LAVA: LIFETIME-AWARE VM ALLOCATION WITH LEARNED DISTRIBUTIONS AND ADAPTATION TO MISPREDICTIONS

Jianheng Ling [*1] **Pratik Worah** [*1] **Yawen Wang** [*1] **Yunchuan Kong** [*1] **Chunlei Wang** [1] **Clifford Stein** [1]
Diwakar Gupta [1] **Jason Behmer** [1] **Logan A. Bush** [1] **Prakash Ramanan** [1] **Rajesh Kumar** [1] **Thomas Chestna** [1]
Yajing Liu [1] **Ying Liu** [1] **Ye Zhao** [1] **Kathryn S. McKinley** [*1] **Meeyoung Park** [*1] **Martin Maas** [*2]

ABSTRACT

Scheduling virtual machines (VMs) on hosts in cloud data centers dictates efficiency and is an NP-hard problem with incomplete information. Prior work improved VM scheduling with predicted VM lifetimes. Our work further improves lifetime-aware scheduling using repredictions with lifetime distributions versus one-shot prediction. Our approach repredicts and adjusts VM and host lifetimes when incorrect predictions emerge. We also present novel approaches for defragmentation and regular system maintenance, which are essential to our data center reliability and optimizations, and are not explored in prior work. We show repredictions deliver a fundamental advance in effectiveness over one-shot prediction.

We call our novel combination of distribution-based lifetime predictions and scheduling algorithms *Lifetime Aware VM Allocation (LAVA)*. LAVA reduces resource stranding and increases the number of empty hosts, which are critical for large VM scheduling, cloud system updates, and reducing dynamic energy consumption. Our approach runs in production within Google's hyperscale cloud data centers, where it improves efficiency by decreasing stranded compute and memory resources by ∼3% and ∼2% respectively. It increases empty hosts by 2.3-9.2 pp in production, reducing dynamic energy consumption, and increasing availability for large VMs and cloud system updates. We also show a reduction in VM migrations for host defragmentation and maintenance. In addition to our fleet-wide production deployment, we perform simulation studies to characterize the design space and show that our algorithm significantly outperforms the prior state of the art lifetime-based scheduling approach.

1 INTRODUCTION

Cloud data centers run an increasing fraction of the world's compute. Efficient resource use in data centers is thus critical, from both an economic and environmental perspective. In Infrastructure-as-a-Service (IaaS) environments, VM scheduling to physical hosts determines a large portion of end-to-end efficiency. Poor assignment of VMs may *strand host resources* – making the remaining host resources too small or too imbalanced to accommodate additional VMs. VM allocation also impacts the number of *empty hosts*, which impacts idle power optimizations, resource utilization, the ability to provision large VMs (obtainability), and system maintenance velocity (e.g., for security patches).

Much prior work addresses workload allocation in data centers (Schwarzkopf et al., 2013; Verma et al., 2015; Cortez et al., 2017; Gog et al., 2016; Grandl et al., 2016; Zhang et al., 2014), including leveraging workload predictions (Delimitrou & Kozyrakis, 2014b; 2013; Park et al., 2018). The closest related work predicts job lifetimes (time between start and exit) to assist the scheduler in making better placement decisions (Yadwadkar et al., 2014; Ferguson et al., 2012; Tumanov et al., 2016; Cortez et al., 2017; Buchbinder et al., 2021). Most recently, Barbalho et al. (Barbalho et al., 2023) show bin packing improvements using Lifetime Alignment (LA) scheduling. LA predicts the lifetime of a VM at creation time, and then treats the predicted lifetime as fixed. As a result, mispredictions may tie up an entire host and, over time, model mistakes will accumulate. For example, assume a model with a 1% chance of predicting a long-lived VM (running for months) as short-lived. If, over time, the scheduler places 70 presumably short-lived VMs on a host, there is a greater than 50% chance that one of them is long-lived, which would prevent the host from freeing up until that VM exits. Correspondingly, Barbalho et al. show that their best algorithm achieves maximum efficiency with perfect prediction accuracy and degrades rapidly as the accuracy decreases ((Barbalho et al., 2023), Figure 8). They thus introduce an alternative algorithm that achieves lower efficiency but is more tolerant to mispredictions.

[*]Equal contribution [1]Google [2]Google DeepMind. Correspondence to: Martin Maas <mmaas@google.com>.

Proceedings of the 8th MLSys Conference, Santa Clara, CA, USA, 2025.

Our work achieves both tolerance to mispredictions and high efficiency by *repredicting* VM and host lifetimes to correct mispredictions and improve accuracy over time. We take inspiration from C++ memory allocation work that increases the robustness of lifetime-based allocation by reassessing predictions on the fly (Maas et al., 2020). For example, if a VM outlives its predicted lifetime, the algorithms change which other VMs are preferred to be co-scheduled with it on the VM's host. In addition to host bin packing, we address significant challenges not explored by prior work, including host defragmentation, maintenance, and stranding, which are essential components of Google's cloud system design and approach to efficiency.

Our key contributions are A) VM lifetime reprediction and adaptation to mispredictions, such as when a VM outlives its initial prediction; B) a novel ML model of VM lifetime probability distributions, which captures hard-to-predict VM behavior and prediction uncertainty; and C) three novel lifetime repredicting scheduling algorithms: 1) Non-Invasive Lifetime Aware Scheduling (NILAS) incorporates predictions into an existing scoring function. 2) Lifetime Aware VM Allocation (LAVA) more fundamentally redesigns the scheduler around lifetimes. 3) Lifetime-Aware ReScheduling (LARS) exploits lifetimes to reduce VM disruptions during host defragmentation and maintenance.

We deployed our approach fleet-wide across Google's planet-scale cloud infrastructure. We evaluate it in production and in high-fidelity simulations with production traces, and compare these results to prior work. Some of the key findings of our paper include the following:

- This paper shows how to exploit our key insight – using probability distribution models of VM lifetimes – to co-design algorithms that repredict VM lifetimes and actively respond to mispredictions.
- While NILAS and the prior state-of-the-art (SOTA) LA algorithm place VMs with similar lifetime together, the key idea of LAVA is the opposite – it puts shorter-lived VMs on hosts with one or more longer-lived VM, creating hosts with a wider range of lifetimes. This approach avoids extending the time at which the longest-lived VM will exit and the host will be empty.
- We show the NILAS, LAVA, and LARS algorithms increase empty host availability (2.3-9.2 percentage points (pp)), reduce stranding (∼2-3%), and reduce VM disruptions (4.5%). Note that a consistent 1 pp improvement in stranding or empty hosts can be equivalent to saving 1% of a cluster's capacity.
- Using production data and extensive simulation studies, we show that these algorithms are efficient and robust, outperforming the prior SOTA.

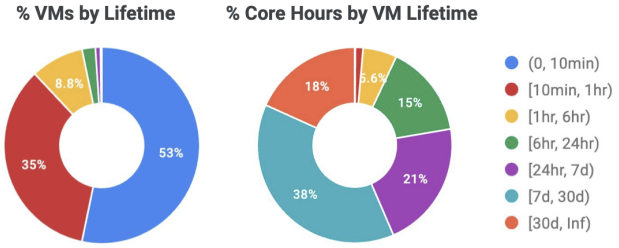


Figure 1. Distribution of VM lifetimes of scheduled VMs vs. their resource consumption.

- We incorporate lifetime-based scheduling in a complex environment that features dynamic resource management based on VM usage, VM live migrations, and hierarchical scheduling. We show reductions in resource stranding and VM migrations, in addition to improved bin packing quality.

Similar to Maas et al. (Maas et al., 2020), we find that on-the-fly adjustment to mispredictions, versus one-off predictions, opens up a fundamentally different class of algorithms and optimizations, in this case for VM scheduling, and perhaps beyond. Our production deployment shows the practicality of this approach, and of deploying ML in the lower layers of the systems infrastructure stack more broadly.

