Optimal Online Multi-Instance Acquisition in IaaS Clouds

Wei Wang, *Student Member, IEEE,* Ben Liang, *Senior Member, IEEE,* and Baochun Li, *Fellow, IEEE*

Abstract—Infrastructure-as-a-Service (IaaS) clouds offer diverse instance purchasing options. A user can either run instances on demand and pay only for what it uses, or it can prepay to reserve instances for a long period, during which a usage discount is entitled. An important problem facing a user is how these two instance options can be dynamically combined to serve time-varying demands at minimum cost. Existing strategies in the literature, however, require either exact knowledge or the distribution of demands in the long-term future, which significantly limits their use in practice. Unlike existing works, we propose two practical *online algorithms*, one deterministic and another randomized, that dynamically combine the two instance options online without any knowledge of the future. We show that the proposed deterministic (resp., randomized) algorithm incurs no more than $2-\alpha$ (resp., $e/(e-1+\alpha)$) times the minimum cost obtained by an optimal *offline algorithm* that knows the exact future *a priori*, where α is the entitled discount after reservation. Our online algorithms achieve the best possible competitive ratios in both the deterministic and randomized cases, and can be easily extended to cases when short-term predictions are reliable. Simulations driven by a large volume of real-world traces show that significant cost savings can be achieved with prevalent IaaS prices.

Index Terms—IaaS cloud computing, cost management, reserved instance, multi-instance reservation, online algorithm.

1 Introduction

Enterprise spending on Infrastructure-as-a-Service (IaaS) cloud is on a rapid growth path. According to [2], the public cloud services market is expected to expand from $109 billion in 2012 to $207 billion by 2016, during which IaaS is the fastest-growing segment with a 41.7% annual growing rate [3]. IaaS cost management therefore receives significant attention and has become a primary concern for IT enterprises.

Maintaining optimal cost management is especially challenging, given the complex pricing options offered in today's IaaS services market. IaaS cloud vendors, such as Amazon EC2, ElasticHosts, GoGrid, etc., apply diverse instance (i.e., virtual machine) pricing models at different commitment levels. At the lowest level, cloud users launch *on-demand* instances and pay only for the incurred instance-hours, without making any long-term usage commitments, e.g., [4], [5], [6]. At a higher level, there are *reserved instances* wherein users prepay a one-time upfront fee and then reserve an instance for months or years, during which the usage is either free, e.g., [5], [6], or is priced under a significant discount, e.g., [4]. Table 1 gives a pricing example of on-demand and reserved instances in Amazon EC2.

Acquiring instances at the cost-optimal commitment level plays a central role for cost management. Simply operating the entire load with on-demand instances can be highly cost inefficient. For example, in Amazon EC2, three years of continuous on-demand service cost 3 times more than reserving instances for the same period [4]. On the other hand, naively switching to a long-term commitment incurs a huge amount of upfront payment (more than 1,000 times the on-demand rate in EC2 [4]), making reserved instances extremely expensive for sporadic workload. In particular, with time-varying loads, a user needs to answer two important questions: (1) when should I reserve instances (timing), and (2) how many instances should I reserve (quantity)?

TABLE 1
Pricing of on-demand and reserved instances (Light Utilization, Linux, US East) in Amazon EC2, as of Feb. 10, 2013.

Instance Type	Pricing Option	Upfront	Hourly
Standard Small	On-Demand	$0	$0.08
	1-Year Reserved	$69	$0.039
Standard Medium	On-Demand	$0	$0.16
	1-Year Reserved	$138	$0.078

Recently proposed instance reservation strategies, e.g., [7], [8], [9], heavily rely on long-term predictions of future demands, with historic workloads as references. These approaches, however, suffer from several limitations in practice. First, historic workloads might not be available, especially for startup companies who have just switched to IaaS services. In addition, not all workloads are amenable to prediction. In fact, it is observed in real production applications that workload is highly variable and statistically nonstationary [10], [11], and as a result, history may reveal limited information about the future. Moreover, due to the long span of a reservation period (i.e., months to years), workload prediction is usually required over a very long period of time, say, years. It

- *W. Wang, B. Liang and B. Li are with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, M5S 3G4. E-mail: {weiwang, liang}@ece.utoronto.ca, bli@ece.toronto.edu*
- *Part of this paper has appeared in [1]. This new version contains substantial revision with additional derivations, proofs, and explanations.*

would be very challenging, if not impossible, to make sufficiently accurate predictions over such a long term. For all these reasons, instance reservations are usually made conservatively in practice, based on empirical experiences [12] or professional recommendations, e.g., [13], [14], [15].

In this paper, we are motivated by a practical yet fundamental question: Is it possible to reserve instances in an *online* manner, with limited or even no *a priori* knowledge of the future workload, while still incurring *near-optimal* instance acquisition costs? To our knowledge, this paper represents the first attempt to answer this question, as we make the following contributions.

With dynamic programming, we first characterize the optimal offline reservation strategy as a benchmark algorithm (Sec. 3), in which the exact future demand is assumed to be known *a priori*. We show that the optimal strategy suffers "the curse of dimensionality" [16] and is hence computationally intractable. This indicates that optimal instance reservation is in fact very difficult to obtain, even given the entire future demands.

Despite the complexity of the reservation problem in the offline setting, we present two *online* reservation algorithms, one deterministic and another randomized, that offer *the best provable* cost guarantees *without* any knowledge of future demands beforehand. We first show that our deterministic algorithm incurs no more than $2-\alpha$ times the minimum cost obtained by the benchmark optimal offline algorithm (Sec. 4), and is therefore $(2-\alpha)$-*competitive*, where $\alpha \in [0,1]$ is the entitled usage discount offered by reserved instances. This translates to a worst-case cost that is 1.51 times the optimal one under the prevalent pricing of Amazon EC2. We then establish the more encouraging result that, our randomized algorithm improves the competitive ratio to $e/(e-1+\alpha)$ in expectation, and is 1.23-competitive under Amazon EC2 pricing (Sec. 5). Both algorithms achieve the *best possible* competitive ratios in the deterministic and randomized cases, respectively, and are simple enough for practical implementations.

While our online algorithms are designed without assuming knowledge of future demand, with minor modifications, they can be applied to accommodate workloads for which *short-term prediction* is reliable (Sec. 6). We show that for such workloads, our algorithms make better decisions of instance reservation by leveraging the prediction results, leading to even lower instance acquisition cost.

In addition to our theoretical analysis, we have also evaluated both proposed online algorithms via large-scale simulations (Sec. 7), driven by Google cluster-usage traces [17] with 40 GB workload demand information of 933 users in one month. Our simulation results show that, under the pricing of Amazon EC2 [4], our algorithms closely track the demand dynamics, realizing substantial cost savings compared with several alternatives.

Though we focus on cost management of acquiring compute instances, our algorithms may find wide applications in the prevalent IaaS services market. For example, Amazon ElastiCache [18] also offers two pricing options for its web caching services, i.e., the On-Demand Cache Nodes and Reserved Cache Nodes, in which our proposed algorithms can be directly applied to lower the service costs.

2 Optimal Cost Management

We start off by briefly reviewing the pricing details of the on-demand and reservation options in IaaS clouds, based on which we formulate the online instance reservation problem for optimal cost management.

2.1 On-demand and Reservation Pricing

On-Demand Instances: On-demand instances let users pay for compute capacity based on usage time without long-term commitments, and are uniformly supported in leading IaaS clouds. For example, in Amazon EC2, the hourly rate of a Standard Small Instance (Linux, US East) is \$0.08 (see Table 1). In this case, running it on demand for 100 hours costs a user \$8.

On-demand instances resemble the conventional pay-as-you-go model. Formally, for a certain type of instance, let the hourly rate be p. Then running it on demand for h hours incurs a cost of ph. Note that in most IaaS clouds, the hourly rate p is set as fixed in a very long time period (e.g., years), and can therefore be viewed as a constant.

Reserved Instances: Another type of pricing option that is widely supported in IaaS clouds is the reserved instance. It allows a user to reserve an instance for a long period (months or years) by prepaying an upfront reservation fee, after which, the usage is either free, e.g., ElasticHosts [5], GoGrid [6], or is priced with a heavy discount, e.g., Amazon EC2 [4]. For example, in Amazon EC2, to reserve a Standard Small Instance (Linux, US East, Light Utilization) for 1 year, a user pays an upfront \$69 and receives a discount rate of \$0.039 per hour within 1 year of the reservation time, as oppose to the regular rate of \$0.08 (see Table 1). Suppose this instance has run 100 hours before the reservation expires. Then the total cost incurred is \$69 + 0.039×100 = \$72.9.

Reserved instances resemble the wholesale market. Formally, for a certain type of reserved instance, let the reservation period be τ (counted by the number of hours). An instance that is reserved at hour i would expire before hour $i+\tau$. Without loss of generality, we assume the reservation fee to be 1 and normalize the on-demand rate p to the reservation fee. Let $\alpha \in [0,1]$ be the received discount due to reservation. A reserved instance running for h hours during the reservation period incurs a discounted running cost αph plus a reservation fee, leading to a total cost of $1+\alpha ph$. In the previous example, the normalized on-demand rate $p = 0.08/69$; the received discount due to reservation is $\alpha = 0.039/0.08 = 0.49$; the running hour $h = 100$; and the normalized overall cost is

$$1 + \alpha ph = 72.9/69 \ .$$

In practice, cloud providers may offer multiple types of reserved instances with different reservation periods and utilization levels. For example, Amazon EC2 offers 1-year and 3-year reservations with light, medium, and high utilizations [4]. For simplicity, we limit the discussion to one type of such reserved instances chosen by a user based on its rough estimations. We also assume that the on-demand rate is far smaller than the reservation fee, i.e., $p \ll 1$, which is always the case in IaaS clouds, e.g., [4], [5], [6].

2.2 The Online Instance Reservation Problem

In general, launching instances on demand is more cost efficient for sporadic workload, while reserved instances are more suitable to serve stable demand lasting for a long period of time, for which the low hourly rate would compensate for the high upfront fee. The cost management problem is to optimally combine the two instance options to serve the time-varying demand, such that the incurred cost is minimized. In this section, we consider making instance purchase decisions *online*, without any *a priori* knowledge about the future demands. Such an online model is especially important for startup companies who have limited or no history demand data and those cloud users whose workloads are highly variable and non-stationary — in both cases reliable predictions are unavailable. We postpone the discussions for cases when short-term demand predictions are reliable in Sec. 6.

