t)$ |
| $\beta$ | Ratio of average price of spot to on-demand instances, the average price of spot instances is $\beta p$ per hour | $q$ | Termination probability of a running on-demand instance in the next time slot |
| $r_t$ | Number of reservation contracts admitted at time $t$ | $\lambda_t$ | Discount factor for upfront reservation fee at time $t$ |
| $o_t$ | Number of on-demand pay-as-you-go instances accepted at time $t$ | $\|Z\|$ | Number of reserved capacity utilization class intervals |
| $s_t$ | Number of spot instances accepted at time $t$ | $\bar{u}$ | Mean reserved capacity utilization |
| $d_t^r$ | Predicted number of reservation contracts at time $t$ | $e_t^r$ | Number of expired reservations by the end of time $t$ |
| $d_t^o$ | Predicted number for on-demand pay-as-you-go requests at time $t$ | $V(\varsigma_t)$ | Expected revenue obtained from $t = 0$ to $\tau - 1$ |
| $d_t^s$ | Predicted number of spot instances at time $t$ | $P(\varsigma_{t+1}\|\varsigma_t, r_t)$ | Transition probability from $\varsigma_t$ to $\varsigma_{t+1}$ given the chosen action $r_t$ |
| $l_t^r$ | Number of previously admitted reservation contracts live at time $t$ | $\gamma(\varsigma_t, r_t)$ | The revenue for each state-action pair |
| $l_t^o$ | Number of previously running on-demand instances active at time $t$ | $B$ | Number of instances per block of capacity |
| $l_t^s$ | Total number of previously running spot instances active at time $t$ | $T$ | Number of billing cycles (hours) per time slot |
| $i_t$ | Reserved capacity utilization class interval to which $u_t$ belongs | $z_i$ | Representative value of the class interval $i$ |

window is maximized:

$$\max_{r_t} \sum_{t=0}^{\Gamma-1} r_t\varphi + \alpha p u_t(l_t^r + r_t) + p(l_t^o + o_t) + \beta p(l_t^s + s_t)\,, \quad (2)$$

where the first term is the revenue from the upfront reservation fees and the second, third and fourth terms are the revenues per time slot from running reserved, on-demand and spot instances respectively. We can define the maximization problem as:

$$\begin{aligned} \max_{r_t} \sum_{t=0}^{\Gamma-1} & r_t\varphi + \alpha p u_t\big(l_t^r + r_t\big) + p\big(l_t^o + o_t\big) + \beta p\big(l_t^s + s_t\big) \\ \text{s.t} \quad & l_t^r + r_t + l_t^o + o_t \le C\,, \\ & u_t(l_t^r + r_t) + l_t^o + o_t + l_t^s + s_t \le C\,, \\ & \forall t = 0, \ldots, \Gamma - 1\,. \end{aligned} \quad (3)$$

The first constraint ensures that the number of live reservations and running on-demand instances remains within the provider's capacity, thereby ensuring that no SLA violations on the reservation contracts can occur. The second constraint limits the total amount of running instances over all pricing plans to that capacity.

The optimization problem (3) is non-trivial and by no means easy to solve. The root cause of the problem's complexity lies in the fact that the number of running instances in each slot for each pricing plan depends on the history of admitted requests in previous slots. Moreover, the duration that instances remain active in the system is not known a priori as the provider is often unaware of the application type running on the VM instance.

### 3.3 Optimal Capacity Control with Dynamic Programming

We devise a stochastic dynamic programming formulation to tackle problem (3) in this section. We formulate the problem as a Markov decision process defined by a four-tuple ($\varsigma$, $r$, $\gamma$, $P$) where $\varsigma$ is the state space, $r$ is the action space, $\gamma$ is the reward function, and $P$ stands for the transition probabilities that govern how the state of the process changes as actions are taken over time. The decision problem consists of $\tau$ stages indexed 0 to $\tau - 1$, each representing a time slot. The provider must decide on the number of admitted reservation contracts ($r_t$) at each time slot $t$ to maximize its revenue.

Before we formulate the details of our dynamic programming solution, we first introduce a number of assumptions made for solving the optimization problem. After that, using a set of recursive *Bellman* equations [5], we show that the problem can be broken down into simpler sub-problems, each of which can be solved optimally. Finally, we present two additional algorithms as the dynamic programming approach, while optimal, is computationally prohibitive for large-scale cloud providers. For reference, Table 2 summarizes the symbols used throughout the paper and their definitions.

#### 3.3.1 Assumptions

In general, the lifetime $h_j$ of an on-demand instance $j$, i.e., the amount of time between booting the instance and its

termination, is not known to the provider in advance. To make the analysis tractable, we assume, in line with [2], that the $h_j$'s are exponentially i.i.d. (independent and identically distributed) random variables. In our discrete setting, this means that $h_j$ follows a *geometric distribution* [5] with a probability mass function (pmf) of $P(h_j = k) = (1-q)^{k-1}q$ for $k = 1, 2, 3, \ldots$, where $q$ is the probability that the customer terminates the currently running instance in the next time slot. Since the expected value of $h_j$ is $1/q$, the expected payment over the lifetime of an on-demand instance is $E[h_j p] = p/q$.

In practice, the spot market's underlying market mechanism must be run at each time slot, involving bids from newly arrived requests and currently running spot instances. In fact, the provider does not distinguish between newly submitted requests and those requests that are admitted previously in each round of the auction [2]. Moreover, spot instances can be terminated by the provider at any time by adjustment of the market clearing price. This allows the provider to shape the load according to the available capacity and user bids. Therefore, to avoid the resulting complexity with respect to the lifetime of spot instances, we assume that the load for the spot market in each time slot is independent of the previous slots and is solely defined by demand on that time slot ($d_t^s$), i.e., $l_t^s = 0$. The load prediction component in our framework therefore computes $d_t^s$ based on the aggregated load of the spot market in past time slots, that is, $d_t^s$ implicitly includes $l_t^s$.

We treat $u_t$, the reserved capacity utilization at time $t$, as a categorical random variable. We categorize the reserved capacity utilization range into a set of $|Z|$ classes. Each class is associated with a utilization interval, denoted by $i$, of which the midpoint is used as the representative value of the corresponding class. The representative value of the $i$'th class interval is denoted by $z_i \in Z$ with $0 \leq i < |Z|$. For instance, if we take $|Z| = 5$, the utilization range of $[0, 1]$ is divided to five class intervals of $[0, 0.2], [0.2, 0.4], \ldots, [0.8, 1]$. $Z = \{0.1, 0.3, 0.5, 0.7, 0.9\}$ is used as a set of discrete values for categorizing the reserved capacity utilization. If $u_t$ lies within $[0.2, 0.4]$, then it belongs to class interval $1$ and the class interval representative value of $0.3$ is used as the utilization value at time $t$. Note that the class interval to which the reserved capacity utilization at time $t$ belongs is denoted by $i_t$, and in our calculation, we use the representative value of that class as the reserved capacity utilization at time $t$ (i.e., $z_{i_t}$). Treating $u_t$ as a discrete random variable is necessary for the dynamic programming solution we propose. The number of class intervals can be chosen depending on the desired granularity of the analysis. The provider is assumed to have sufficient load history available in order to derive the pmf of $u_t$ a priori, i.e., $P(u_t = z_{i_t})$ is known for all $z_{i_t}$.