2 Background

We first give an overview of VM lifetime properties, our production system, and metrics. Next, we describe the prior SOTA Lifetime Alignment algorithm (LA) (Barbalho et al., 2023), to which we compare qualitatively and quantitatively.

2.1 Characterizing VM Lifetimes in Production

In our production environment, most VMs are short-lived, but most core hours are consumed by long-running VMs, similar to the generational hypothesis in garbage collection (Lieberman & Hewitt, 1983; Ungar, 1984). Figure 1 shows our VM lifetime distributions. While 88% of all VMs live for less than 1 hour, 98% of resources are consumed by VMs that live for 1 hour or more (measured in *CPU cores* × *time occupied*). Without lifetime-specific handling of VMs, each VM would introduce the same amount of fragmentation, although only the placement of a small fraction of them has an impact on the overall resource efficiency. Prioritizing the bin packing quality of all VMs equally is thus highly inefficient, which motivates lifetime-based VM scheduling.

A key element of our approach is to view VM lifetimes from the perspective of distributions (Cumulative Density Functions or CDFs), as opposed to just averages. We identify that VM features useful for prediction do not fully determine

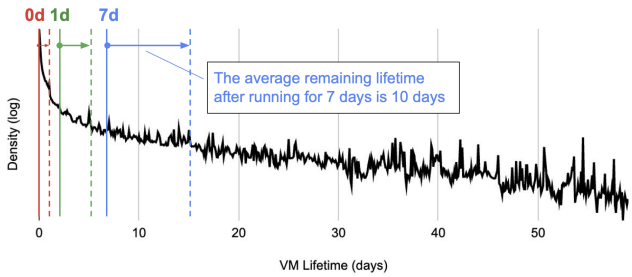


Figure 2. Lifetime distribution (PDF) of VM lifetimes. When the VM is scheduled, the expected (average) lifetime is 0.2 days. After it has run for 1 day, the expected remaining lifetime is 4 days. After 7 days, the expected remaining lifetime is 10 days.

VM lifetimes. The best a traditional model could do would thus be to predict the expected (or average) lifetime for each VM. Figure 2 shows the average lifetime may not be particularly meaningful when there are both long and short-lived VMs with the same features. We therefore model and predict the lifetime of VMs as probability distributions. Note that this type of approach has a long history in survival analysis (Wang et al., 2019) and in areas such as storage systems (Zhou & Maas, 2021). However, our novelty is predicting distributions and co-designing VM scheduling algorithms to leverage them in detecting and updating mispredictions.

2.2 VM Allocation

Our fleet consists of data centers in many geographic regions that run a mix of first and third party workloads. Cloud workloads have distinct host *pools* for distinct VM *families*. VM families represent different products, such as VMs optimized for high performance or cost efficiency. Broadly, VM families fall into two categories. 1) Performance-optimized, slice-of-hardware families assign each VM a fixed partition of a host's resources, e.g., CPU cores, DRAM, SSD, and GPUs. 2) Dynamically sized families share unused resources based on VM and host usage patterns and thus the amount of virtual CPUs and memory of VMs may on occasion exceed the physical resources on a host. This paper focuses on the performance-optimized C2 family and cost-optimized E2 family, representative of these categories.

A scheduling framework called Borg (Verma et al., 2015) assigns VMs to hosts. Borg maintains a global view of all VMs and hosts in a given host pool. When a VM creation request arrives, the scheduler computes the set of feasible hosts, i.e., hosts with sufficient resources that match any hard constraints. For each feasible host, it computes a score that determines the preference between them. This score incorporates a number of different business goals, such as spreading tasks across failure domains and bin packing efficiency. This scheduling setup is common (Barbalho et al., 2023; Hindman et al., 2011; Verma et al., 2015).

Borg's *Waste Minimization* bin packing algorithm explicitly optimizes for producing empty *shapes* of resources in multiple dimensions (e.g., CPU $\times$ DRAM $\times$ SSD) that match anticipated workload patterns. Borg deployed this approach several years ago because it achieves higher obtainability for large VMs and improves resource efficiency compared to *Best Fit*, the previous scheduler, which was similar to prior work (Barbalho et al., 2023). However, Borg's scoring function poses challenges for incorporating VM lifetimes, because it uses lexicographic ordering – one scoring dimension is evaluated at a time, with tie-breakers resolved by the next-lower scoring function. Bin packing is among its final dimensions, ensuring that improving bin packing does not degrade other business objectives.

2.3 Constraints & Objectives

Our work aims to maximize resource utilization and minimize fragmentation without degrading other business objectives. We measure the following optimization metrics.

Empty Hosts. We measure the percentage of hosts in a pool that is empty. Empty hosts are required to schedule large VMs that consume most or all of the host's resources. They also increase host maintenance velocity when rolling out a kernel or microcode security patch: By increasing empty hosts, applying the update to empty hosts first, and preferring new VMs land on updated hosts, we speed up maintenance and reduce VM disruptions due to live migrations. Additionally, Google puts empty hosts in low power mode and moves empty hosts between pools, configuring them differently to serve various capacity needs. As such, 1 pp of increase in empty hosts directly corresponds to 1% more capacity available for these use cases. Other metrics of bin packing quality include *packing density* (Barbalho et al., 2023), and are equivalent (Appendix D).

Resource Stranding. This metric measures free resource shapes and how many future VMs may be scheduled in them. Periodic *inflation* simulations measure stranding. We take a representative mix of VMs and simulate scheduling as many as possible until capacity is exhausted. The remaining resources on hosts represent *stranded* resources that cannot fit new VMs. For example, a host may contain free memory but no free CPUs. This memory is thus stranded. The metric is similar to admission control scoring functions (Sajal et al., 2023) and captures unusable capacity when the zone is full.

VM Live Migrations. We aim to minimize live migrations. Live migration and fragmentation are closely connected. For some VM families, periodic live migrations will defragment resources. Live migration and resource efficiency may thus be traded against one another. This trade-off is controlled by policy and predates our work. It represents a fundamental difference from some other clouds. Maintenance events trigger live migrations as well.

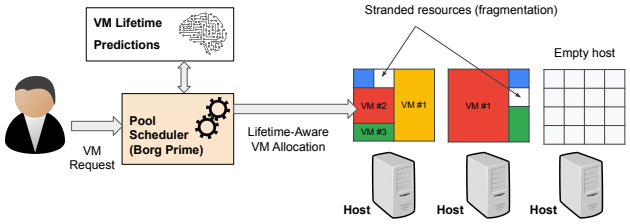


Figure 3. Overview of our VM allocation setup.

2.4 Baseline: Lifetime Alignment (LA)

We compare against LA (the *Lifetime Alignment* algorithm) from Barbalho et al. (Barbalho et al., 2023). While other work on lifetime-based VM scheduling predates it, LA supersedes this work and, to our knowledge, is the only published approach deployed in production data centers. At a high level, our setup (Figure 3) resembles Barbalho et al. We use machine learning to predict the lifetimes of VMs and incorporate predictions into scheduling decisions. Note that their work uses Best Fit scoring. They do not correct mispredictions nor do they consider maintenance and defragmentation, contributions of our paper.

For every VM allocation request, LA runs a gradient-boosting model to predict its *lifetime class*, which is over an exponentially increasing time interval (2^i to 2^{i+1}). Their models run on special inference servers. The allocator then preferentially assigns VMs to a host with the same lifetime class, using Best Fit. The lifetime class of a host is defined dynamically: it is the longest *remaining* time of any VM on that host, based on the VM's *initial* lifetime prediction. If there is no host of the same lifetime class, LA picks another suitable host. If no such host exists, it assigns the VM to a new (previously empty) host.