Since IaaS instances are billed in an hourly manner, we slot the time to a sequence of hours indexed by $t = 0, 1, 2, \dots$ At each time t, demand d_t arrives, meaning that a user requests d_t instances, $d_t = 0, 1, 2, \dots$ To accommodate this demand, the user decides to use o_t on-demand instances and $d_t - o_t$ reserved instances. If the previously reserved instances that remain available at time t are fewer than $d_t - o_t$, then new instances need to be reserved. Let r_t be the number of instances that are *newly reserved* at time t, $r_t = 0, 1, 2, \dots$ The overall cost incurred at time t is the on-demand cost $o_t p$ plus the reservation cost $r_t + \alpha p(d_t - o_t)$, where r_t is the upfront payments due to new reservations, and $\alpha p(d_t - o_t)$ is the cost of running $d_t - o_t$ reserved instances.

The cost management problem is to make instance purchase decisions online, i.e., r_t and o_t at each time t, before seeing future demands $d_{t+1}, d_{t+2}, \dots$ The objective is to minimize the overall instance acquiring costs. Suppose demands last for an arbitrary time T (counted by the number of hours). We have the following *online instance reservation* problem:

$$\begin{aligned} \min_{\{r_t, o_t\}} \quad & C = \sum_{t=1}^{T} (o_t p + r_t + \alpha p(d_t - o_t)) \ , \\ \text{s.t.} \quad & o_t + \sum_{i=t-\tau+1}^{t} r_i \geq d_t \ , \\ & o_t, r_t \in \{0, 1, 2, \dots\}, t = 1, \dots, T \ . \end{aligned} \tag{1}$$

Here, the first constraint ensures that all d_t instances demanded at time t are accommodated, with o_t on-demand instances and $\sum_{i=t-\tau+1}^{t} r_i$ reserved instances that remain active at time t. Note that instances that are reserved before time $t - \tau + 1$ have all expired at time t, where τ is the reservation period. For convenience, we set $r_t = 0$ for all $t \leq 0$.

The main challenge of problem (1) lies in its online setting. Without knowledge of future demands, the online strategy may make purchase decisions that turn out later not to be optimal. Below we clarify the performance metrics to measure how far away an online strategy may deviate from the optimal solution.

2.3 Measure of Competitiveness

To measure the cost performance of an online strategy, we adopt the standard *competitive analysis* [19]. The idea is to bound the gap between the cost of an interested online algorithm and that of the optimal offline strategy. The latter is obtained by solving problem (1) with the exact future demands $d_1, \dots, d_T$ given *a priori*. Formally, we have

Definition 1 (Competitive analysis): A *deterministic* online reservation algorithm A is *c-competitive* (c is a constant) if for all possible demand sequences $\mathbf{d} = \{d_1, \dots, d_T\}$, we have

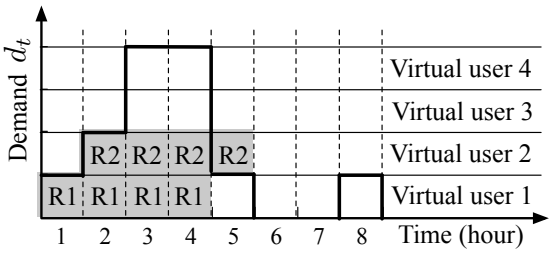
$$C_A(\mathbf{d}) \leq c \cdot C_{\text{OPT}}(\mathbf{d}) \ , \tag{2}$$

where $C_A(\mathbf{d})$ is the instance acquiring cost incurred by algorithm A given input $\mathbf{d}$, and $C_{\text{OPT}}(\mathbf{d})$ is the optimal instance acquiring cost given input $\mathbf{d}$. Here, $C_{\text{OPT}}(\mathbf{d})$ is obtained by solving the instance reservation problem (1) *offline*, where the exact demand sequence $\mathbf{d}$ is assumed to know *a priori*.

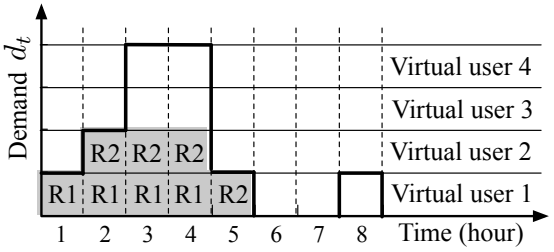
A similar definition of the competitive analysis also extends to the *randomized* online algorithm A, where the decision making is drawn from a random distribution. In this case, the left-hand side of (2) is simply replaced by $\mathbf{E}[C_A(\mathbf{d})]$, the expected cost of randomized algorithm A given input $\mathbf{d}$. (See [19] for a detailed discussion.)

Competitive analysis takes an optimal offline algorithm as a benchmark to measure the cost performance of an online strategy. Intuitively, the smaller the competitive ratio c is, the more closely the online algorithm A approaches the optimal solution. Our objective is to design *optimal online algorithms* with the smallest competitive ratio.

We note that the instance reservation problem (1) captures the Bahncard problem [20] as a special case when a user demands no more than one instance at a time, i.e., $d_t \leq 1$ for all t. The Bahncard problem models online ticket purchasing on the German Federal Railway, where one can opt to buy a Bahncard (reserve an instance) and to receive a discount on all trips within one year of the purchase date. It has been shown in [20], [21] that the lower bound of the competitive ratio is $2-\alpha$ and $e/(e-1+\alpha)$ for the deterministic and randomized



(a) Each virtual user optimally reserves instances separately.



(b) Optimal instance reservations.

Fig. 1. Simply separating the instance reservation problem into a set of independent Bahncard problems leads to inefficient use of reserved instances. In the example, the on-demand hourly rate is 0.4. The reservation fee is 1, and the reservation period is 4 hours. The reservations are highlighted as shaded areas.

Bahncard algorithms, respectively. Because the Bahncard problem is a special case of our problem (1), we have

Lemma 1: The competitive ratio of problem (1) is *at least* $2-\alpha$ for deterministic online algorithms, and is at least $e/(e-1+\alpha)$ for randomized online algorithms.

However, we show in the following that the instance reserving problem (1) is by no means a trivial extension to the Bahncard problem, mainly due to the time-multiplexing nature of reserved instances.

2.4 Challenges

A natural way to extend the Bahncard solutions in [20] is to decompose problem (1) into separate Bahncard problems. To do this, we introduce a set of *virtual users* indexed by 1, 2, ... Whenever demand d_t arrives at time t, we view the original user as d_t virtual users 1, 2, ..., d_t, each requiring one instance at that time. Fig. 1a illustrates an example with 4 virtual users. Virtual user 1 demands one instance in the first 5 hours and hour 8; virtual user 2 demands one instance in hour 2 to hour 4; both virtual users 3 and 4 have demands in hour 3 and 4. Each virtual user then reserves instances (i.e., buy a Bahncard) separately to minimize its cost, which is exactly a Bahncard problem [20].

However, such a decomposition does *not* lead to an equivalent problem as the original one. To see this, consider the following example. Suppose the on-demand hourly rate is $p = 0.4$. The reservation fee is 1, and the reservation spans 4 hours. Fig. 1a shows the reservation outcome with decomposition, where each virtual user *optimally* reserves instances based on its demand. We see that two reservations, denoted R1 and R2, are made by virtual users 1 and 2. Reservation $R1$ is used to serve the demands of virtual user 1 from hour 1 to hour 4, while reservation $R2$ is used to serve the demands of virtual user 2 from hour 2 to hour 4. Such a reservation result is obviously not optimal. As we see from Fig. 1b, in the optimal scheme, reservation $R2$ should be used by virtual user 1 to serve its demand in hour 5, during which virtual user 2 has no demand.

In general, letting each virtual user reserve instances independently leads to inefficient use of reserved instances: An instance reserved by one virtual user, even idle, cannot be shared with another, who still needs to pay for its own demand, incurring unnecessary costs.

A "simple" fix is to allow virtual users to use idle reservations of others. While conceptually simple, the algorithm needs to specify *how* idle reservations are distributed. In particular, suppose there are k reserved instances idle at time t, required by $m > k$ busy virtual users. Should these k instances be randomly distributed to k out of m virtual users? Or should they be distributed to k least-indexed virtual users? Given that different distribution strategies result in different algorithmic behaviours — whether a virtual user has been awarded "free" instances in the past would critically determine its future reservation decisions — one needs to justify which alternative gives the best cost performance. However, such justification remains a non-trivial open problem. We shall revisit this discussion in Sec. 7.

All challenges above reveal that the instance reservation problem (1) is a more complex generalization of the Bahncard problem. Indeed, as we shall see in the next section, even with full knowledge of the entire future demand, the best offline algorithm we are aware of is computationally prohibitive, which is not the case for the Bahncard problem.

3 The Optimal Offline Strategy and Its Intractability

In this section we consider the benchmark *offline cost management* strategy for problem (1), in which the exact future demands are given *a priori*. The offline setting is an integer programming problem and is generally difficult to solve. We derive the optimal solution via dynamic programming. However, such an optimal offline strategy suffers from "the curse of dimensionality" [16] and is computationally intractable.

We start by defining states. A state at time t is defined as a $(\tau-1)$-tuple $\mathbf{s}_t = (s_{t,1}, \ldots, s_{t,\tau-1})$, where $s_{t,i}$ denotes the number of instances that are reserved *no later than* t

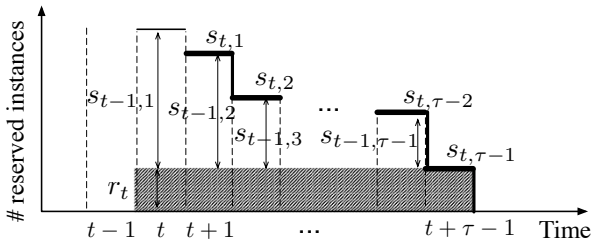


Fig. 2. Illustration of a transition into state $\mathbf{s}_t = (s_{t,1}, \ldots, s_{t,\tau-1})$ from state $\mathbf{s}_{t-1} = (s_{t-1,1}, \ldots, s_{t-1,\tau-1})$ according to (3). The shaded area stands for the r_t reservations made at time t.

and *remain active* at time $t+i$, i.e.,

$$s_{t,i} = \sum_{j=t+i-\tau+1}^{t} r_j, \quad i = 1, \ldots, \tau-1 \ .$$

We use a $(\tau-1)$-tuple to define a state because an instance that is reserved no later than t will no longer be active at time $t+\tau$ and thereafter. Clearly, $s_{t,1} \geq \cdots \geq s_{t,\tau-1}$ as reservations gradually expire.

