Virtual Machine Allocation with Lifetime Predictions

Hugo Barbalho [1*] Patricia Kovaleski [1*] Beibin Li [1*] Luke Marshall [1] Marco Molinaro [1] Abhisek Pan [2] Eli Cortez [2] Matheus Leao [2] Harsh Patwari [3] Zuzu Tang [2] Tamires Vargas Capanema Santos [1] Larissa Rozales Gonçalves [1] David Dion [2] Thomas Moscibroda [2] Ishai Menache [1]

Abstract

The emergence of machine learning technology has motivated the use of ML-based predictors in computer systems to improve their efficiency and robustness. However, there are still numerous algorithmic and systems challenges in effectively utilizing ML models in large-scale resource management services that require high throughput and response latency of milliseconds. In this paper, we describe the design and implementation of a VM allocation service that uses ML predictions of the VM lifetime to improve packing efficiencies. We design lifetime-aware placement algorithms that are provably robust to prediction errors and demonstrate their merits in extensive real-trace simulations. We significantly upgraded the VM allocation infrastructure of Microsoft Azure to support such algorithms that require ML inference in the critical path. A robust version of our algorithms has been recently deployed in production, and obtains efficiency improvements expected from simulations.

1 Introduction

Cloud computing has revolutionized the way computing resources are consumed. The emergence of cloud computing is attributed to lowering the risks for end-users (e.g., scaling-out resource usage based on demand) while allowing providers to reduce their costs by efficient management and operation at scale. However, to realize the full economic potential, cloud providers have to focus attention on maximizing the efficiency of cloud operations. This can be achieved by actions such as increasing resource utilization, reducing power stranding, and matching projected demand to hardware supply. Recent advances in Machine Learning (ML) tools and their accessibility in mature cloud offerings have thus motivated their internal use for improving cloud efficiency, e.g., in Microsoft (Cortez et al., 2017) and Google (Gao, 2014) datacenters.

Arguably, the "holy grail" of cloud resource management is the Virtual Machine (VM) allocation system. Customers rent VMs on demand until they decide to terminate usage. In turn, cloud resource managers place VMs on physical servers that have enough capacity to serve them. Allocation decisions have a direct impact on resource efficiency and return on investment. Inefficient placement might result in fragmentation and unnecessary over-provisioning of physical resources. In fact, even one percent of improvement in packing efficiency can lead to cost savings of hundreds of millions of dollars (Hadary et al., 2020).

From an algorithmic perspective, the VM allocation problem can be mapped to a (multi-dimensional) *dynamic* bin packing problem (Coffman et al., 1983). "Dynamic" here implies that each item (VM) is characterized by both its size (CPU, memory, etc.) and *lifetime* (i.e., the duration spent in the system) as opposed to static bin packing, which does not include a temporal dimension. While the size is known to the scheduler upon the VM request, lifetime can only be predicted. Our goal in this work is therefore to incorporate real-time ML predictions of lifetimes to guide the online allocation decisions.

Including lifetimes in VM allocator decisions poses several design challenges spanning ML algorithms, combinatorial algorithms, and systems. From an ML perspective, the prediction problem itself is quite involved because the actual lifetime has high variability even across VMs with very similar characteristics §2.4. Algorithmically, the online allocation mechanisms must be *robust* to prediction errors and perform adequately well even when the prediction accuracy is mediocre. Finally, the end-to-end system must produce predictions and make them available within a few milliseconds while ensuring high availability and reliability expected from cloud-scale systems.

In this paper, we design an end-to-end system that addresses the above challenges. Our system has been recently deployed in Microsoft Azure's infrastructure, serving millions

[*]Equal contribution [1]Microsoft Research, Redmond, USA [2]Microsoft, Redmond, USA [3]University of Washington, Seattle, USA. Correspondence to: Patricia Kovaleski <pkovaleski@microsoft.com>, Beibin Li <beibin.li@microsoft.com>, Ishai Menache <ishai@microsoft.com>.

Proceedings of the 6^{th} MLSys Conference, Miami Beach, FL, USA, 2023. Copyright 2023 by the author(s).

of VM requests per day. To enable this, we first had to carefully pick an ML algorithm that best addresses our accuracy, performance, and resource usage (e.g., memory footprint) requirements §3.2. Next, we designed new online algorithms with lifetime predictions; one important design choice here is to rely on a coarse-grained classification of lifetime, which is robust to prediction errors §3.1. Based on these ideas, we implemented a simple lifetime-aware allocation mechanism that accounts for different considerations and constraints beyond packing quality (e.g., "respect" other system objectives §4.2). Our system encapsulates in real-time the predicted class into the VM request and has fallback mechanisms in cases of excess prediction latencies and corrupt data. Furthermore, the system allows for efficient retraining of the underlying ML models in case of changing conditions. Our system design decouples control, inference, and allocation services, which can facilitate using ML for other aspects of the allocation problem in the future.

We evaluate our proposed solution in both simulations with real traces, and production measurements. To understand the possible benefits of lifetime prediction, our first set of simulations considers a setting where prediction accuracy is perfect. In this idealized setting, our lifetime aware algorithms outperform the previous packing algorithm that has been used in production until recently by $3-5\%$. We then study the effect of prediction errors and additional system constraints on a variety of lifetime aware algorithms. We demonstrate an interesting trend where a simple algorithm that uses the predicted information only implicitly (e.g., by prioritizing the packing quality of projected long-lived VMs) is more robust at higher prediction error levels. In contrast, some algorithms that use the VM lifetime more explicitly (intuitively, trying to match the lifetime of the VM with the lifetime of the node) perform better as the ML models become more accurate. Based on these results, we deploy in production an algorithm that strikes the right balance between simplicity and robustness. Production measurements taken from multiple regions reaffirm the benefits of lifetime awareness.

2 Background and motivation

2.1 VM allocation at scale

Public cloud providers such as Amazon Web Services, Microsoft Azure, and Google Cloud Platform maintain enormous inventories of millions of servers spread all over the world. For instance, Hadary et al. (2020) describe in detail the scale and organization of Microsoft Azure's cloud computing service. The cloud inventory that we consider in this paper is organized into more than 200 datacenters in over 60 regions. Each region can be further divided into one to three availability zones. An availability zone can comprise up to a few hundred thousand servers, which vary widely in configurations and capabilities. A single instance of the allocation service manages VM allocations over all servers in an availability zone while handling up to a few thousand VM allocation requests per minute.

The requests also vary tremendously in their resource and capability requirements. The requests are expressed in terms of VM sizes, where each size is associated with specific CPU (or GPU), memory, network, and disk requirements. A zone can support a large number of VM sizes. Requests can vary in other dimensions beyond VM size as well, such as by priority or fault domain constraints. The allocation service assigns each incoming request to a server that can fit the requested VM while attempting to optimize several objectives (which might also be conflicting). While live-migrations are possible, they are considered costly to both provider and customer, hence it is preferable to use them infrequently. With all this in mind, the overall allocation strategy must strike a delicate balance between making high-quality VM-server assignment over a large inventory and handling the peak throughput demand of diverse allocation requests. This description provides a fairly typical account of the challenges faced by the leading public cloud providers.

Despite these challenges, the scale and complexity of this space also provides system designers with unprecedented opportunities to impact business margin and customer satisfaction. In the context of packing efficiency, cloud providers have stated that even a percent point reduction in fleet fragmentation can lead to cost savings in the order of hundreds of million USD per year. This provides a strong motivation to continuously explore new strategies to achieve even relatively modest packing efficiency improvements in the cloud inventory.

As mentioned in the introduction, VM allocation is a *dynamic* bin packing problem, in which a VM may arrive and exit at arbitrary times. The *static* version of the problem, in which items never leave the system, is a classic optimization problem that has been studied for decades (Garey et al., 1972) with many algorithms such as First-Fit, Best-Fit, Next-Fit, etc. available; see (Christensen et al., 2017) for a recent survey. Traditionally, cloud providers have used heuristics based on these static algorithms to achieve reasonable packing efficiency (Hadary et al., 2020; Verma et al., 2015).

Hadary et al. (2020) reports that such heuristics can sustain a fleet-wide packing efficiency of around 80-90%[1]. Although they work fairly well in practice, improving it further has been challenging for many reasons. For instance, optimizing packing often conflicts with other objectives such as avoiding noisy-neighbor interference, reducing allocation time, or handling more requests concurrently. More fundamentally,

[1]In terms the *packing density*; see §4 for a precise definition.

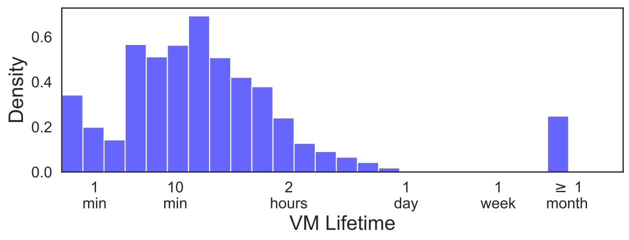

Figure 1. VM lifetime distribution.

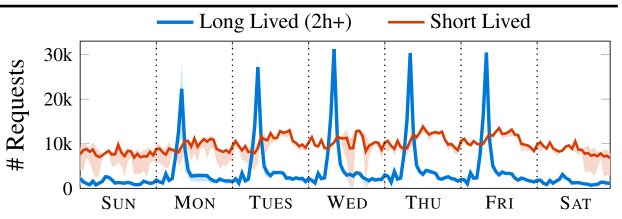

Figure 2. Number of VM requests per hour in an Availability Zone, averaged over day-of-week for a month, and split by their lifetime. Shaded regions represent 25th and 75th percentiles.

we show that using only static packing strategies (thereby ignoring temporal information) might lead to significant packing inefficiencies. However, using lifetime information for a public cloud workload is extremely challenging because this information is not provided by the customer, and estimating it is difficult due to workload diversity and lack of high-quality a-priori information about workload characteristics.

In this paper, we specifically focus on predicting and using VM lifetime information to improve the packing efficiency of public cloud inventory. We show that incorporating lifetime awareness can lead to significant improvements in packing efficiency. We also show that while achieving high-quality lifetime prediction can be very challenging for a public cloud environment, lifetime-aware packing logic can be designed to work with noisy predictions to achieve significant improvements.

2.2 Diversity and periodicity in VM lifetimes

Figure 1 shows a sampled lifetime distribution for VMs allocated in a public cloud instance across the globe over a period of three months. We see that VM lifetimes follow a long-tail distribution: while most VMs live in the system for less than one hour, some VMs can live longer. We can also see a small subset of VMs with lifetimes of more than a month. Interestingly, the median lifetime of VMs is about 16 minutes and the average is more than a day.

Figure 2 shows the distribution of long and short-lived requests over one week in an Availability Zone. We observe that VM requests show strong periodicity in their lifetimes and strong correlation between the time of day and the likelihood of staying longer in the system. In fact, using domain knowledge, we are aware that some long-lived VMs span the duration of the workday in the corresponding region. These repetitive patterns indicate the feasibility of VM lifetime prediction even for diverse and largely opaque public cloud workloads.

2.3 VM lifetimes matter

In this section, we provide an example to illustrate why using lifetime information is crucial for effective VM-machine assignments. As the baseline static strategy, we consider an "Any-Fit" algorithm that ignores VM lifetimes and assigns each incoming VM to some available non-empty machine where it fits, only consuming an empty machine if necessary (this encompasses the First-Fit, Best-Fit, etc.).

Example 1. *Suppose that all machines are identical, and each VM consumes $\frac{1}{k}$ resources of a machine, with integer $k \geq 2$. At time 0, $k-1$ "short" VMs with lifetime $L = 1$ arrive, and is immediately followed by a "long" VM with lifetime $L \gg 1$. At this point, the Any-Fit algorithm has allocated all these k VMs to a single machine. If we repeat this sequence k times (see Figure 3.a), with a tiny delay between repetitions, we see that k machines are used for L units of time with very low packing-density. In contrast, if we use lifetime information during allocation we can pack all k "long" VMs into one machine and use $k-1$ machines for "short" VMs (see Figure 3.b). This way, no machine resource is ever wasted.*

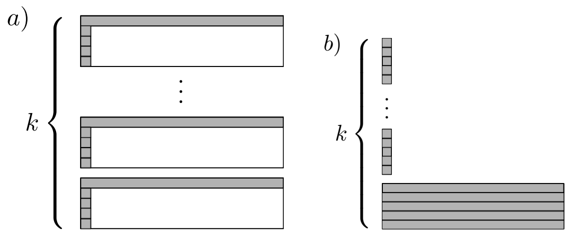

Figure 3. Each row represents a machine and each gray bar a VM. The horizontal axis represents time; (a) Allocation obtained by an Any-Fit algorithm ignores lifetimes with poor density and significant waste of resources; (b) Lifetime-aware allocation packs all "long" VMs on the same machine, with no wasted resources.