In an MDP formulation, each state $\varsigma_t$ at time $t$ is to depend solely on the state at time $t-1$ ($\varsigma_{t-1}$) and be independent of all earlier states $\varsigma_{t-2}, \varsigma_{t-3}, \ldots, \varsigma_0$. For our optimization problem $\varsigma_t$ represents the state of the market which clearly depends on the total number of live reservations at time $t$.

Clearly, with a reservation period of size $\tau$, the total number of live reservations at time $t$ depends on $r_{t-\tau+1}, \ldots, r_{t-1}$, as reservations admitted earlier than $t-\tau+1$ will no be longer available at time $t$. To make $\varsigma_t$ only dependent on $\varsigma_{t-1}$, one could resort to the inclusion of $\tau - 1$ values in the state, each one representing the number of instances reserved at time $t-i$, $i = \tau - 1, ..., 1$. This leads to a high-dimensional state space. Note that $\tau$ is often large (e.g., the number of hours in one year) and the number of instances that are reserved at each time slot $t$ can be as large as $C$. Iteration over the possible states in the problem space therefore results in exponential time complexity, leading to the *curse of dimensionality* [22].

In practical online cases, the provider is interested in finding the admission threshold at the current time instant. Moreover, the impact of admitting a reservation at time $t$ is only affected by future events in the reservation period $[t, t+\tau-1]$. We therefore limit the prediction window $\Gamma = \tau$. This significantly reduces the dimensionality of each state as every admitted reservation in the window remains live until the end of the prediction window.

### 3.3.2 Dynamic Programming Formulation

Let us now define the four components of our MDP, i.e., $\varsigma$, $r$, $\gamma$ and $P$.

We define $\varsigma_t = (l_t^r, l_t^o, i_t)$, with $l_t^r$ the number of reservations that remain live from previous time slots in time slot $t$ and $l_t^o$ the total number of running on-demand instances remain active from previous time slots. All information about the load in the data center at time $t$ can be obtained from $\varsigma_t$. Apart from total number of live reservations and active on-demand instances, the number of running reserved instances can be computed based on $z_{i_t}$, that is, $(l_t^r + r_t) \times z_{i_t}$. The number of spot instances is also bounded by the available capacity or the spot market demand and can be calculated as follows:

$$s_t = \min\big(C - \big(l_t^r + r_t\big)z_{i_t} - \big(l_t^o + o_t\big), d_t^s\big). \tag{4}$$

The MDP consists of $\tau$ stages indexed $0$ to $\tau - 1$, each representing a time slot. The provider must decide to perform one of the possible *actions* (possible choices on the number of reservations) to admit $r_t$ reservation contracts at stage $t$, with $0 \leq r_t \leq d_t^r$.

The amount of the revenue obtained by the provider in each stage depends on the current state ($\varsigma_t$) and the provider's choice for $r_t$. The revenue of each state-action pair (i.e., the reward function $\gamma$) is therefore defined as:

$$\gamma(\varsigma_t, r_t) = \lambda_t r_t \varphi + \alpha p\big(l_t^r + r_t\big)z_{i_t} + p\big(l_t^o + o_t\big) + \beta p s_t\,, \tag{5}$$

where the consecutive terms are the total revenue of reservations, reserved, on-demand and spot instances respectively.

We define $\lambda_t$ as a discount factor that linearly scales the reservation fee with respect to the remaining time until the end of the prediction window. This measure is required as the prediction window is taken to be as large as the reservation period, which in itself is required for making sound optimization decisions. For a reservation admitted at time $t$ and expiring at time $t+\tau-1$, a total of $\tau - t$ time slots lie within the prediction window for all $0 \leq t < \tau$. Therefore, we apply a discount on the premium fee ($\varphi$) proportional to the effective reservation period in the window. In each stage $t$, we thus define $\lambda_t = (\tau - t)/\tau$ with $0 \leq t < \tau$.

Suppose there are $n$ on-demand instances in the data center at time $t-1$ (i.e., $n = l_{t-1}^o + o_{t-1}$) and right before $t$, $X$ of them are terminated by customers. This results in

$l_t^o = n - X$ active instances remaining at the beginning of $t$. According to the assumption of the geometric lifetime of on-demand instances, one can see that $X$ follows a *binomial distribution* [5] with $P(X = k) = Bin(k; n, q)$, where $Bin(k; n, q) = \binom{n}{k} q^k (1-q)^{n-k}$. Here, $q$ is the probability that the instance is terminated in the next time slot.

As stated before, each admitted reservation within the window remains active until the last stage ($t = \tau - 1$). However, at the beginning of each time slot $t$, some reservations expire as they are admitted before time $t = 0$. We define $e_t^r$ as the number of reservations that are expired by the end of $t$. Therefore, for a window of size $\tau$, $(e_0^r, \ldots, e_{\tau-2}^r)$ encodes all information regarding expired reservations in each stage. $(e_0^r, \ldots, e_{\tau-2}^r)$ can easily be obtained based on the provider's history of admitted reservation contracts.

From the above discussion, it follows that $\varsigma_{t+1}$ can be computed based on $\varsigma_t$ only. In fact, total number of live reservations at time $t$ solely depends on $l_t^r$, $e_t^r$ and $r_t$ by the relation $l_{t+1}^r = l_t^r + r_t - e_t^r$.

From the *memorylessness*[5] property of the geometric distribution, $l_{t+1}^o$ can also be easily computed only using the previous state. Finally, according to the definition, $z_{i_{t+1}}$ is independent of $z_{i_t}$. Therefore, we make an important observation, that state $\varsigma_{t+1}$ only depends on state $\varsigma_t$ at the previous time and is *independent* of earlier states $\varsigma_0, \ldots, \varsigma_{t-1}$.

Let $P(\varsigma_{t+1}|\varsigma_t, r_t)$ denote the transition probability to $\varsigma_{t+1}$ given state $\varsigma_t$ and action $r_t$. Given $k = (l_t^o + o_t) - l_{t+1}^o$, the desired transition probability is computed as follows:

$$P(\varsigma_{t+1}|\varsigma_t, r_t) = P(u_{t+1} = z_{i_{t+1}}) \times Bin(k; l_t^o + o_t, q)\,, \qquad (6)$$

where $P(u_{t+1} = z_{i_{t+1}})$ is the probability that the reserved capacity utilization at stage $t+1$ falls in the class interval $i_{t+1}$ and $Bin(k; l_t^o + o_t, q)$ denotes the probability that $k$ on-demand instances are terminated in a transition from $\varsigma_t$ to $\varsigma_{t+1}$. Since these two events are independent, the probability of both occurring is the product of their probabilities. Note that the probability of change in the reserved capacity from $l_t^r + r_t$ to $l_{t+1}^r$ given the exact values of $r_t$ is 1, as it is known how many of the reservation contracts expire at the end of time slot $t$ based on the admittance history.

Now we can characterize the problem of revenue maximization through optimal admittance of reservation contracts by the following *Bellman equations* [5]:

$$V(\varsigma_t) = \max_{r_t} [\gamma(\varsigma_t, r_t) + \sum_{\varsigma_{t+1}} P(\varsigma_{t+1}|\varsigma_t, r_t) V(\varsigma_{t+1})]\,, \qquad (7)$$

where $V(\varsigma_t)$ is the expected revenue obtained from $t$ to $\tau - 1$.