We make an important observation, that state $\mathbf{s}_t$ only depends on states $\mathbf{s}_{t-1}$ at the previous time, and is *independent* of earlier states $\mathbf{s}_{t-2}, \ldots, \mathbf{s}_1$. Specifically, suppose state $\mathbf{s}_{t-1}$ is reached at time $t-1$, and at the beginning of the next time t, r_t new instances are reserved. These newly reserved r_t instances will add to the active reservations starting from time t, leading state $\mathbf{s}_{t-1}$ to transit into state $\mathbf{s}_t$ following the transition equations below:

$$s_{t,i} = \begin{cases} s_{t-1,i+1} + r_t, & i = 1, \ldots, \tau-2; \\ r_t, & i = \tau-1 \ . \end{cases} \tag{3}$$

To see $s_{t,\tau-1} = r_t$, let us consider state $\mathbf{s}_t$ shown in Fig. 2. By definition of $\mathbf{s}_t$, at time $t+\tau-1$, there are $s_{t,t+\tau-1}$ reservations that remain effective. All these reservations must be made at time t, because instances reserved before t have all expired at time $t+\tau-1$.

Let $V(\mathbf{s}_t)$ be the minimum cost of serving demands $d_1, \ldots, d_t$ up to time t, conditioned upon the fact that state $\mathbf{s}_t$ is reached at time t. We have the following recursive Bellman equations:

$$V(\mathbf{s}_t) = \min_{\mathbf{s}_{t-1}} \big\{ V(\mathbf{s}_{t-1}) + c(\mathbf{s}_{t-1}, \mathbf{s}_t) \big\}, \quad t > 0, \tag{4}$$

where $c(\mathbf{s}_{t-1}, \mathbf{s}_t)$ is the transition cost, and the minimization is over all states $\mathbf{s}_{t-1}$ that can transit to $\mathbf{s}_t$ following the transition equations (3). The Bellman equations (4) indicate that the minimum cost of reaching $\mathbf{s}_t$ is given by the minimum cost of reaching a previous state $\mathbf{s}_{t-1}$ plus the transition cost $c(\mathbf{s}_{t-1}, \mathbf{s}_t)$, minimized over all possible previous states $\mathbf{s}_{t-1}$. Let

$$X^+ = \max\{0, X\} \ .$$

The transition cost is defined as

$$c(\mathbf{s}_{t-1}, \mathbf{s}_t) = o_t p + r_t + \alpha p (d_t - o_t) \ , \tag{5}$$

where

$$r_t = s_{t,\tau-1},$$

$$o_t = (d_t - r_t - s_{t-1,1})^+,$$

and the transition from $\mathbf{s}_{t-1}$ to $\mathbf{s}_t$ follows (3). The rationale of (5) is as follows. By the transition equations (3), state $\mathbf{s}_{t-1}$ transits to $\mathbf{s}_t$ by reserving $r_t = s_{t,\tau-1}$ instances at time t. Adding the $s_{t-1,1}$ instances that have been reserved before t, we have $r_t + s_{t-1,1}$ reserved instances remain active at time t. We therefore need $o_t = (d_t - r_t - s_{t-1,1})^+$ on-demand instances at that time.

The boundary conditions of Bellman equations (4) are

$$V(\mathbf{s}_0) = s_{0,1}, \quad \text{for all } \mathbf{s}_0 = (s_{0,1}, \ldots, s_{0,\tau-1}), \tag{6}$$

because an initial state $\mathbf{s}_0$ indicates that a user has already reserved $s_{0,1}$ instances at the beginning.

With the analyses above, we see that the dynamic programming defined by (3), (4), (5), and (6) optimally solves the offline instance reserving problem (1). Therefore, it gives the optimal cost $C_{\text{OPT}}(\mathbf{d})$ in theory.

We now analyze the complexity of dynamic programming presented above. To solve the Bellman equations (4), one has to go through all $t = 1, 2, \ldots, T$ and compute $V(\mathbf{s}_t)$ for all states $\mathbf{s}_t$. For every time t, since a state $\mathbf{s}_t$ is defined in a high-dimensional space — recall that $\mathbf{s}_t$ is defined as a $(\tau-1)$-tuple — the number of states is $O(\bar{d}^{\tau-1})$, where $\bar{d} = \max_t d_t$ is the peak demand. For each state $\mathbf{s}_t$, computing $V(\mathbf{s}_t)$ requires looping over all states $\mathbf{s}_{t-1}$ that can transit to it. By (3), there could be as many as $O(\bar{d})$ such states. Therefore, computing $V(\mathbf{s}_t)$ for all states $\mathbf{s}_t$ at time t requires $O(\bar{d}^\tau)$ time. The overall computational complexity is $O(T\bar{d}^\tau)$. Dynamic programming hence offers a *pseudo-polynomial time* solution (i.e., the algorithm is polynomial given bounded peak demand $\bar{d}$ and reservation period τ). However, given that $\bar{d}$ and τ could be very large in practice, the algorithm is computationally intractable. This is known as "the curse of dimensionality" suffered by high-dimensional dynamic programming [16].

While it remains open to show whether the offline problem (1) is NP-hard, dynamic programming is the best algorithm we are aware of. Its computational intractability suggests that optimal cost management in IaaS clouds is in fact a very complicated problem, *even if future demands can be accurately predicted*. Note that this is not the case for the aforementioned Bahncard problem, where the demand is bounded by 1 at all times (i.e., $\bar{d} = 1$), in which dynamic programming reduces to an $O(T)$ algorithm. This serves as another evidence that the instance reservation problem (1) is by no means a simple extension to the Bahncard problem.

4 Optimal Deterministic Online Strategy

In this section, we present a deterministic online reservation strategy that incurs no more than $2-\alpha$ times the

minimum cost. As indicated by Lemma 1, this is also the best that one can expect from a deterministic algorithm.

4.1 The Deterministic Online Algorithm

We start off by defining a *break-even point* at which a user is indifferent between using a reserved instance and an on-demand instance. Suppose an on-demand instance is used to accommodate workload in a time interval that spans a reservation period, incurring a cost c. If we use a reserved instance instead to serve the same demand, the cost will be $1 + \alpha c$. When $c = 1/(1-\alpha)$, both instances cost the same, and are therefore indifferent to the user. We hence define the break-even point as

$$\beta = 1/(1-\alpha) \ . \tag{7}$$

Clearly, the use of an on-demand instance is well justified *if and only if* the incurred cost does not exceed the break-even point, i.e., $c \leq \beta$.

Our deterministic online algorithm is summarized as follows. By default, all workloads are assumed to be operated with on-demand instances. At time t, upon the arrival of demand d_t, we check the use of on-demand instances in a recent reservation period, starting from time $t-\tau+1$ to t, and reserve a new instance whenever we see an on-demand instance incurring more costs than the break-even point. Formally, let $\mathbb{1}(\cdot)$ be an *indicator function*, i.e.,

$$\mathbb{1}(X) = \begin{cases} 1, & X \text{ is true;} \\ 0, & \text{otherwise.} \end{cases}$$

Algorithm 1 presents the detail of the aforementioned deterministic online algorithm.

Algorithm 1 Deterministic Online Algorithm A_β

1. Let x_i be the number of reserved instances at time i, Initially, $x_i \leftarrow 0$ for all $i = 0, 1, \ldots$
2. Upon the arrival of demand d_t, loop as follows:
3. **while** $p \sum_{i=t-\tau+1}^{t} \mathbb{1}(d_i > x_i) > \beta$ **do**
4. Reserve a new instance: $r_t \leftarrow r_t + 1$.
5. Update the number of reservations that can be used in the future: $x_i \leftarrow x_i + 1, i = t, \ldots, t+\tau-1$.
6. Add a "phantom" reservation to the recent period, indicating that the history has already been "processed": $x_i \leftarrow x_i + 1, i = t-\tau+1, \ldots, t-1$.
7. **end while**
8. Launch on-demand instances: $o_t \leftarrow (d_t - x_t)^+$.
9. $t \leftarrow t+1$, repeat from 2.

Fig. 3 helps to illustrate Algorithm 1. Whenever demand d_t arrives, we check the recent reservation period from time $t-\tau+1$ to t. We see that an on-demand instance has been used at time i if demand d_i exceeds the number of reservations x_i (both actual and phantom), $i = t-\tau+1, \ldots, t$. The shaded area in Fig. 3 represents the use of an on-demand instance in the recent period, which incurs a cost of $p \sum_{i=t-\tau+1}^{t} \mathbb{1}(d_i > x_i)$. If this cost exceeds

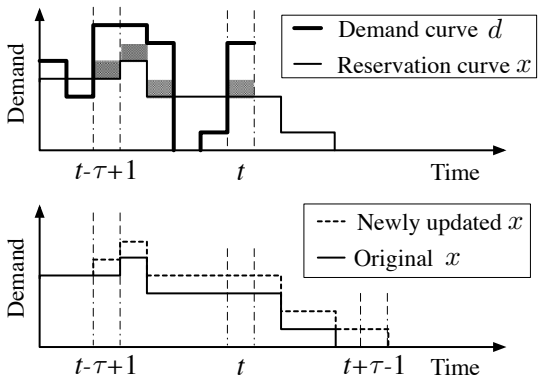


Fig. 3. Illustration of Algorithm 1. The shaded area in the top figure shows the use of an on-demand instance in the recent period. An instance is reserved at time t if the use of this on-demand instance is not well justified. The bottom figure shows the corresponding updates of the reservation curve x.

the break-even point β (line 3 of Algorithm 1), then such use of an on-demand instance is not well justified: We *should have* reserved an instance before at time $t-\tau+1$ and used it to serve the demand (shaded area) instead, which *would have* lowered the cost. As a compensation for this "mistake," we reserve an instance at the current time t (line 4), and will have one more reservation to use in the future (line 5). Since we have already compensated for the misuse of an on-demand instance (the shaded area), we add a "phantom" reservation to the history so that such a mistake will not be counted multiple times in the following rounds (line 6). This leads to an update of the reservation number $\{x_i\}$ (see the bottom figure in Fig. 3).

As a concrete example, we apply Algorithm 3 to the example of Fig. 1. It is easy to verify that two instances are reserved: the first reservation is made at time 3, while the second is made at time 4.

Unlike the naive extension of the Bahncard algorithm described in Sec. 2.4, Algorithm 1 jointly reserves instances by taking both the currently active reservations (i.e., x_t) and the historic records (i.e., x_i, $i < t$) into consideration (line 3), without any knowledge of the future. We will see later in Sec. 7 that such a joint reservation significantly outperforms the Bahncard extension where instances are reserved separately.

4.2 Performance Analysis: $(2-\alpha)$-Competitiveness

The "trick" of Algorithm 1 is to make reservations "lazily": no instance is reserved unless the misuse of an on-demand instance is seen. Such "lazy behaviour" turns out to guarantee that the algorithm incurs no more than $2-\alpha$ times the minimum cost.

Let A_β denote Algorithm 1 and let OPT denote the optimal offline algorithm. We now make an important observation, that OPT reserves at least the same amount of instances as A_β does, for any demand sequence.