Barbalho et al. report that LA achieves high improvements in packing density with accurate predictions. While LA performs live migrations occasionally, they are not reported or part of the scheduling approach (Barbalho et al., 2023). Because they find the multi-class version of LA is sensitive to mispredictions, they introduce DPBFR, which uses lifetime predictions only to adjust the quantization of Best Fit scoring. They deployed DPBFR in production, as it is more robust to mispredictions, but it achieves lower maximum improvement compared to the binary version of LA. We thus compare to a faithful implementation of their best algorithm, called LA-Binary.

While LA (Barbalho et al., 2023) tackled optimizing packing density (empty hosts), we extend the state-of-the-art by reducing stranding and our LARS algorithm decreases the number of migrations needed due to maintenance and defragmentation by determining the migration order. In addition, since we create more empty hosts via lifetime-aware scheduling, we further reduce the need for defragmentation.

2.5 Lifetime-aware C++ Memory Allocation

We observe that VM scheduling is similar to explicit memory allocation in C/C++ and other languages. The memory allocator needs to place objects, that never move, on pages such that pages are either maximally used or become empty (Maas et al., 2020). VM scheduling is similar. VMs correspond to objects, hosts to pages, stranding to internal fragmentation, and empty hosts to external fragmentation. A key difference is that VM allocation is multi-dimensional. It manages multiple resources, including CPUs, memory, and SSD, while memory is one-dimensional (object size).

LLAMA (Maas et al., 2020) is a C++ memory allocator that uses an ML model to predict object lifetimes. Every page is assigned a lifetime class (10ms, 100ms,...) and objects are initially assigned to huge pages with the same lifetime class. Once a page is full, emerging gaps are filled with objects that are *at least one lifetime class lower*. Once all original objects are free, LLAMA reduces the lifetime class of the page by one. This process continues until the page is empty. LLAMA detects mispredictions. If a page is not free before the expected time, LLAMA under-predicted the lifetime of an object, and it increases the lifetime class of the page by one to correct the misprediction. This mechanism minimizes fragmentation while tolerating mispredictions. The latter is critical, since a single misprediction may fragment an entire huge page and misprediction probability increases exponentially with the number of allocations.

We take inspiration from LLAMA, designing our models and allocators to detect and correct mispredictions. We cannot directly adopt LLAMA because VM allocation is multi-dimensional and bin packing is not the only business objective. We do, however, incorporate the key insights of LLAMA into new algorithms, NILAS, LAVA, and LARS.

3 ML Lifetime Prediction Model

We design ML models that predict the remaining lifetime of a VM both at the time it is scheduled and when re-evaluating the VM and host later, to correct previous mispredictions. Intuitively, reprediction addresses two fundamental issues.

1. VM lifetimes depend on many factors that cannot be described by its features, e.g., human behavior or external events. As such, some VMs are fundamentally impossible to predict. Predicting a single VM lifetime thus limits accuracy, as shown in Figure 2.

2. Estimating the actual lifetime of a previously scheduled VM throughout its execution is not always possible with a single prediction. For example, if a particular VM type's lifetime is bi-modal (e.g., 1 day or 1 week), observing that the VM has been running for 2 days means that it is expected to live for another 5 days.

Our model addresses these limitations by predicting a probability distribution as opposed to a single value. This approach is well-known in survival analysis (Wang et al., 2019). Survival models capture the "uncertainty" associated with a particular set of feature values. If a VM is highly predictable based on the features, the distribution will be narrow, while difficult-to-predict VMs have wider distributions. Here, we use distributions to update our VM estimates based on the lifetime so far (uptime). Instead of relying on the initial prediction, we compute the *conditional expectation* $\mathbb{E}(T_r|T_u)$ for an updated estimate of a VM's lifetime over time: "Given a VM has been running for interval T_u, what is the expected remaining lifetime T_r?". This value is directly calculated from the PDF in Figure 2.

This approach avails itself to a range of models and we compare a few (Appendix B), including different forms of survival models, neural networks, and gradient-boosted decision trees (GBDTs). GBDTs performed best in our experiments. Since they are not distribution-based, we pass an additional parameter T_u into the model that represents the uptime of the VM so far (i.e., the x axis of the PDF), and we predict the remaining lifetime T_r. We *augment* every training example by turning it into multiple examples with different values of T_u: specifically, 12.5%, 25%, ... of the original lifetime. This implementation turns a regression model into a survival model with good accuracy. We train the model directly on $\mathbb{E}(T_r|T_u)$ as the label.

There are two ways to use such a model: 1) When scheduling a VM, run the model repeatedly to materialize a mapping from uptime to the expected remaining lifetime, which may require a large number of model invocations to get sufficient precision. 2) Run the model repeatedly when re-evaluating a VM, which saves resources since it only runs the model for values that are actually needed, but only works if model latency is low. We choose the latter.

Our models use a range of features, including the *zone*, *VM family*, *VM shape*, whether or not the VM attaches to local SSD, and internal project metadata (Appendix A). We use a production-grade machine learning library called Yggdrasil (Guillame-Bert et al., 2023). We generate our training data by querying a large-scale internal data warehouse that contains historical data of VMs. We use a combination of distributed SQL queries and data processing pipelines to generate a set of labeled *examples* and feed them into the training framework. Our data sets are on the order of 1M examples for training. The resulting model is loaded from a file system and runs in binaries via a range of language bindings. We periodically retrain our model on recent data.

We train a single joint model for our entire global fleet, across a large number of our data centers. We experimented with training a separate model for each pool, but this approach is disadvantageous from a rollout perspective, more difficult to maintain, and we found that it neither performs better nor significantly improves resource savings. Our model achieves 99% precision at 70% recall when used to classify VMs between short and long-lived according to a 7 day threshold. We use this model throughout all of our experiments, to ensure comparability across all algorithms.

An important design consideration is how to deploy the models. For instance, Barbalho et al. (Barbalho et al., 2023) run their models on separate inference servers. They cache prediction results to tolerate unavailability of servers, increasing complexity. Running model inference on separate servers also requires handling model rollouts and verification separately since they are not integrated with existing rollout mechanisms. We choose instead to embed the model directly into the Borg binary. This deployment reduces possible failure modes. By piggybacking on Borg rollouts, we update models with the same frequency as Borg, testing the two together. This approach lowers model overhead and achieves a median latency of 9 us (Figure 8), which is 780× lower than the median latency for LA (Barbalho et al., 2023). With this low latency, we can run our model more frequently, including for lifetime-guided defragmentation (Section 4.4) and to update and correct mispredictions.

We deployed our model in production over a year ago. It maintained all our production standards, while improving empty machines, stranding, and VM migrations.

4 Scheduling Algorithms

We show that repredicting lifetimes offers a fundamental advantage over one-shot prediction in our multi-dimensional scheduling approach. We introduce three algorithms. NILAS is non-invasive and becomes active when a set of hosts satisfy all other criteria and thus have the same score without the bin packing score (Section 4.2). LAVA, in contrast, makes lifetime-aware scheduling a fundamental part of the scheduler (Section 4.3). Note that while LA and NILAS place VMs with similar lifetime together, the key idea of LAVA is the opposite – it adds many short lived VMs to hosts with one or more long lived VM. The intent is to create hosts that have a wider range of different lifetimes, with an upper bound that is decreasing over time. LARS exploits lifetimes to reduce VM migrations during defragmentation and maintenance (Section 4.4).