In (7), the maximum revenue the provider can obtain at state $\varsigma_t$ by optimally choosing $r_t$ is given by the expected maximum revenue over all possible states $\varsigma_{t+1}$. The boundary conditions of (7) are given by $V(\varsigma_\tau) = 0$ for all $\varsigma_\tau$. The above analysis converts problem (3) into a dynamic programming problem (7).

5. In probability theory, memorylessness is a property of those distributions (e.g., the exponential distributions and the geometric distributions), wherein any derived probability from a set of random samples is distinct and has no information of earlier samples.

#### 3.3.3 Complexity of Optimal Capacity Control

Equation (7) represents a Markov decision process that can be solved by numerical dynamic programming through backward induction. It commences the search for a solution by simulating the load for each pricing plan based on the predicted demand in the last stage $\tau - 1$ and calculating the optimal number of reservations to be admitted in that stage. Using results for the last stage, it then proceeds to determine the optimal solution for the previous stage (*backward induction*). This process continues until the optimal solution at stage $t = 0$ is obtained.

The number of possible actions at each stage is at most $C + 1$ taking into consideration $d_t^r \leq C$. The number of possible states at stage $t$ is at most $(C+1)^2 \times |Z|$ since $0 \leq l_t^r, l_t^o \leq C$. In each stage $t$, the maximization must be done over every possible action for all states which by itself requires a computation of expected revenue over all possible states at stage $t+1$. Therefore, the time complexity of a single-stage calculation is $O(C \times (C^2 \times |Z|)^2)$. As there are $\tau$ stages, the overall computational complexity is $O(\tau \times C^5 \times |Z|^2)$.

The space complexity of the dynamic algorithm to solve (7) is $O(\tau \times C^2 \times |Z|)$. This follows from the fact that the number of possible states at stage $t$, is at most $(C+1)^2 \times |Z|$ and we have $\tau$ different stage. Note that, we do not require to store values for all the states in the algorithm, as different states at each stage only depend on the previous stage. Therefore, the overall space complexity can easily be reduced to $O(2 \times C^2 \times |Z|)$ or equally $O(C^2 \times |Z|)$.

For IaaS cloud providers with large capacity (e.g, $C = 10^5$) and a long reservation period (e.g., $\tau = 1$ year) finding the exact solution of (7) is computationally prohibitive as decisions need to be made in real time. However, solving problem (7) at the granularity of a single VM and a billing cycle of an hour is not essential for large cloud providers with a large amount of cash flow. Thus, we define a *pseudo optimal* algorithm based on larger blocks of capacity and time that approximates the optimal solution and can solve the problem in a reasonable time. We also propose a *heuristic* algorithm which significantly reduces the time complexity at the price of sacrificing a fraction of the revenue.

## 4 Proposed Algorithms

### 4.1 Pseudo Optimal Algorithm

The *Pseudo Optimal* algorithm reduces the dimensionality of the problem. Define $B$ as the number of VM instances per block of capacity (e.g., $B = 100$ *VMs*) and $T$ the number of billing cycles per time slot (e.g., $T = 168$ *hours*). We apply the same approach presented in Section 3.3, while increasing the granularity of the problem formulation with respect to capacity and time. We therefore map the values of the original problem variables onto representative values given the chosen block sizes and use these in Algorithm 1. For example, for $B = 100$, all capacity values are rounded to the nearest multiple of 100. On line (1) of Algorithm 1, the revenue of each state-action pair is therefore scaled in terms of $T$ and $B$. Note that all previously used notations related to the capacity or time must be interpreted in multiples of $B$ and $T$, e.g., if $T = 24$ *hours* and the reservation period is $365 \times 24$ *hours* (365 days) then $\tau = 365$. Likewise, if $B = 100$ and the

total number of reservations that remain live at time $t$ equals 500 then $l_t^r = 5$.

**Algorithm 1.** Pseudo Optimal Algorithm

**Input:** $t, l_t^r, l_t^o, i_t$
**Output:** $maxrev$

1: **dp** $\leftarrow \{-1\}$ ▷matrix **dp** is used for memoization and all cells are initialized with -1.
2: **function** V($t$, $l_t^r$, $l_t^o$, $i_t$)
3: **if dp**$[t][l_t^r][l_t^o][i_t] \neq -1$ **then**
4: **return dp**$[t][l_t^r][l_t^o][i_t]$
5: **end if**
6: **if** $t = \tau$ **then**
7: **dp**$[t][l_t^r][l_t^o][i_t] = 0$
8: **return** 0
9: **end if**
10: $maxrev \leftarrow 0$
11: **for** $r_t \leftarrow 0$ **to** $\min(C - l_t^r - l_t^o, d_t^r)$ **do**
12: $rev \leftarrow 0$
13: $l_{t+1}^r \leftarrow l_t^r + r_t - e_t^r$
14: $o_t \leftarrow \min(C - l_t^r - l_t^o - r_t, d_t^o)$
15: $s_t \leftarrow \min(C - (l_t^r + r_t)z_{i_t} - l_t^o - o_t, d_t^s)$
16: $\lambda \leftarrow (\tau - t)/\tau$
17: $\gamma(\varsigma_t, r_t) \leftarrow B\lambda r_t\varphi + BT(\alpha p(l_t^r + r_t)z_{i_t} + p(l_t^o + o_t) + \beta p s_t)$
18: **for** $l_{t+1}^o \leftarrow 0$ **to** $l_t^o + o_t$ **do**
19: **for** $i_{t+1} \leftarrow 0$ **to** $|Z|$ **do**
20: $P(\varsigma_{t+1}|\varsigma_t, r_t) \leftarrow P(u_{t+1} = z_{i_{t+1}}) \times Bin(l_t^o + o_t - l_{t+1}^o; l_t^o + o_t, q)$
21: $rev \leftarrow rev + \gamma(\varsigma_t, r_t) + P(\varsigma_{t+1}|\varsigma_t, r_t) \times \mathrm{V}(t+1, l_{t+1}^r, l_{t+1}^o, i_{t+1})$
22: **end for**
23: **end for**
24: **if** $rev \geq maxrev$ **then**
25: $maxrev \leftarrow rev$
26: **end if**
27: **end for**
28: **dp**$[t][l_t^r][l_t^o][i_t] \leftarrow maxrev$
29: **return** $maxrev$
30: **end function**

Reducing the granularity of the optimization problem not only reduces the problem size but also removes the necessity for accurately predicting future demand at a fine-grained level of VMs and billing cycles. Depending on the prediction technique used, a reformulation of the problem at a higher granularity might therefore be better aligned with the actual accuracy of the predictions made.

### 4.2 Heuristic Algorithm with a Low Computational Complexity

The pseudo optimal algorithm proposed in the previous section can be run in an acceptable time frame if $T$ and $B$ are taken to be sufficiently large. But it still suffers from the prohibitively high polynomial order for small values of $T$ and $B$. Therefore, we propose our *heuristic* algorithm that can find an approximated solution for any values of $T$ and $B$ in $O(\tau \times C)$.

