Google

LAVA: Lifetime-Aware VM Allocation with Learned Distributions and Adaptation to Mispredictions

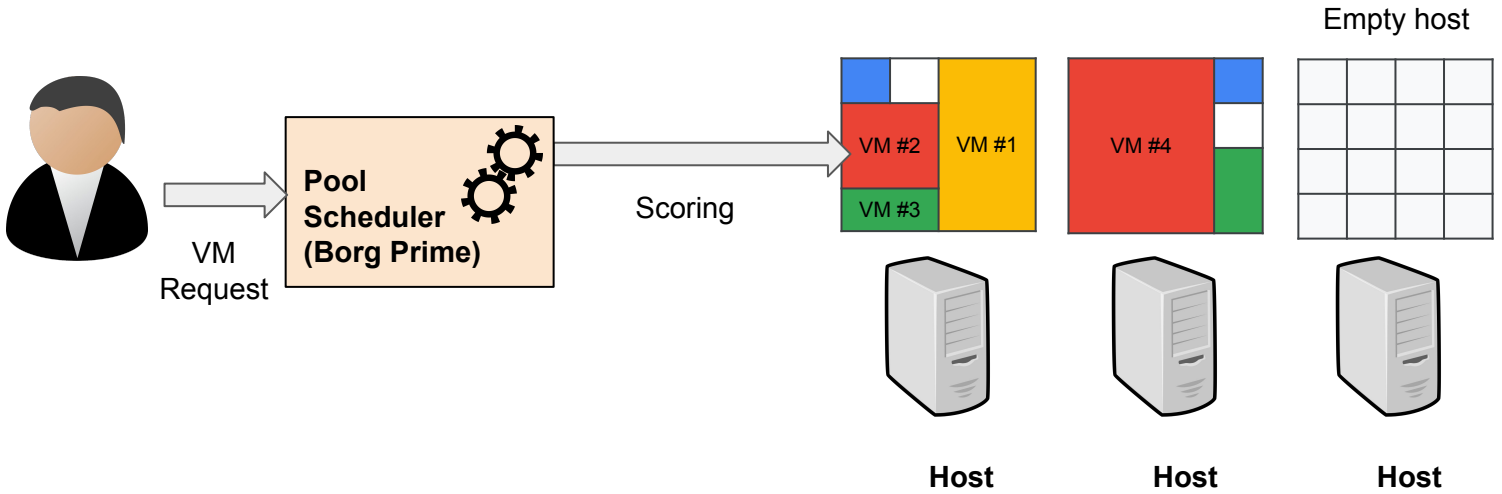
Speakers: Yunchuan Kong & Kathryn S McKinley

MLSys 2025 – Last Session (Thursday May 15 4:30)

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

VM Scheduling at Google

Scheduling **Google Cloud VMs** is a large-scale problem: **10-100 scheduling request/s** on **100s to 10,000+ hosts**, **O(100-10,000) VMs active** per cluster. Our scheduler, *Borg Prime*, schedules VMs 1) when they **arrive**, 2) during **host maintenance**, and 3) to **defragment** the fleet.



VM Scheduling Objectives

- **Maximize empty hosts** to put empty hosts in a low power state, improve large VM obtainability, improve maintenance speed, use hosts elsewhere, etc.
- **Minimize stranding** to fill all resource dimensions on each host, maximizing useful CPU, memory, and SSD.
- **Minimize VM disruptions** when defragmenting the fleet or performing maintenance operations.

Lifetime-Aware Scheduling

In 2020, we invented **lifetime-based memory allocation for C++**, where we allocated objects to huge pages based on ML-predicted lifetimes while correcting for mispredictions.

We observed that **the same approach applies to cloud VMs**.

Learning-based Memory Allocation for C++ Server Workloads

Martin Maas, David G. Andersen*†, Michael Isard, Mohammad Mahdi Javanmard*‡, Kathryn S. McKinley, Colin Raffel

Google Research †Carnegie Mellon University ‡Stony Brook University

Abstract

Modern C++ servers have memory footprints that vary widely over time, causing persistent heap fragmentation of up to 2× from long-lived objects allocated during peak memory usage. This fragmentation is exacerbated by the use of huge (2MB) pages, a requirement for high performance on large heap sizes. Reducing fragmentation automatically is challenging because C++ memory managers cannot move objects.

This paper presents a new approach to huge page fragmentation. It combines modern machine learning techniques with a novel memory manager (LLAMA) that manages the heap based on object lifetimes and huge pages (divided into blocks and lines). A neural network-based language model predicts lifetime classes using symbolized calling contexts. The model learns context-sensitive per-allocation site lifetimes from previous runs, generalizes over different binary versions, and extrapolates from samples to unobserved calling contexts. Instead of size classes, LLAMA's heap is organized by *lifetime* classes that are dynamically adjusted based on observed behavior at a block granularity.

LLAMA reduces memory fragmentation by up to 78% while only using huge pages on several production servers. We address ML-specific questions such as tolerating mispredictions and amortizing expensive predictions across application execution. Although our results focus on memory allocation, the questions we identify apply to other system-level problems with strict latency and resource requirements where machine learning could be applied.

CCS Concepts • **Computing methodologies → Supervised learning;** • **Software and its engineering → Allocation / deallocation strategies;**

Keywords Memory management, Machine Learning, Lifetime Prediction, Profile-guided Optimization, LSTMs

* Work done while at Google.

ASPLOS '20, March 16–20, 2020, Lausanne, Switzerland

ACM ISBN 978-1-4503-7102-5/20/03.
https://doi.org/10.1145/3373376.3378525

ACM Reference Format:
Martin Maas, David G. Andersen, Michael Isard, Mohammad Mahdi Javanmard, Kathryn S. McKinley, Colin Raffel. 2020. 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), March 16–20, 2020, Lausanne, Switzerland.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3373376.3378525

1 Introduction

Optimizing interactive web services, many of which are written in C++, requires meeting strict latency requirements while minimizing resource usage. Users abandon services if response times are too slow and data center costs are directly proportional to resource usage. Multithreaded services require large heaps both to minimize the number of deployed instances and to handle multiple requests simultaneously. Hardware has not kept pace with these demands. While memory sizes have increased, Translation Lookaside Buffers (TLB) have not, because address translation is on the critical path. One solution is increasing TLB reach with huge (2 MB) pages, i.e., each entry covers more memory. Huge pages reduce TLB misses, improving performance by up to 53% [33, 37]. Looking forward, 1 GB pages are already available and variable-sized ranges can eliminate even more TLB misses [27, 33]. Future virtual memory systems may hence predominantly rely on huge pages and ranges.