Our simulation studies also confirm the insight illustrated by the above example. In §5, we show that lifetime-aware packing algorithms can achieve up to 5% improvement in packing efficiency if we have perfect prediction of VM lifetimes. The algorithmic challenge here is that existing algorithms have not been designed with the goal of being robust to lifetime prediction errors produced by machine learning models, see §6. Indeed, as we discuss next, prediction error is prevalent in our production environment.

2.4 Real-world VM lifetime prediction is hard

Several factors make achieving high-quality lifetime predictions extremely challenging in a real-world production system. We find that VMs with similar features can still show significant variance in their lifetime distributions across different time points, which in turn requires extreme care in temporal feature extraction. The skewed and long-tailed lifetime distribution also makes high quality prediction challenging because common ML training approaches are tuned under Gaussian assumptions. Additionally, since inference is done as part of the hot-path in VM allocation request processing, strict milliseconds bounds on allocation latency might preclude the use of more sophisticated deep learning models or features that require high extraction latency (see §5). Several critical features are not available during extraction time either because they depend on the behaviors exhibited by running VMs or they are traditionally not relevant for allocation decision and hence are not available in the VM allocation workflow (e.g., the VM name or guest OS image). Lastly, prediction quality might be compromised by missing data due to temporary data losses in the storage system or systemic data pruning used to reduce system overheads.

3 Basic algorithms

In this section, we introduce fundamental algorithms for two crucial components: (1) allocating VMs with predicted lifetime information and (2) predicting VM lifetimes.

3.1 Robust lifetime-aware allocation algorithms

To design an allocation algorithm that is robust to lifetime prediction errors in a principled way, we start by considering a simplified theoretical model to guide our design decisions.

3.1.1 Allocation algorithm with provable guarantees

Model. In the *Dynamic Bin Packing Problem with Lifetime Predictions*, each VM j has a size (resource demand) $s_j \in [0, 1]$, a predicted lifetime $\hat{\ell}_j$, and an *unknown and random* true lifetime L_j (which becomes known only after the VM finishes). We assume the L_j's are independent random variables. The VMs $1, 2, \dots, n$ come one-by-one over time, and when VM j arrives it needs to be assigned to some unit-sized machine, without knowledge of future VMs. Migration of VMs is not allowed. The effectiveness of the assignment performed by the algorithm is measured as the total number of machines used over time, namely $cost = \int_0^T [\#\text{ machines used at time } t]\, \mathrm{d}t$, where T is the horizon of the problem. The goal is then to minimize this cost. Let OPT denote the cost of the *offline optimal solution*, namely the best solution possible when all VMs are known upfront, including the VMs' true lifetimes. We design an online algorithm that is *provably competitive* with this clairvoyant solution.

Algorithm. The intuition comes again from Example 1. VMs with "long" predicted lifetimes should be packed together to avoid unnecessary active machines. Since there is no clear threshold for declaring a VM "short" or "long", the algorithm considers multiple classes of predicted lifetime lengths and packs all VMs of a given class together in their dedicated machines. More precisely, for $i \in \mathbb{Z}$, let $I_i = \{j : \hat{\ell}_j \in [2^{i-1}, 2^i)\}$ be the set of all VMs for which the predicted lifetime $\hat{\ell}_j$ is in the interval $[2^{i-1}, 2^i)$.

Algorithm 1

```
Process each sub-instance $I_i$ separately (i.e., each $I_i$ has
its own set of machines) and allocate the VMs in $I_i$ to its
machines using First-Fit (i.e., among the machines that
can currently accommodate the VM's size, choose the
one with oldest opening time).
```

Notice that this algorithm can indeed be run online, since when a VM arrives we know to which sub-instance I_i it belongs based on its predicted lifetime.

We prove that this algorithm is competitive against the optimal offline solution, as long as the predictions are not "too far" from the true lifetimes, as captured in the following assumption.

Assumption 1. *There are constants $\alpha, \beta \geq 1$ such that for every job j:*

1. *$\hat{\ell}_j \geq \frac{1}{\alpha} \cdot L_j$ with probability 1*
2. *$\hat{\ell}_j \leq \beta \cdot \mathbb{E}L_j$*

Letting μ denote the ratio of largest to smallest lifetime in an instance, i.e., the smallest value such that $\frac{L_j}{L_{j'}} \leq \mu$ for all j, j', we have the following guarantee for our algorithm.

Theorem 1. *If Assumption 1 holds with parameters α, β, then Algorithm 1 has expected cost at most $O(\alpha\beta \log(\alpha\beta\mu)) \cdot \mathbb{E}\mathrm{OPT}$.*

We remark that no algorithm without predictions can do better than $O(\mu) \cdot \mathbb{E}\mathrm{OPT}$ (see (Li et al., 2014) and Theorem 2 in the appendix), which is much worse than the guarantee from Theorem 1 (for constant α, β, say). When there is *perfect* knowledge of the lifetimes, Azar & Vainstein (2019) gave an algorithm with guarantee $O(\sqrt{\log \mu}) \cdot \mathbb{E}\mathrm{OPT}$, which is best possible (Buchbinder et al. (2021) obtain improved approximation when additional information is available, namely the load of future jobs). Thus, the main contribution here is that our guarantee holds even under *noisy* lifetime information, which is inevitable in public cloud settings.

See Appendix A for the proof of Theorem 1, and a proof that Assumption 1 cannot be relaxed for such guarantees to hold.

3.1.2 The Lifetime Alignment algorithm

Using Algorithm 1 as a starting point, we design the *Lifetime Assignment* (LA) algorithm that improves over some shortcomings of our previous algorithm. LA introduces a new concept that has considerable impact in practice: It dynamically updates the lifetime classification of the machines based on currently running VMs. This contrasts with both Algorithm 1 and the previous algorithms of Azar & Vainstein (2019) and Buchbinder et al. (2021), which use static machine classification determined by the lifetime of the first VM assigned to it. Thus, previous algorithms might assign a long-lived VM to a machine that is about to become idle, which impacts efficiency.

In addition, LA handles different practical aspects of the problem that have not been covered in the above-mentioned papers, such as multi-dimensional allocation (e.g., CPU, memory, disk) and machines with heterogeneous capacities.

We now describe the LA algorithm, detailing these and other improvements:

1. We partition the lifetimes into an arbitrary set of intervals $I'_0, I'_1, \ldots, I'_k$ (I'_0 contains the smallest values), instead of the doubling-based partition of Algorithm 1. This allows us to use fewer intervals, making the algorithm more robust to prediction errors. We say that a VM is of *class i* if its predicted lifetime belongs to I'_i

2. Forcing the assignment of VMs to machines of their own class can be wasteful (e.g., if there are few VMs of a class). So, we assign a VM to an *existing* machine of the same class, if possible, else we assign it to a machine of different class. Class 0 ("small") VMs are an exception: no machine-class priority is used, since these VMs can help filling the leftover capacity of machines, and do not have a long-term impact on the system.

3. Since a machine may receive VMs of a wider range of lifetimes, we define the class of a machine *dynamically*. We look at the "predicted remaining lifetime" of the machine (i.e., the largest predicted amount of time that a VM in the machine will still take), and use the interval I'_i in which it falls as the class of the machine.

4. We use Best-Fit instead of First-Fit to assign a VM to one of its preferred machines, since the former typically produces better assignments. For Best-Fit, we define the aggregate "size" s_j of VM j as a weighted sum of its demand across the multiple resources and the same is done with the resource capacities of a machine i to obtain its "size" c_i and "current occupation" o_i. Best-Fit assigns VM j to the machine of largest occupation o_i among those where $s_j \leq c_i - o_i$ (i.e., where it "fits" considering these aggregate size/occupation/capacities).

With these considerations, we arrive at the LA algorithm, described in Algorithm 2.

Algorithm 2 Lifetime Alignment (LA)

```
For each incoming VM $j$
  $S \leftarrow$ set of active machines where VM $j$ can fit
  If $S$ is empty, assign VM $j$ to a new machine
  If $S$ is non-empty
    If VM $j$ is not of class 0, and there is a machine in
    $S$ of the same class as $j$, assign it to one of these
    machines using Best-Fit
    Otherwise, assign VM $j$ to a machine in $S$ using
    Best-Fit
```

3.2 Learning to predict lifetime

To resolve real-world challenges on label heterogeneity in VM lifetime prediction, we define lifetime-specific learning tasks, examine cloud operation-related features, and test different temporal feature extraction methods.

3.2.1 The learning task

Recall that our LA algorithm partitions the VMs into classes based on predicted lifetimes. From an ML perspective, this motivates the use of classification models. Accordingly, we discretize the lifetimes into buckets, and may perform either binary classification or multi-class classification (depending on number of classes that we choose for the LA algorithm). While the subsequent discussion on features and models is relevant for both of these classification tasks, we focus here on binary classification for simplicity; see Appendix §B.3 for details on multi-class classification models (we also examined regression models for other potential uses of VM lifetime, see §B.2). For binary classification, we must choose a cut off threshold that defines whether the VM is considered short or long lived. Based on some considerations that would be clarified later (see §4.4 and Appendix C), we have chosen that threshold to be two hours.

3.2.2 Features

We describe below some of the features that we use for classification. See §B.1 for a detailed account.

VM centric features. We examined accessible features within a VM request, including allocation time, operating system (OS), resource group, and other related information.

Customer centric features. The historical behavior of a customer is indicative of their future VM demands characteristics. As some evidence, Figure 4 shows that the normalized standard deviation (STD) of VM lifetimes for a given customer is fairly low for VMs that stay in the system for long periods. We thus include customers' VM lifetime statistics in the feature sets. *Such features are deidentified for compliance with privacy considerations.*

Additional temporal features. To automatically capture different behavioral features from the time-series of VM-request data of a customer, we also support using deep

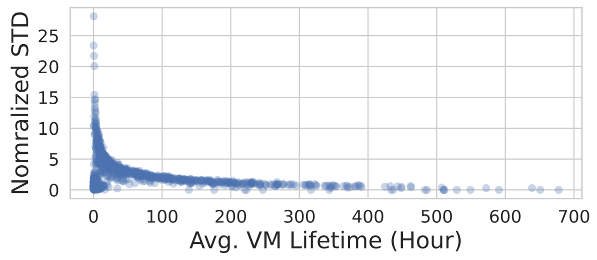

Figure 4. VM Lifetime from different customers, where each point represents one customer. The average VM runtime and normalized standard deviation STD (aka, coefficient of variation) for the customer are visualized.

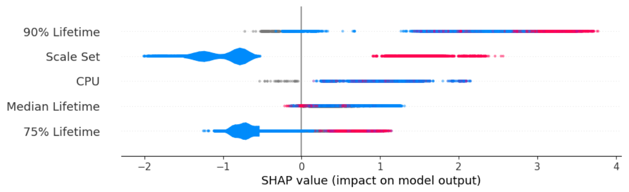

Figure 5. Importance of some representative VM features.

learning models. In particular, we may use recurrent neural networks and the central-attention mechanism (CAM) to extract a feature embedding vector for each deidentified customer; see Appendix §B.4 for more details.

Feature importance. Figure 5 evaluates representative features using SHapley Additive exPlanation (SHAP) values (Lundberg & Lee, 2017), which measure the additive feature importance, borrowing ideas from cooperative game theory.

3.2.3 *Models*

After examining several ML models, we decided to use LightGBM (LGB) (Ke et al., 2017) as our main model, and applied grid search and FLAML (Wang et al., 2021) for hyper-parameter tuning. Combining the inference speed advantage from LGB and feature representation strength from deep learning, we also applied the CAM to calculate customer embedding vectors offline, used as additional features to improve LGB's accuracy[2].

4 System Architecture and Implementation

The ML models and lifetime-aware allocation algorithms from the previous sections were integrated into the VM allocation system of Microsoft Azure. Figure 6 presents a high-level architecture of the implemented system. In the following section, we describe the details of both the offline and online components that comprise our design. We also highlight certain multi-objective optimization goals that go beyond packing quality. Finally, we discuss the practical challenges we have faced during the development and deployment of the system and our solutions thereof.

[2]We plan to use these for improved precision; see §5.1.

4.1 System design

Our system design must accommodate various real-world requirements. To start, prediction inference must take fewer than 30 milliseconds to avoid delays in VM allocations. Additionally, our lifetime-aware logic within the allocator must be robust to prediction errors while accounting for additional system considerations (described in §4.2–§4.3).