The idea behind the heuristic algorithm is that whenever the provider admits a reservation it might require to reject upcoming future on-demand requests in order to fully guarantee the availability of the reserved instances. Clearly, the admission of a reservation contract is well justified *if and only if* the revenue loss due to rejections of on-demand instances does not exceed the total revenue the reservation generates. Two main factors can affect that revenue: 1) the utilization of the reserved capacity, and 2) the demand in the spot market. The more the reservation is utilized, the higher revenue it generates in total. As stated in earlier sections of this paper, the provider is able to accommodate spot instances in the reserved capacity without any concern for the availability of the reserved instances, since spot instances can be terminated as the need arises. Therefore, if admission of a reservation provides capacity for accommodation of a spot request that might be rejected otherwise due to the lack of capacity, this additional revenue must be taken into account by the revenue management system.

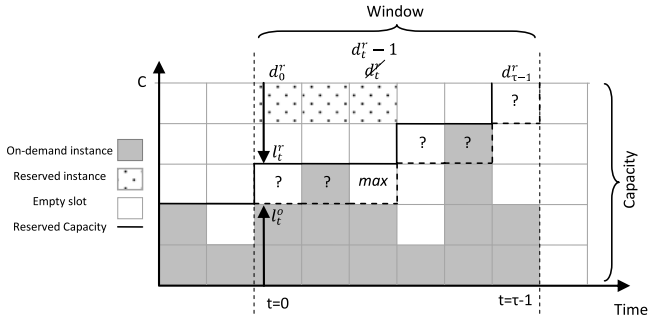


Fig. 2. Illustration of Algorithm 2. Each small block shows the capacity unit per time unit (e.g., instance-hour). Schematically, reserved instances occupy the available capacity top-down and on-demand instances use the capacity bottom-up. For clarity, spot instances are not shown in the figure.

The heuristic algorithm has two main simplifications compared to the *pseudo optimal* algorithm. First, instead of using the instance lifetime distribution to estimate load induced by on-demand requests, the future load is generated assuming all arriving requests pertain to instances with the same lifetime equal to the estimated mean lifetime. Second, it relies on the average utilization of the reserved capacity $\bar{u}$ to control admission of reservation contracts. In fact, $\bar{u}$ is the expected value of the categorical pmf related to reserved capacity utilization.

Algorithm 2 presents the details of the proposed heuristic and Fig. 2 illustrates the operation of the algorithm. As we estimate the load for on-demand instances beforehand, with a slight abuse of notation, let $l_t^r$ and $l_t^o$ denote the number of live reservations (reserved capacity) and the number of running on-demand instances at time slot $t$, respectively. $l_t^o$ is computed according to the previously instantiated VMs (before time $t = 0$) and the arriving demand ($d_t^r$) assuming each on-demand instance's lifetime equals the mean lifetime. The shaded area in the bottom of Fig. 2 exemplifies such a load. Using $e_t^r$ and the history of reservation contracts admitted earlier than $t = 0$, the initial value of $l_t^r$ is computed within the prediction window.

The algorithm attempts to admit as many reservation contracts as possible by filling the blocks from the end of the window to the beginning (denoted by the question marks in Fig. 2). In each iteration of the inner loop (see Line 8), it adds one unit to $l_t^r$, computes the revenue of adding a new reservation, and adds this value to the sum of the total revenue. The corresponding price of the on-demand instances ($B \times T \times p$) must be deducted from the total revenue until this point, if

the admission of a reservation overlaps with on-demand instance load for the specific capacity block. Thus, the summation is performed in Line 16, assuming a rejection of an on-demand request does not occur, or in Line 19 when a rejection occurs (denoted by the question marks in blocks with a white or gray color in Fig. 2, respectively).

**Algorithm 2.** Heuristic Algorithm

**Input:** $l^r, l^o$ $\triangleright$ $l^r_t$ is the initial reserved capacity at time $t$ for those requests admitted before time $t = 0$. $l^o_t$ indicates the number of on-demand instances at time $t$ taking into account previously instantiated VMs, arriving demand, and assuming that every on-demand instance's lifetime equals the mean lifetime.

**Output:** r

1: **function** HEURISTIC($l^r$, $l^o$)
2: $r \leftarrow \{0\}$ $\triangleright$ Create the array $r$ with size of $\tau$ and initialize all elements with zero.
3: **loop**
4: $max \leftarrow -\infty$
5: $index \leftarrow -1$
6: $sum \leftarrow 0$
7: $clr \leftarrow false$
8: **for** $t \leftarrow \tau - 1$ **to** $0$ **do**
9: $l^r_t \leftarrow l^r_t + 1$
10: **if** $l^r_t > C$ **then**
11: $clr \leftarrow true$
12: **break**
13: **end if**
14: $ls \leftarrow 0$
15: **if** $l^r_t + l^o_t < C$ **then**
16: $sum \leftarrow sum + T \times B(p\alpha\bar{u})$
17: $ls \leftarrow d^s_t - (C - l^r_t - l^o_t)$
18: **else**
19: $sum \leftarrow sum + T \times B(p\alpha\bar{u} - p)$
20: $ls \leftarrow d^s_t$
21: **end if**
22: **if** $(l^r_t - 1) \times (1 - \bar{u}) < ls$ **then**
23: $sum \leftarrow sum + T \times B(1 - \bar{u})\beta p$
24: **end if**
25: $\lambda \leftarrow (\tau - t)/\tau$
26: **if** $sum + B\varphi\lambda \geq max$ **and** $d^r_t > 0$ **then**
27: $max \leftarrow sum + B\varphi\lambda$
28: $index \leftarrow t$
29: **end if**
30: **end for**
31: **if** $index = -1$ **or** $max < 0$ **then**
32: **break**
33: **end if**
34: **if** $clr$ **then**
35: $k \leftarrow 0$
36: **else**
37: $k \leftarrow t$
38: **end if**
39: **for** $t \leftarrow k$ **to** $index - 1$ **do**
40: $l^r_t \leftarrow l^r_t - 1$
41: **end for**
42: $r_{index} \leftarrow r_{index} + 1$
43: $d^r_t \leftarrow d^r_t - 1$
44: **end loop**
45: **return** $r$
46: **end function**

This computation takes into account the potential revenue that spot instances can generate as well. Accordingly, the spot market's demand that must be accommodated in the underutilized reserved capacity ($ls$) is computed. The condition in Line 15 checks whether there is underutilized capacity outside the reserved capacity to accommodate spot instances or not. If there is such a capacity, then only that part of spot market's demand accommodated in the reserved capacity is taken into account (see Line 17); otherwise, the total demand in the spot market is accommodated in the reserved capacity, i.e., $ls \leftarrow d^s_t$ (see Line 20). In this process, if the generated revenue of $ls$ is not used (compensated) by previously admitted requests (see Line 22), then the revenue which is proportional to the underutilized reserved capacity is added to the sum (Line 23).

Lines 26-29 keep track of the maximum revenue found thus far for each iteration of the outer loop (Line 3). The upfront reservation fee that is proportional to the effective part of the reservation period in the window is also taken into account ($B\varphi\lambda$ in Line 27). After the maximum value and its corresponding time slot have been found, if there is available reservation demand on that time slot ($d^r_t > 0$), a reservation contract is admitted ($r_t = r_t + 1$ ) and $d^r_t$ is reduced by one unit (see Fig. 2). The process finishes when the maximum value is negative (Line 31). If the reservation load exceeds the available capacity (Line 10), a boolean variable $clr$ (*capacity limit reached*) is set to true. This aids in finding a starting point to undo added reservation contracts after the break statement at line 12 (see the loop in Lines 39 to 41). The undo process is then performed for all time slots starting from the first time slot (k = 0) or from the break point in the iteration (k = t).