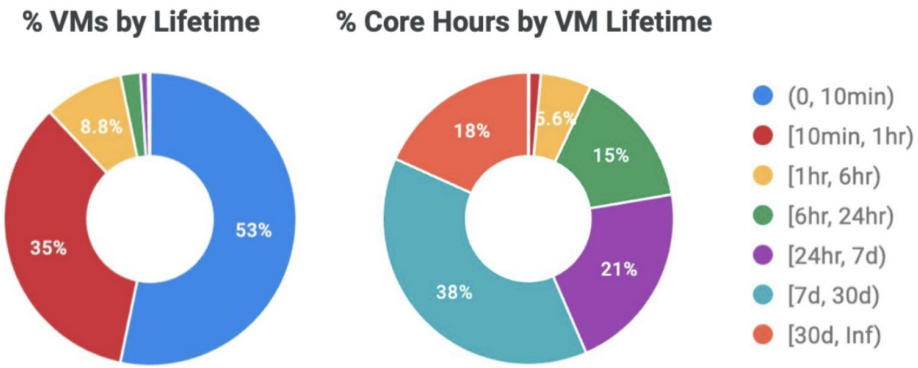
These trends require workloads to efficiently use huge pages. While Operating Systems (OS) have explored transparent huge pages [37, 45], they either trade performance for space, increasing the physical memory footprint by up to 23% and 69% on server workloads [37], or break up huge pages, sacrificing performance (TLB hits) and depleting contiguous physical memory for all workloads on the machine [37, 45]. If the C++ memory allocator is not huge page aware, it may further defeat the OS. Only one C++ memory allocator in the literature uses huge pages, but its evaluation uses microbenchmarks [36]. To our knowledge, no current memory allocator efficiently manages memory entirely with huge pages without incurring significant fragmentation.

We identify a root cause of huge page fragmentation in long-running servers: allocations of long-lived objects at peak memory usage. Since C++ allocators cannot move objects, using huge pages increase the probability of one long-lived object preventing a page from being released to the

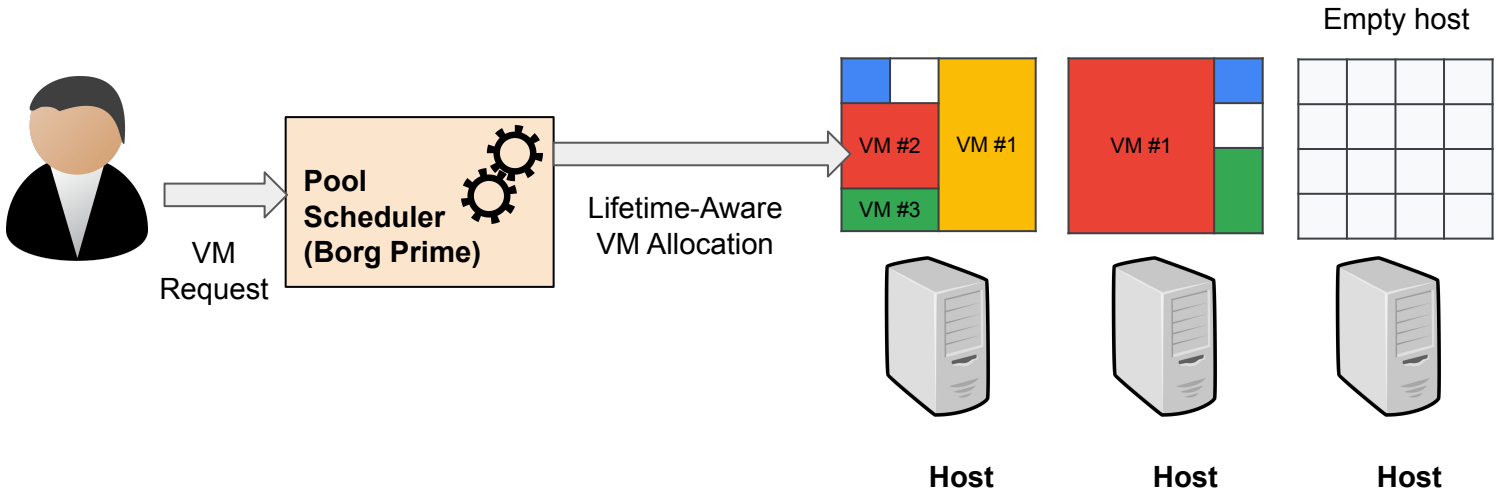
Google

Key Insight

88% of VMs are short-lived, but only **2% of core hours** are consumed by short-lived VMs. Weighting all VMs the same in bin packing is **fundamentally inefficient**.