The system infrastructure has to provide a framework to train, test, deploy, and validate different ML models in production. A schematic view of the infrastructure is given in Figure 6. At a high level, we split the learning and deployment into *offline* and *online* components. Offline components process historical cloud data and train ML models that are consumed by the online system. The online system is comprised of the following subsystems: i) *Control service* to process incoming VM requests, ii) *Inference servers* that provide lifetime predictions, and iii) *Allocation servers* that decide the physical VM placement.

4.1.1 *Offline model and feature updates*

As described in §3.2.2, our model exploits features of the specific VM request, as well as aggregated statistics calculated from its customer. Due to the fluid nature of client usage patterns, the underlying data distributions change over time. Thus, our ML models and customer features require frequent training or fine-tuning and recalculation to maintain prediction quality. Figure 14 in Appendix shows the model performance drop as the time since the last training period increases. The training and updates are performed offline using historical datasets computed from daily logs of VM workloads; we provide some details below.

Training: A set of MapReduce jobs aggregate the workload data on a weekly basis to produce a complete dataset, which is then sampled appropriately for training and validation.

Customer features: These are updated daily via MapReduce jobs. The outcome of the feature generation process is stored in our feature store (Kakantousis et al., 2019) that is part of the Inference Server. An example of such data is the average lifetime of VMs from each customer.

Quality monitoring: All of the predictions from our model are logged and periodically compared against the actual lifetime of VMs. This allows for additional post-hoc analysis that can offer more insights and lead to adjustments of the model training or validation.

4.1.2 *Online services*

Machine Learning Pipeline. When a VM request enters the system, the control service processes it and relies on the ML client library to inject a lifetime prediction into its traits before the allocation servers start processing the

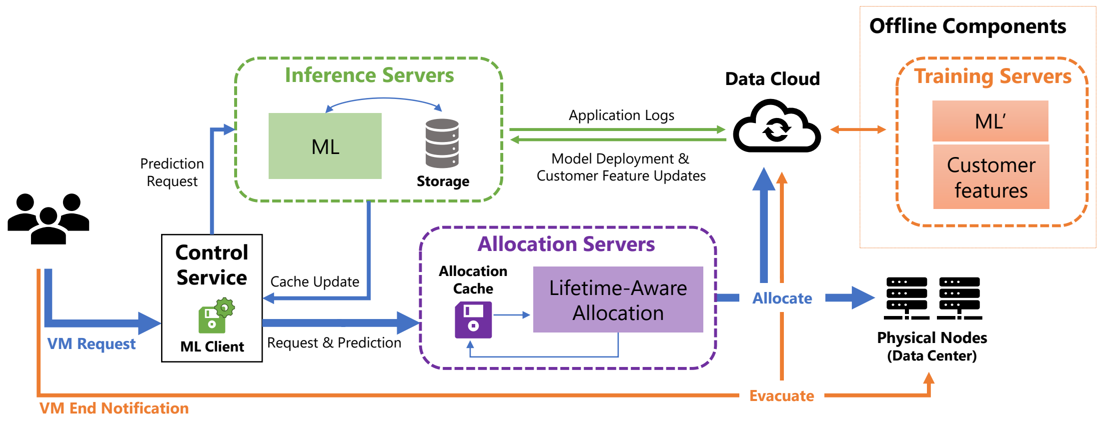

Figure 6. The allocation framework. ML training, inference, and allocations are separated into different servers. The ML model is periodically updated by the offline training servers. Local storage in the inference servers includes features store, client library, customer embedding store, previous model store. The data cloud stores all the logs and information generated during the VM allocation and the deallocation processes. Other details, such as the allocator communications with nodes and data centers, are omitted here for brevity.

request. If the inference takes too long, the request is processed without a prediction. The exact time-budget for each request depends on the current demand of requests, number of allocation servers, inventory size, etc., but this budget is typically small (milliseconds). Thus, our ML pipeline is designed to be as fast as possible. Our system reduces network call latency overheads by implementing two strategies as part of the ML client library: (1) pre-loading artifacts independently of allocation server calls and (2) caching lifetime predictions in the control service (see §4.3). Pre-loading artifacts consists of downloading static feature data and business policies that can be used for the execution of the prediction requests.

The inference service is deployed in multiple Availability Zones (AZs) through phased deployment, where the model is validated in a set of AZs before moving forward, taking up to three days from training to reach all clusters. A validation failure results in an on-call investigation and reverting to the previous model until the validation passes.

Allocation Pipeline. As Figure 6 shows, the allocation servers receive the incoming VM allocation request along with the lifetime prediction. The servers handle availability requirements (e.g., fault tolerant, geographical diversity requirements) and determine the suitable node for an incoming VM using a collection of rules. A small subset of these rules addresses the packing quality of the incoming VM; we refer to this subset in short as the *packing logic*. In this work, we expand and refine the packing logic to account for the predicted lifetime of the VM. More details about the allocator system can be found in the Protean paper (Hadary et al., 2020); in fact, the work we present here builds on and improves upon the basic Protean infrastructure.

4.2 Allocation objectives

We consider the following metrics when evaluating the quality of our packing logic:

- Packing Density (PD): PD measures the average number of allocated cores on non-empty machines. Formally, the packing density at any given time is the ratio between the number of allocated cores and the total number of cores on the non-empty machines.[3] In practice, PD is a better proxy for packing quality than the number of used nodes (which was used for our theoretical guarantees in §3.1) since it normalizes load across a heterogeneous fleet.
- Filtering Factor (FF): As mentioned above, the allocation is determined through a collection of rules evaluated sequentially. Each rule can be viewed as trimming down the set of candidate nodes that are best for the given VM. We define the FF to be the percent of nodes that are filtered out by the packing logic; namely the difference between the number of candidate nodes before and after the packing logic is evaluated, divided by the total size of the inventory. Here, a smaller value is considered better, as downstream rules (which are evaluated after the packing logic) would still be meaningful. Aggressive rules with high FF values can lead to a small number of candidate nodes for subsequent rules, making them ineffective. Thus, it is essential to prevent over-filtering of nodes by allocation rules. We allow relatively high FF values for packing related rules due to their direct impact on efficiency.
- Rejections: A rejection occurs when a VM request cannot be satisfied. Rejections should be kept to a minimum (e.g., less than 0.1% of VM requests).

[3]PD can be defined similarly for other resources, such as memory; we focus on CPU as it is typically the bottleneck resource.

When designing the packing logic, we have to take into account an inherent trade-off between the packing quality metrics (PD) and the filtering factor (FF). Intuitively, the more sophisticated or nuanced the packing logic is, the better the chances that the packing quality will increase, albeit at the expense of higher FF.

4.3 Accounting for practical challenges

Rule smoothing. As described earlier, the allocation logic consists of a hierarchy of many rules that address different business needs and considerations. Having a strict prioritization among these rules requires "smoothing" some of them so that all rules can contribute; see (Hadary et al., 2020) for more background and motivation for rule smoothing. One way of smoothing rules is by quantizing the score of some rules into a small number of buckets. For example, we have the Prefer Best Fit rule (PBFR) which scores the nodes based on how well they will be packed after the insertion of the requested VM (akin to the best-fit heuristic (Panigrahy et al., 2011))[4]; the original score, which is continuous, is quantized to a small number of discrete values. Fortunately, our new lifetime aware logic (see Algorithm 2) already has a discrete structure, in which a VM is assigned to a *class* of nodes based on its predicted lifetime. Hence, we can easily translate that logic into a rule that is not too aggressive.

Caching of inference results. An AZ can receive up to thousands of allocation requests per minute at peak. Without any further enhancements, we observed that the ML system struggles to fulfill all request bursts within the time budget, resulting in a drop of almost 2% in service rate. To address this, we observe that ML inference exhibits "locality". For example, a customer may request a large number of VMs of the same type. We thus cache inference results at each control service, indexed by the request type. We achieve a cache hit close to 60% (this can be improved in the future with distributed caching optimizations).

These inference results are cached in the ML client using an in-memory data structure indexed by the set of features that the VM lifetime model takes as input. Due to the use of temporal features, entries in the cache automatically expire after some time. This serves to maintain a manageable cache size. We allowed the cache size to grow as needed to accommodate bursts and observed in production that the cache size never exceeds a few megabytes, as shown in Figure 7.

ML availability. Predictions from the inference servers may not always be available due to a variety of system related issues such as server failures, disk overheating, computation

[4]Since the allocation is multi-dimensional, we use a weighted sum over the different resources; higher weights are given to scarce resources (Panigrahy et al., 2011; Hadary et al., 2020).

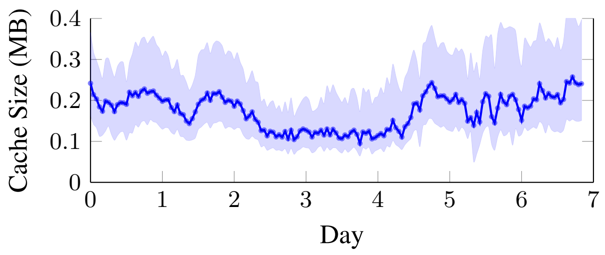

Figure 7. Memory footprint of the cache holding ML inference results. The graph depicts measurements extracted from 30 machines over a one-week period. The solid line represents the median cache size and the shaded region captures the $25\% - 75\%$ confidence interval.

bottlenecks, extreme request bursts, and networking problems. Another potential issue is trying to infer the request of a completely new customer. When the inference servers fail to yield lifetime prediction, our packing logic relies only on best-fit (i.e., accounting only for the size of the VM). More details are provided in §4.4.

4.4 Putting it all together

ML Model. After a careful study, we decided to deploy an ML model based on LGB, which uses easily accessible features from the feature store during inference time. As we elaborate in §5.1, we made this choice by evaluating various trade-offs in performance, model size, inference speed, and interpretability for human administrators. While we have trained models with different numbers of classes, we currently use in production a simple binary classification model (i.e., the VM is predicted to be either short or long lived). Overall, the deployed ML model produces an inference within the time budget for 99.5% of VM requests (see §5.3 for more statistics). As a fallback for inference timeout, we simply set the prediction to be "short lived". This is based on the intuition that classifying a short lived VM as long-lived has worse consequences than vice-versa (cf. Example 1).

Packing logic. We have implemented packing logic that accounts for lifetime while taking into account the system constraints discussed above (§4.3). We developed two different versions of this logic that differ by how they account for the lifetime information.

V1. The first version is a simple modification of PBFR (Prefer Best Fit rule), which we term DPBFR ('D' for dynamic). If the VM is predicted to be long-lived, use k_l buckets for the best-fit score quantization; otherwise use k_s buckets, where $k_l > k_s$ (in our production deployment we use $k_l = 5$, $k_s = 3$). The idea behind this rule stems from the intuition that long-lived VM matter more when it comes to their placement. Thus, intuitively, we wish to use more of our "FF budget" for these long-lived VMs (see

Appendix C.1 for details)[5].

V2. The second version involves more explicit use of the VM lifetime prediction. We introduce a new rule termed Lifetime Awareness Rule (LAR), which is essentially a two-class version of Algorithm 2. To further account for the VM size itself, LAR is followed by PBFR (with k_s buckets).

Due to its simplicity and excellent robustness properties (see §5.2), we currently have V1 deployed in production. We plan to integrate V2 in the future as we continue to refine our ML models.

Safeguard. While we have carefully tested our system and algorithms, we must take additional necessary precautions when relying on ML. In particular, we wish to prevent a scenario where the new packing logic exhibits a very high filtering factor (FF), e.g., due to mistakenly classifying too many VMs as longed-lived because of some unforeseen data issue. To address this, we implement a *safeguard* that mitigates such behavior. The safeguard is based on the following property: if all VMs were to be classified as short-lived, then we effectively end up with just the original PBFR (this property holds for both V1 and V2).

The safeguard therefore keeps track of the amount of requests that have been classified as longed-lived in the last T minutes (say, $T = 10$). If the percent of such classifications exceeds a certain threshold (say, 50%) then all following requested will be statically classified as short-lived for a certain timeout period. We note that the safeguard is rarely invoked (less than 1% of the requests are impacted by the safeguard), hence it does not have a meaningful impact on our metrics.

5 Evaluation

In this section, we ask the following questions: (1) what is the preferred choice of an ML model for our purposes, taking into account accuracy, performance, and resource overheads? §5.1 (2) What is the potential benefit of our lifetime-aware packing and how does our lifetime-aware packing perform in the face of inaccurate predictions? §5.2 (3) What are the actual performance gains that we obtain in our large-scale production deployment? §5.3

5.1 ML model evaluation

Training and testing. We have tested our ML models for a period of three months in 2022. At each weekend, we re-train models using the previous week's data and then apply the latest models for the following week.

Metrics. We use area-under-ROC-curve (AUC), Average Precision (AP), and F-1 score as the main accuracy metrics. A detailed tuning analysis with precision, recall, and Precision@Recall scores are in Table 3 in the Appendix.