The computational complexity of Algorithm 2 is $O(\tau \times C)$, as in the worst case all available blocks in the window must be investigated. The algorithm has two nested loops, an outer loop at line 3 that iterates over the capacity to find the best allocation of reservation contracts maximizing revenue at each time slot and an inner loop at line 8 that finds the best time slot to accept the next reservation in the window. The outer and inner loop at most iterate $C$ and $\tau$ times, respectively, which leads to above computational complexity. The space complexity of the algorithm is $O(\tau)$ as the algorithm only needs to store $\tau$ values of the vector of $r$. The heuristic is thus considerably more efficient than the pseudo optimal algorithm in terms of computational complexity which makes it suitable for online admission control.

## 5 Revenue Management Framework

In this section, we briefly discuss how Algorithms 1 and 2 could be used in a real-world system for online decision making on the admittance of reservation requests. The algorithms are integrated in an admission control module part of a revenue management framework outlined in Fig. 3, of which we discuss the modules in the following.

The *collector* collects and stores demand information for the different price plans. It also tracks the number of rejected requests for those individual plans. The collected information is used by the *prediction module* and the *reserved capacity analyzer* to be fed into the *admission controller*.

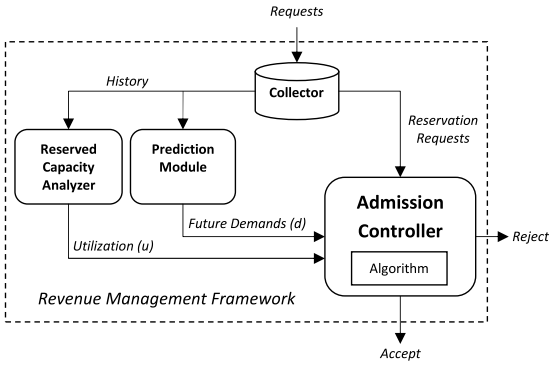


Fig. 3. Key modules of the revenue management framework.

The main role of the reserved capacity analyzer is to obtain the categorical probability distribution of the reserved capacity utilization for the pseudo optimal algorithm, or the expected value ($\bar{u}$) for the heuristic algorithm. In our simulation in Section 6, the probability distribution is dynamically derived from the history of the data center load. During each time slot, the reserved capacity analyzer measures the period of time that the utilized reserved capacity falls into the different utilization class intervals introduced in Section 3.3. It then computes the probability of each utilization class interval occurring based on the statistics collected in each time slot. Eventually, the categorical pmf is generated by averaging the computed probabilities of the last $\tau$ time slots. We also use the expected value of the distribution to set $\bar{u}$ in case of the heuristic algorithm.

The prediction module forecasts future demands for a window of size $\tau$ for each market. Forecasting future demand is a well-studied area in the literature [3], [21] and it is beyond the scope of this paper to present the best forecasting method here. Hence, we adopt a basic method for forecasting future demands in our simulation, which can be replaced with a customized prediction method in practical implementations. There the prediction module forecasts demands based on the history of the data center load by assuming the observed demands for past $\tau$ time slots would be repeated for the future $\tau$ time slots.

For the reservation market, the number of reservation contracts received by the provider per time slot is rounded to the nearest multiple of $B$. A similar transformation is used for the demand in the on-demand market. For spot instances, the prediction module computes the average load per time slot and rounds it to the nearest capacity block representative value ($B$). That is, the area below the spot market's load curve is computed and divided by the slot time duration. The prediction module also incorporates the rejected demands into the predicted future demand using the number of rejected requests per slot and the mean lifetime of instances. The number of rejected requests in each slot is divided by multiplication of the mean lifetime of VMs and the size of the time slots. Using the above framework, the provider adaptively updates the required parameters by the admission control algorithm.

At the beginning of each time slot, the predicted future demands and the computed pmf of $u_t$ are fed into the admission control module. It then calculates the maximum number of reservations ($r_t$) that must be admitted in this time slot based on the admission control algorithm. The admission control module accepts reservation contracts as long as the received demand is lower than $r_t$ during the time slot ($t = 0$). Note that the admission control algorithm is repeated for each time slot and only $r_t$ at the first time slot in the window (i.e., $t = 0$) is used to perform actions during the time slot. The produced result by the admission control algorithm remains valid as long as the observed demand is lower than the predicted demand or $r_t < d_t^r$ for the current slot. The algorithm in the admission controller module runs periodically and is executed at the beginning of each time slot. It then uses the updated information from the reserved capacity analyzer and prediction modules.

## 6 Performance Evaluation

In this section, we conduct two different groups of experiments. First, we use a large-scale trace to evaluate the revenue management framework with the proposed heuristics for admission control. Then, we further evaluate the performance of our algorithms using small-scale simulations that allow for a comparison to the optimal solution found by the dynamic programming approach.

### 6.1 Framework Evaluation

#### 6.1.1 Workload Setup

To our knowledge, no publicly available workload traces of real-world IaaS clouds currently exist, as such information is often regarded by providers as being strictly confidential. Recently, Google has published a data set pertaining to workloads on Google Compute Clusters [23]. This data set includes the resource requirements of tasks submitted by users to a cluster of 12,000 physical machines over a time period of 29 days. Although the Google cluster does not constitute an actual public IaaS cloud, we argue that its usage can represent demands of public cloud users to some extent as it records the execution of actual cloud application services provided by Google.

An issue however is that these traces do not include any details on VM instances used to execute the application-level requests. We therefore need to generate VM requests for each user as if the user was running the trace workload in a virtualized IaaS cloud such as EC2. In this regard, it is worth mentioning that in the Google cluster, tasks of different users might be scheduled onto a single machine, while in a public IaaS cloud a customer's VM executes only requests originating from applications that are hosted by that customer. In the following, we provide the details of the VM scheduling algorithm that is used to generate VM requests based on the workload of each user.

*VM scheduling.* The trace includes records of a user or application submitting several *tasks*, each of which has resource requirements related to CPU, memory and disk [23]. As 93 percent of the Google cluster nodes have the same computing capability [15], we assume that the cluster nodes are homogeneous.

We align the compute capacity of a VM instance (i.e., our capacity unit) to that of a node in the cluster. This enables us to accurately map resource requirements of tasks in the trace to VM instances.

For each user, we use the following simple scheduling algorithm to instantiate and terminate VM instances based

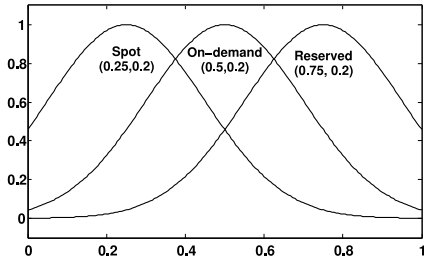


Fig. 4. Three Gaussian functions for different pricing plans.

on the resource requirements of the tasks. Whenever a user submits a task, the scheduling algorithm checks if there is available capacity in the pool of currently running VM instances, otherwise it instantiates a new VM instance. The algorithm groups the VM requests that are instantiated at the same time into a single request for multiple VMs that is sent to the provider. As such, we obtain VM requests for each user and create a trace of $250,171$ requests. The scheduling algorithm also terminates VM instances when there is no running task on the VM.