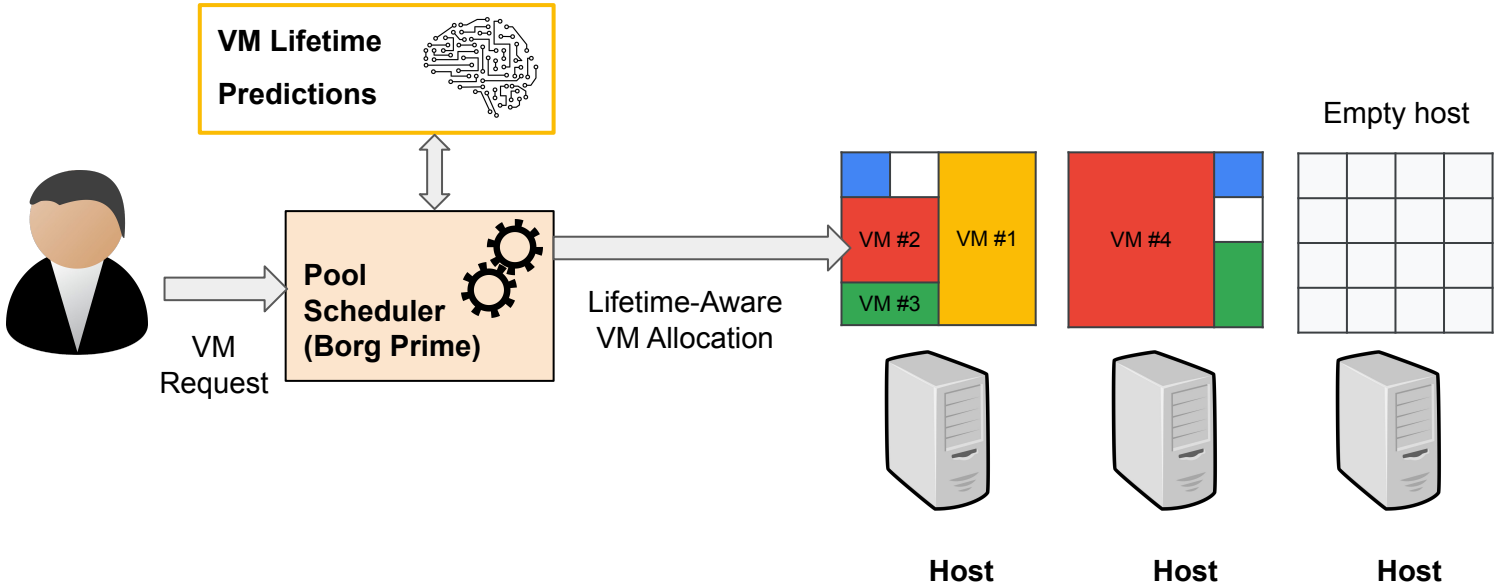


High-Level Setup



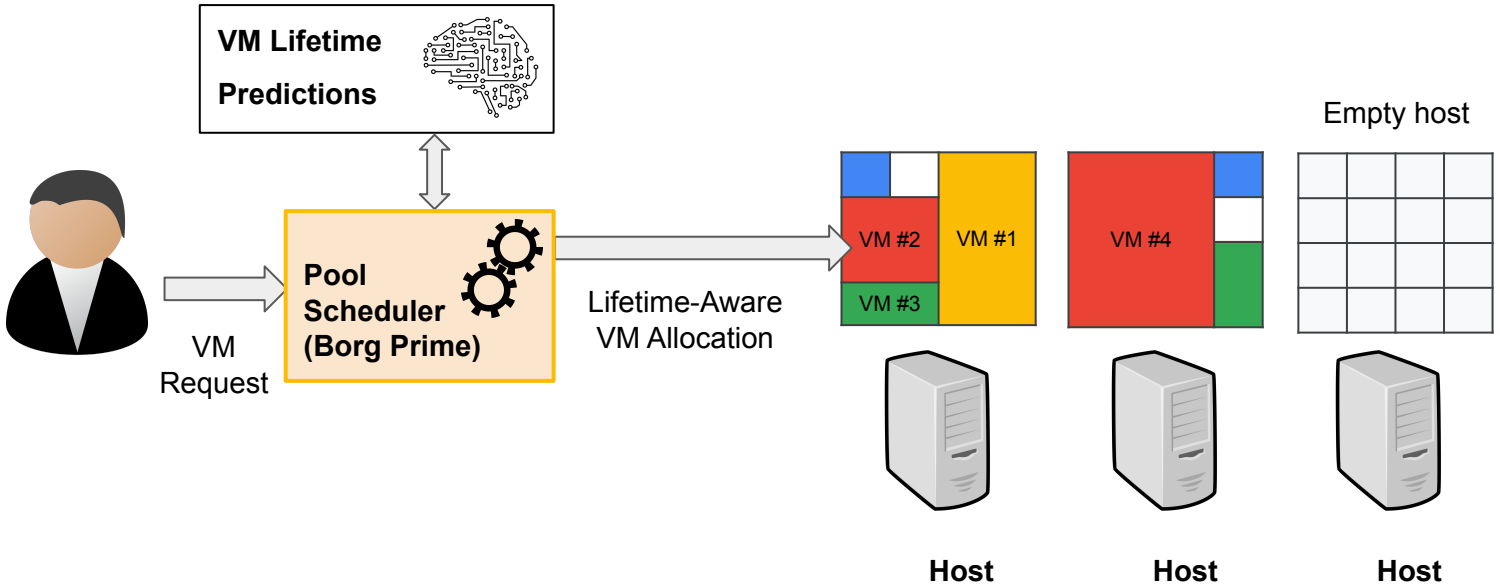
High-Level Setup

ML Model predicts the lifetime of a VM when it arrives and **repredicts** lifetimes of VMs on candidate hosts.



High-Level Setup

New **cluster scheduling algorithms** use these predictions to improve scheduling objectives.



Related Work

Lots of prior work on prediction-based scheduling:

- **Many papers, going back 10+ years**: Quasar, Paragon, Wrangler, 3Sigma, Jockey, TetriSched, etc.
- The closest work is Barbalho et al. from MLSys'23, introduced the **Lifetime Alignment (LA) algorithm**.

↪ **Google's environment is unique** and our approach is thus quite different.

Our Key Novel Innovations

- **Adaptation to mispredictions**: **Continually repredict** VM and host lifetime based on new observations.
- **New Algorithmic Approach**: Sometimes place VMs with very different, not similar, lifetimes together.
- **Lifetime-aware Maintenance & Defragmentation**: Exploit lifetimes to reduce the number of migrations.

↪ **We improve SOTA scheduling quality**

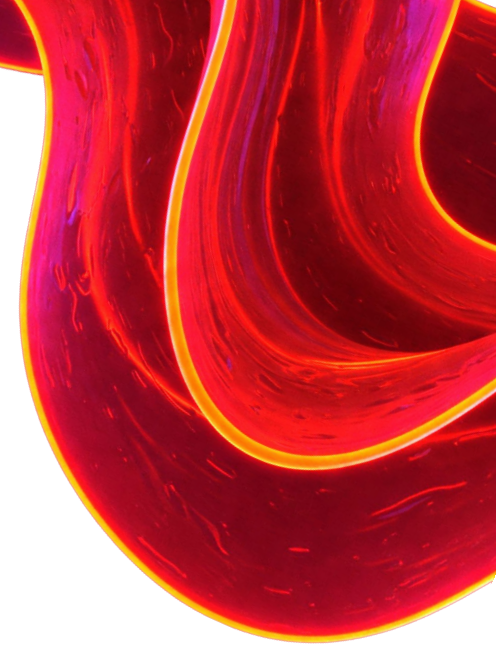
Talk Outline

1 **Algorithms & Approach**
Lifetime-aware VM scheduling techniques

2 **Production Experience**
Deploying lifetime-aware scheduling in Google Cloud

3 **Evaluation**
Evaluating our approach in production and simulation

PART I
Algorithms & Approach
VM lifetime models and lifetime-aware VM scheduling

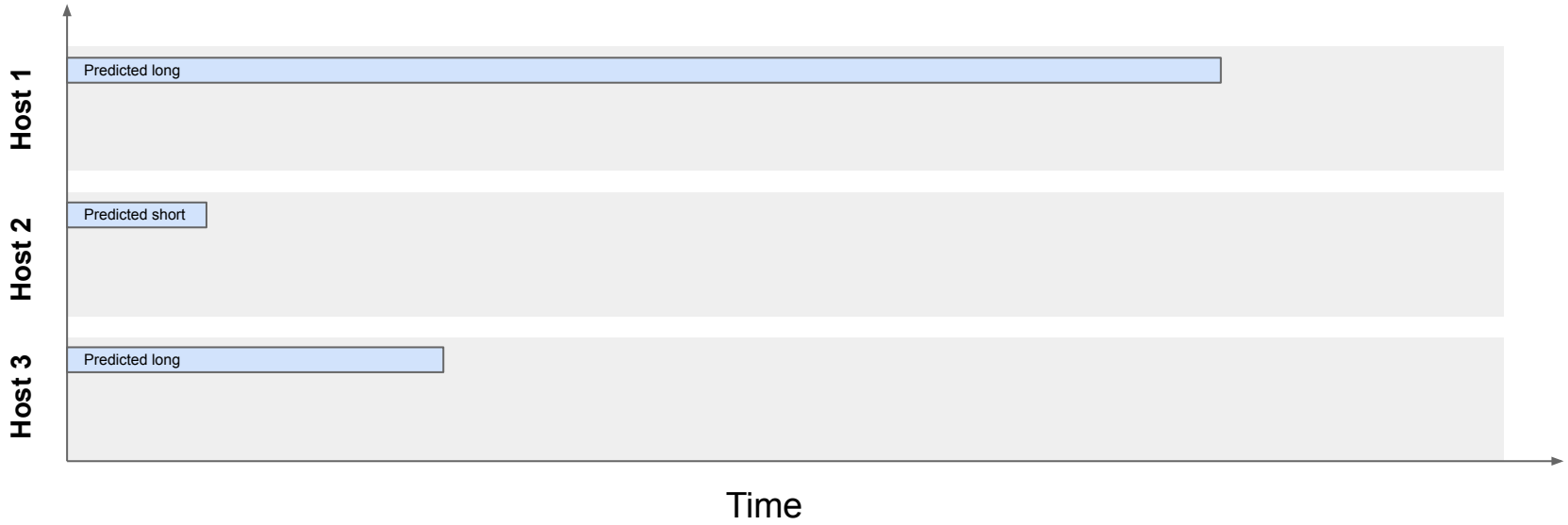


Basic Algorithm (Non-Invasive)

Existing scheduling algorithm, with lifetime as tie-breaker.