Lemma 2: Given an arbitrary demand sequence, let n_β be the number of instances reserved by A_β, and let n_{OPT} be the number of instances reserved by OPT. Then $n_\beta \leq n_{\text{OPT}}$.

The proof of Lemma 2 is given in the appendix.[1] Lemma 2 can be viewed as a result of the "lazy behaviour" of A_β, in which instances are reserved just to compensate for the previous "purchase mistakes." Intuitively, such a conservative reservation strategy leads to fewer reserved instances.

We are now ready to analyze the cost performance of A_β, using the optimal offline algorithm OPT as a benchmark.

Proposition 1: Algorithm 1 is $(2-\alpha)$-*competitive*. Formally, for any demand sequence,

$$C_{A_\beta} \leq (2-\alpha) C_{\text{OPT}} \ , \tag{8}$$

where C_{A_β} is the cost of Algorithm 1 (A_β), and C_{OPT} is the cost of the optimal offline algorithm OPT.

Proof: Suppose A_β (resp., OPT) launches o_t (resp., o_t^*) on-demand instances at time t. Let $\text{Od}(A_\beta)$ be the costs incurred by these on-demand instances under A_β, i.e.,

$$\text{Od}(A_\beta) = \sum_{t=1}^{T} o_t p \ . \tag{9}$$

We refer to $\text{Od}(A_\beta)$ as the *on-demand costs* of A_β. Similarly, we define the on-demand costs incurred by OPT as

$$\text{Od}(\text{OPT}) = \sum_{t=1}^{T} o_t^* p \ . \tag{10}$$

Also, let

$$\text{Od}(A_\beta \backslash \text{OPT}) = \sum_{t=1}^{T} (o_t - o_t^*)^+ p \tag{11}$$

be the on-demand costs incurred in A_β that are not incurred in OPT. We see

$$\text{Od}(A_\beta \backslash \text{OPT}) \leq \beta n_{\text{OPT}} \tag{12}$$

by noting the following two facts: First, demands $\sum_{t=1}^{T} (o_t - o_t^*)^+$ are served by at most n_{OPT} reserved instances in OPT. Second, demands that are served by the same reserved instance in OPT incur on-demand costs of at most β in A_β (by the definition of A_β). We therefore bound $\text{Od}(A_\beta)$ as follows:

$$\begin{aligned} \text{Od}(A_\beta) &\leq \text{Od}(\text{OPT}) + \text{Od}(A_\beta \backslash \text{OPT}) \\ &\leq \text{Od}(\text{OPT}) + \beta n_{\text{OPT}} \ . \end{aligned} \tag{13}$$

Let S be the cost of serving all demands with on-demand instances, i.e.,

$$S = \sum_{t=1}^{T} d_t p \ .$$

1. The appendix is given in an online supplementary document.

We bound the cost of OPT as follows:

$$\begin{aligned} C_{\text{OPT}} &= \text{Od}(\text{OPT}) + n_{\text{OPT}} + \alpha(S - \text{Od}(\text{OPT})) && (14) \\ &\geq \text{Od}(\text{OPT}) + n_{\text{OPT}} + \alpha\beta n_{\text{OPT}} && (15) \\ &\geq n_{\text{OPT}}/(1-\alpha) \ . && (16) \end{aligned}$$

Here, (15) holds because in OPT, demands that are served by the same reserved instance incur at least a break-even cost β when priced at an on-demand rate p.

With (13) and (16), we bound the cost of A_β as follows:

$$\begin{aligned} C_{A_\beta} &= \text{Od}(A_\beta) + n_\beta + \alpha(S - \text{Od}(A_\beta)) \\ &\leq (1-\alpha)\text{Od}(A_\beta) + n_{\text{OPT}} + \alpha S && (17) \\ &\leq (1-\alpha)(\text{Od}(\text{OPT}) + \beta n_{\text{OPT}}) + \alpha S + n_{\text{OPT}} && (18) \\ &= C_{\text{OPT}} + n_{\text{OPT}} && (19) \\ &\leq (2-\alpha) C_{\text{OPT}} \ . && (20) \end{aligned}$$

Here, (17) holds because $n_\beta \leq n_{\text{OPT}}$ (Lemma 2). Inequality (18) follows from (13), while (20) is derived from (16). □

By Lemma 1, we see that $2-\alpha$ is already the best possible competitive ratio for deterministic online algorithms, which implies that Algorithm 1 is optimal in a view of competitive analysis.

Proposition 2: Among all online deterministic algorithms of problem (1), Algorithm 1 is *optimal* with the smallest competitive ratio of $2-\alpha$.

As a direct application, in Amazon EC2 with reservation discount $\alpha = 0.49$ (see Table 1), algorithm A_β will lead to no more than 1.51 times the optimal instance purchase cost.

Despite the already satisfactory cost performance offered by the proposed deterministic algorithm, we show in the next section that the competitive ratio may be further improved if randomness is introduced.

5 Optimal Randomized Online Strategy

In this section, we construct a randomized online strategy that is a random distribution over a family of deterministic online algorithms similar to A_β. We show that such randomization improves the competitive ratio to $e/(e-1+\alpha)$ and hence leads to a better cost performance. As indicated by Lemma 1, this is the best that one can expect without knowledge of future demands.

5.1 The Randomized Online Algorithm

We start by defining a family of algorithms similar to the deterministic algorithm A_β. Let A_z be a similar deterministic algorithm to A_β with β in line 3 of Algorithm 1 replaced by $z \in [0, \beta]$. That is, A_z reserves an instance whenever it sees an on-demand instance incurring more costs than z in the recent reservation period. Intuitively, the value of z reflects the *aggressiveness* of a reservation strategy. The smaller the z, the more aggressive the strategy. As an extreme, a user will always reserve when $z = 0$. Another extreme goes to $z = \beta$ (Algorithm 1), in

which the user is very conservative in reserving new instances.

Our randomized online algorithm picks a $z \in [0, \beta]$ according to a density function $f(z)$ and runs the resulting algorithm A_z. Specifically, the density function $f(z)$ is defined as

$$f(z) = \begin{cases} (1-\alpha)e^{(1-\alpha)z}/(e-1+\alpha), & z \in [0, \beta), \\ \delta(z-\beta) \cdot \alpha/(e-1+\alpha), & \text{o.w.}, \end{cases} \quad (21)$$

where $\delta(\cdot)$ is the *Dirac delta function*. That is, we pick $z = \beta$ with probability $\alpha/(e-1+\alpha)$. It is interesting to point out that in other online rent-or-buy problems, e.g., [22], [21], [23], the density function of a randomized algorithm is usually continuous.[2] However, we note that a continuous density function does not lead to the minimum competitive ratio in our problem. Algorithm 2 formalizes the descriptions above.

Algorithm 2 Randomized Online Algorithm

1. Randomly pick $z \in [0, \beta]$ according to a density function $f(z)$ defined by (21)
2. Run A_z

The rationale behind Algorithm 2 is to strike a suitable balance between reserving "aggressively" and "conservatively." Intuitively, being aggressive is cost efficient when future demands are long-lasting and stable, while being conservative is efficient for sporadic demands. Given the unknown future, the algorithm randomly chooses a strategy A_z, with an expectation that the incurred cost will closely approach the *ex post* minimum cost. We see in the following that the choice of $f(z)$ in (21) leads to the optimal competitive ratio $e/(e-1+\alpha)$.

5.2 Performance Analysis: $e/(e-1+\alpha)$-Competitiveness

To analyze the cost performance of the randomized algorithm, we need to understand how the cost of algorithm A_z relates to the cost of the optimal offline algorithm OPT. The following lemma reveals their relationship. The proof is given in the appendix.

Lemma 3: Given an arbitrary demand sequence $d_1, \ldots, d_T$, suppose algorithm A_z (resp., OPT) launches $o_{z,t}$ (resp., o_t^*) on-demand instances at time t. Let C_{A_z} be the instance acquiring cost incurred by algorithm A_z, and n_z the number of instances reserved by A_z. Denote by

$$D_z = \sum_{t=1}^{T} (o_t^* - o_{z,t})^+ p \quad (22)$$

the on-demand costs incurred by OPT that are not by algorithm A_z. We have the following three statements.

(1) The cost of algorithm A_z is at most

$$C_{A_z} \leq C_{\text{OPT}} - n_{\text{OPT}} + n_z + (1-\alpha)(z n_{\text{OPT}} - D_z). \quad (23)$$

2. The density function in these works is chosen as $f(z) = e^z/(e-1), z \in [0,1]$, which is a special case of ours when $\alpha = 0$.

(2) The value of D_z is at least

$$D_z \geq \int_z^\beta n_w \mathrm{d}w - (\beta - z) n_{\text{OPT}} \ . \quad (24)$$

(3) The cost incurred by OPT is at least

$$C_{\text{OPT}} \geq \int_0^\beta n_z \mathrm{d}z \ . \quad (25)$$

With Lemma 3, we bound the expected cost of our randomized algorithm with respect to the cost incurred by OPT.

Proposition 3: Algorithm 2 is $e/(e-1+\alpha)$-competitive. Formally, for any demand sequence,

$$\mathbf{E}[C_{A_z}] \leq \frac{e}{e-1+\alpha} C_{\text{OPT}} \ , \quad (26)$$

where the expectation is over z between 0 and β according to density function $f(z)$ defined in (21).

Proof: Let $F(z) = \int_0^z f(x)\mathrm{d}x$, and $E_F = \int_0^z x f(x)\mathrm{d}x$. From (23), we have

$$\begin{aligned} \mathbf{E}[C_{A_z}] &\leq C_{\text{OPT}} - n_{\text{OPT}} + \int_0^\beta f(z) n_z \mathrm{d}z \\ &\quad + (1-\alpha) \int_0^\beta f(z)(z n_{\text{OPT}} - D_z)\mathrm{d}z \quad (27) \\ &\leq C_{\text{OPT}} - \alpha n_{\text{OPT}} E_F + \int_0^\beta f(z) n_z \mathrm{d}z \\ &\quad - (1-\alpha) \int_0^\beta n_w \int_0^w f(z)\mathrm{d}z\mathrm{d}w \quad (28) \\ &= C_{\text{OPT}} + \int_0^\beta (f(z) - (1-\alpha)F(z)) n_z \mathrm{d}z \\ &\quad - \alpha n_{\text{OPT}} E_F \ , \quad (29) \end{aligned}$$

where the second inequality is obtained by plugging in inequality (24).