*Labeling requests with different pricing plans.* After generation of the VM requests, they need to be assigned to one of the pricing plans offered by the provider. In IaaS public clouds, customers adopt a given pricing plan based on their applications' requirements and cost considerations. Customers who are interested to run their application at very low compute prices and who require a large amount of capacity for a short period of time often rely on spot instances. The average lifetime of these instances is relatively short as instances face interruptions from time to time. The reverse holds for reserved instances as they usually execute applications with steady state or predictable long-term usage. Applications with short term, spiky, or unpredictable workloads that cannot tolerate interruption usually rely on on-demand instances, which have an average lifetime in between that of the other two categories.

On the basis of the above discussion we use the following, necessarily synthetic, approach to associate each request to one of the pricing plans. First, we normalize the lifetime of VM requests to the maximum lifetime in the traces and sort requests in ascending order of their lifetime. Next, we label requests based on random numbers generated according to three Gaussian distributions shown in Fig. 4. This results in $17{,}000$ reserved instance requests, $120{,}000$ spot instance requests and $113{,}171$ on-demand instance requests.

*Reservation requests.* Up to this point, we generated workload traces for on-demand, reserved and spot instances. In order to generate requests for obtaining an actual reserved contract, we devise an online lazy reservation strategy for each user. Whenever the user submits a request directed to a reserved instance and does not have enough reserved capacity to handle the request, a new reservation contract is acquired. This way, we assure that there is enough reserved capacity at each point in time to run all reserved instances of the user. If more than one contract must be acquired at the same time, they are grouped in a single reservation contract for multiple instances. A cost-conscious user might rely on more advanced workload prediction techniques to optimize the timing and volume of reserved contracts acquired, see for example Van den Bossche et al. [24] or Chaisiri et al. [18].

### 6.1.2 Simulation Setup

To evaluate our approach, we extend CloudSim [25] with support for the different pricing plans discussed in this paper and the proposed revenue management system. CloudSim is a discrete-event Cloud simulator that includes models of virtualized computing infrastructures and various VM provisioning policies.

*Pricing.* We adopt the pricing details of Amazon EC2 in the us-east region at the time of writing. The VM configuration used for evaluating the revenue management system is aligned with Amazon EC2 *standard small instances*. Rates of \$0.06, \$0.021 and \$0.012 per hour are used for the on-demand, reserved, and spot instances, respectively and accordingly $\alpha \simeq 0.35$ and $\beta = 0.2$. Similar to Amazon EC2, spot instances are not charged for their last partial hour upon their termination. On-demand or reserved instances that are terminated by their owner are charged for a discrete number of hours, with a partial hour of usage accounted for as a full hour.

Since the Google traces only span 29 days, we map each 5 minutes of workload data to one hour by linear scaling, resulting in a total simulation time of 12 months.

We assume each reservation is effective for two months ($\tau = 60$ days) and that the upfront reservation fee is \$22.849 which is proportional to Amazon EC2's value of $\varphi$ for a standard small instance (Linux, us-east, medium utilization) for a one-year term.

*Benchmark algorithm.* We compare the proposed *pseudo optimal* and *heuristic* algorithms with a benchmark algorithm that uses no admission control referred to as *no-control*. As its name implies, it admits all reservation contracts and gives preference to them over requests from the on-demand and spot markets. All reported revenues in Section 6.1.3 are normalized to the outcome of the *no-control* algorithm.

### 6.1.3 Experimental Results

We evaluate the revenue performance of the *pseudo optimal* and *heuristic* algorithms, varying $C$ from 600 to 3,400 with a step size of 400. We configure $B = 100$, $T = 75$ and $|Z| = 5$. The first and last two months of the 12-month simulation period are used as warm-up and cool-down periods, their respective outcomes are omitted from the experiment data. The lifetime of the on-demand instances in our workload trace does not precisely follow a geometric distribution. However, the admission controller assumes the mean lifetime of on-demand instances to be equal to the expected value of the geometric distribution, i.e., $1/q$. We therefore set $q$ to $T$ divided by the mean lifetime of on-demand instances in the workload.

The box plots in Fig. 5 show the revenue normalized to the *no-control* algorithm for 30 runs of the experiment. As expected, the revenue management system significantly improves revenue, especially under resources scarcity. As capacity increases, revenue gains decrease due to the fact that less opportunities for admission control arise. In no condition does the system lead to lower revenues compared to the *no-control* policy however. For $C = 3{,}400$, when the

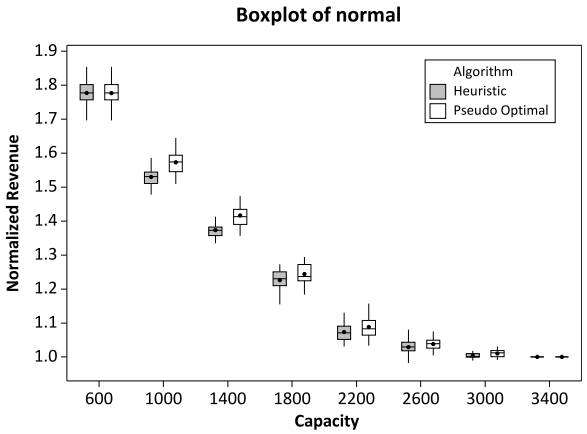


Fig. 5. The revenue performance of the proposed revenue management framework under different algorithms normalized to the outcome of the no admission control algorithm ($B = 100$ and $T = 75$).

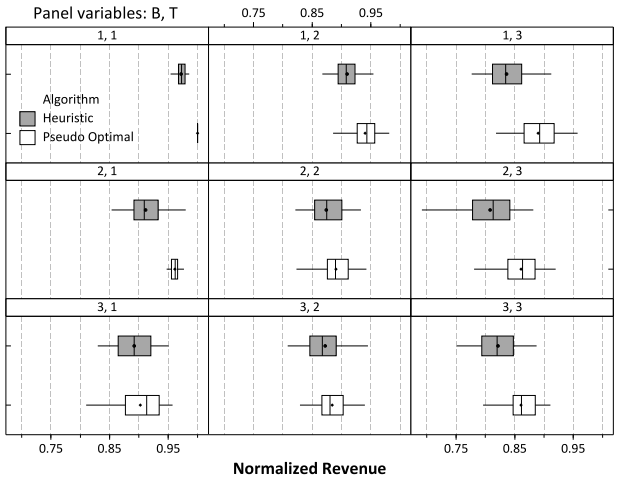


Fig. 6. The revenue performance of the pseudo optimal and heuristic algorithms with different values of B and T. All values are normalized to the outcome of the optimal solution. ($q = 0.2$).

demand to supply ratio (DSR) is sufficiently low and there is no resource contention, both algorithms generate the same revenue as *no-control*. Note that at a capacity level of 600 with a correspondingly high DSR, the *no-control* algorithm assigns the whole capacity to reservation contracts and all underutilized reserved capacity to the spot market. Our admission control algorithms increase revenue drastically under these conditions. In such cases however, a real-world provider would likely increase $C$ instead of entirely relying on admission control. An investment decision we would like to address in future developments of our revenue management framework.