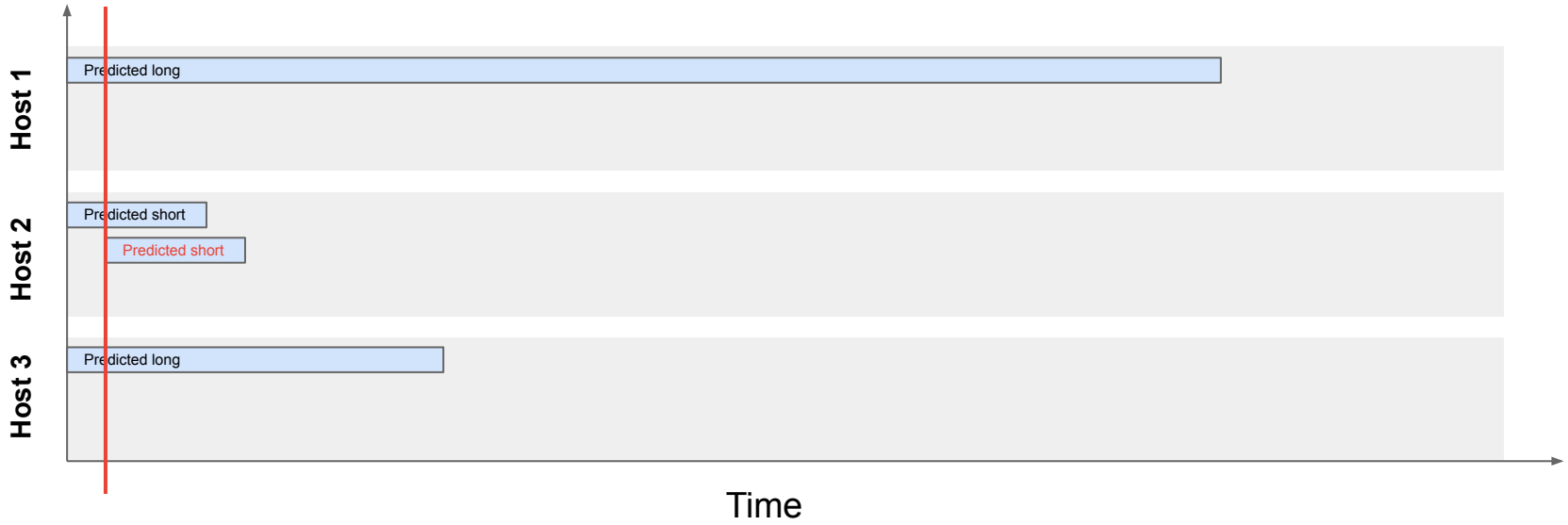
Baseline: LA (MLSys'23)

Predict a lifetime once when the VM arrives and place it on the host with the closest exit time based on its VMs.



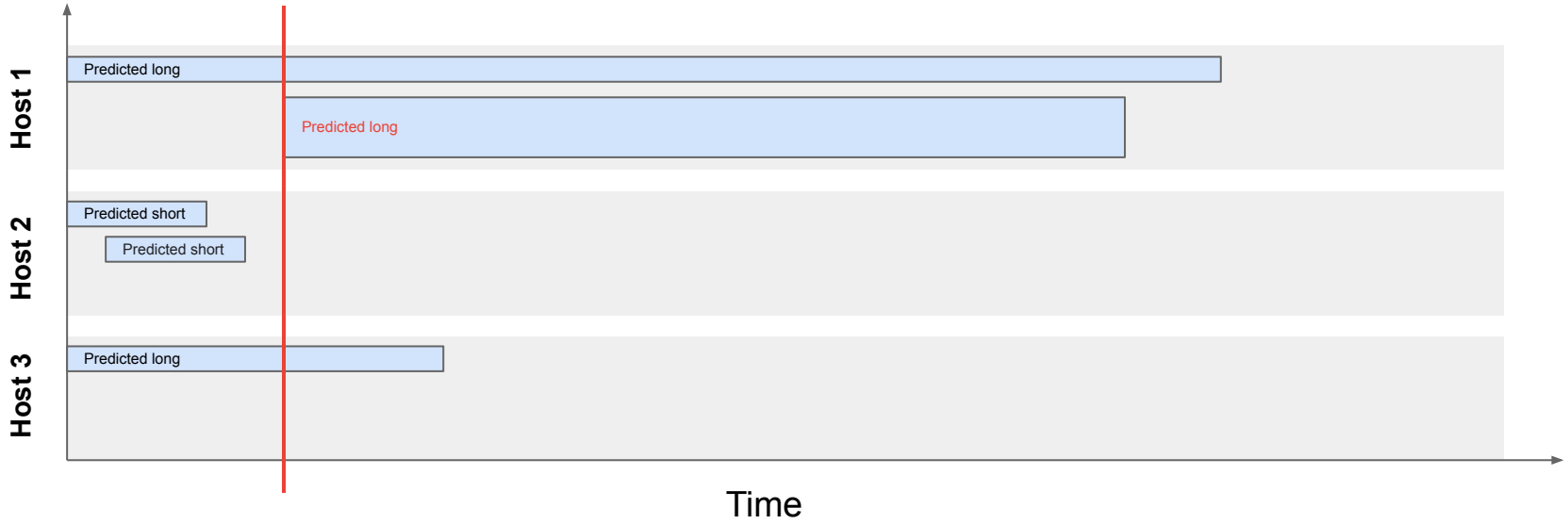
Baseline: LA (MLSys'23)

Predict a lifetime once when the VM arrives and place it on the host with the closest exit time based on its VMs.



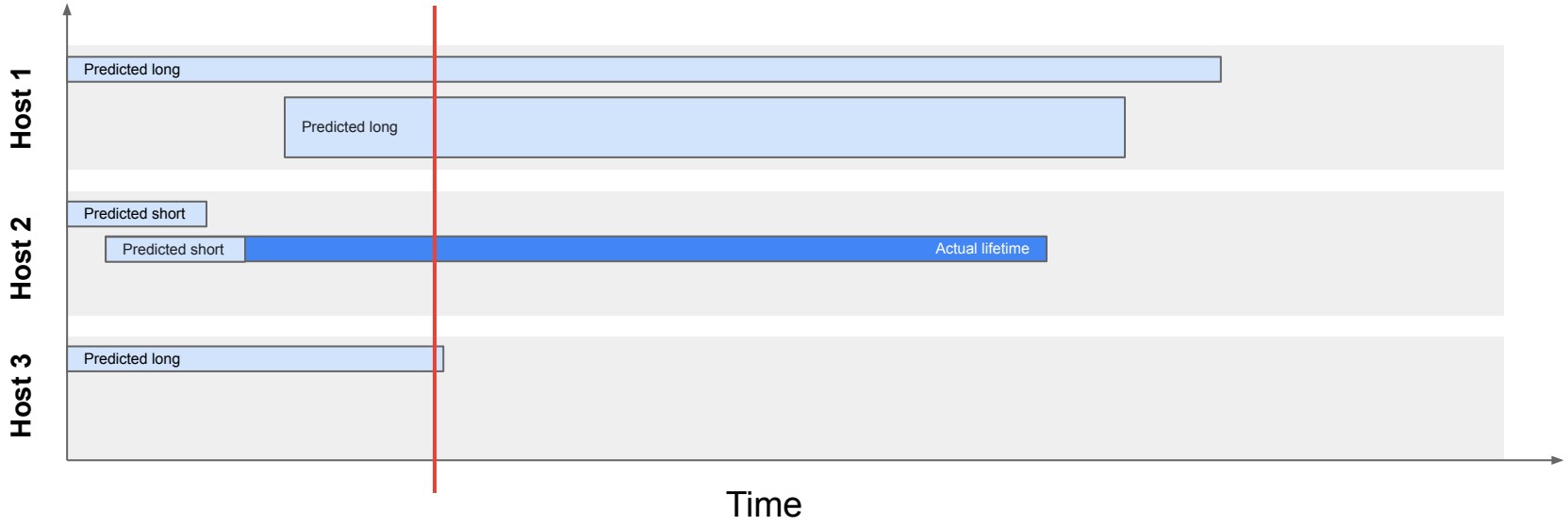
Baseline: LA (MLSys'23)

Predict a lifetime once when the VM arrives and place it on the host with the closest exit time based on its VMs.