[5] We note that the cutoff threshold for classifying short vs. long lived VMs also affects the FF. The smaller this threshold is, the more VMs would be classified as long-lived, which in turn would increase FF.

Table 1. Machine learning performance over 3 months. Random coin flip would result in a F-1 score of 17%.

Features	ML	t_{inf} (μs)	AP	F-1	AUC
Small	LGB	0.1	46%	45%	89%
	CAM	0.3	73%	47%	84%
	L + G	0.2	47%	45%	89%
	L + C	0.2	50%	47%	90%
Large	LGB	0.2	62%	62%	94%
	CAM	0.4	63%	63%	93%
	L + G	0.2	63%	63%	94%
	L + C	0.2	63%	64%	95%

Results. Table 1 shows various ML models' performance with two different sets of features, termed *small* and large (the small set is a subset of the large set); see Appendix §B.1. Choosing a model is determined based on multiple criteria including the inference memory footprint, the inference time per VM request (t_{inf}), and the prediction accuracy of the model.

The LGB method has the smallest memory footprint with 20 MB (for the small set) and 51 MB (for the large set) per million customers. Other methods require loading temporal feature embeddings extracted from deep learning models, resulting in at least 40X memory footprint (implying also larger caches). The CAM model incurs 4X inference latency compared to LGB, while the other hybrid models (i.e., L + G = LGB with GRU features, L + C = LGB with CAM features), with their relatively lightweight backbones, still incur up to 2X inference latency compared to LGB.

From a prediction accuracy standpoint, all of the models display similar characteristics. Notably, CAM achieves 17% higher AP but 5% lower AUC over LGB for the small feature set. This is because CAM focused more on the subtleties of short lived VMs. As such, the benefits of using CAM over LGB are doubtful. Additionally, the deep learning models take longer time to train. In summary, LGB exhibits low inference latency, small memory footprint, and competitive prediction accuracy when compared to other models; hence, LGB is our current chosen model for production deployment.

5.2 Evaluating the benefits of lifetime-aware packing

The production system evolves continuously and multiple changes may affect packing quality. Hence, simulations serve as an important tool for algorithmic comparisons. In this section, we rely on realistic simulations of the allocation system with the same testing dataset used in §5.1.

Methodology. Our simulations use real traces and configurations from several zones as input and can be considered

Table 2. Performance under idealized setting. Results are averaged over ten different instances.

Method	Density Avg. (STD)	Imp. (%)
Best Fit (no quant.)	82.12% (± 1.80%)	-
Lifetime Alignment	85.06% (±0.05%)	3.58%
OFFLINE	90.11%	9.73%

a reasonably accurate representation of reality. In particular, our historical traces from production (spanning three months) include VM requests, the actual lifetimes of the VMs, and when required, the ML predictions of the lifetimes. Our event-driven simulator includes a lightweight emulation of the allocator (e.g., supports a subset of the rules). The simulator still provides an excellent approximation of the system and is able to scale adequately to large inventories.

In addition to simulating online allocation algorithms through our simulation, we have implemented an *offline* combinatorial heuristic (OFFLINE) that obtains perfect knowledge of all VM requests as input. OFFLINE helps establish the potential range of algorithmic improvement.

Results. To understand the potential benefits of lifetime awareness, we first consider an *idealized* (unrealistic) setting where the lifetime predictions are perfect. Furthermore, for the sake of the experiment, we relax rule smoothness considerations (see §4.2). Under this setting, the baseline algorithm is a non-quantized version of best-fit, where a VM is simply allocated to the node that leaves the least amount of node fragmentation (see 4.3); from a packing perspective, the non-quantized best-fit is obviously better than its quantized version. We compare this better baseline to our Lifetime Alignment (LA) algorithm, and obtain PD improvements of around 3.5% (Table 2), a huge gain which can be attributed to lifetime awareness. For reference, we also compare the results to OFFLINE, which can be regarded as an upper bound for any online algorithm. The results show us that the maximum margin for improvement is around 10%, however most of it might not be achievable due to the shortcomings of online vs. offline allocations.

We next examine the performance of our algorithms in a more realistic setting where the ML predictions are prone to errors and we further have to smoothen our packing logic rules. Figure 8 summarizes our results. The vertical line in the figure corresponds to simulations with the actual outputs of the ML model. As shown, DPBFR (recently deployed in production) obtains average gains of 0.6% over PBFR, while the binary version of our LA algorithm has larger gains of around 1.5%. Based on these results, we expect to gradually deploy the latter in production. Interestingly, the multi-class version of the LA algorithm is less robust to prediction errors and achieves negligible gains over baseline. We note that the addition of lifetime awareness has not affected the FF, which is roughly the same for all algorithms, including the

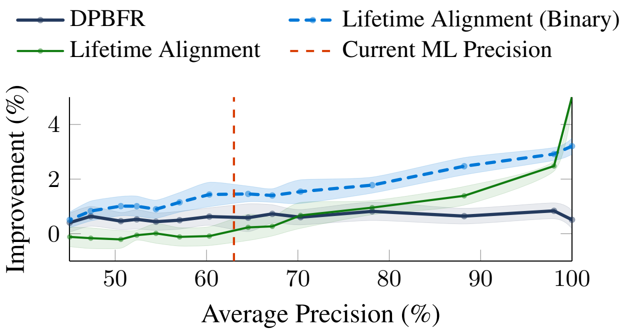

Figure 8. Real-world data simulation with noise

baseline (ranges between $21 - 25\%$); the rejections levels are negligible for all algorithms.

To further understand the impact of prediction noise (and in particular, the potential gains that can be obtained with better ML precision), we have synthetically modified the prediction outcomes so that they correspond to different values of average precision (see Appendix §B.6 for details on our noise generation methodology). We observe a few key trends: while DPBFR gains are modest compared to the LA algorithms, DPBFR exhibits quite stable performance across the spectrum, which is a desirable property for cloud providers (e.g., for capacity planning). Furthermore, the binary version of LA obtains the best improvement almost throughout, but it is surpassed by the multi-class version at around 97% precision; such levels of precision are highly unlikely in practice, hence we do not expect to transition to that version.

5.3 Real-World Production Results

We next provide concrete evidence that the system we developed and deployed in production can perform according to our requirements in regards to system performance, machine learning prediction quality, and allocation quality.

ML system performance. To measure the effectiveness of our solution, specifically the ML client library and the inference service, we collected production telemetry to evaluate the ML cache policy hit rate and, when there is a cache miss, the latency to get the prediction from inference servers and make them available to allocation servers.

On average, we process around *20 Million* prediction requests daily. Of these, *12 million* are locally processed and resolved by the ML client library and only *8 million* are processed by the inference servers; namely, we observe a 60% cache hit rate.

Figure 9 shows the VM lifetime latency distribution and different percentiles for uncached requests. The 50^{th}, 95^{th}, and 99^{th} percentile values are at 7.0, 20.0, and 40.0 milliseconds, respectively – less than 2% of the prediction requests that were not available in the cache do not meet the

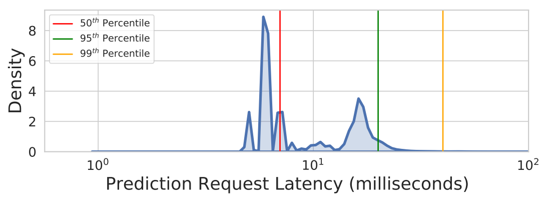

Figure 9. VM lifetime latency distribution and percentiles.

time budget requirement. Combined with the cache impact, 99.2% of requests meet the time budget.

We can calculate performance metrics by comparing the production predictions given by the model with the ground truth observed after the VM departs. Recall that the model deployed in production is an LGB with the small feature set in Table 1. Sampling over 200k predictions over all zones, we have an AUC of 89.6% and an AP of 59.2%, which is slightly better than the expected performance from simulations.

Allocation quality. It is quite challenging to assess the impact of our new algorithms in a production environment. The production allocation system is continually evolving, with many changes potentially affecting the packing quality simultaneously. In addition, changes in workload can significantly influence results. Therefore, we need to carefully choose metrics that directly measure the impact of our changes. One such metric that is robust to environment changes is the *instantaneous* packing density at each allocation. Specifically, for each VM request, we can measure the utilization of its selected node. If we partition the requests into short and long lived, we expect that, on average, VMs that have been predicted to be long lived will have improved packing density. To measure this in production, we evaluated ten clusters over a period of three months and observed that there was an instantaneous packing density improvement of 2% for long lived VMs, on average.

6 Related Work

Resource management for large clouds. Our work adds to a large body of work on hyper-scale cloud computing clusters, see (Verma et al., 2015; Burns et al., 2016; Schwarzkopf et al., 2013; Delimitrou & Kozyrakis, 2013; 2014; Delimitrou et al., 2015; Newell et al., 2021; Tang et al., 2020; Goudarzi & Pedram, 2012; Gharehpasha et al., 2021) and references therein.

ML for cloud systems. ML for systems has been a very active area of research over the last decade or so; see, e.g., (Wu & Xie, 2022) for a survey. We briefly survey below recent related works on ML for cloud systems, which are more relevant to our context. The Resource Central paper (Cortez et al., 2017) lays out key principles for using ML for cloud resource management. However, while the paper motivates some scenarios, there is no concrete guidance for how to use lifetimes for VM allocation. Ambati et al. (2020) applies ML models to predict survival rates and average number of cores to accommodate harvest VMs. Li et al. (2021); Wang et al. (2020) optimize the placement of long-running application (LRA) containers through reinforcement learning techniques. Wang et al. (2022) applies ML to increase node agent efficiency in cloud platforms. Finally, Kumbhare et al. (2021) uses ML to predict future workload for power oversubscription. None of the above papers utilizes real-time predictions of VM lifetimes.

Dynamic bin packing. Different versions of dynamic bin packing with perfect lifetime information have been studied for decades (Coffman et al., 1983), see (van Stee, 2012) for a survey. This has seen renewed interest motivated by VM allocation; the recent works (Li et al., 2015a; Azar & Vainstein, 2019; Buchbinder et al., 2021) present online algorithms under the assumptions, respectively, of no lifetime information, perfect lifetime information, and lifetime plus total load information. Offline versions of the problem have also been studied, see (Brandao & Pedroso, 2016; Aydın et al., 2020) and references therein. Li et al. (2015b) present an application of dynamic bin packing to VM allocations in cloud gaming.

7 Conclusion

We design and implement an end-to-end lifetime-aware VM allocation system, which serves millions of VM requests per day. Our allocation algorithms are robust to prediction errors and lead to efficiency gains, observed both in simulation and production measurements. Our system design decouples control, inference, and allocation services, allowing us to infuse ML into other aspects of the allocation problem in the future (e.g., over-subscription and live migration).

8 Acknowledgements

We thank the anonymous reviewers for their valuable suggestions. We also thank Saurabh Agarwal, Konstantina Mellou and Ricardo Bianchini for useful discussions.

References

Ambati, P., Goiri, Í., Frujeri, F., Gun, A., Wang, K., Dolan, B., Corell, B., Pasupuleti, S., Moscibroda, T., Elnikety, S., et al. Providing slos for resource-harvesting vms in cloud platforms. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 735–751, 2020.

Aydın, N., Muter, İ., and Birbil, Ş. İ. Multi-objective temporal bin packing problem: An application in cloud computing. *Computers & Operations Research*, 121:

104959, 2020.

Azar, Y. and Vainstein, D. Tight bounds for clairvoyant dynamic bin packing. *ACM Trans. Parallel Comput.*, 6 (3), oct 2019. ISSN 2329-4949. doi: 10.1145/3364214.

Brandao, F. and Pedroso, J. P. Bin packing and related problems: General arc-flow formulation with graph compression. *Computers & Operations Research*, 69: 56–67, 2016.

Buchbinder, N., Fairstein, Y., Mellou, K., Menache, I., and Naor, J. Online virtual machine allocation with lifetime and load predictions. *ACM SIGMETRICS Performance Evaluation Review*, 49(1):9–10, 2021.

Burns, B., Grant, B., Oppenheimer, D., Brewer, E., and Wilkes, J. Borg, Omega, and Kubernetes. *Queue*, 14(1): 70–93, January 2016. ISSN 1542-7730. doi: 10.1145/2898442.2898444.

Christensen, H. I., Khan, A., Pokutta, S., and Tetali, P. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, 24: 63–79, 2017.

Coffman, Jr, E. G., Garey, M. R., and Johnson, D. S. Dynamic bin packing. *SIAM Journal on Computing*, 12 (2):227–258, 1983.

Cortez, E., Bonde, A., Muzio, A., Russinovich, M., Fontoura, M., and Bianchini, R. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pp. 153–167, 2017.