4.1 Theoretical Foundation

We show analytically that repredicting lifetimes and correcting for mispredictions fundamentally improves how well any algorithm can do. Specifically, if the initial error in lifetime prediction is a positive constant, then the number of hosts required without correcting for mispredictions will exceed the number for the same best fit algorithm with

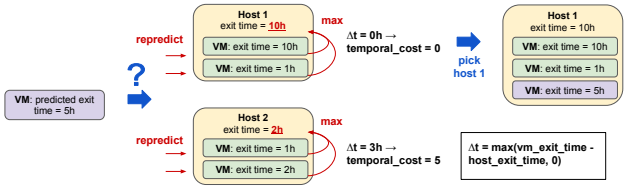


Figure 4. Overview of the NILAS algorithm.

repredictions by $\Omega(m)$, where m is the number of hosts. Appendix E includes the precise theorem and proof.

4.2 NILAS: Non-Invasive Lifetime-Aware Scheduling

The goal of NILAS (Figure 4) is to create more empty machines by scheduling VMs on a host where all other VMs are likely to exit at a similar time or later. Given a VM, NILAS ranks potential hosts based on the exit times for the VMs already scheduled on that host. It computes the maximum of the predicted exit time of all VMs on the host. This score is conceptually similar to LA (Barbalho et al., 2023), with a crucial difference: Instead of using the original VM lifetime predictions, NILAS repredicts the remaining lifetime (i.e., the *updated* exit time) for all VMs, using their uptime so far.

NILAS corrects any earlier mispredictions as follows. Consider a host where all remaining VMs have exceeded their originally predicted lifetime. Without correcting mispredictions, the scheduler has to assume that this host will soon become empty and will avoid scheduling long-lived VMs on this host. We hypothesize that this choice is the reason why LA is sensitive to prediction accuracy, leading the authors to not deploy LA. In contrast, NILAS schedules new VMs on hosts using the repredicted *remaining* lifetimes of all VMs on each host. As such, NILAS continues to improve its predictions if the original prediction was wrong.

NILAS is integrated into an existing hierarchical scheduler (Section 2.2) in a non-invasive way. When scoring a host, we compute ΔT = max(predicted_vm_exit_time - host_exit_time, 0), where the host_exit_time is the maximum of the *repredicted* remaining VM lifetimes on the host. We then quantize this value with bucket boundaries {0m, 30m, 60m, 90m, 2h, 3h, 4h, 6h, 12h, 24h, 168h}. We call the index of the bucket that ΔT falls into the *temporal cost*. For example, if $\Delta T = 70m$, the temporal cost is 2. This quantization integrates into our lexicographic scoring function (Section 2.2). By quantizing the temporal cost, we create equivalence classes within which hosts can be packed based on additional considerations. This feature integrates well with our multi-dimensional and shape-based bin packing.

We add the temporal cost into our lexicographic scoring function one level above the bin packing score. The temporal cost never has an effect on any higher-ranked business metrics. It only becomes the deciding factor if all those scores are the same. The score is thus *non-invasive*. It is local to the bin packing score and is equivalent to modifying the bin packing score to add a third dimension (lifetime).

4.3 LAVA: Lifetime-Aware VM Allocation

A limitation of NILAS is that mispredictions may gradually bump up the host lifetime. By trying to match the new VM to the exit times of other VMs on a host, the host may never free up because each under-predicted VM further bumps up the exit time, causing even more VMs to be scheduled, which in turn bumps up the exit time further.

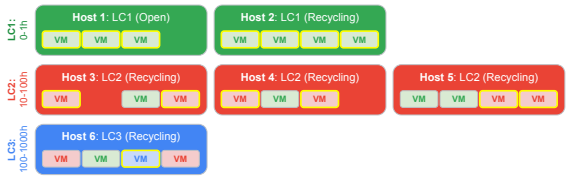
In preference to filling resource gaps on hosts with VMs that exit at a similar time, LAVA attempts to fill gaps with VMs that are at least 10× shorter lived, so that even mispredictions are unlikely to increase the host lifetime. Further, LAVA updates the host's exit time when detecting a *major* (more than 10×) misprediction. We use order of magnitude differences to limit the impact of small amounts of over or under prediction. We now describe LAVA in more detail.

LAVA divides lifetime predictions into lifetime classes: <1h, 1-10h, 10-100h, 100-1000h. We refer to these lifetime classes as LC1, LC2, L3, and L4. Each host is also assigned a lifetime class. LAVA runs an additional coarse-grained host scoring function based on their lifetime classes and breaks ties with NILAS.

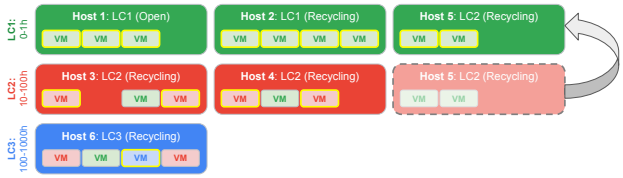
Each host has a lifetime class and is in one of three states: *empty*, *open*, or *recycling*, similar to LLAMA (Maas et al., 2020). When a new VM arrives and there is no eligible *open* or *recycling* host, LAVA places the VM on an empty host, assigns the host the *open* state, and assigns the host the lifetime class of the VM. As long as this host is open, LAVA will only schedule other VMs of the same lifetime class on it (Figure 5a). Once over 90% of the resources (CPU or memory) of an open host are occupied, LAVA transitions the host to the *recycling* state. All the VMs that were present on the host when it transitioned state are labeled *residual VMs* (i.e., the VMs of the original lifetime class of the host). *Recycling* hosts only receive VMs that are at least one lifetime class lower, e.g., LAVA only places new VMs with a predicted lifetime of 10 hours or shorter on 100-hour recycling hosts.

LAVA handles mispredictions as follows. If all predictions are correct, the total lifetime of a host does not exceed 1.1× its original lifetime class. There are two cases: overpredicted and underpredicted lifetimes.

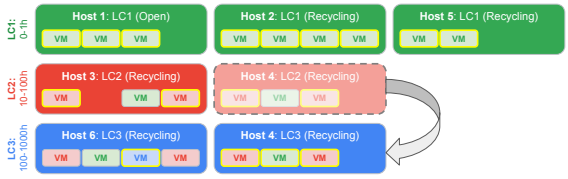
If lifetimes are overpredicted, VMs will exit early, creating new gaps on the host. The approach of placing shorter-lived VMs in gaps ensures that we keep a host occupied, while minimizing the likelihood that we extend its lifetime. Once all residual VMs exit, we know that all remaining VMs are



(a) LAVA preferentially places new VMs on *recycling* hosts of a higher lifetime class. If none is available, it chooses *open hosts* of the same lifetime class. Otherwise, it opens a new host.



(b) When all residual VMs (highlighted) on a host exit, LAVA reduces the lifetime class by one. All remaining VMs turn residual.



(c) If a host exceeds its deadline, there was a misprediction and LAVA increases its lifetime class by one. Its VMs become residual.

Figure 5. Overview the LAVA Algorithm.

of the next-shorter lifetime class. We therefore re-classify the host as one lifetime class lower (Figure 5b), classify all remaining VMs as the new residual VMs, and fill emerging gaps with even shorter-lived VMs. This process repeats until the host is empty.

Conversely, when lifetimes are underpredicted, the host will not become empty within the expected time interval. We detect this case by establishing a time-out for each host after which we move it to the next-higher lifetime class (Figure 5c). All VMs that are on the host at the time of expiration become the new residual VMs.

LAVA is also integrated into Borg's existing hierarchical scheduler, but adds another scoring function before NILAS. This higher-ranked scoring function keeps track of additional states such as predicted host lifetime class, host state, and deadline for each host. The size of this additional state is negligible compared to other per-VM state.

When a new VM request arrives, LAVA scores the hosts in the following order (in decreasing preference): *recycling* hosts whose lifetime classes are greater than the VM lifetime class (the closer the more preferred), *open* hosts whose lifetime class are equal to the VM lifetime class, any non-empty host, and lastly empty hosts. In contrast to NILAS, which acts as a tie-breaker for the existing algorithm, LAVA makes decisions that the baseline could not have made, and is therefore *invasive*.