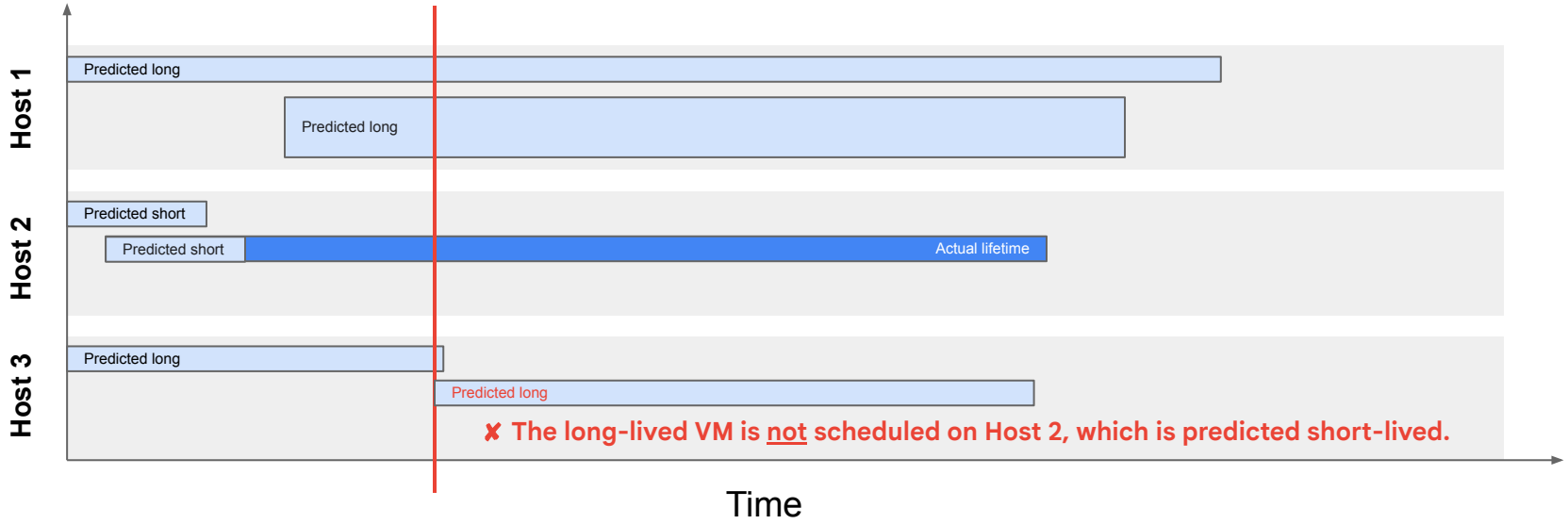
Baseline: LA (MLSys'23)

Predict a lifetime once when the VM arrives and place it on the host with the closest exit time based on its VMs.



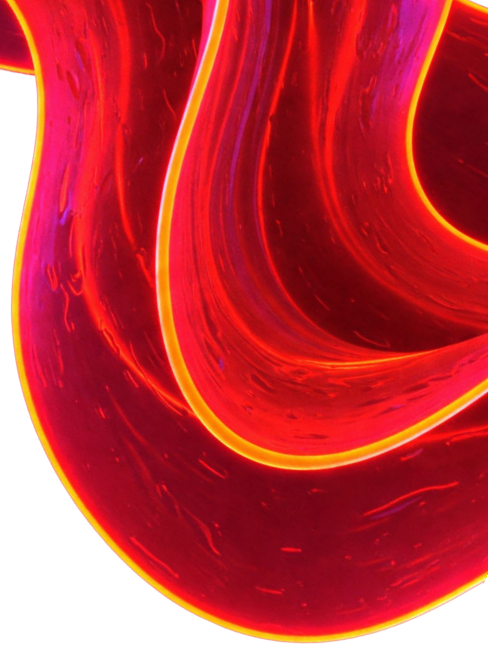
Baseline: LA (MLSys'23)

Predict a lifetime once when the VM arrives and place it on the host with the closest exit time based on its VMs.



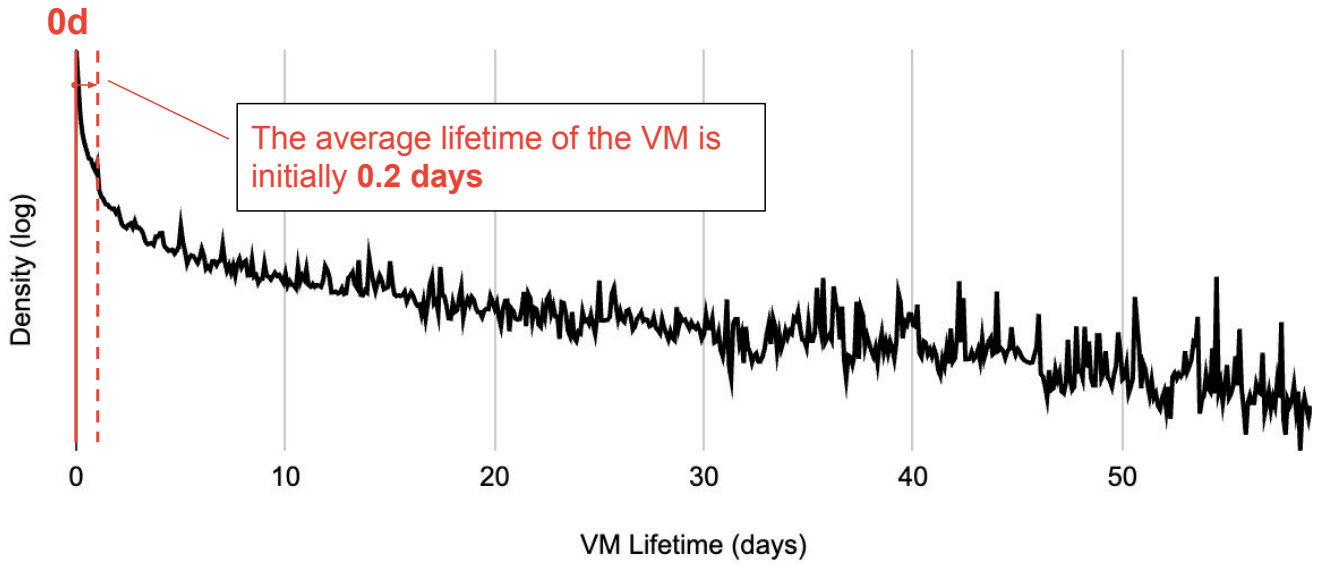
Key Idea

Revisit and correct previous mispredictions when more information is known.