Now divide both sides of inequality (29) by C_{OPT} and apply inequality (25). We have

$$\frac{\mathbf{E}[C_{A_z}]}{C_{\text{OPT}}} \leq 1 + \frac{\int_0^\beta (f(z) - (1-\alpha)F(z)) n_z \mathrm{d}z - \alpha n_{\text{OPT}} E_F}{\int_0^\beta n_z \mathrm{d}z} \ . \quad (30)$$

Plugging $f(z)$ defined in (21) into (30) and noting that $n_\beta \leq n_{\text{OPT}}$ (Lemma 2) lead to the desired competitive ratio. □

By Lemma 1, we see that no online randomized algorithm is better than Algorithm 2 in terms of the competitive ratio.

Proposition 4: Among all online randomized algorithms of problem (1), Algorithm 2 is optimal with the smallest competitive ratio $e/(e-1+\alpha)$.

We visualize in Fig. 4 the competitive ratios of both deterministic and randomized algorithms against the hourly discount α offered by reserved instances. Compared with the deterministic algorithm, we see that introducing randomness significantly improves the competitive ratio in all cases. The two algorithms become exactly the same when $\alpha = 1$, at which the reservation offers

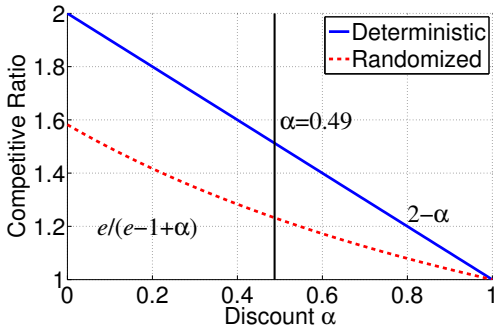


Fig. 4. Competitive ratios of both deterministic and randomized algorithms.

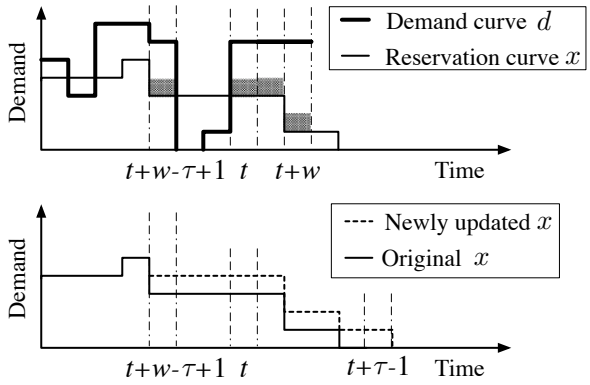


Fig. 5. Illustration of Algorithm 3. The shaded area in the top figure shows the use of an on-demand instance in $[t+w-\tau+1, t+w]$. An instance is reserved at time t if the use of this on-demand instance is not well justified. The bottom figure shows the corresponding updates of the reservation curve x.

no discount and will never be considered. In particular, when it comes to Amazon EC2 with reservation discount $\alpha = 0.49$ (standard 1-year reservation, light utilization), the randomized algorithm leads to a competitive ratio of 1.23, compared with the 1.51-competitiveness of the deterministic alternative.

6 Cost Management with Short-Term Demand Predictions

In the previous sections, our discussions focus on the extreme cases, with either full future demand information (i.e., the offline case in Sec. 3) or no *a priori* knowledge of the future (i.e., the online case in Sec. 4 and 5). In this section, we consider the middle ground in which short-term demand prediction is reliable. Note that this is not an uncommon case in practice. For example, websites typically see diurnal patterns exhibited on their workloads, based on which it is possible to have a demand prediction window that is weeks into the future. Both our online algorithms can be easily extended to utilize these prediction results to improve reservation decisions.

We begin by formulating the instance reservation problem with limited information of future demands. Let w be the *prediction window*. That is, at any time t, a user can predict its future demands $d_{t+1}, \ldots, d_{t+w}$ in the next w hours. Since only short-term predictions are reliable, we can safely assume that the prediction window is less than the reservation period, i.e., $w < \tau$. The instance reservation problem resembles the online reservation problem (1), except that the instance purchase decisions made at each time t, i.e., the number of reserved instances (i.e., r_t) and on-demand instances (i.e., o_t), are based on both history and future demands predicted, i.e., $d_1, \ldots, d_{t+w}$. The competitive analysis (Definition 1) remains valid in this case.

The Deterministic Algorithm: We extend our deterministic online algorithm as follows. As before, all workloads are *by default* served by on-demand instances. At time t, we can predict the demands up to time $t+w$. Unlike the online deterministic algorithm, we check the use of on-demand instances in a reservation period across *both history and future*, starting from time $t+w-\tau+1$ to $t+w$. A new instance is reserved at time t whenever we see an on-demand instance incurring more costs than the break-even point β and the currently effective reservations are less than the current demand d_t. Algorithm 3, denoted A_β^w, formalizes the description above.

Algorithm 3 Deterministic Algorithm A_β^w with Prediction Window w

1. Let x_i be the number of reserved instances at time i, Initially, $x_i \leftarrow 0$ for all $i = 0, 1, \ldots$
2. Upon the arrival of demand d_t, loop as follows:
3. **while** $p\sum_{i=t+w-\tau+1}^{t+w} \mathbb{1}(d_i > x_i) > \beta$ and $x_t < d_t$ **do**
4. Reserve a new instance: $r_t \leftarrow r_t + 1$.
5. Update the number of reservations that can be used in the future, i.e., $x_i \leftarrow x_i + 1, i = t, \ldots, t+\tau-1$.
6. Add a "phantom" reservation to the history, indicating that the history has already been "processed", i.e., $x_i \leftarrow x_i + 1$ for all $i = t+w-\tau+1, \ldots, t-1$.
7. **end while**
8. Launch on-demand instances: $o_t \leftarrow (d_t - x_t)^+$.
9. $t \leftarrow t+1$, repeat from 2.

Fig. 5 illustrates an example of applying Algorithm 3 with the same demand input as Fig. 3. In the top figure, at time t, we know the demand curve up to time $t+w$. The shaded area represents the use of an on-demand instance in $[t+w-\tau+1, t+w]$. Since the incurred on-demand cost exceeds the break-even point, a new instance is reserved at time t, and the reservation curve is updated accordingly as shown in the lower figure.

The Randomized Algorithm: The randomized algorithm can also be constructed as a random distribution over a family of deterministic algorithms similar to A_β^w. In particular, let A_z^w be similarly defined as algorithm A_β^w

with β replaced by $z \in [0, \beta]$ in line 3 of Algorithm 3. The value of z reflects the aggressiveness of instance reservation. The smaller the z, the more aggressive the reservation strategy. Similar to the online randomized, we introduce randomness to strike a good balance between reserving aggressively and conservatively. Our algorithm randomly picks $z \in [0, \beta]$ according to the same density function $f(z)$ defined by (21), and runs the resulting algorithm A_z^w. Algorithm 4 formalizes the description above.

Algorithm 4 Randomized Algorithm with Prediction Window w

1. Randomly pick $z \in [0, \beta]$ according to a density function $f(z)$ defined by (21)
2. Run A_z^w

It is easy to see that both the deterministic and the randomized algorithms presented above improve the cost performance of their online counterparts, due to the knowledge of future demands. Therefore, we have Proposition 5 below. We will evaluate their performance gains via trace-driven simulations in the next section.

Proposition 5: Algorithm 3 is $(2-\alpha)$-competitive, and Algorithm 4 is $e/(e-1+\alpha)$-competitive.

Finally, while both algorithms can be applied when the prediction window is longer than the reservation period (i.e., $w > \tau$), demand information predicted beyond the reservation period will have no effect on the algorithms. This suggests that one might need to develop alternative algorithms when long-term prediction is reliable. Discussions on such algorithms are beyond the scope of this paper.

7 Trace-Driven Simulations

So far, we have analyzed the cost performance of the proposed algorithms in a view of competitive analysis. In this section, we evaluate their performance for practical cloud users via simulations driven by a large volume of real-world traces.

7.1 Dataset Description and Preprocessing

Long-term user demand data in public IaaS clouds are often confidential: no cloud provider has released such information so far. For this reason, we turn to Google cluster-usage traces that were recently released in [17]. Although Google is not a public IaaS cloud, its cluster-usage traces record the computing demands of its cloud services and Google engineers, which can represent the computing demands of IaaS users to some degree. The dataset contains 40 GB of workload resource requirements (e.g., CPU, memory, disk, etc.) of 933 users over 29 days in May 2011, on a cluster of more than 11K Google machines.

Demand Curve: Given the workload traces of each user, we ask the question: How many computing instances would this user require if it were to run the same workload in a public IaaS cloud? For simplicity, we set an instance to have the same computing capacity as a cluster machine, which enables us to accurately estimate the run time of computational tasks by learning from the original traces. We then schedule these tasks onto instances with sufficient resources to accommodate their requirements. Computational tasks that cannot run on the same server in the traces (e.g., tasks of MapReduce) are scheduled to different instances. In the end, we obtain a demand curve for each user, indicating how many instances this user requires in each hour. Fig. 6 illustrates such a demand curve for a user.

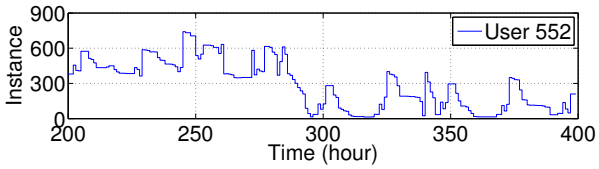


Fig. 6. The demand curve of User 552 in Google cluster-usage traces [17], over 1 month.

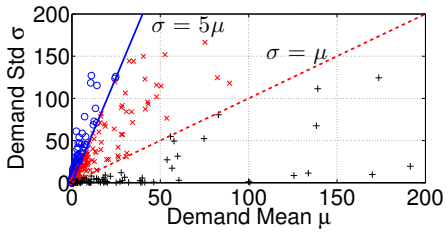
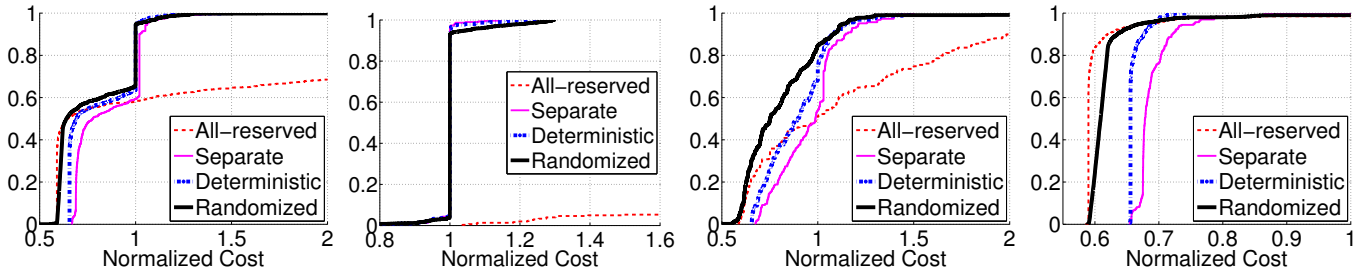


Fig. 7. User demand statistics and group division.