4.4 LARS: Defragmentation & Maintenance

The above algorithms focus on reducing fragmentation by improving VM placement. Borg also performs software and hardware maintenance activities, applying security, microcode, host OS, and VM software updates. It also actively defragments hosts for some VM families. These activities live migrate VMs (Ruprecht et al., 2018). Live migration uses pre- and post-copy techniques that minimize VM execution time disruptions to seconds. However, total migration times are proportional to VM memory and SSD usage, and consume capacity on both hosts. In simulations, we model that both hosts are busy for a conservative 20 minutes.

Borg reserves a small fraction of hosts for defragmentation, maintenance, and to tolerate hardware failures, and thus these activities do not impact available capacity. We show how using lifetimes can reduce the number of VMs that are disrupted during defragmentation and maintenance. We call the algorithm LARS. The rest of our exposition focuses on defragmentation for conciseness (details in Appendix H).

For some VM families, we defragment hosts when the availability of empty hosts in a particular pool drops below a certain threshold. The defragmenter searches for suitable hosts to defragment, preferring hosts with few VMs and excess resources and to move the VMs to hosts with available slots that match the VM shapes.

LARS first chooses a small number of hosts to defragment based on their excess resources and marks them as unavailable for scheduling. It then uses predicted lifetimes to determine the order in which to migrate VMs. It starts with the longest-predicted VM and selects a target host to which to migrate it. Choosing this VM reduces the number of migrations, as shorter-lived VMs exit while LARS migrates longer-lived VMs. Once the host is empty, LARS marks it as available to schedule for any lifetime class.

Note that LARS uses the same algorithm to choose a target host for migration as for initial VM placement. As such, NILAS and LAVA use repredictions from our distribution-based model to place the VM on a better-suited host compared to the prior algorithm that was not lifetime-aware.

5 Implementation

Our algorithms are integrated into Borg's scheduling code. The scheduler runs within *Borg Prime*, which handles incoming VM requests and assigns them to hosts. The Borg Prime binary is replicated across a number of instances, with one of them serving as the elected leader. All replicas run the same binary and a lower layer implements consensus.

As mentioned in Section 3, we compile our model into the Borg Prime binary and roll it out with Borg. This coupling has the advantage that canary deployments (the first step of our rollout process) test the entire end-to-end approach. We deploy our software regularly and could update our model as often as once a week. However, we found that the models' accuracy remains high even a month later as shown in Figure 10. Note that while we found it highly advantageous from a latency and complexity perspective to compile the model into the binary, our approach is not restricted to this deployment scenario.

We carefully measured the CPU and DRAM overheads of our approach in production. Binary size increases by less than 10 MB, and both the memory and CPU consumption of our approach are (generally) indistinguishable from noise. This lack of observable overhead may seem counter-intuitive, but since a VM record has so much other state and Borg performs many other computation during scoring, the extra bytes of memory and the 9 us of model latency are not visible. We process on the order of 10-100 scheduling requests per second in each cluster, and even hypothetically repredicting 100 VMs in the process would add less than 1 ms. In practice, we generally repredict many fewer VMs, since higher-ranking scoring functions filter out many hosts and we re-score in parallel VMs only on considered hosts.

In some very large zones, re-scoring still became a bottleneck. We therefore added a host lifetime score cache (Appendix G.3). For any host that 1) is assigned a new VM, or 2) an existing VM exits, or 3) the host expected lifetime expires, we re-score the host. For all other hosts, we re-score them according to a configurable time interval (e.g., 60s).

5.1 Simulations

NILAS/LAVA: We developed an event-driven simulator to explore our approach. We extract production traces of VM start, exit, and restart events within a particular pool and then replay this trace against a simulated instance of the scheduler. The simulated scheduler uses our production code base and is thus highly accurate. We validated our simulator against production (Appendix F has more details).

LARS: To calculate the benefit of LARS (lifetime-based defragmentation), we built a modified simulator, using the same production traces. From our traces, we collect the list of migrations that were performed during a time interval. In production, we limit the number of live migrations that can be in progress simultaneously to batches of 3. We simulate this behavior by assuming that all migrations are performed in a certain order (in our baseline, defined by the trace), but have to wait until a *slot* is available. This approach has the effect that some VMs exit while others are migrating. LARS modifies this order based on lifetime predictions.

5.2 Production Measurements

Measuring the impact of scheduling policies in production is challenging. Since workloads and total capacity in a pool are constantly changing, it is difficult to attribute changes in a metric to a particular modification. We therefore use an A/B testing methodology. We split the hosts into two halves and apply our scheduling algorithm to one of them. Our production metrics report empty hosts, stranding, and model prediction for the entire pool. However, stranding is only accurate for entire pools and cannot differentiate A/B hosts. We report live production experiments in four pools, where we apply the algorithm to the entire pool and evaluate scheduling quality. These experiments also ensure that our A/B approach does not distort behavior. Note that these A/B deployments are at production scale. The cluster/zone sizes for this product range from 100s to 10,000+ hosts with O(100-10,000) VMs active at a time.

NILAS has been deployed fleet-wide for one VM family for about a year, at the time of publication. Time series data from the fleet-wide production rollout confirmed our simulation numbers and A/B experiments, and show a fleet-wide increase of empty hosts by 4 pp for this VM family.

5.3 LA-Binary Comparison

Our main comparison is to LA-Binary, a faithful implementation of LA, but embedded in the Borg binary versus a separate model service (Barbalho et al., 2023). Since our data center environment is different, perfectly replicating their setup is impossible as, e.g., the ML model features are different. We are faithful to LA's description and favorable to LA-Binary when in doubt. For an apples-to-apples comparison, we use the exact same model when comparing LA-Binary to NILAS and LAVA, but without repredictions for LA-Binary. The predictor classifies VMs as short or long lived, using all our model features, but with the same two-hour cutoff threshold as in their paper.

6 Evaluation

This section reports results from production and simulations on production data center traces. Our simulations are highly accurate (Appendix F). We report relative improvements and omit industry-competitive sensitive data.

6.1 End-to-end Simulation Results (NILAS & LAVA)

We first compare the performance of the different algorithms in simulation using C2 family production traces. Figure 6 shows simulation data from C2 traces from 24 pools that cover a wide range of sizes, geographies, and usage patterns. The traces are from May-June 2024 and have a duration of seven weeks. We measure the average percentage of empty

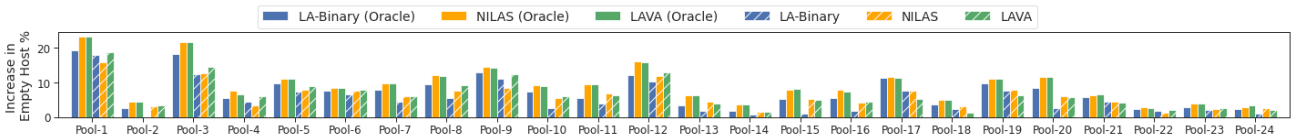


Figure 6. Empty host improvements comparing NILAS, LAVA with LA-Binary, to our production baseline for 24 C2 pools in simulation. On average, LAVA increases empty hosts by 6.5 pp, compared to 6.1 pp for NILAS, and 5.0 pp for LA-Binary.

hosts and report changes relative to the production baseline. For instance, if LAVA achieves 21% and the baseline achieved 20%, we report a 1 percentage point (pp) improvement. We run all algorithms both with our actual lifetime model and a perfect (oracular) predictor as a comparison point. Similar to the LA paper (which increased packing density by 2% in production), a consistent 1 pp improvement represents a large gain in this scenario, and can be equivalent to 1% of a cluster's capacity if used to power down or divest hosts, or offset holdbacks.