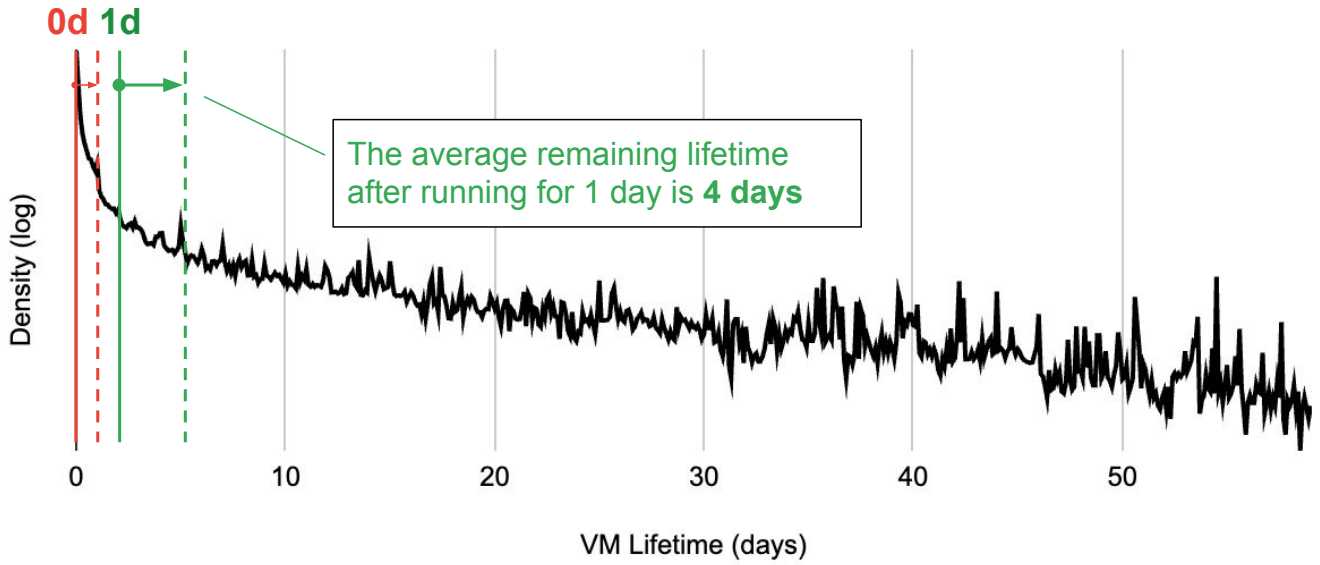
Distribution-Based Predictions

Instead of predicting the lifetime once, predict updated **conditional lifetimes** based on running for X days.



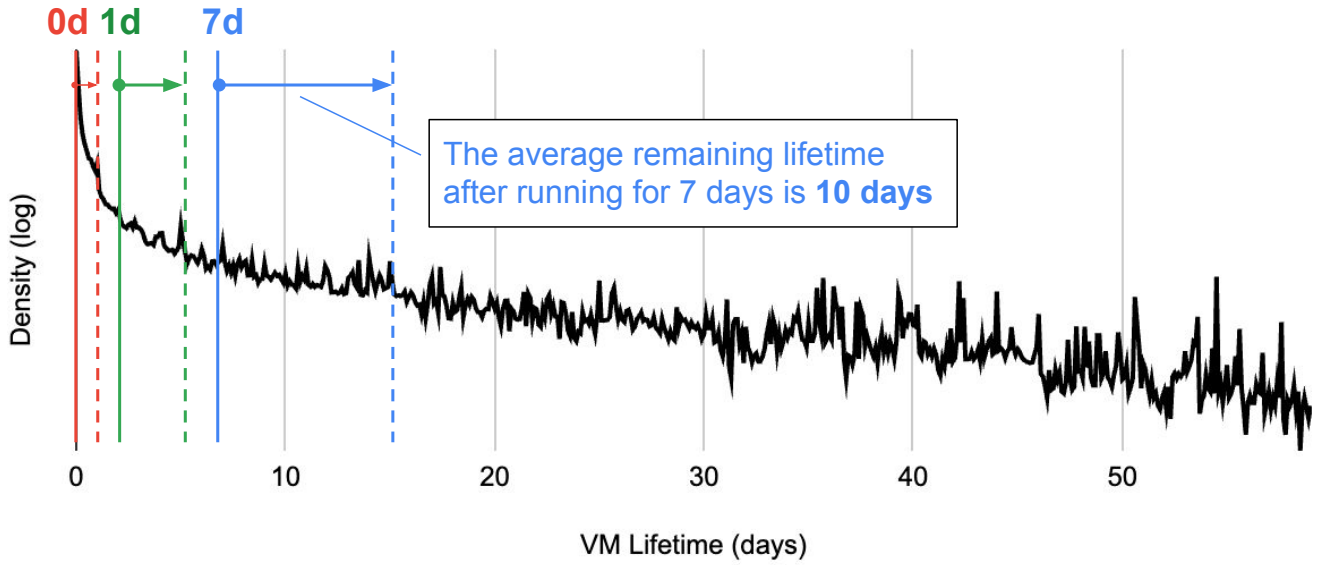
Distribution-Based Predictions

Instead of predicting the lifetime once, predict updated **conditional lifetimes** based on running for X days.



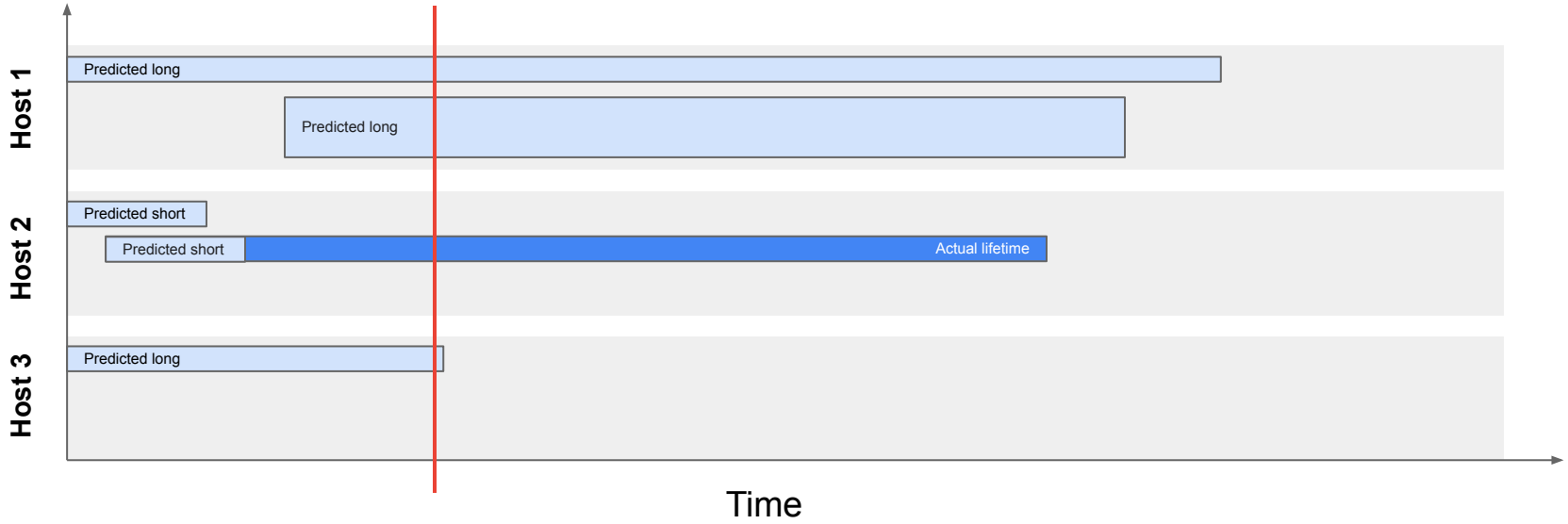
Distribution-Based Predictions

Instead of predicting the lifetime once, predict updated **conditional lifetimes** based on running for X days.