According to Fig. 5, the *pseudo optimal* algorithm generates slightly more revenue than the heuristic algorithm; however, as stated before it has a significantly higher computational complexity. The heuristic algorithm generates a competitively higher revenue with a significantly lower order of computational complexity. Therefore, in online cases, it can operate with reasonable delay using smaller values for $T$ and $B$.

### 6.2 Evaluation of the Proposed Heuristic Algorithms

In the previous section, we showed that the revenue management system performs well even though a simple demand prediction model and large values of $T$ and $B$ are used. Due to high computational complexity of the optimal algorithm, it is infeasible to compare our proposed algorithms with the optimal algorithm in the large-scale scenario. In addition, prediction model errors and specific characteristics of the workload do not allow us to conduct fair experiments to show how close the algorithms can approximate the optimal solution. In this section, we evaluate the efficiency of the algorithms in comparison with the optimal solution in scenarios of smaller scale, and investigate the impact of system parameters on the performance of the proposed algorithms.

We set both capacity ($C$) and the reservation period ($\tau$) to 30. With exception of the reservation fee which is updated to the 30-hour period, i.e., $\$0.48$, all pricing related variables retain their values. The amount of requests per pricing plan is generated based on a Poisson distribution with parameter $\lambda = 1.5$ requests per hour. We used a Binomial distribution with parameters $q = 0.5$ and $n = |Z| = 5$ for the categorical pmf related to the reserved capacity utilization. All reported revenue values are normalized to the outcome of the optimal algorithm. Each experiment is carried out 30 times. For each experiment, we generate requests randomly according to the corresponding probability distributions. Afterwards, we schedule the arriving requests for a period of $\tau$ based on the computed actions by each algorithm separately. To compute the expected revenue, we apply the same computed actions for 1,000 runs in each of which the lifetime of on-demand instances are randomly generated based on the Binomial distribution and its related parameter $q$.

Fig. 6 shows box plots of the normalized revenue for the pseudo optimal and heuristic algorithms with different values of $B$ and $T$ when $q = 0.2$. The figure shows as $T$ and $B$ increase, the revenue performance of the algorithms decrease. The top left panel demonstrates a head-to-head comparison of the revenue performance of the heuristic and pseudo optimal algorithm with $T = 1$ and $B = 1$ (i.e., the optimal solution).

One important observation which may not be obvious from the figure is that even though the increase in the values $B$ and $T$ decreases the performance, the decrease is smaller when the two values are increased simultaneously. The reason is that scaling in only one dimension without considering the other causes the rounding errors to increase. In other words, an increase in $T$ enlarges the number of requests in one time slot and dividing large values to a predefined value of $B$ results in a smaller rounding error.

Our sensitivity analysis reveals the only parameter which has significant effect on the revenue performance of the algorithms is $q$. Fig. 7 shows the box plots for the revenue performance of the heuristic algorithm with regards to $q$. As shown in the figure, as $q$ increases, the revenue performance of the algorithm improves compared to the optimal solution. This is due to the fact that the optimal solution takes the probability distribution of the instance lifetime into account, while the heuristic algorithm only uses the mean lifetime value to maximize revenue. Larger values of $q$ result in smaller lifetime values, consequently the heuristic algorithm's estimated load shape more closely approximates the real load shape, until it eventually perfectly

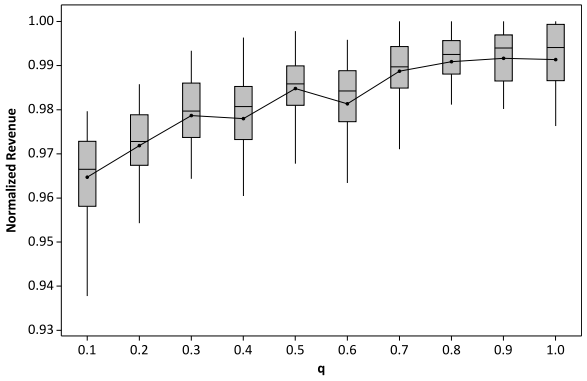


Fig. 7. Impact of q, the termination probability of the running on-demand instance in the next time slot, on the revenue performance of the heuristic algorithm with $B = 1$ and $T = 1$. All values are normalized to the outcome of the optimal solution.

matches at $q = 1$. This leads to a lower error for the solution found by the heuristic algorithm, around one percent at $q = 1$. Finally, it is worth mentioning that the low computational complexity and considerably high revenue performance of the heuristic algorithm make it a suitable choice by cloud providers aimed at revenue maximization in practical online cases.

## 7 Conclusions and Future Work

In this paper, we presented a revenue management framework to tackle the problem of optimal capacity control for allocating resources to customers of an IaaS cloud provider who are segmented into different cloud markets, i.e., reservation, on-demand pay-as-you and spot markets. The main challenge is that the provider must find an optimal capacity to admit demands from the reservation market such that the expected revenue is maximized. We consider the stochastic lifetime of on-demand requests and reserved capacity utilization and we formulate the problem as a finite horizon Markov decision process. Finding the optimal solution is computationally prohibitive in practical settings. We therefore present two algorithms namely *pseudo optimal* and *heuristic* that reduce the computational complexity. Large-scale simulations driven by Google cluster usage traces with Amazon EC2 pricing data are conducted to evaluate the revenue performance of the proposed revenue management framework using our admission control algorithms. We compare the performance of these algorithms to the optimal solution small-scale scenarios. Our experimental results suggest that significant revenue increases can be attained with the proposed revenue management approach given that sufficient resource contention is present in the system.

The broad literature of revenue management provides many meaningful future directions for this study. More research needs to be done on modeling customer reactions to a negative admission control decision (e.g., switching to a different price plan). Future research also needs to be done to incorporate our proposed reserved capacity control with support for investment decisions on extending the infrastructure in a real-world system. Another future direction of this work involves the extension of the revenue management framework with overbooking strategies.

## Acknowledgments

The authors would like to thank Yaser Mansouri and Mehran Garmehi for many helpful discussions and the rest of the CLOUDS lab members for their comments on improving the paper. A. Nadjaran Toosi is the corresponding author.

## References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Comput. Syst.*, vol. 25, no. 6, pp. 599–616, 2009.

[2] W. Wang, B. Li, and B. Liang, "Towards optimal capacity segmentation with hybrid cloud pricing," in *Proc. 32nd IEEE Int. Conf. Distrib. Comput. Syst.*, Jun. 2012, pp. 425–434.

[3] Q. Zhang, Q. Zhu, and R. Boutaba, "Dynamic resource allocation for spot markets in cloud computing environments," in *Proc. 4th IEEE Int. Conf. Utility Cloud Comput.*, Dec. 2011, pp. 178–185.

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

[5] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ, USA: Wiley, 2005.

[6] L. R. Weatherford and S. E. Bodily, "A taxonomy and research overview of perishable-asset revenue management: Yield management, overbooking, and pricing," *Oper. Res.*, vol. 40, no. 5, pp. 831–844, 1992.

[7] T. Püschel and D. Neumann, "Management of cloud infastructures: Policy-based revenue optimization," in *Proc. Int. Conf. Inf. Syst.*, Dec. 2009, pp. 2303–2314.