Crow, E. L. and Shimizu, K. *Lognormal distributions*. Marcel Dekker New York, 1987.

Delimitrou, C. and Kozyrakis, C. Paragon: QoS-aware scheduling for heterogeneous datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 77–88, 2013. ISBN 9781450318709. doi: 10.1145/2451116.2451125.

Delimitrou, C. and Kozyrakis, C. Quasar: Resource-efficient and QoS-aware cluster management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 127–144, 2014. ISBN 9781450323055. doi: 10.1145/2541940.2541941.

Delimitrou, C., Sanchez, D., and Kozyrakis, C. Tarcil: reconciling scheduling speed and quality in large shared clusters. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, pp. 97–110, 2015.

Diederik, P. K., Welling, M., et al. Auto-encoding variational bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, volume 1, 2014.

Eban, E., Schain, M., Mackey, A., Gordon, A., Rifkin, R., and Elidan, G. Scalable learning of non-decomposable objectives. In *Artificial intelligence and statistics*, pp. 832–840. PMLR, 2017.

Gao, J. Machine learning applications for data center optimization. 2014.

Garey, M. R., Graham, R. L., and Ullman, J. D. Worst-case analysis of memory allocation algorithms. In *Proceedings of the Fourth Annual ACM Symposium on Theory of Computing*, STOC '72, pp. 143–150, New York, NY, USA, 1972. Association for Computing Machinery. ISBN 9781450374576. doi: 10.1145/800152.804907.

Gharehpasha, S., Masdari, M., and Jafarian, A. Virtual machine placement in cloud data centers using a hybrid multi-verse optimization algorithm. *Artificial Intelligence Review*, 54:2221–2257, 2021.

Goudarzi, H. and Pedram, M. Energy-efficient virtual machine replication and placement in a cloud computing system. In *2012 IEEE Fifth International Conference on Cloud Computing*, pp. 750–757. IEEE, 2012.

Hadary, O., Marshall, L., Menache, I., Pan, A., Greeff, E. E., Dion, D., Dorminey, S., Joshi, S., Chen, Y., Russinovich, M., et al. Protean: VM Allocation Service at Scale. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 845–861, 2020.

Kakantousis, T., Kouzoupis, A., Buso, F., Berthou, G., Dowling, J., and Haridi, S. Horizontally scalable ml pipelines with a feature store. In *Proc. 2nd SysML Conf., Palo Alto, USA*, 2019.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

Kumbhare, A. G., Azimi, R., Manousakis, I., Bonde, A., Frujeri, F., Mahalingam, N., Misra, P. A., Javadi, S. A., Schroeder, B., Fontoura, M., et al. Prediction-based power oversubscription in cloud platforms. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 473–487, 2021.

Li, S., Wang, L., Wang, W., Yu, Y., and Li, B. George: Learning to place long-lived containers in large clusters with operation constraints. In Curino, C., Koutrika, G., and Netravali, R. (eds.), *SoCC '21: ACM Symposium on Cloud Computing, Seattle, WA, USA, November 1 - 4,* *2021*, pp. 258–272. ACM, 2021. ISBN 978-1-4503-8638-8. doi: 10.1145/3472883.3486971.

Li, Y., Tang, X., and Cai, W. On dynamic bin packing for resource allocation in the cloud. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '14, pp. 2–11, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450328210. doi: 10.1145/2612669.2612675.

Li, Y., Tang, X., and Cai, W. Dynamic bin packing for on-demand cloud resource allocation. *IEEE Transactions on Parallel and Distributed Systems*, 27(1):157–170, 2015a.

Li, Y., Tang, X., and Cai, W. Play request dispatching for efficient virtual machine usage in cloud gaming. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(12):2052–2063, 2015b.

Lundberg, S. M. and Lee, S.-I. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

Moerkotte, G., Neumann, T., and Steidl, G. Preventing bad plans by bounding the impact of cardinality estimation errors. *Proceedings of the VLDB Endowment*, 2(1):982–993, 2009.

Newell, A., Skarlatos, D., Fan, J., Kumar, P., Khutornenko, M., Pundir, M., Zhang, Y., Zhang, M., Liu, Y., Le, L., et al. Ras: Continuously optimized region-wide datacenter resource allocation. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pp. 505–520, 2021.

Panigrahy, R., Talwar, K., Uyeda, L., and Wieder, U. Heuristics for vector bin packing. *Microsoft Research Technical Report*, 2011.

Qi, Q., Luo, Y., Xu, Z., Ji, S., and Yang, T. Stochastic optimization of areas under precision-recall curves with provable convergence. *Advances in Neural Information Processing Systems*, 34:1752–1765, 2021.

Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M., and Wilkes, J. Omega: flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys)*, pp. 351–364, 2013.

Tang, C., Yu, K., Veeraraghavan, K., Kaldor, J., Michelson, S., Kooburat, T., Anbudurai, A., Clark, M., Gogia, K., Cheng, L., et al. Twine: A unified cluster management system for shared infrastructure. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 787–803, 2020.

van Stee, R. SIGACT news online algorithms column 20: the power of harmony. *SIGACT News*, 43(2):127–136, 2012. doi: 10.1145/2261417.2261440.

Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., and Wilkes, J. Large-Scale Cluster Management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems*, pp. 1–17, 2015.

Wang, C., Wu, Q., Weimer, M., and Zhu, E. Flaml: A fast and lightweight automl library. *Proceedings of Machine Learning and Systems*, 3:434–447, 2021.

Wang, L., Weng, Q., Wang, W., Chen, C., and Li, B. Metis: Learning to schedule long-running applications in shared container clusters at scale. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–17, 2020. doi: 10.1109/SC41405.2020.00072.

Wang, Y., Crankshaw, D., Yadwadkar, N. J., Berger, D., Kozyrakis, C., and Bianchini, R. Sol: safe on-node learning in cloud platforms. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 622–634, 2022.

Wu, N. and Xie, Y. A survey of machine learning for computer architecture and systems. *ACM Computing Surveys (CSUR)*, 55(3):1–39, 2022.

Yan, L., Dodier, R. H., Mozer, M., and Wolniewicz, R. H. Optimizing classifier performance via an approximation to the wilcoxon-mann-whitney statistic. In *Proceedings of the 20th international conference on machine learning (icml-03)*, pp. 848–855, 2003.

Yuan, Z., Yan, Y., Sonka, M., and Yang, T. Large-scale Robust Deep AUC Maximization: A New Surrogate Loss and Empirical Studies on Medical Image Classification. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 3020–3029, Montreal, QC, Canada, October 2021. IEEE. ISBN 978-1-66542-812-5. doi: 10.1109/ICCV48922.2021.00303.

A PROOF OF THEOREM 1

Throughout this section, we use *jobs* or *items* to denote the VMs, following the standard terminology in the bin packing literature.

A.1 Proof of Theorem 1

By scaling the problem, without loss of generality assume that all lifetimes L_j are between α and $\alpha\mu$ with probability 1. Thus, given Assumption 1, the predicted lifetimes $\hat{\ell}_j$ are between 1 and $\alpha\beta\mu$, and we have $\log(\alpha\beta\mu)$ (non-empty) intervals $I_1, I_2, \ldots, I_{\log(\alpha\beta\mu)}$. To simplify the notation we use $\bar{\mu} := \alpha\beta\mu$.

We say that job j has *class* i (or is an i*-job*) if its predicted lifetime belongs to the interval I_i, and say that a machine has class i if it receives jobs of class i. Also, let a_j denote the arrival time of job j.

The following lemma is the core of the upper bound for the cost of the algorithm. Essentially for every interval J, it upper bound the maximum number of machines of class i simultaneously active during this interval by the total size of the i-jobs that arrive in a slightly bigger interval. To make this precise, for an interval J, let $size_i(J)$ be the total size of the i-items that arrive at some time in the interval J.

Lemma 1. *Fix a class $i \in [\log \bar{\mu}]$, and an interval J over the grid with spacing $\alpha 2^i$, i.e., $J = [c \cdot \alpha 2^i, (c+1) \cdot \alpha 2^i)$ for an integer $c \geq 0$. Let m be the maximum number of machines of class i simultaneously active during a time in J, in the execution of Algorithm 1. Then, letting $J'' = [(c-2) \cdot \alpha 2^i, (c+1) \cdot \alpha 2^i)$,*

$$m \;\leq\; 4\, size_i(J'') \;+\; 2\,.$$

Proof. Let $J' = [(c-1) \cdot \alpha 2^i, (c+1) \cdot \alpha 2^i)$. Assume that at most $\frac{m}{2}$ i-jobs of size at least $\frac{1}{2}$ ("big jobs") arrive in the interval J', else $size_i(J'') \geq \frac{m}{4}$ and the lemma directly follows.

Now consider a time t in J such that there are m active machines of class i, and let us denote them by M. All these machines must have received at least one i-job that arrived in the interval $J' \cap [0, t]$, otherwise such machine would be inactive at time t, since any i-job j that arrives prior to interval J' finishes before time

$$(c-1) \cdot \alpha 2^i + L_j \;\leq\; (c-1) \cdot \alpha 2^i + \alpha \hat{\ell}_j \;\leq\; (c-1) \cdot \alpha 2^i + \alpha 2^i \;\leq\; t.$$

Moreover, since at most $\frac{m}{2}$ big i-jobs arrived in J', at least half of the machines in M receive a non-big i-job that arrived in $J' \cap [0, t]$; let $M' \subseteq M$ denote these machines.

Consider the "newest" (newest opening time) of the machines in M', and let j be a non-big i-job that arrived in $J' \cap [0, t]$ that is assigned to this newest machine. Notice that then job j arrives, all other machines in M' are active (they are older than the "newest", and they remain active at least until time $t \leq a_j$). Thus, all these machines were all under consideration for the First-Fit rule. Since this rule chose the "newest" machine, it means that all the other machines in M' could not fit job j at this time. Since j is a non-big job, it means that all these $\frac{m}{2} - 1$ machines have size occupation at least $\frac{1}{2}$ at this time. Again, all this size occupation must have been incurred due to i-jobs released in the interval J'' (older jobs would be done by time $a_j \in J'$). Thus, the total size of i-jobs that arrive in the interval J'' is at least $(\frac{m}{2} - 1) \cdot \frac{1}{2}$, and the lemma follows. □

To continue upper bounding the cost of the algorithm, we say that the *predicted volume* of job j is $s_j \hat{\ell}_j$. Also, let span (which is a random quantity) denote the total amount of time where at least one job is active, namely span $=$ area$(\cup_j [a_j, a_j + L_j])$. Actually we need to work with the following "predicted span with lifetimes multiplied by α" $\widehat{\text{span}} :=$ area$(\cup_j [a_j, a_j + \alpha\hat{\ell}_j])$. We then have the following upper bound on the cost of the algorithm.

To continue upper bounding the cost of the algorithm, we say that the *predicted volume* of job j is $s_j \hat{\ell}_j$. Also, let span (which is a random quantity) denote the total amount of time where at least one job is active, namely span $=$ area$(\cup_j [a_j, a_j + L_j])$. Actually we need to work with the following "predicted span with lifetimes multiplied by α" $\widehat{\text{span}} :=$ area$(\cup_j [a_j, a_j + \alpha\hat{\ell}_j])$. We then have the following upper bound on the cost of the algorithm.

Lemma 2. *Algorithm 1 has expected cost at most*

$$24\alpha\textit{[total predicted volume of jobs]} + 8(\log \bar{\mu}) \cdot \widehat{\text{span}}.$$

Proof. Fix a class i. For an integer c, define the intervals $J_c =: [c \cdot \alpha 2^i, (c+1) \cdot \alpha 2^i)$ and $J''_c = [(c-2) \cdot \alpha 2^i, (c+1) \cdot \alpha 2^i)$. Also, let C_i be the set of integral c's such that some i-job arrives at some time in interval the J_c, i.e. only the intervals indexed by C_i matter.

Finally, let m^i_c be the maximum number of machines of class i simultaneously active during a time in J, in the execution of Algorithm 1. The total cost that the algorithm pays in interval J_c due to the active machines of class i is at most $|J_c| \cdot m^i_c$ (i.e. majorizes the cost by assuming that all m^i_c machines are active over the whole interval J_c). Moreover, from Lemma 1, we have that $m^i_c \leq 4 size_i(J''_c) + 2$. Then adding over all $c \in C_i$, we obtain that the total cost of the algorithm due to machines of class i is at most

$$\sum_{c \in C_i} |J_c| \cdot m^i_c \;\leq\; \alpha 2^i \cdot \sum_{c \in C_i} \Big(4 size_i(J''_c) + 2 \Big)$$
$$= 4\alpha 2^i \cdot \sum_{c \in C_i} size_i(J''_c) + \alpha 2^{i+1} |C_i|. \quad (1)$$

Now notice that the interval J''_c only intersects with the next 2 intervals J''_{c+1} and J''_{c+2}; more precisely, for every $k = 0, 1, 2$, the intervals $\{J''_c\}_{c : c = k \mod 3}$ are disjoint.