NILAS outperforms LA-Binary on average, achieving an additional 1.1 pp increase in empty hosts. With more optimized misprediction mitigation, LAVA achieves the best average improvement of 1.5 pp compared to LA-Binary. We also observe that NILAS consistently outperforms LA when both have oracular lifetimes: 9.5 pp vs. 7.5 pp on average, confirming our theoretical results that reprediction is fundamentally more powerful than one-shot prediction.

As with many NP-hard problems, we observe occasional inversions when comparing techniques on large numbers of production pools with highly varying workloads. The different production pools we evaluated in Figure 6 vary significantly in size, utilization, VM workload distribution, and accuracy of the lifetime model. Isolating the specific factors driving these performance variations is challenging due to the complex interplay of these variables and direct correlations with individual factors were weak at best.

6.2 Production Results (NILAS)

Prior to fleet-wide rollout, we ran production pilots in waves on C2 and E2 traces. For the first two waves, we conducted A/B experiments in three pools and measured the impact. Table 1 shows our statistically significant empty host improvements, ranging from 2% to 10% for the entire pool.

For the third wave, we piloted NILAS on the entire pool. We employ a Bayesian Structural Time Series method CausalImpact (Brodersen et al., 2015) to evaluate the pre-post causal effect of NILAS (Figure 7). We observed a significant empty host improvement of 4.9% (Table 1). Moreover, for the whole-pool pilot, we additionally measure the causal effect of NILAS on stranded CPU and memory, where 1 pp of stranding reduction translates directly into 1% of capacity. NILAS reduced CPU stranding by 3% (95% CI: [-5.5%, -0.49%]) and memory stranding by 2% (95% CI: [-3.8%, -0.23%]). We also conducted a whole-pool pilot for an E2 pool. We note that C2 is slice-of-hardware while E2 is dynamically sized. The fact that the approach worked for both types of VM families is a non-obvious result, since dynamic resizing of VMs further complicates bin packing.

Table 1. NILAS Empty Host Improvements in Pilot Pools

Pilot Pool	Type	Change in Empty Hosts
C2 Wave 1 pool	A/B	+2.3 pp (p-value = 0.01)
C2 Wave 2 pool 1	A/B	+2.7 pp (p-value $<$ 0.01)
C2 Wave 2 pool 2	A/B	+9.2 pp (p-value $<$ 0.01)
C2 Wave 3 pool	All	+4.9 pp (95% CI: [0.54, 9.2])
E2 Wave 1 pool	All	+6.1 pp (95% CI: [1.9, 10.0])

Table 2. VM Migration Reductions Using LARS on Two Traces.

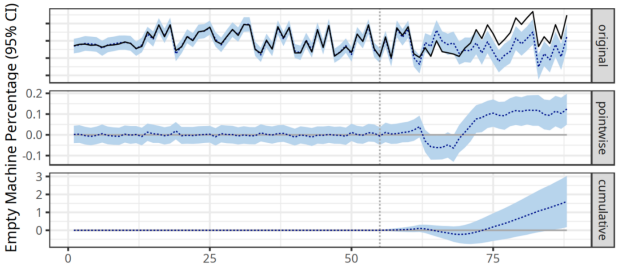
		Migrations		
	Scheduled	Baseline	LARS	Reduction
1	48,239	37,108	35,505	4.32%
2	53,597	36,307	34,655	4.55%

6.3 Lifetime-based Host Defragmentation (LARS)

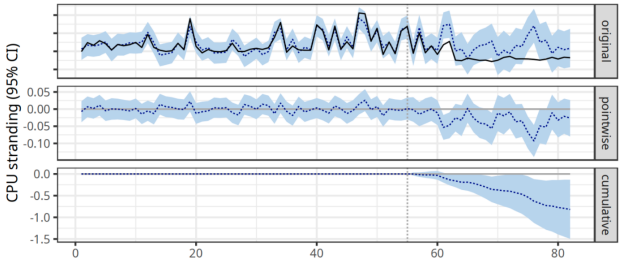
Using production traces, we simulate defragmentation on two different one-month time intervals from a number of pools with customer workloads, including C2 and E2 VMs. In this experiment, we use oracle lifetimes. LARS does not affect which hosts are picked for migration at a given time, but orders the VMs on each in-migration host so that the VMs with the longest remaining lifetime are migrated first (Appendix H). The effect is that a larger number of VMs terminate before they would have been migrated, resulting in a reduction of around 4.5% of live migrations.

6.4 ML Model

Our lifetime model achieves 70% recall at 99% precision, which is significantly higher than other model types we tried (Appendix B). We measured the latency of our model and find that most predictions complete in under 10 us (Figure 8). This latency improves upon (Barbalho et al., 2023) by 780×, and shows the benefit of compiling the model into the Borg Prime binary. Fast predictions make it practical to frequently re-run the model to correct mispredictions and to use it during maintenance and defragmentation.



(a) Empty hosts.



(b) CPU stranding.

Figure 7. Time series from our Wave 3 rollout, produced by CausalImpact (Brodersen et al., 2015). Each sub-figure has three panels. The first panel shows empty machines and a counterfactual prediction (dotted line) post-launch (labels omitted due to sensitivity). The second panel shows the difference between observed data and counterfactual predictions, i.e., the point-wise causal effect, as estimated by the model. The third panel adds the point-wise contributions from the second panel, plotting the cumulative effect.

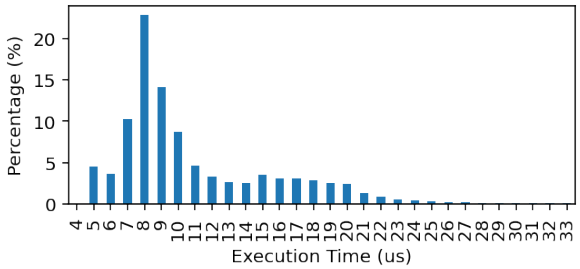


Figure 8. Histogram of model execution latencies.

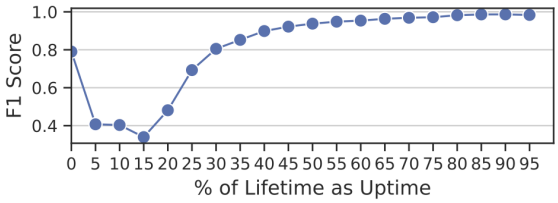


Figure 9. Making repredictions of VM remaining lifetime using its uptime significantly improves the model accuracy.

We analyze the impact of model features in Appendix A. We also analyze the sensitivity of NILAS and LAVA to prediction accuracy and show that our improvements apply across accuracies (Appendix G.1). Finally, we performed a number of ablations to show that NILAS with oracle lifetimes and positioned high in the scoring function achieves close to the theoretical maximum of empty hosts (Appendix G.2). The gap between production performance and theoretical optimum is a combination of prediction accuracy and priority within the scoring function.

6.5 Impact of Repredictions

To further understand the utility of repredictions based on the current VM uptime, we vary the amount of uptime used as input feature to the model and assess its impact on model accuracy. For each VM in the test trace, we divide its total lifetime into 20 quantiles. We then show the F1 score for predicting whether or not a VM lives beyond 168 hours (7 days) when the uptime is set to a particular quantile. Figure 9 illustrates the relationship between varying amount of uptime and model accuracy. We observe that without using reprediction (no uptime/0^{th} quantile), the model only has a F1 score of 0.8. The F1 score quickly rises above 0.9 after the 8^{th} quantile (i.e., once the VM's uptime has reached 40% of its total lifetime). Observing how long a VM has lived and repredicting its remaining lifetime thus leads to more accurate predictions. We find repredicting longer-lived VMs is critical to avoid spreading them across hosts and causing fragmentation.