User Classification: To investigate how our online algorithms perform under different demand patterns, we classify all 933 users into three groups by the *demand fluctuation level* measured as the ratio between the standard deviation σ and the mean μ.

Specifically, *Group 1* consists of users whose demands are highly fluctuating, with $\sigma/\mu \geq 5$. As shown in Fig. 7 (circle 'o'), these demands usually have small means, which implies that they are highly sporadic and are best served with on-demand instances. *Group 2* includes users whose demands are less fluctuating, with $1 \leq \sigma/\mu < 5$. As shown in Fig. 7 (cross 'x'), these demands cannot be simply served by on-demand or reserved instances alone. *Group 3* includes all remaining users with relatively stable demands ($0 \leq \sigma/\mu < 1$). As shown in Fig. 7 (plus '+'), these demands have large means and are best served with reserved instances. Our evaluations are carried out for each user group.

Pricing: Throughout the simulation, we adopt the pricing of Amazon EC2 standard small instances with the on-demand rate \$0.08, the reservation fee \$69, and the discount rate \$0.039 (Linux, US East, 1-year light utilization). The break-even point is 1683 instance-hours, or 70 days. Since the Google traces only span one month, we proportionally shorten the on-demand billing cycle from one hour to one minute, and the reservation period



(a) Cost CDF (all users) (b) Cost CDF (Group 1) (c) Cost CDF (Group 2) (d) Cost CDF (Group 3)

Fig. 8. Cost performance of online algorithms without *a priori* knowledge of future demands. All costs are normalized to All-on-demand.

from 1 year to 6 days (i.e., $24 \times 365 = 8760$ minutes $=$ 6 days) as well. The break-even point is linearly scaled to 28 instance-hours.

7.2 Evaluations of Online Algorithms

We start by evaluating the performance of online algorithms without any *a priori* knowledge of user demands.

Benchmark Online Algorithms: We compare our online deterministic and randomized algorithms with three benchmark online strategies. The first is *All-on-demand*, in which a user never reserves and operates all workloads with on-demand instances. This algorithm, though simple, is the most common strategy in practice, especially for those users with time-varying workloads [12]. It is worth mentioning that the proposed online algorithms reduce to All-on-demand for workload that ends before the break-even time, which is the optimal strategy as the use of a reserved instance is not well justified. The second algorithm is *All-reserved*, in which all computational demands are served via reservations. The third online algorithm is the improved Bahncard extension described in Sec. 2.4, where virtual users reserve instances separately. We consider two distribution strategies: (1) randomly distributing idle reservations (if any) to busy virtual users, and (2) distributing idle reservations to the least-indexed virtual users. We have found that the former incurs slightly less cost than the latter in most cases. We hence adopt it as the third benchmark algorithm, referred to as *Separate*. All three benchmark algorithms, as well as the two proposed online algorithms, are carried out for each user in the Google traces. All the incurred costs are *normalized to All-on-demand*.

Cost Performance: We present the simulation results in Fig. 8, where the CDF of the normalized costs are given, grouped by users with different demand fluctuation levels. We see in Fig. 8a that when applied to all 933 users, both the deterministic and randomized online algorithms realize significant cost savings compared with all three benchmarks. In particular, when switching from All-on-demand to the proposed online algorithms, more than 60% users cut their costs. About 50% users save more than 40%. Only 2% incur slightly more costs than before. For users who switch from All-reserved to our randomized online algorithms, the improvement is even more substantial. As shown in Fig. 8a, cost savings are almost guaranteed, with 30% users saving more than 50%. We also note that Separate, while generally outperforming All-on-demand and All-reserved, incurs more costs than our online algorithms.

We next compare the cost performance of all five algorithms at different demand fluctuation levels. As expected, when it comes to the extreme cases, All-on-demand is the best fit for Group 1 users whose demands are known to be highly busty and sporadic (Fig. 8b), while All-reserved incurs the least cost for Group 3 users with stable workloads (Fig. 8d). These two groups of users, should they know their demand patterns, would have the least incentive to adopt advanced instance reserving strategies, as naively switching to one option is already optimal. Even in these extreme cases, our online algorithms, especially the randomized one, remain highly competitive, incurring only slightly higher cost.

However, the acquisition of instances is not always a black-and-white choice between All-on-demand and All-reserved. As we observe from Fig. 8c, for Group 2 users, a more intelligent reservation strategy is essential, since naive algorithms, either All-on-demand or All-reserved, are always highly risky and can easily result in skyrocketing cost. Our online algorithms, on the other hand, become the best choices in this case, outperforming all three benchmark algorithms by a significant margin.

Table 2 summarizes the average cost performance for each user group. We see that our online algorithms remain highly competitive in all cases. In addition, the complexity of our online algorithms is also very low, both operating in $O(\tau \bar{d})$ time, where $\bar{d}$ is the peak demand. Note that this complexity is also required by Separate. Only naive strategies (i.e., All-on-demand and All-reserved) have lower complexity of $O(1)$, but at a price of much higher instance acquisition cost.

7.3 The Value of Short-Term Predictions

While our online algorithms perform sufficiently well without knowledge of future demands, we show in this subsection that more cost savings are realized by their extensions when short-term demand predictions are reliable. Since we are not concerned with the detailed

TABLE 2
Average cost performance (normalized to All-on-demand).

Algorithm	All users	Group 1	Group 2	Group 3
All-on-demand	1.00	1.00	1.00	1.00
All-reserved	16.48	48.99	1.25	0.61
Separate	0.86	1.02	0.96	0.70
Deterministic	0.81	1.00	0.89	0.67
Randomized	0.76	1.02	0.79	0.63

workload prediction techniques, we set the future workload available to both the deterministic and randomized extensions (i.e., Algorithm 3 and 4) within the prediction window. Specifically, we consider three prediction windows that are 1, 2, and 3 months into the future, respectively. To fit these numbers into the one-month Google traces, we apply the same linear scaling mentioned in Sec. 7.1 (i.e., one hour is linearly scaled down to one minute), after which the prediction windows used in the simulations are 12, 24, and 36 hours, respectively. For each prediction window, we run both Algorithm 3 and 4 for each Google user in the traces, and compare their costs with those incurred by the online counterparts without future knowledge (i.e., Algorithm 1 and 2). Figs. 9 and 10 illustrate the simulation results, where all costs are *normalized to Algorithm 1 and 2, respectively*.

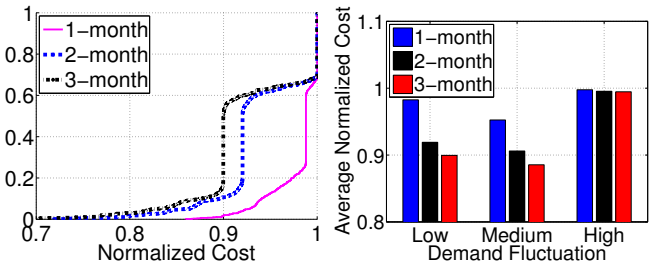
As expected, the more information we know about the future demands (i.e., longer prediction window), the better the cost performance. Yet, the marginal benefits of having long-term predictions are diminishing. As shown in Figs. 9a and 10a, long prediction windows will not see proportional performance gains. This is especially the case for the randomized algorithm, in which knowing the 2-month future demand *a priori* is no different from knowing 3 months beforehand.

Also, we can see in Fig. 9b that for the deterministic algorithm, having future information only benefits those users whose demands are stable or with medium fluctuation. This is because the deterministic online algorithm is almost optimal for users with highly fluctuating demands (see Fig. 8b), leaving no space for further improvements. On the other hand, we see in Fig. 10b that the benefits of knowing future demands are consistent for all users with the randomized algorithm.

8 Related Work

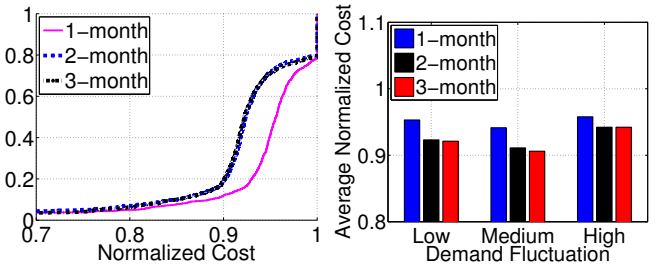
On-demand and reserved instances are the two most prominent pricing options that are widely supported in leading IaaS clouds [4], [5], [6]. Many case studies [12] show that effectively combining the use of the two instances leads to a significant cost reduction.

There exist some works in the literature, including both algorithm design [7], [8], [24], [25] and prototype implementation [9], focusing on combining the two instance options in a cost efficient manner. All these works assume, either explicitly or implicitly, that workloads are statistically stationary in the long-term future and



(a) Cost CDF (b) Average cost in different user groups

Fig. 9. Cost performance of the deterministic algorithm with various prediction windows. All costs are normalized to the online deterministic algorithm (Algorithm 1) without any future information.



(a) Cost CDF (b) Average cost in different user groups

Fig. 10. Cost performance of the randomized algorithm with various prediction windows. All costs are normalized to the online randomized algorithm (Algorithm 1) without any future information.

can be accurately predicted *a priori*. However, it has been observed that in real production applications, ranging from enterprise applications to large e-commerce sites, workload is highly variable and statistically non-stationary [10], [11]. Furthermore, most workload prediction schemes, e.g., [26], [27], [28], are only suitable for predictions over a very short term (from half an hour to several hours). Such limitation is also shared by general predicting techniques, such as ARMA [29] and GARCH models [30]. Some long-term workload prediction schemes [31], [32], on the other hand, are reliable only when demand patterns are easy to recognize with some clear trends. Even in this case, the prediction window is at most days or weeks into the future [31], which is far shorter than the typical workload span. All these factors significantly limit the practical use of existing works.

Our online strategies are tied to the online algorithm literature [19]. Specifically, our instance reservation problem captures a class of rent-or-buy problems, including the ski rental problem [22], the Bahncard problem [20], and the TCP acknowledgment problem [21], as special cases when a user demands no more than one instance at a time. In these problems, a customer obtains *a single item* either by paying a repeating cost (renting) per

usage or by paying a one-time cost (buying) to eliminate the repeating cost. A customer makes *one-dimensional* decisions only on the timing of buying. Our problem is more complicated as a user demands *multiple instances* at a time and makes *two-dimensional* decisions on both the timing and quantity of its reservation. A similar "multi-item rent-or-buy" problem has also been investigated in [23], where cloud servers are dynamically turned on and off to serve time-varying workloads with a minimum energy cost. It is shown in [23] that, by dispatching jobs to servers that are idle or off the most recently, the problem reduces to a set of independent ski rental problems, through which online algorithms are proposed that achieve the best possible competitive ratios for both deterministic and randomized algorithms with or without future look-ahead information. Our problem does not have such a separability structure and cannot be equivalently decomposed into independent single-instance reservation (Bahncard) problems, mainly due to the possibility of time multiplexing multiple jobs on the same reserved instance as shown in Sec. 2.4. It is for this reason that the problem is challenging to solve even in the offline setting. In fact, the Bahncard problem contains the ski rental problem as a special case [20]. Even without the convenient separability structure as in [23], our proposed algorithms have achieved optimality in competitive ratio for the more general multi-instance Bahncard problem.