Thus, $\sum_{c \in C_i : c = k \mod 3} size_i(J''_c) \leq size_i([0,\infty))$, and so $\sum_{c \in C_i} size_i([0,\infty)) \leq 3\, size_i([0,\infty))$. Employing this on (1), the cost of the algorithm due to machines of class i is at most

$$\begin{aligned}&12\alpha 2^i \cdot size_i([0,\infty)) + \alpha 2^{i+1}|C_i| \\ &\leq 24\alpha \cdot \text{[total predicted volume of } i\text{-jobs]} + \alpha 2^{i+1}|C_i|,\end{aligned}$$

the last inequality because the predicted volume of an i-jobs is $s_j \hat{\ell}_j \geq 2^{i-1} s_j$. Adding over all classes i, we have

$$\begin{aligned}&\text{cost of algo} \\ &\leq\ 24\alpha\text{[total predicted vol of jobs]} + \sum_i \alpha 2^{i+1}|C_i|. \quad (2)\end{aligned}$$

Next, we relate each term $\alpha 2^{i+1}|C_i|$ on the right-hand side with $\widehat{\text{span}}$. To see this, for $c \in C_i$, let j_c be any i-job that arrives in J_c. So the "predicted span with lifetime multiplied by α" of job j_c satisfies the containment is contained in

$$\begin{aligned}[a_{j_c}, a_{j_c} + \alpha\hat{\ell}_{j_c}] \ &\subseteq\ [c \cdot \alpha 2^i, (c+1)\cdot \alpha 2^i + \alpha 2^i) \\ &=\ J_c \cup J_{c+1}.\end{aligned}$$

In particular, the intervals on the left-hand side relative to jobs j_c with even c are disjoint, and hence

$$\begin{aligned}\widehat{\text{span}} \ &=\ \text{area}\Big(\bigcup_j [a_j, a_j + \alpha\hat{\ell}_j]\Big) \\ &\geq \sum_{c \in C_i\,:\, c \text{ even}} \text{area}\Big([a_{j_c}, a_{j_c} + \alpha\hat{\ell}_{j_c}]\Big) \\ &\geq \sum_{c \in C_i\,:\, c \text{ even}} \alpha 2^{i-1},\end{aligned}$$

where the last inequality uses the fact that the jobs j_c are of class i and hence $\hat{\ell}_{j_c} \geq 2^{i-1}$. Moreover, the same holds for the odd c's in C_i, and adding these two bounds together we get

$$2\,\widehat{\text{span}}\ \geq\ \sum_{c \in C_i} \alpha 2^{i-1} = \alpha 2^{i-1}|C_i|.$$

Applying this bound on inequality (2) we get that the total cost of the algorithm is at most

$$24\alpha\text{[total predicted volume of jobs]} + 8(\log\bar{\mu}) \cdot \widehat{\text{span}},$$

which concludes the proof of the lemma. □

We now lower bound OPT. It is clear that OPT is lower bounded by the total volume $\sum_j s_j L_j$ of the jobs (since each machine has capacity 1), and also by the span (at least one machine needs to be open during the span). Thus:

Lemma 3. *We have* OPT $\geq$ *[total volume of jobs], and* OPT $\geq$ span.

In order to relate this bound to the upper bound of the Algorithm 1 from Lemma 2, we need to relate these true volume/span to the predicted (scaled) volume/span. The volume par is easy due to Assumption 1:

$$\begin{aligned}\mathbb{E}\,\text{[total volume of jobs]} &= \sum_j s_j \mathbb{E} L_j \geq \frac{1}{\beta}\sum_j s_j \hat{\ell}_j \\ &= \frac{1}{\beta}\text{[total predicted volume of jobs]}. \quad (3)\end{aligned}$$

Relating span and $\widehat{\text{span}}$ is more complicated, and requires another argument based on disjoint intervals, carried out in the next lemma.

Lemma 4. *It holds that* $\mathbb{E}\,\text{span} \geq \frac{1}{2\alpha\beta} \cdot \widehat{\text{span}}$.

Proof. Let U be a minimal set of jobs whose "predicted span with lifetimes multiplied by α" equals $\widehat{\text{span}}$, namely $\text{area}(\cup_{j \in U}[a_j, a_j + \alpha\hat{\ell}_j]) = \widehat{\text{span}}$. Then we have

$$\widehat{\text{span}} \leq \sum_{j \in U}\Big|[a_j, a_j + \alpha\hat{\ell}_j]\Big| = \alpha\sum_{j \in U}\hat{\ell}_j. \quad (4)$$

Moreover, due to the minimality of U, we claim that these intervals are almost disjoint; more precisely, for each time t there are at most 2 intervals $[a_j, a_j + \alpha\hat{\ell}_j]$ with $j \in U$ that contain t. To see this, by means of contradiction assume that there are 3 intervals, indexed by $j_1, j_2, j_3 \in U$, that contain t. Without loss of generality let j_1 be the interval among these that starts earliest. The same interval cannot have finish the latest, else one could remove the jobs j_2 and j_3 from U and still have the desired span property, contradicting the minimality of U. So without loss of generality assume the interval indexed by j_3 has finishes the latest among these 3 intervals. Since both the intervals indexed by j_1 and j_3 intersect at t, we have

$$\begin{aligned}[a_{j_1}, a_{j_1} + \alpha\hat{\ell}_{j_1}] \cup [a_{j_3}, a_{j_3} + \alpha\hat{\ell}_{j_3}] \ &=\ [a_{j_1}, a_{j_3} + \alpha\hat{\ell}_{j_3}] \\ &\supseteq\ [a_{j_2}, a_{j_2} + \alpha\hat{\ell}_{j_2}];\end{aligned}$$

in this case job j_2 can be removed from U, also contradicting its minimality. This proves the claim.

Since from Assumption 1 we have $L_j \leq \alpha\hat{\ell}_j$ for every job j, we have the interval inclusion $[a_j, a_j + L_j] \subseteq [a_j, a_j + \alpha\hat{\ell}_j]$, and so the previous claim implies that for any time t, at most 2 of the intervals $[a_j, a_j + L_j]$ with $j \in U$ contain t; that is, at most 2 jobs in U are active at time t. Therefore, we have the span lower bound

$$\begin{aligned}\text{span} \geq \text{area}\Big(\bigcup_{j \in U}[a_j, a_j + L_j]\Big) &\geq \frac{1}{2}\sum_{j \in U}\Big|[a_j, a_j + L_j]\Big| \\ &= \frac{1}{2}\sum_{j \in U} L_j\,.\end{aligned}$$

Taking expectation and using the fact $\mathbb{E}L_j \geq \frac{1}{\beta}\hat{\ell}_j$ (Assumption 1), we get $\mathbb{E}\,\text{span} \geq \frac{1}{2\beta}\sum_{j\in U}\hat{\ell}_j$. Combining this with inequality (4) concludes the proof of the lemma. □

With these elements, we can now finally conclude the proof of Theorem 1: combining Lemma 3, inequality (3), and Lemma 4, we obtain that

$$\begin{aligned}\text{cost of algo} &\leq 48\alpha\beta\cdot\mathbb{E}\text{OPT} + 16\alpha\beta(\log\bar{\mu})\cdot\mathbb{E}\text{OPT}\\ &\leq \Big(64\alpha\beta\log\bar{\mu}\Big)\,\mathbb{E}\text{OPT}\,.\end{aligned}$$

This concludes the proof.

A.2 Necessity of Assumption 1

We show that Assumption 1 on the quality of the prediction is basically necessary to obtain approximation guarantees that improve over the $O(\mu)$-approximation that can be obtained without any prediction (Li et al., 2014). More precisely, we show that even if with a tiny probability the prediction underestimates the true lifetime, then no algorithm can do well. In particular, it says that while we can relax the assumption $\hat{\ell}_j \in [\frac{1}{\beta}L_j, \alpha L_j]$ so that the upper bound only holds in expectation, we cannot do the same for the lower bound. That is, it is much more important to be careful to not underestimate the lengths. A main difficulty in the analysis is that the algorithm may adapt to the size of the jobs scheduled thus far.

Theorem 2. *For every μ and $\varepsilon > 0$, there is an instance for this problem where $L_j \geq \hat{\ell}_j$ for all j, the probability of underestimation is at most ε (i.e. $\Pr(\hat{\ell}_j < L_j) = \varepsilon$ for all jobs j) but any online algorithm has cost at least $\frac{\mu}{4}\cdot\mathbb{E}\text{OPT}$. (In particular, note that for $\varepsilon \leq 1/\mu$ we have $\mathbb{E}L_j \leq 2\hat{\ell}_j$.)*

For the remainder of the section we prove this theorem. Without loss of generality we may assume that ε is sufficiently small, in particular, we may assume $\varepsilon \leq \frac{1}{3\mu}$. The instance is the following: It is convenient to denote $k := \frac{1}{\varepsilon}$, and assume WLOG that k is an integer. There are $\frac{1}{\varepsilon^2}$ identical jobs, and each has size $s_j = \frac{1}{\varepsilon}$ and true length L_j that is equal to μ with probability $\frac{\varepsilon}{2}$ and equal to 1 with probability $1-\frac{\varepsilon}{2}$. The predictions are all $\hat{\ell}_j = 1$, hence $\Pr(\hat{\ell}_j < L_j) = \varepsilon$. The jobs 1,2,... are released in this order but all of them at almost time 0 (e.g. the release time of job j is $r_j := \frac{j}{2k^2}$, say).

To simplify the notation, we say that job j is "big" whenever $L_j = \mu$, and "small" otherwise.

Consider any algorithm for this instance. Let N_i be the number of jobs this algorithm assigns to bin i (note N_i is a random variable, since the algorithm may adapt to the outcome of the sizes of the jobs it has already scheduled). Let B_i be the indicator of the event that one of the jobs assigned to bin i turned out to be big. The main lemma is the following.

Lemma 5. *For every bin i we have*

$$\mathbb{E}B_i \geq \frac{\varepsilon}{2}\cdot\mathbb{E}N_i.$$

Before proving this lemma we show how it implies the desired result. Notice that every bin that has a big job remains open until at least time μ, thus the total cost of the algorithm can be lower bounded as μ times the number of bins that have a big job, i.e. $\mu\cdot\sum_i B_i$. Since the total number of jobs $\sum_i N_i$ equals k^2, the previous lemma gives

$$\text{cost of alg} \geq \mu\cdot\mathbb{E}\sum_i B_i \geq \frac{\varepsilon\mu}{2}\cdot\mathbb{E}\sum_i N_i = \frac{k\mu}{2}. \quad (5)$$

On the other hand, OPT can pack all the small and big jobs separately (i.e., each machine only has big or only has small jobs). Since each bin accommodates k jobs, there is such a packing using at most $\frac{\#\text{big jobs}}{k}+1$ bins for the big jobs and k bins for the small jobs. Each of the first set of bins contributes with at most $r_{k^2}+\mu \leq 2\mu$ to the cost, and each of the second set of bins contributes with 1. This gives an upper bound on OPT of

$$\mathbb{E}\text{OPT} \leq 2\mu\cdot\left(\frac{\mathbb{E}[\#\text{big jobs}]}{k}+1\right)+k = 3\mu+k \leq 2k, \quad (6)$$

where the second inequality follows from the fact $\mathbb{E}[\#\text{big jobs}] = k^2\frac{\varepsilon}{2} = \frac{k}{2}$, and the last inequality from the assumption that $\varepsilon \leq \frac{1}{3\mu}$. Combining the bounds from (5) and (6) we obtain that the cost of any algorithm is at least $\frac{\mu}{4}\cdot\mathbb{E}\text{OPT}$, proving the desired result.

In order to conclude the proof, we now prove Lemma 5.

Proof of Lemma 5. Fix a bin i for the rest of the proof. Let A_j be the indicator that the algorithm assigned job j to machine i, and let Big_j be the indicator that job j is big, i.e. $Big_j = \mathbf{1}(L_j = \mu)$. Then we can express the indicator B_i that bin i was assigned a job that has materialized as big as $B_i = \max_j A_j Big_j$. It is easy to see that fo every scenario we have

$$\begin{aligned}B_i &\geq \sum_j A_j Big_j\cdot\Big(1-\sum_{j'<j}A_{j'}Big_{j'}\Big)\\ &= \sum_j A_j Big_j - \sum_{j'<j}(A_j Big_j)\cdot(A_{j'}Big_{j'}). \quad (7)\end{aligned}$$

To lower bound the expected value of B_i, we compute the expected value of the right-hand side.