[8] T. Meinl, A. Anandasivam, and M. Tatsubori, "Enabling cloud service reservation with derivatives and yield management," in *Proc. IEEE 12th Conf. Commerce Enterprise Comput.*, Nov. 2010, pp. 150–155.

[9] M. Macías, J. O. Fitó, and J. Guitart, "Rule-based SLA management for revenue maximisation in cloud computing markets," in *Proc. Int. Conf. Netw. Serv. Manage.*, Oct. 2010, pp. 354–357.

[10] M. M. Kashef, A. Uzbekov, J. Altmann, and M. Hovestadt, "Comparison of two yield management strategies for cloud service providers," in *Proc. Grid Pervasive Comput.*, 2013, pp. 170–180.

[11] A. Anandasivam, S. Buschek, and R. Buyya, "A heuristic approach for capacity control in clouds," in *Proc. IEEE Conf. Commerce Enterprise Comput.*, Jul. 2009, pp. 90–97.

[12] S. K. Nair and R. Bapna, "An application of yield management for internet service providers," *Naval Res. Logistics*, vol. 48, no. 5, pp. 348–362, 2001.

[13] H. Khazaei, J. Misic, and V. Misic, "Performance analysis of cloud centers under burst arrivals and total rejection policy," in *Proc. IEEE Global Telecommun. Conf.*, Dec. 2011, pp. 1–6.

[14] M. Mazzucco and M. Dumas, "Reserved or On-demand instances? A revenue maximization model for Cloud providers," in *Proc. 4th IEEE Int. Conf. Cloud Comput.*, Jul. 2011, pp. 428–435.

[15] W. Wang, D. Niu, B. Li, and B. Liang, "Dynamic cloud resource reservation via cloud brokerage," in *Proc. IEEE 33rd Int. Conf. Distrib. Comput. Syst.*, Jul. 2013, pp. 400–409.

[16] Y.-J. Hong, J. Xue, and M. Thottethodi, "Dynamic server provisioning to minimize cost in an IAAS cloud," in *Proc. ACM SIGMETRICS Joint Int. Conf. Meas. Model. Comput. Syst.*, 2011, pp. 147–148.

[17] K. Vermeersch, "A broker for cost-efficient QoS aware resource allocation in ec2," Master's thesis, Department of Mathematics and Computer Science, Univ. of Antwerp, Antwerpen, Belgium, 2011.

[18] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimization of resource provisioning cost in cloud computing," *IEEE Trans. Serv. Comput.*, vol. 5, no. 2, pp. 164–177, Apr. 2012.

[19] T. Truong-Huu and C.-K. Tham, "A novel model for competition and cooperation among cloud providers," *IEEE Trans. Cloud Comput.*, vol. 2, no. 3, pp. 251–265, Jul.-Sep. 2014.

[20] A. N. Toosi, R. K. Thulasiram, and R. Buyya, "Financial option market model for federated cloud environments," in *Proc. 5th IEEE/ACM Int. Conf. Utility Cloud Comput.*, Nov. 2012, pp. 3–12.

[21] K. Papagiannaki, N. Taft, Z.-L. Zhang, and C. Diot, "Long-term forecasting of internet backbone traffic: Observations and initial models," in *Proc. 22nd Annu. Joint Conf. IEEE Comput. Commun. Soc.*, Mar. 2003, pp. 1178–1188.

[22] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Hoboken, NJ, USA: Wiley, 2007, vol. 703.
[23] C. Reiss, J. Wilkes, and J. L. Hellerstein. (2011, Nov). Google cluster-usage traces: Format + schema. Google Inc., Mountain View, CA, USA. [Online]. Available: http://code.google.com/p/googleclusterdata/wiki/TraceVersion2
[24] R. V. den Bossche, K. Vanmechelen, and J. Broeckhove, "Optimizing IaaS reserved contract procurement using load prediction," in *Proc. 7th IEEE Int. Conf. Cloud Comput.*, Jun. 2014, pp. 1–8.
[25] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of Cloud computing environments and evaluation of resource provisioning algorithms," *Softw.: Prac. Exp.*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

**Adel Nadjaran Toosi** received the BSc degree in 2003 and the MSc degree in 2006, both in computer science and software engineering from the Ferdowsi University of Mashhad, Iran. He is working towards the PhD degree at the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, the University of Melbourne, Australia. He was awarded International Research Scholarship (MIRS) and Melbourne International Fee Remission Scholarship (MIFRS) supporting his PhD studies. His research interests include distributed systems and cloud computing. His main focus is on pricing strategies and economics-inspired mechanisms for cloud computing. He is the member of the IEEE.

**Kurt Vanmechelen** is a post-doctoral researcher and lecturer at the University of Antwerp (UA), Belgium in the Department of Mathematics and Computer Science. His research focuses on the interplay between economics and computer-science related aspects to systems and services. His research interests include resource management in general, and market-based resource management in computational grids, clouds, and smart grids in particular. In addition, he is an active member of the parallel discrete-event simulation community. He is the member of the IEEE.

**Kotagiri Ramamohanarao** (Rao) received the PhD degree from Monash University. He was awarded the Alexander von Humboldt Fellowship in 1983. He has been at the University of Melbourne since 1980 and was appointed as a professor in computer science in 1989. He held several senior positions including Head of Computer Science and Software Engineering, Head of the School of Electrical Engineering and Computer Science at the University of Melbourne and Research Director for the Cooperative Research Center for Intelligent Decision Systems. He served on the editorial boards of the *Computer Journal*. At present he is on the editorial boards for *Universal Computer Science, and Data Mining*, IEETKDE and VLDB (*Very Large Data Bases*) Journal. He was the program co-chair for VLDB, PAKDD, DASFAA, and DOOD conferences. He received distinguished contribution award for Data Mining. He is a fellow of the Institute of Engineers Australia, a fellow of Australian Academy Technological Sciences and Engineering, and a fellow of Australian Academy of Science. He was awarded Distinguished Contribution Award in 2009 by the Computing Research and Education Association of Australasia. He is a steering committee member of the IEEE ICDM, PAKDD, and DASFAA.

**Rajkumar Buyya** is professor and future fellow of the Australian Research Council, and the director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft, a spin-off company of the University, commercializing its innovations in Cloud Computing. He has authored more than 425 publications and four text books including "Mastering Cloud Computing" published by McGraw Hill and Elsevier/Morgan Kaufmann, 2013 for Indian and international markets, respectively. He is one of the highly cited authors in computer science and software engineering worldwide. Microsoft Academic Search Index ranked him as the world's top author in distributed and parallel computing between 2007 and 2012. Software technologies for grid and cloud computing developed under his leadership have gained rapid acceptance and are in use at several academic institutions and commercial enterprises in 40 countries around the world. He has led the establishment and development of key community activities, including serving as foundation chair of the IEEE Technical Committee on Scalable Computing and five IEEE/ACM conferences. His these contributions and international research leadership are recognized through the award of "2009 IEEE Medal for Excellence in Scalable Computing" from the IEEE computer society. Manjrasoft's Aneka Cloud technology developed under his leadership has received "2010 Frost & Sullivan New Product Innovation Award" and "2011 Telstra Innovation Challenge, People's Choice Award". He is currently serving as the foundation editor-in-chief (EiC) of the *IEEE Transactions on Cloud Computing*. He is a fellow of the IEEE.