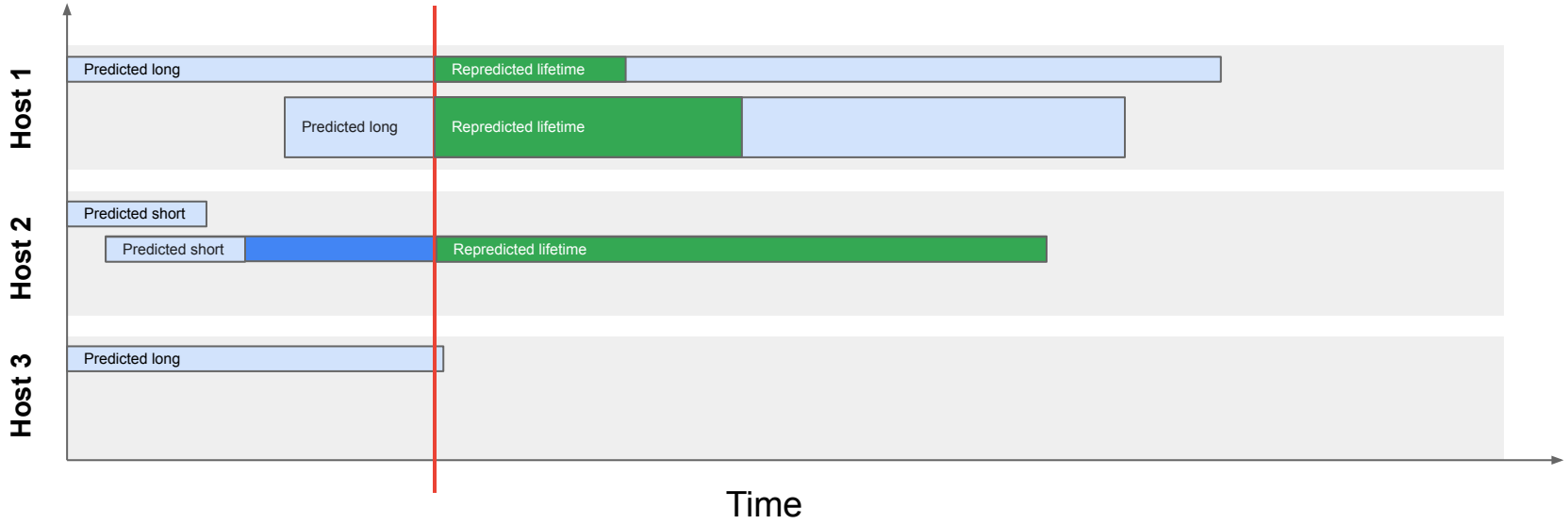
Algorithm 1: NILAS

When scoring a host, repredict all the lifetimes of the VMs on the host based on the uptime so far.



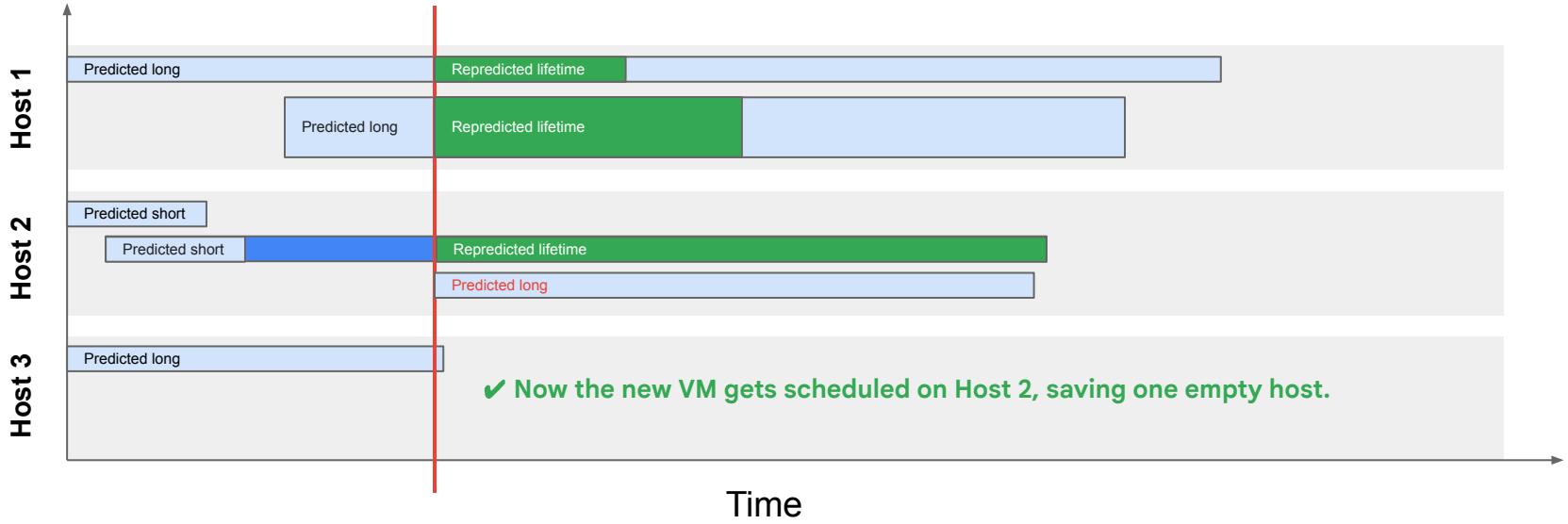
Algorithm 1: NILAS

When scoring a host, repredict all the lifetimes of the VMs on the host based on the uptime so far.



Algorithm 1: NILAS

When scoring a host, repredict all the lifetimes of the VMs on the host based on the uptime so far.