For the product terms on the right-hand size, we have

$$\begin{aligned}&\mathbb{E}\Big[(A_j Big_j)\cdot(A_{j'}Big_{j'})\Big]\\ &= \mathbb{E}\Big[A_j\cdot A_{j'}\cdot Big_{j'}\cdot\mathbb{E}[Big_j \mid A_j\cdot A_{j'}\cdot Big_{j'}]\Big]\\ &= \mathbb{E}\Big[A_j\cdot A_{j'}\cdot Big_{j'}\cdot\frac{\varepsilon}{2}\Big],\end{aligned}$$

the last equation because the size of job j is independent of where j itself and the previously released job j' are assigned, and also independent to whether j' is big or not. Fixing j' and adding this over all job $j > j'$ and using the fact that at most k jobs can be assigned to machine i (since their the job sizes are $\frac{1}{k}$), i.e., $\sum_j A_j \le k = \frac{1}{\varepsilon}$, we have

$$\sum_{j:j>j'} \mathbb{E}\Big[(A_j Big_j)\cdot(A_{j'} Big_{j'})\Big] = \frac{\varepsilon}{2}\mathbb{E}\Big[A_{j'} Big_{j'} \cdot \sum_{j:j>j'} A_j\Big] \le \frac{1}{2}\cdot \mathbb{E}[A_{j'} Big_{j'}]\,.$$

Applying this to inequality (7) we get

$$\mathbb{E}B_i \ge \frac{1}{2}\sum_j \mathbb{E}[A_j Big_j] = \frac{1}{2}\sum_j \mathbb{E}A_j \cdot \mathbb{E}Big_j = \frac{\varepsilon}{2}\sum_j \mathbb{E}A_j = \frac{\varepsilon}{2}\mathbb{E}N_i,$$

the first equation again because the size of job j is independent to where it is assigned. This concludes the proof of the lemma. □

B More Machine Learning Details

B.1 Features

As discussed, we have ablated around one hundreds available features among all accessible information. We also included simple statistics for each customer: 25, 50, 75, 90 percentiles, mean, and standard deviation of VM lifetime in the past half, one, and three months for a given customer. Then, we decided to only have two set of features based on the features' acquisition time and their importance on the ML performance through SHAP value.

Small feature set contains seven features: allocation hour and day, counts of all the customer's VMs and long-lived VMs in the past two months, VM requested size and location.

Large features set contains all features in the small feature set, and it also contains additional eleven features, including the 50, 75, 90 percentile of the customers' VM lifetime in the past 2 months, the standard deviation of the customer's VM lifetime, the number of cores, the amount of memory requested, the OS disk type, extensions requested, resource group name, computer name, and the setting for VM scale set. A more detailed SHAP value plot is shown in Figure 10.

B.2 Regression

As indicated by the Assumption 1 used in the guarantee of Algorithm 1, the ratio between ground truth VM lifetime and its prediction is an important indicator of errors from online VM allocation algorithms. Moreover, as indicated in Figure

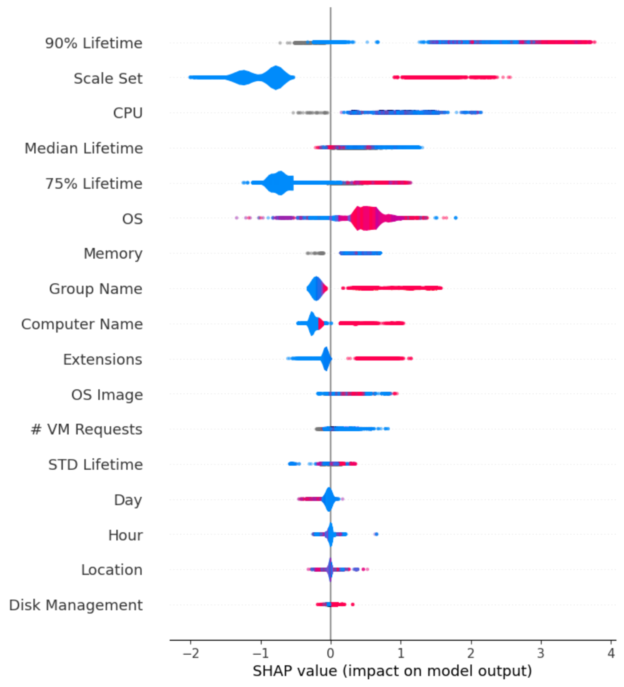

Figure 10. SHAP value for the large set of features.

1, most VMs' lifetimes are in lognormal distribution or a mixture of lognormal distributions. Here, by considering the longtail but ignoring the spike of longlived VMs at the tail, we assume the labels are lognormal (Crow & Shimizu, 1987), such that the log of the label is distributed according to a normal distribution. We use the Q-error (Moerkotte et al., 2009) to evaluate the ratio between the ground truth and the predicted; letting y denote the ground truth and y' denote the prediction, the Q-error is:

$$Q = \max\left(\frac{\max(y,1)}{\max(y',1)}, \frac{\max(y',1)}{\max(y,1)}\right). \tag{8}$$

We use the mean absolute error of the logarithmic values as our regression loss, which is equivalent to minimizing the average of Q-error (Theorem 3). This optimization objective has two desirable properties: (1) it assumes the labels are in lognormal distribution so that standard MAE or MSE loss can be applied to the log values; (2) it minimizes the ratio between the ground truth and prediction.

Theorem 3 (MAE-Log Loss and Q-Error)**.** *When ground truth and predictions are both positive values, optimizing mean absolute error between log value of ground truth and log value of the prediction is equivalent to minimize the Q-error.*

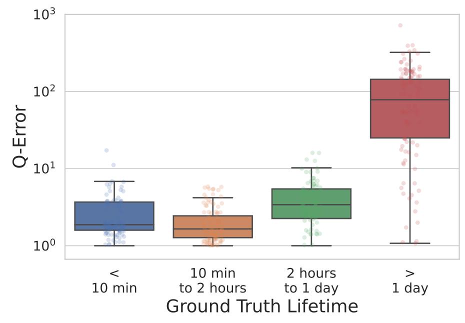

Figure 11. Regression testing errors.

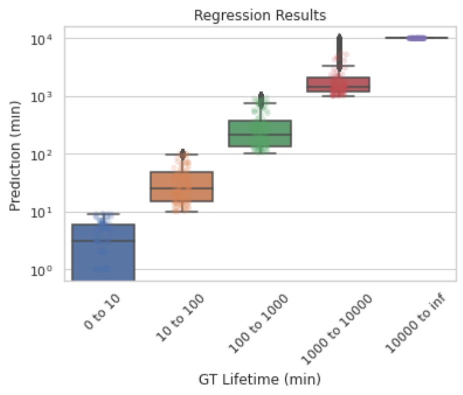

Figure 12. Regression testing results.

Proof. Assuming $y, y' \in [1, \infty)$, we have:

$$\begin{aligned}\arg\min Q(y,y') &= \arg\min \max(y/y', y'/y) \\ &= \arg\min \max(\log y/y', \log y'/y) \\ &= \arg\min \max(\log y - \log y', \\ &\qquad \log y' - \log y) \\ &= \arg\min |\log y - \log y'|. \qquad (9)\end{aligned}$$

□

For the regression task, we use Geometric Mean of Q-error $GMQ = \sqrt[n]{Q_1 Q_2 \cdot Q_n}$ between the ground truth and predicted lifetime, where n is the number of predicted items. As shown in Figure 11, the regression performance is worse for VMs with longer lifetime, and the Q-error could reach 100 for VMs with lifetime of more than a day. Figure 12 shows the prediction distribution for VMs with different ground truth lifetime.

B.3 Multi-Class Classification

Following the theoretical guarantee from Algorithm 1, we use buckets of doubling widths, using cut-off thresholds of the form $15 \cdot 2^i$, more precisely, 15, 30, ..., and 3840 minutes.

We use the standard cross entropy loss for the 10-way classification experiments. All tested models, with different sets of features and backbones, achieved similar results for multi-class classification of 61 - 65% weighted F-1 score. The accuracy for VMs that lived in less than 15 minutes is high, but the accuracy for the rest is extremely low, which might affect VM allocation. If we grant more granularity to VMs with lifetime up to 15 minutes, the packing density would not be significantly affected, because those VMs would exit the system very soon.

B.4 CAM Details

Targeting to automatically learn customers' behaviors for VM lifetime prediction, we create a novel central-attention mechanism (CAM) based on existing self-attention and multi-instance learning. This model architecture enables the system to learn information from long sequence ($> 10^4$ tokens) and store customer's embedding in a vector.

Multi-Instance Attention: our preliminary experiment shows customers' historical VMs contain useful information to predict the lifetime of future VM requests, which inspired the feature extraction methods as described in the previous section. Here, we create a multi-instance learning-based method to extract information from historical VM requests for each customer.

Denote p as the number of dimensions for the attention, $X \in \mathbb{R}^{t\times d}$ as an input time-series data with t time points and d features at each time point, $C \in \mathbb{R}^{c\times d}$ as c central tokens, f as a neural network-based estimator. Based on the self-attention mechanism, we define:

- Query: $Q = f_q(C) \in \mathbb{R}^{c\times p}$.
- Key: $K = f_k(C, X) \in \mathbb{R}^{t\times p}$
- Value: $V_x = f_v(X) \in \mathbb{R}^{t\times p}$, $V_c = f_v(C) \in \mathbb{R}^{c\times p}$, and $V = [V_c; V_x] \in \mathbb{R}^{(t+c)\times p}$
- Attention: $A_C = softmax(\frac{QK^T}{\sqrt{d_p}})V_x$
- Output: $Y = [A_c; V_x] \in \mathbb{R}^{(c+t)\times p}$
- Multi-head Output: $[Y_0; Y_1, ..., Y_h] \in \mathbb{R}^{(c+t)\times(p\times h)}$ with h heads.

Integrating Binary and Multi-class Classification: In neural network-based models, we use 10 neurons to represent 10 geometric intervals for the lifetime span with the softmax function for the multi-class classification. For binary classification, we calculate cumulative sum across these neurons. Denoting $\{c_i\}_i$ as the list of cut-off thresholds mentioned earlier and value x, we have:

$$\Pr(\text{lifetime} \geq x) = \sum_{c_i < x} \Pr\Big(\text{lifetime} \in [c_{i-1}, c_i]\Big) \quad (10)$$

Other Design Considerations: runtime and latency is a crucial challenge to apply giant deep learning models for system application, and we offshore the main computation offline, as discussed in §4.

B.5 Other ML Methods We Tried

Here, we discuss some methods that we have tried but haven't included in the study. We believe all these approaches have potential to be applied in lifetime-aware VM allocation, but more future studies are needed to strengthen them for this challenging application.

Cox Proportional Hazard (CPH): The Cox models are widely recognized as a valuable tool for predicting lifetime using right-censored data. Compared to the volume of VMs we received every day, only a small fraction of VMs remain active, and they are what we refer to as right-censored data. Our research has shown that incorporating the Proportional Hazard loss in our deep learning system has little effect on the final outcome. In addition, most online allocation algorithms in use today do not heavily rely on the proportional hazard score. Nevertheless, the CPH model may prove to be useful in data centers where there is an abundance of right-censored VMs, or in cases where the training data is limited.

Contrastive Learning: contrastive learning is a great tool and improved our machine learning prediction performance. However, the optimization community lacks correspondingly online allocation algorithms using Contrastive Learning's predictions or ranking predictions. In the future, if more robust allocation algorithms are invented, this approach can be tested further.

AUC and PaR Optimization: several optimization objectives have been proposed to improve AUC score or Precision@Recall in the past decades, including Wilcoxon-Mann-Whitney Statistics (Yan et al., 2003), non-decomposable objectives (Eban et al., 2017), compositional optimization (Qi et al., 2021), surrogate loss (Yuan et al., 2021), and many other methods. We tested some of them in our systems but only have seen incremental effects on our ML models. Our data size is large enough for standard optimizer to achieve satisfied results.

Variational Bayesian Inference: we applied standard reparameterization approach (Diederik et al., 2014) to approximate the lognormal distribution of lifetime for each VM. Its performances for short-lived VMs are great, but its variational inference for super long-lived VMs (e.g., that lived for more than a few days) is hard to interpret.

Baseline ML Models: we also tested linear regression, logistic regression, support vector machines, and multi-layer perceptron. As expected, these approaches all underperformed more advanced models.

Standard MSE and MAE Losses for Regression: as discussed, these simple loss functions failed to generalize for the long-tail distribution.