One surprising result is that the the F1 score of quantiles 1-5 is, in fact, lower than the accuracy at the 0^{th} quantile. This is because these uptimes are very close to 0 in absolute terms and thus difficult for the model to disambiguate, particularly since we are operating in the log domain. In contrast, the accuracy at 0% uptime is higher because it is always 0 and the model has seen many training samples with uptime=0. A potential optimization would be to only pass the uptime to the model if it reaches a particular threshold (e.g., 30 seconds), to avoid this initial dip in accuracy.

6.6 Decrease in Model Accuracy Over Time

We find that the accuracy of our model remains stable for several months after deployment (Figure 10), but does require periodic retraining on the order of once a month to maintain its high accuracy. There are at least two reasons why accuracy drops over time. 1) New workloads arrive that the model was not trained on. 2) Workloads change their behavior. We looked at production data during this time period and observed gradual shifts in resource usage that is consistent with shifting workloads.

7 Production Experience

NILAS has been running in production since early 2024. We now share general insights from our experience of integrating ML into a mature production system. Our project started in 2020 and we quickly developed initial versions of

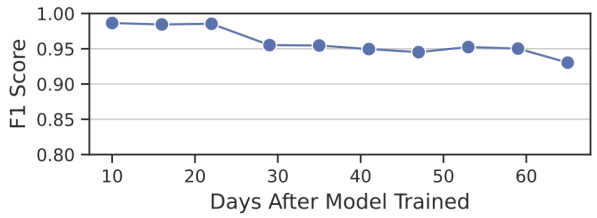


Figure 10. The model remains high accuracy weeks after training, making its maintenance overhead low for production deployment.

our VM lifetime models and algorithms in simulation. Most time was spent on putting the approach in production.

Explainable models One key requirement was that models had to be interpretable and explainable to be accepted in a production environment. We started with a lookup table approach where each entry contained a survival curve produced using Kaplan Meier (Kaplan & Meier, 1958). However, we found that the resulting models underperformed (Appendix B) and were not space-efficient. GBDTs showed the best combination of accuracy and explainability.

Model Deployment We initially considered running the model within separate inference servers, but switched to an in-binary approach. Running our models at $<$10us latency became critical for our reprediction-based approach. We also found that depending on separate servers would have added a circular dependency from a reliability standpoint, since those servers would themselves run on top of Borg. For the same reason, we included the model in the binary as opposed to loading it from, e.g., a distributed file system.

Rollout A major concern in our production environment was how to roll out the model. Rolling out the model independently of Borg risked the model becoming a single point of failure by bypassing existing verification approaches (e.g., gradual rollout and canary deployments). We found that treating the model like any other code change and rolling it out with new versions of the binary side-stepped this issue by subjecting models to the same careful production testing. We also relied extensively on simulations for backtesting and tuning design parameters, to avoid production changes.

Production Monitoring During production rollout, we carefully monitored the behavior of NILAS, by adding additional telemetry and monitoring dashboards. This telemetry captures, e.g., the increase in CPU and memory caused by loading the model in production, the increase in time each scheduling pass takes, and how often the new cost component becomes the tie breaker. This instrumentation confirms the system is working as intended and debugs production issues. For example, in one of our initial deployments, we found a bug where the model was called incorrectly.

8 Related Work

Prediction-based job scheduling has a long history. Some work uses reinforcement learning (Mao et al., 2019; Li et al., 2021; Wang et al., 2022), e.g., for placing containers. Other work predicts properties of jobs and uses them to improve scheduling for scale-out (Delimitrou & Kozyrakis, 2014a), interference (Delimitrou & Kozyrakis, 2013), or goodput (Jayaram Subramanya et al., 2023).

Job lifetimes are particularly useful for scheduling. Jockey (Ferguson et al., 2012) and Aria (Verma et al., 2011) predict lifetimes (called runtimes) in data processing frameworks and use them to improve scheduling of dataflow-based jobs. TetriSched (Tumanov et al., 2016) uses runtime predictions in conjunction with an MILP solver to schedule jobs in an HPC-style setting. Wrangler predicts stragglers (Yadwadkar et al., 2014). 3Sigma (Park et al., 2018) predicts various properties of Google job traces and uses them for scheduling. Note that most of this work uses runtime predictions in domain-specific scenarios and does not leverage general machine learning techniques to schedule.

In Cloud Computing, Microsoft's Resource Central system (Cortez et al., 2017) performs predictions on VMs and uses them for various tasks, including VM scheduling and admission control (Sajal et al., 2023). Microsoft's LA algorithm that performs lifetime-based VM scheduling is most similar to our work (Barbalho et al., 2023; Buchbinder et al., 2021). Our work differs in that instead of performing one-shot predictions when the VM starts running, it repredicts VM lifetimes as needed, updating lifetimes based on new knowledge, delivering good improvements in empty hosts over LA, as shown in Section 6. We show how to incorporate lifetimes in our existing scheduler by integrating the models into the binary. Our models run in-process at very low latency and our new scheduling algorithms tolerate mispredictions. LAVA changes the objective function from LA and NILAS – instead of placing VMs with similar lifetimes on the same host, LAVA seeks to make more hosts empty by *not* extending the host lifetime, by placing short lived VMs on a host with one or more long lived VM(s).

9 Conclusion

We presented a new approach to lifetime-aware VM allocation in cloud data centers. Our approach continually repredicts VM and host lifetimes, and corrects for past mistakes. NILAS is deployed fleet-wide for one VM family at Google, where it shows significant improvements by increasing empty hosts and reducing stranding.

Acknowledgements: We would like to thank Deniz Altınbüken, Mike Dahlin, Vahab Mirrokni, Andreas Terzis and the anonymous reviewers for their feedback. We also want to thank Anshul Kapoor for critical contributions to the project after the paper had been submitted. Finally, we thank Wan Chen, Eileen Feng, Vijayan Satyamoorthy, Dmitry Shiraev, Samuel Smith and Sirui Sun for their contributions to this work.

REFERENCES

Barbalho, H., Kovaleski, P., Li, B., Marshall, L., Molinaro, M., Pan, A., Cortez, E., Leao, M., Patwari, H., Tang, Z., Santos, T., Gonçalves, L. R., Dion, D., Moscibroda, T., and Menache, I. Virtual machine allocation with lifetime predictions. In *MLSys*, June 2023. URL https://www.microsoft.com/en-us/research/publication/virtual-machine-allocation-with-lifetime-predictions/.

Brodersen, K. H., Gallusser, F., Koehler, J., Remy, N., and Scott, S. L. Inferring causal impact using bayesian structural time-series models. *The Annals of Applied Statistics*, 9(1):247–274, 2015. ISSN 19326157. URL http://www.jstor.org/stable/24522418.

Buchbinder, N., Fairstein, Y., Mellou, K., Menache, I., and Naor, J. S. Online virtual machine allocation with lifetime and load predictions. In *Abstract Proceedings of the 2021 ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '21, pp. 9–10, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380720. doi: 10.1145/3410220.3456278. URL https://doi.org/10.1145/3410220.3456278.

Chen, T. and Guestrin, C. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 785–794, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939785. URL http://doi.acm.org/10.1145/2939672.2939785.

Chollet, F. et al. Keras. https://keras.io, 2015.

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 International Symposium on Operating Systems Principles (SOSP)*, October 2017. URL https://www.microsoft.com/en-us/research/publication/resource-central-understanding-predicting-workloads-improved-resource-management-large-cloud-platforms/.

Cox, D. R. Regression Models and Life-Tables. *Journal of the Royal Statistical Society: Series B (Methodological)*, 34(2):187–202, 12 2018. ISSN 0035-9246. doi: 10.1111/j.2517-6161.1972.tb00899.x. URL https://doi.org/10.1111/j.2517-6161.1972.tb00899.x.