Besides instance reservation, online algorithms have also been applied to reduce the cost of running a file system in the cloud. The recent work [33] introduces a *constrained ski-rental problem* with extra information of query arrivals (the first or second moment of the distribution), proposing new online algorithms to achieve improved competitive ratios. This work is orthogonal to this paper as it takes advantage of additional demand information to make rent-or-buy decisions for a single item. Other related work orthogonal to this paper includes [34], where an online auction is designed for cloud providers to price cloud resources.

9 Concluding Remarks and Future Work

Acquiring instances at the cost-optimal commitment level for time-varying workloads is critical for cost management to lower IaaS service costs. In particular, when should a user reserve instances, and how many instances should it reserve? Unlike existing reservation strategies that require knowledge of the long-term future demands, we propose two online algorithms, one deterministic and another randomized, that dynamically reserve instances without knowledge of the future demands. We show that our online algorithms incur near-optimal costs with the best possible competitive ratios, i.e., $2-\alpha$ for the deterministic algorithm and $e/(e-1+\alpha)$ for the randomized algorithm. Both online algorithms can also be easily extended to cases when short-term predictions are reliable. Large-scale simulations driven by 40 GB Google cluster-usage traces further indicate that significant cost savings are derived from our online algorithms and their extensions, under the prevalent Amazon EC2 pricing.

One of the issues that we have not discussed in this paper is the combination of different types of reserved instances with different reservation periods and utilization levels. For example, GoGrid offers monthly and yearly reservations with 25% and 50% discounts, respectively, while Amazon EC2 offers 1-year and 3-year reserved instances with light, medium, and high utilizations. Effectively combining these reserved instances with on-demand instances could further reduce instance acquisition costs. We note that when a user demands no more than one instance at a time and the reservation period is infinite, the problem reduces to *Multislope Ski Rental* [35]. However, it remains unclear if and how the results obtained for Multislope Ski Rental could be extended to instance acquisition with multiple reservation options.

References

[1] W. Wang, B. Li, and B. Liang, "To reserve or not to reserve: Optimal online multi-instance acquisition in IaaS clouds," in *Proc. USENIX Intl. Conf. on Autonomic Computing (ICAC)*, 2013.

[2] "Gartner Says Worldwide Cloud Services Market to Surpass $109 Billion in 2012," https://www.gartner.com/it/page.jsp?id=2163616.

[3] "The Future of Cloud Adoption," http://cloudtimes.org/2012/07/14/the-future-of-cloud-adoption/.

[4] Amazon EC2 Pricing, http://aws.amazon.com/ec2/pricing/.

[5] ElasticHosts, http://www.elastichosts.com/.

[6] GoGrid Cloud Hosting, http://www.gogrid.com.

[7] Y. Hong, M. Thottethodi, and J. Xue, "Dynamic server provisioning to minimize cost in an IaaS cloud," in *Proc. ACM SIGMETRICS*, 2011.

[8] C. Bodenstein, M. Hedwig, and D. Neumann, "Strategic decision support for smart-leasing Infrastructure-as-a-Service," in *Proc. 32nd Intl. Conf. on Info. Sys. (ICIS)*, 2011.

[9] K. Vermeersch, "A broker for cost-efficient qos aware resource allocation in EC2," Master's thesis, University of Antwerp, 2011.

[10] C. Stewart, T. Kelly, and A. Zhang, "Exploiting nonstationarity for performance prediction," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3, pp. 31–44, 2007.

[11] R. Singh, U. Sharma, E. Cecchet, and P. Shenoy, "Autonomic mix-aware provisioning for non-stationary data center workloads," in *Proc. IEEE/ACM Intl. Conf. on Autonomic Computing (ICAC)*, 2010.

[12] AWS Case studies, http://aws.amazon.com/solutions/case-studies/.

[13] "Cloudability," http://cloudability.com.

[14] "Cloudyn," http://www.cloudyn.com.

[15] "Cloud Express by Apptio," https://www.cloudexpress.com.

[16] W. Powell, *Approximate Dynamic Programming: Solving the curses of dimensionality*. John Wiley and Sons, 2011.

[17] "Google Cluster-Usage Traces," http://code.google.com/p/googleclusterdata/.

[18] "Amazon ElastiCache," http://cloudability.com.

[19] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[20] R. Fleischer, "On the Bahncard problem," *Theoretical Computer Science*, vol. 268, no. 1, pp. 161–174, 2001.

[21] A. Karlin, C. Kenyon, and D. Randall, "Dynamic TCP acknowledgment and other stories about e/(e-1)," *Algorithmica*, vol. 36, no. 3, pp. 209–224, 2003.

[22] A. Karlin, M. Manasse, L. McGeoch, and S. Owicki, "Competitive randomized algorithms for nonuniform problems," *Algorithmica*, vol. 11, no. 6, pp. 542–571, 1994.

[23] T. Lu, M. Chen, and L. L. Andrew, "Simple and effective dynamic provisioning for power-proportional data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1611–1171, 2013.
[24] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimization of resource provisioning cost in cloud computing," *IEEE Trans. Services Comput.*, vol. 5, no. 2, pp. 164–177, 2012.
[25] W. Wang, D. Niu, B. Li, and B. Liang, "Dynamic cloud resource reservation via cloud brokerage," in *Proc. IEEE ICDCS*, 2013.
[26] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, "Energy-aware server provisioning and load dispatching for connection-intensive internet services," in *Proc. USENIX NSDI*, 2008.
[27] B. Guenter, N. Jain, and C. Williams, "Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning," in *Proc. IEEE INFOCOM*, 2011.
[28] Í. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, "Greenhadoop: leveraging green energy in data-processing frameworks," in *Proc. ACM EuroSys*, 2012.
[29] G. Box, G. Jenkins, and G. Reinsel, *Time Series Analysis: Forecasting and Control*. Prentice Hall, 1994.
[30] T. Bollerslev, "Generalized autoregressive conditional heteroskedasticity," *Journal of Econometrics*, vol. 31, no. 3, pp. 307–327, 1986.
[31] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal, "Dynamic provisioning of multi-tier internet applications," in *Proc. IEEE/ACM Intl. Conf. on Autonomic Computing (ICAC)*, 2005.
[32] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Capacity management and demand prediction for next generation data centers," in *Proc. IEEE Intl. Conf. on Web Services (ICWS)*, 2007.
[33] A. Khanafer, M. Kodialam, and K. P. N. Puttaswamy, "The constrained ski-rental problem and its application to online cloud cost optimization," in *Proc. IEEE INFOCOM*, 2013.
[34] H. Zhang, B. Li, H. Jiang, F. Liu, A. V. Vasilakos, and J. Liu, "A framework for truthful online auctions in cloud computing with heterogeneous user demands," in *Proc. of IEEE INFOCOM*, 2013.
[35] Z. Lotker, B. Patt-Shamir, D. Rawitz, S. Albers, and P. Weil, "Rent, lease or buy: Randomized algorithms for multislope ski rental," in *Proc. Symp. on Theoretical Aspects of Computer Science (STAC)*, 2008.

Wei Wang received the B.Engr. and M.A.Sc. degrees from the Department of Electrical Engineering, Shanghai Jiao Tong University, in 2007 and 2010. He is currently a Ph.D. candidate in the Department of Electrical and Computer Engineering at the University of Toronto. His general research interests cover the broad area of distributed system, with special emphasis on cloud computing, data analytic systems, and computer networks. He is also interested in problems at the intersection of cloud computing and economics.

Ben Liang received honors-simultaneous B.Sc. (valedictorian) and M.Sc. degrees in Electrical Engineering from Polytechnic University in Brooklyn, New York, in 1997 and the Ph.D. degree in Electrical Engineering with Computer Science minor from Cornell University in Ithaca, New York, in 2001. In the 2001 - 2002 academic year, he was a visiting lecturer and post-doctoral research associate at Cornell University. He joined the Department of Electrical and Computer Engineering at the University of Toronto in 2002, where he is now a Professor. His current research interests are in mobile communications and networked systems. He is an editor for the IEEE Transactions on Wireless Communications and an associate editor for the Wiley Security and Communication Networks journal, in addition to regularly serving on the organizational or technical committee of a number of conferences. He is a senior member of IEEE and a member of ACM and Tau Beta Pi.

Baochun Li received the B.Engr. degree from the Department of Computer Science and Technology, Tsinghua University, China, in 1995 and the M.S. and Ph.D. degrees from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 1997 and 2000. Since 2000, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, where he is currently a Professor. He holds the Nortel Networks Junior Chair in Network Architecture and Services from October 2003 to June 2005, and the Bell Canada Endowed Chair in Computer Engineering since August 2005. His research interests include large-scale multimedia systems, cloud computing, peer-to-peer networks, applications of network coding, and wireless networks. Dr. Li was the recipient of the IEEE Communications Society Leonard G. Abraham Award in the Field of Communications Systems in 2000. In 2009, he was a recipient of the Multimedia Communications Best Paper Award from the IEEE Communications Society, and a recipient of the University of Toronto McLean Award. He is a member of ACM and a Fellow of IEEE.

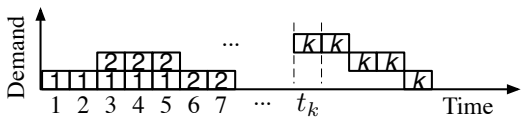


Fig. 11. A reservation can be pictorially represented as a reservation strip. For example, reservation 2 is represented as $(2, 2, 2, 1, 1)$ with $t_2 = 3$.

APPENDIX A
PROOF OF LEMMA 2

We first present a general description for a reserved instance, based on which we reveal the connections between reservations in A_β and OPT.