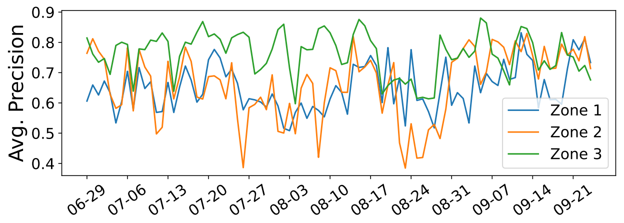

Figure 13. Performance Across Different Availability Zones. The x-axis is the date, and y-axis is the Average Precision score.

Reinforcement Learning (RL): we also tested reinforcement learning and Markov decision process to replace existing bin packing algorithms. Training time, convergence, and generalizability are main obstacles to apply RL for large data centers. The RL approach has not outperform the production Best Fit algorithms yet, but we believe RL-inspired approaches could be promising methods for AI-based VM allocation in the future.

B.6 Simulation Experiment Details

To examine different packing algorithms, we designed a simple strategy to synthesize noises to ground truth lifetime. To establish a fair comparison for different algorithms with different label types (e.g., binary classification, multi-class classification, different categorization thresholds, etc.), we use the regression performance distribution obtained in Table 11 and Table 12. Then, we convert the noisy labels into different bins.

B.7 More machine learning results

Here, we discuss more interesting ML results and findings. First, we observed different ML performance across different zones (Figure 13) and performance drop over time if the model is not re-trained (Figure 14).

The behaviors at different zones and geographical locations might be caused by different cultures, time-zone differences, and other customer preferences. For instance, in some locations, people would start an VM around 8 AM and use it for several hours. While this phenomena would not happen in other zones.

The performance drop over time also indicates the distribution shift occurred in cloud management system, where users and systems would have change their behavior over time. Hence, adapting to the new distribution is also critical for applying ML models to system management. We resolve this data shift problem by re-training a model each weekend, and more thoughtful techniques could be applied in the future.

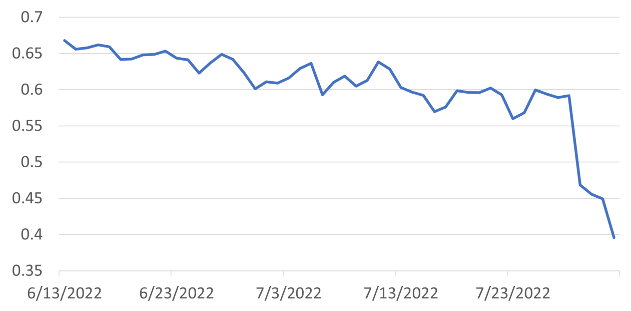

Figure 14. F1 score metric calculated day-by-day for an ML model trained until June 12.

C DPBFR: Method Details

To study how lifetime information can be valuable for VM allocation, we adapt the Prefer Best Fit algorithm mentioned on Section 4.3 to apply different quantization values depending on the lifetime of the VM. If the VM is short-lived (for a defined threshold. E.g.: 60 min) we apply the default quantization. But if the VM is long-lived we apply a better, and more aggressive quantization.

C.1 Gains of using lifetime during packing

In this round of experiments, we selected two quantization sets to test as the “better” quantization for long-lived VM:

- Quantization with 5 buckets: The higher the number of buckets, the higher the precision of the selected nodes. Currently we use quantization with 3 buckets in production, by increasing the number of buckets we can narrow the number of nodes selected.
- No quantization: At the limit, considering an infinite number of buckets, not using quantization is the best that we could do. This is our upper limit.

We want to answer if there is gain in packing performance when using lifetime information. To answer that we will compare the results obtained when applying 5 buckets or no quantization for long-lived VMs with the baseline (applying 3 buckets for all VMs). We also need to define what a long-lived VM is and analyze how different thresholds impact the metrics.

In the real scenario we won’t have perfect predictions for lifetime. The machine learning model will make mistakes and those mistakes need to be considered in our experiments because they will impact the final performance of the algorithm. We also need to understand how noisy predictions impact the metrics. For that, we mimic the model precision by inserting noise (randomly making mistakes at a rate consistent with the results from the model). In this set of experiments, the noise applied was: 73% of recall for the long-lived VMs and 96% of recall for the short-lived VMs.

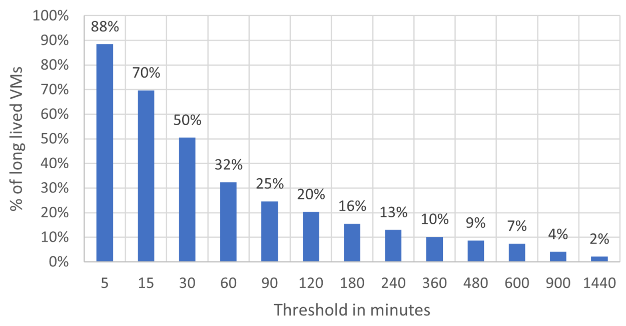

Figure 15. Lifetime distribution for different thresholds in 80 clusters from generation 7.

The dataset utilized for the simulation consists of 80 clusters from generation 7 that were randomly selected amount all zones. Each cluster has 3 months of data, and the simulation starts with a snapshot of the system from June to August 2021 . Figure 15 shows the dataset lifetime distribution. Overall, the percentage of long-lived VMs in the dataset is small; half of VMs live longer than 30min, only 32% live longer than 1h, and 20% live longer than 2h.

Figure 16 shows the improvement we achieve in packing density (PD), and Figure 17 shows the increase in FF, by using different quantization approaches to a group of VMs. Each line is a different experiment, in which a new quantization setup is compared to the baseline. The x axis represents the threshold in minutes that defines a long-lived VM, and the y axis represents the improvement over the baseline in percentage points. For a threshold of 0 min (the left-most point) all VMs are classified as long-lived, and as the threshold increases, the percentage of long lived VM decreases. The experiments considered are:

1. Blue line: Apply no quantization for the long-lived VMs and keep using the default quantization for the short-lived VMs.
2. Orange line: Apply 5 buckets quantization for the long-lived VMs and keep using the default quantization for the short lived VMs.
3. Grey line: Apply no quantization for the long-lived VMs and keep using the default quantization for the short-lived VMs (the same as the blue line) but inserting noise in the lifetime predictions.
4. Green line: Randomly applies no quantization to the same percentage of VMs that are long-lived. That is, for each threshold, we calculate the amount of VMs that are long-lived and use this percentage to randomly choose the VMs that will receive the better quantization.

Figure 16 shows that all experiments have a positive gain compared to the baseline, and the smaller lifetime thresholds

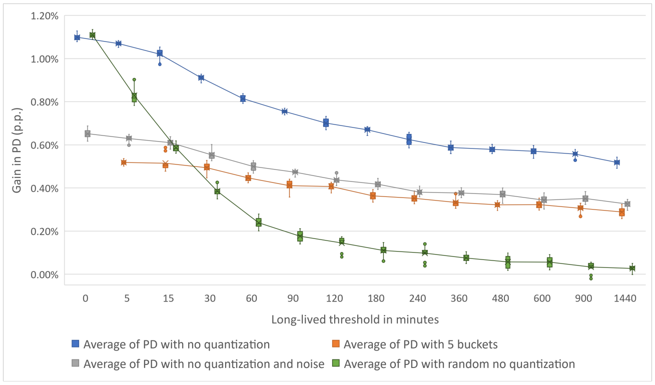

Figure 16. Gain in core packing density for 80 clusters from gen7 considering no quantization, no quantization with noise, 5 buckets quantization and random no quantization.

lead to the higher deltas. This is expected because with a small threshold more VMs will be considered long lived and receive improved quantization. Thus, as the threshold increases the number of VMs affected decreases, resulting in smaller gains. This effect can be clearly seen in the green line, for which the amount of improvement follows the percentage of long-lived VMs at each threshold, as shown in Figure 15, since it chooses the VMs at random to receive better quantization.

We see the impact of choosing long-lived VMs instead of random VMs to apply the improved quantization comparing the blue and green line. When choosing the long-lived VMs we achieve a much better performance, especially when the percentage of affected VMs is very low. For instance, for a threshold of 1440 min only 2% of the VMs are affected, and the performance of the blue line is over 3.5 p.p. higher than the green line.

As expected, the blue line works as our upper limit, because we apply the most aggressive quantization and consider the actual lifetime instead of model predictions (working as "perfect predictions"). With that said, another interesting comparison is the one between the blue and the grey line. The considerable gap between them shows us the amount of performance that is lost when we insert noisy predictions. The machine learning model will make mistakes, but improving its performance can decrease this gap.

Figure 17 shows the increase in FF for those experiments when varying the long-lived threshold from 0 to 1440 minutes. The smaller lifetime thresholds lead to bigger increases in FF because a higher number of VMs are being allocated using the improved, but more aggressive, quantization. As the number of VMs affected by the new quantization decreases, the addition to the FF also decreases. The FF is what limits our choice of lifetime threshold because we cannot increase it too much. It is important to notice that when choosing the long lived VMs to apply the new quantization (blue line) the increase in FF is smaller

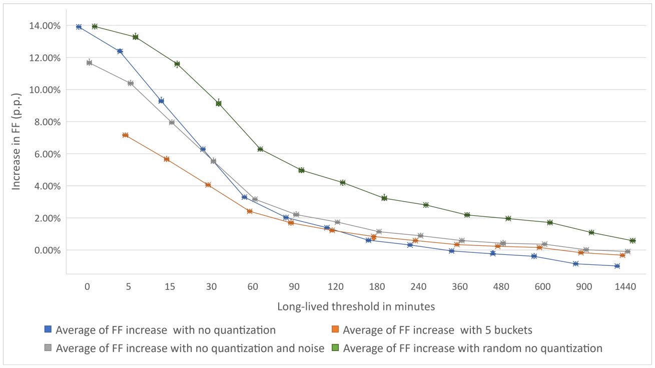

Figure 17. Increase in FF for 80 clusters from gen7 considering no quantization, no quantization with noise, 5 buckets quantization and random no quantization.

Table 3. Binary Classification results in P@R (Precision at Recall) metric.

Threshold	$P@R_{0.70}$	$P@R_{0.75}$	$P@R_{0.80}$	$P@R_{0.85}$
60 min	73.7%	70.6%	67.4%	63.3%
120 min	60.4%	56.2%	51.6%	46.7%
180 min	57.8%	52.4%	47.1%	41.3%
240 min	57.0%	52.2%	46.7%	40.8%
300 min	56.7%	52.2%	47.2%	41.2%
360 min	54.7%	49.4%	44.4%	38.8%

than the random VM selection (green line), reinforcing our assumption of the importance of VM lifetime to the allocation process.

These experiments proved our belief that VM lifetime is valuable information to be used during allocation. Figure 16 shows the gain we have when applying improved quantization to long-lived VMs, while Figure 17 helps us determine the lifetime threshold to consider a VM long-lived. Based on these experiments, we decided to aim at a 0.5p.p. increase in PD with long-lived threshold varying from 1h to 6h.

C.2 Parameter selection

To decide on the best set of values for the "improved quantization" we performed an extensive parameter sweep over sets of 4, 5 and 6 buckets. The evaluation dataset was upgraded to 170 traces of generations 5, 6 and 7 to better represent the system's reality. After many simulations, the best set of values for each configuration (4, 5 or 6 buckets) was chosen and we moved on to noise evaluation.

There were packing density improvement as we add quantization buckets in the perfect prediction case. However, when adding noise to the system, the 5 buckets set proved to be the most robust. We decided not to explore more buckets divisions and follow our experiments with the 5-buckets setting.

We evaluated how each lifetime threshold performs given

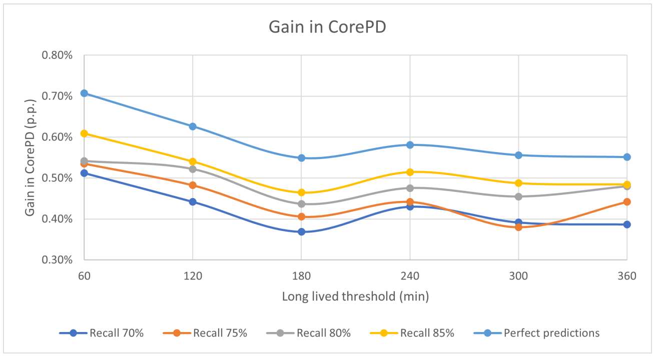

Figure 18. Gain in core packing density for 170 clusters from gen7, gen6 and gen8 for different recall values.

different recall values for the long-lived class. We explored a lifetime threshold between 1 to 6 hours, and recall values from 70% to 85%. For each value of recall, we used the corresponding metrics obtained by the current binary lifetime model to mimic the errors during simulation. Table 3 shows the long-lived precision used for each setting. Figure 18 shows the gain in core packing density and there were no significant changes in FF when varying the recall and threshold values. Based on that, we decided to aim for 85% recall in our models.