Davidson-Pilon, C. lifelines: survival analysis in python. *Journal of Open Source Software*, 4(40):1317, 2019. doi: 10.21105/joss.01317. URL https://doi.org/10.21105/joss.01317.

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 '13, pp. 77–88, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450318709. doi: 10.1145/2451116.2451125. URL https://doi.org/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 '14, pp. 127–144, New York, NY, USA, 2014a. Association for Computing Machinery. ISBN 9781450323055. doi: 10.1145/2541940.2541941. URL https://doi.org/10.1145/2541940.2541941.

Delimitrou, C. and Kozyrakis, C. Quasar: resource-efficient and qos-aware cluster management. *SIGPLAN Not.*, 49 (4):127–144, feb 2014b. ISSN 0362-1340. doi: 10.1145/2644865.2541941. URL https://doi.org/10.1145/2644865.2541941.

Ferguson, A. D., Bodik, P., Kandula, S., Boutin, E., and Fonseca, R. Jockey: guaranteed job latency in data parallel clusters. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pp. 99–112, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450312233. doi: 10.1145/2168836.2168847. URL https://doi.org/10.1145/2168836.2168847.

Gog, I., Schwarzkopf, M., Gleave, A., Watson, R. N. M., and Hand, S. Firmament: Fast, centralized cluster scheduling at scale. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 99–115, Savannah, GA, November 2016. USENIX Association. ISBN 978-1-931971-33-1. URL https://www.usenix.org/conference/osdi16/technical-sessions/presentation/gog.

Grandl, R., Kandula, S., Rao, S., Akella, A., and Kulkarni, J. J. Graphene: Packing and dependency-aware scheduling for data-parallel clusters. In *Operating Systems Design and Implementation*. USENIX, October 2016. URL https://www.microsoft.com/en-us/research/publication/graphene-packing-dependency-aware-scheduling-data-parallel-clusters/.

Guillame-Bert, M., Bruch, S., Stotz, R., and Pfeifer, J. Yggdrasil decision forests: A fast and extensible decision forests library. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, pp. 4068–4077, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701030. doi: 10.1145/3580305.3599933. URL https://doi.org/10.1145/3580305.3599933.

Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R., Shenker, S., and Stoica, I. Mesos: A platform for Fine-Grained resource sharing in the data center. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*, Boston, MA, March 2011. USENIX Association. URL https://www.usenix.org/conference/nsdi11/mesos-platform-fine-grained-resource-sharing-data-center.

Jayaram Subramanya, S., Arfeen, D., Lin, S., Qiao, A., Jia, Z., and Ganger, G. R. Sia: Heterogeneity-aware, goodput-optimized ml-cluster scheduling. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, pp. 642–657, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702297. doi: 10.1145/3600006.3613175. URL https://doi.org/10.1145/3600006.3613175.

Kaplan, E. L. and Meier, P. Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*, 53(282):457–481, 1958. ISSN 01621459, 1537274X. URL http://www.jstor.org/stable/2281868.

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 *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '21, pp. 258–272, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450386388. doi: 10.1145/3472883.3486971. URL https://doi.org/10.1145/3472883.3486971.

Lieberman, H. and Hewitt, C. A real-time garbage collector based on the lifetimes of objects. *Communications of the ACM*, 26(6):419–429, June 1983. ISSN 0001-0782. doi: 10.1145/358141.358147. URL https://doi.org/10.1145/358141.358147.

Maas, M., Andersen, D. G., Isard, M., Javanmard, M. M., McKinley, K. S., and Raffel, C. Learning-based memory allocation for c++ server workloads. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, pp. 541–556, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371025. doi: 10.1145/3373376.3378525. URL https://doi.org/10.1145/3373376.3378525.

Mao, H., Schwarzkopf, M., Venkatakrishnan, S. B., Meng, Z., and Alizadeh, M. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, pp. 270–288, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359566. doi: 10.1145/3341302.3342080. URL https://doi.org/10.1145/3341302.3342080.

Park, J. W., Tumanov, A., Jiang, A., Kozuch, M. A., and Ganger, G. R. 3sigma: distribution-based cluster scheduling for runtime uncertainty. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355841. doi: 10.1145/3190508.3190515. URL https://doi.org/10.1145/3190508.3190515.

Pölsterl, S. scikit-survival: A library for time-to-event analysis built on top of scikit-learn. *Journal of Machine Learning Research*, 21(212):1–6, 2020. URL http://jmlr.org/papers/v21/20-729.html.

Ruprecht, A., Jones, D., Shiraev, D., Harmon, G., Spivak, M., Krebs, M., Baker-Harvey, M., and Sanderson, T. VM live migration at scale. In *Proceedings of the 14th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '18, pp. 45–56, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355797. doi: 10.1145/3186411.3186415. URL https://doi.org/10.1145/3186411.3186415.

Sajal, S. M., Marshall, L., Li, B., Zhou, S., Pan, A., Mellou, K., Narayanan, D., Zhu, T., Moscibroda, T., Dion, D., and Menache, I. Kerveros: Efficient and scalable cloud admission control. In *OSDI*. USENIX, July 2023. URL https://www.microsoft.com/en-us/research/publication/kerveros-efficient-and-scalable-cloud-admission-control/.

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 '13, pp. 351–364, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450319942. doi: 10.1145/2465351.2465386. URL https://doi.org/10.1145/2465351.2465386.

Tumanov, A., Zhu, T., Park, J. W., Kozuch, M. A., Harchol-Balter, M., and Ganger, G. R. Tetrisched: global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342407. doi: 10.1145/2901318.2901355. URL https://doi.org/10.1145/2901318.2901355.

Ungar, D. Generation scavenging: A non-disruptive high performance storage reclamation algorithm. In *Proceedings of the First ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, SDE 1, pp. 157–167, New York, NY, USA, 1984. Association for Computing Machinery. ISBN 0897911318. doi: 10.1145/800020.808261. URL https://doi.org/10.1145/800020.808261.

Verma, A., Cherkasova, L., and Campbell, R. H. Aria: automatic resource inference and allocation for mapreduce environments. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pp. 235–244, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306072. doi: 10.1145/1998582.1998637. URL https://doi.org/10.1145/1998582.1998637.

Verma, A., Pedrosa, L., Korupolu, M. R., Oppenheimer, D., Tune, E., and Wilkes, J. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.

Wang, P., Li, Y., and Reddy, C. K. Machine learning for survival analysis: A survey. *ACM Comput. Surv.*, 51(6), feb 2019. ISSN 0360-0300. doi: 10.1145/3214306. URL https://doi.org/10.1145/3214306.

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*, ASPLOS '22, pp. 622–634, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392051. doi: 10.1145/3503222.3507704. URL https://doi.org/10.1145/3503222.3507704.

Yadwadkar, N. J., Ananthanarayanan, G., and Katz, R. Wrangler: Predictable and faster jobs using fewer resources. In *ACM Symposium on Cloud Computing (SoCC)*, November 2014. URL https://www.microsoft.com/en-us/research/publication/wrangler-predictable-faster-jobs-using-fewer-resources/.

Zhang, Z., Li, C., Tao, Y., Yang, R., Tang, H., and Xu, J. Fuxi: a fault-tolerant resource management and job scheduling system at internet scale. *Proc. VLDB Endow.*, 7(13):1393–1404, aug 2014. ISSN 2150-8097. doi: 10.14778/2733004.2733012. URL https://doi.org/10.14778/2733004.2733012.

Zhou, G. and Maas, M. Learning on distributed traces for data center storage systems. In Smola, A., Dimakis, A., and Stoica, I. (eds.), *Proceedings of Machine Learning and Systems*, volume 3, pp. 350–364, 2021. URL https://proceedings.mlsys.org/paper_files/paper/2021/file/efe0df3ea4a53a04614ad79e7a8a57de-Paper.pdf.