Suppose an algorithm reserves n instances 1, 2, ..., n at time $t_1 \leq t_2 \leq \cdots \leq t_n$, respectively. Also suppose at time t, the active reservations are $i, i+1, \ldots, j$. Let demand d_t be divided into levels, with level 1 being the bottom. Without loss of generality, we can serve demand at level 1 with reservation i, and level 2 with $i+1$, and so on. This gives us a way to describe the use of a reserved instance to serve demands. Specifically, the reservation k is active from t_k to $t_k+\tau-1$ and is described as a τ-tuple $(l^k_{t_k}, l^k_{t_k+1}, \ldots, l^k_{t_k+\tau-1})$, where l^k_t is the demand level that k will serve at time $t = t_k, \ldots, t_k + \tau - 1$. Such a tuple can be pictorially represented as a *reservation strip* and is depicted in Fig. 11.

We next define a *decision strip* for every reserved instance in A_β and will show its connections to the reservation strips in OPT. Suppose an instance is reserved at time t in A_β, leading x_i's to update in line 5 and 6 of Algorithm 1. We refer to the bottom figure of Fig. 3 and define the decision strip for this reservation as the region between the original x curve (the solid line) and the newly updated one (the dotted line). Fig. 12 plots the result, where the shaded area is derived from the top figure of Fig. 3. Such a decision strip captures critical information of a reserved instance in A_β. As shown in Fig. 12, the first τ times of the strip, referred to as the *pre-reservation part*, reveal the reason that causes this reservation: serving the shaded area on demand incurs more costs than the break-even point β. The last τ times are exactly the reservation strip of this reserved instance in A_β, telling how it will be used to serve demands. We therefore denote a decision strip of a reservation k as a $(2\tau-1)$-tuple $\mathbf{l}_k = (l^k_{t_k-\tau+1}, \ldots, l^k_{t_k}, \ldots, l^k_{t_k+\tau-1})$, where l^k_t is the demand level of strip k at time $t = t_k-\tau+1, \ldots, t_k+\tau-1$. The pre-reservation part is $(l^k_{t_k-\tau+1}, \ldots, l^k_{t_k})$, while the reservation strip is $(l^k_{t_k}, \ldots, l^k_{t_k+\tau-1})$.

We now show the relationship between decision strips of A_β and reservation strips of OPT. Given the demand sequence, Algorithm A_β makes n_β reservations, each corresponding to a decision strip. OPT reserves n_{OPT} instances, each represented as a reservation strip. We say a decision strip of A_β *intersects* a reservation strip of OPT if the two strips share a common area when depicted. The following lemma establishes their connections.

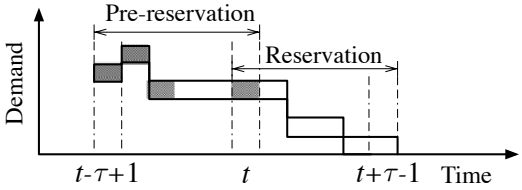


Fig. 12. Illustration of a decision strip for a reserved instance in Fig. 3.

Lemma 4: A decision strip of A_β intersects at least a reservation strip of OPT.

Proof: We prove by contradiction. Suppose there exists a decision strip of A_β that intersects no reservation strip of OPT. In this case, the demand in the pre-reservation part (the shaded area in Fig. 12) are served on demand in OPT, incurring a cost more than the break-even point (by the definition of A_β). This implies that the cost of OPT can be further lowered by serving these pre-reservation demands (the shaded area) with a reserved instance, contradicting the definition of OPT. □

Corollary 1: Any two decision strips of A_β intersect at least two reservation strips of OPT, one for each.

Proof: We prove by contradiction. Suppose there exist two decision strips of A_β, one corresponding to reservation i made at time t_i and another to reservation j made at t_j, $i < j$, that intersect *only one* reservation strip of OPT. (By Lemma 4, any two decision strips must intersect at least one reservation strip of OPT.) By the analyses of Lemma 4, the reservation strip of OPT intersects the pre-reservation parts of both decision strip i and j. It suffices to consider the following two cases.

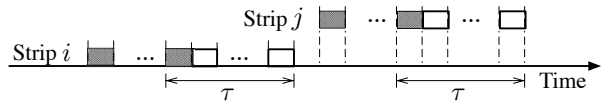
Case 1: Strip i and j do not overlap in time, i.e., $t_i + \tau - 1 < t_j - \tau + 1$. As shown in Fig. 13a, having a reservation strip of OPT intersecting the pre-reservation parts of both i and j requires a reservation period that is longer than τ, which is impossible.

Case 2: Strip i and j overlap in time. In this case, the reservation strip of OPT only intersects strip i. To see this, we refer to Fig. 13b. Because $i < j$, strip i is depicted below j, i.e., $l^i_t < l^j_t$ for all t in the overlap. Suppose the reservation strip of OPT intersects decision strip j at time t. Clearly, t must be in the overlap and $l^i_t < l^j_t$. This implies that at time t in OPT, OPT serves the demand at a lower level l^i_t by an on-demand instance while serving a higher level l^j_t via a reservation, which contradicts the definition of the reservation strip. □

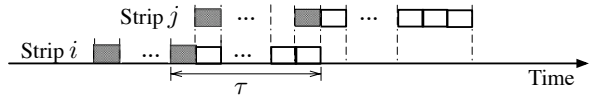
With Lemma 4 and Corollary 1, we see that the n_β decision strips of A_β intersect at least n_β reservation strips of OPT, indicating that $n_\beta \leq n_{\text{OPT}}$.

APPENDIX B
PROOF SKETCH OF LEMMA 3

Statement 1: Following the notations used in the proof of Proposition 1, let S be the total costs of demands when priced at the on-demand rate, and $\text{Od}(A_z)$ be the on-



(a) Strip i and j do not overlap in time.



(b) Strip i and j overlap in time.

Fig. 13. Two cases of strip i and j. They either overlap in time or not.

demand costs incurred by algorithm A_z. It is easy to see

$$\begin{cases} C_{A_z} &= n_z + (1-\alpha)\mathrm{Od}(A_z) + \alpha S; \\ C_{\mathrm{OPT}} &= n_{\mathrm{OPT}} + (1-\alpha)\mathrm{Od}(\mathrm{OPT}) + \alpha S. \end{cases} \tag{31}$$

Plugging (31) into (23), we see that it is equivalent to proving

$$\mathrm{Od}(A_z) \le \mathrm{Od}(\mathrm{OPT}) + z n_{\mathrm{OPT}} - D_z \ . \tag{32}$$

Denote by $\mathrm{Od}(A_z \backslash \mathrm{OPT}) := \sum_{t=1}^{T} (o_{z,t} - o_t^*)^+ p$ the on-demand costs incurred by A_z that are not incurred by OPT. With similar arguments as we made for (12), we see $\mathrm{Od}(A_z \backslash \mathrm{OPT}) \le z n_{\mathrm{OPT}}$. We therefore derive

$$\begin{aligned} \mathrm{Od}(A_z) &= \mathrm{Od}(\mathrm{OPT}) + \mathrm{Od}(A_z \backslash \mathrm{OPT}) - D_z \\ &\le \mathrm{Od}(\mathrm{OPT}) + z n_{\mathrm{OPT}} - D_z \ , \end{aligned}$$

which is exactly (32). □

Statement 2: For any given demands $\{d_t\}$, let $L(n, z)$ be the minimum, over all purchase decisions $D = \{r_t, o_t\}$ with n reserved instances, of the on-demand cost that has been incurred in D but has not been incurred in A_z, i.e.,

$$\begin{aligned} L(n,z) := \inf_{\{r_t,o_t\}} & \textstyle\sum_{t=1}^{T} (o_t - o_{z,t})^+ p \\ \text{s.t.} \quad & \textstyle\sum_{t=1}^{T} r_t = n, \\ & o_t + \textstyle\sum_{i=t-\tau+1}^{t} r_i \ge d_t, \\ & o_t, r_t \in \{0, 1, 2, \dots\},\ t = 1, \dots, T. \end{aligned} \tag{33}$$

We show that for any $u > v \ge z$,

$$L(n_u, z) \ge (v - z)(n_v - n_u) + L(n_v, z) \ . \tag{34}$$

To see this, let $D_u = \{r_{u,t}, o_{u,t}\}$ be the purchase decision that leads to $L(n_u, z)$ and define decision strips for A_v and reservation strips for D_u similarly as we did in Appendix A. With similar arguments of Corollary 1, we see that a reservation strip of D_u intersects at most one decision strip of A_v. As a result, among all n_v decision strips of A_v, there are at least $n_v - n_u$ ones that do not intersect any reservation strips of D_u. We arbitrarily choose $n_v - n_u$ such decision strips and denote their collections as B. Each of these decision strips contains a pre-reservation part with an on-demand cost v[3], of which at most z is also incurred on demand in A_z (by definition of A_z). As a result, an on-demand cost of at least $(v - z)(n_v - n_u)$ is incurred in both D_u and A_v that is not incurred in A_z, i.e.,

$$\sum_{t=1}^{T} (\min\{o_{v,t}, o_{u,t}\} - o_{z,t})^+ p \ge (v - z)(n_v - n_u) \ , \tag{35}$$

where $o_{v,t}$ is the number of on-demand instances launched by A_v at time t. We now reserve a new instance at the starting time of each decision strip in B. Adding to the existing reservations made by D_u, we have $D' = D_u \cup B$ as new reserving decisions with n_v reserved instances (because $|B| = n_v - n_u$). Therefore

$$\begin{aligned} L(n_u, z) &= \mathrm{Od}(D_u \backslash A_z) \\ &\ge \mathrm{Od}(D' \backslash A_z) + (v - z)(n_v - n_u) \\ &\ge L(n_v, z) + (v - z)(n_v - n_u) \ , \end{aligned} \tag{36}$$

where the second inequality holds due to the definition of $L(n_v, z)$ and the fact that D' consists of n_v reserved instances.

The rest of the proof follows the framework of [21]. Taking $u = v + \mathrm{d}v$ and integrating from z to w, for any $z < w \le \beta$, we have

$$L(n_w, z) - L(n_z, z) \ge \int_z^w n_v \mathrm{d}v - (w - z) n_w \ . \tag{37}$$

Observing that $L(n_z, z) = 0$, and that $n_v \le n_w$ for $v > w$, we have

$$L(n_w, z) \ge \int_z^\beta n_v \mathrm{d}v - (\beta - z) n_w \ . \tag{38}$$

Taking $n_w = n_{\mathrm{OPT}}$ and noting that $D_z \ge L(n_{\mathrm{OPT}}, z)$ yields the statement. □

Statement 3: By (31) and noting that $\mathrm{Od}(\mathrm{OPT}) \ge D_z$, we have

$$\begin{aligned} C_{\mathrm{OPT}} &\ge n_{\mathrm{OPT}} + \alpha S + (1 - \alpha) D_z \\ &\ge n_{\mathrm{OPT}} + D_z \ , \end{aligned} \tag{39}$$

where the second inequality is derived by noting $S \ge D_z$. Letting $z \to 0$ and plugging (24) establishes the statement. □

3. This is true when $p \ll 1$.