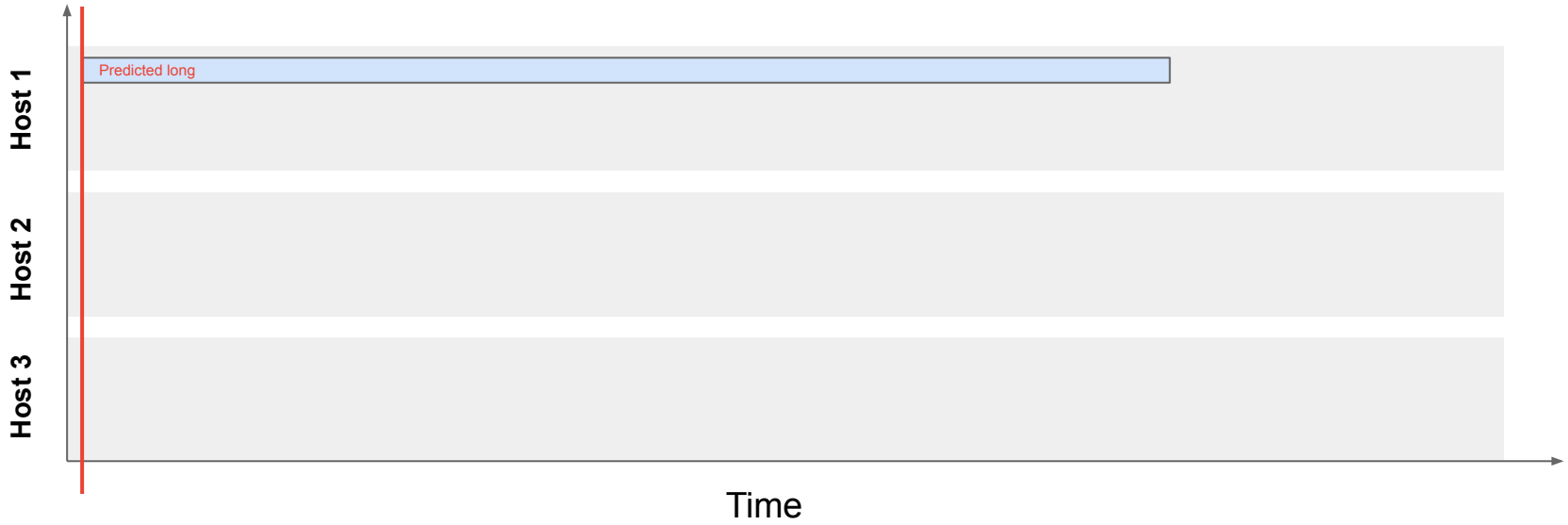


Advanced Algorithm (Invasive)

A fundamental redesign of VM scheduling with lifetimes

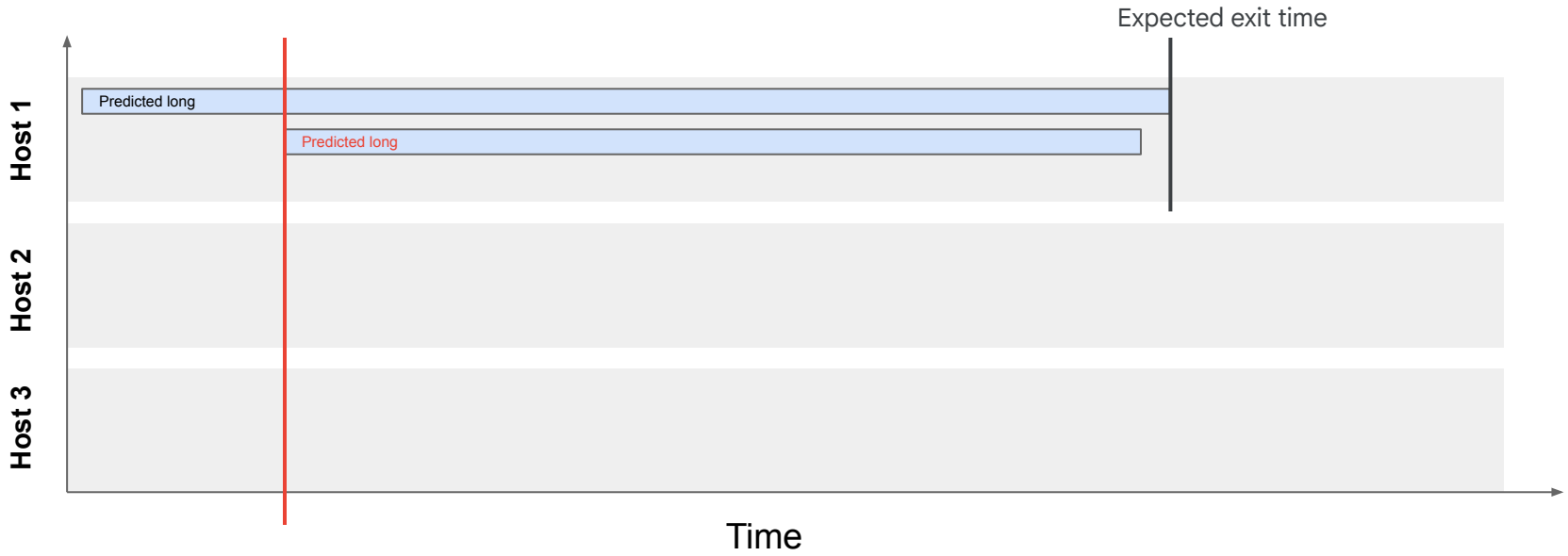
Algorithm 2: LAVA

Problem: Mispredictions push out the exit time.



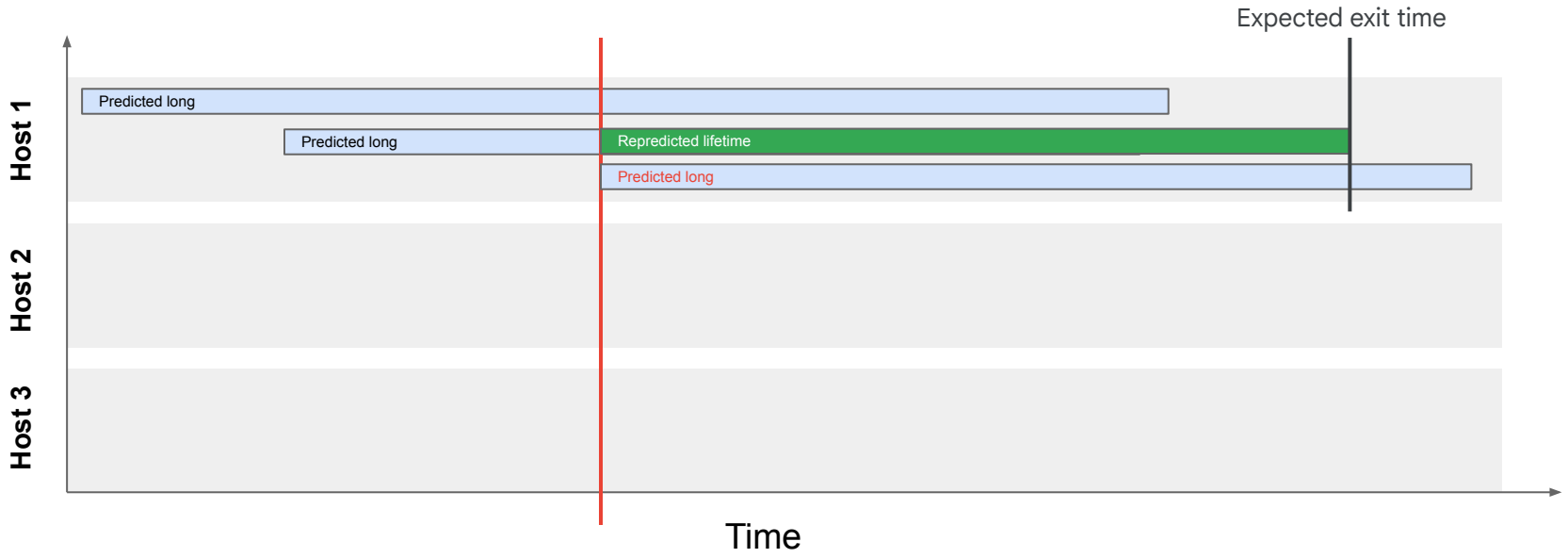
Algorithm 2: LAVA

Problem: Mispredictions push out the exit time.



Algorithm 2: LAVA

Problem: Mispredictions push out the exit time.

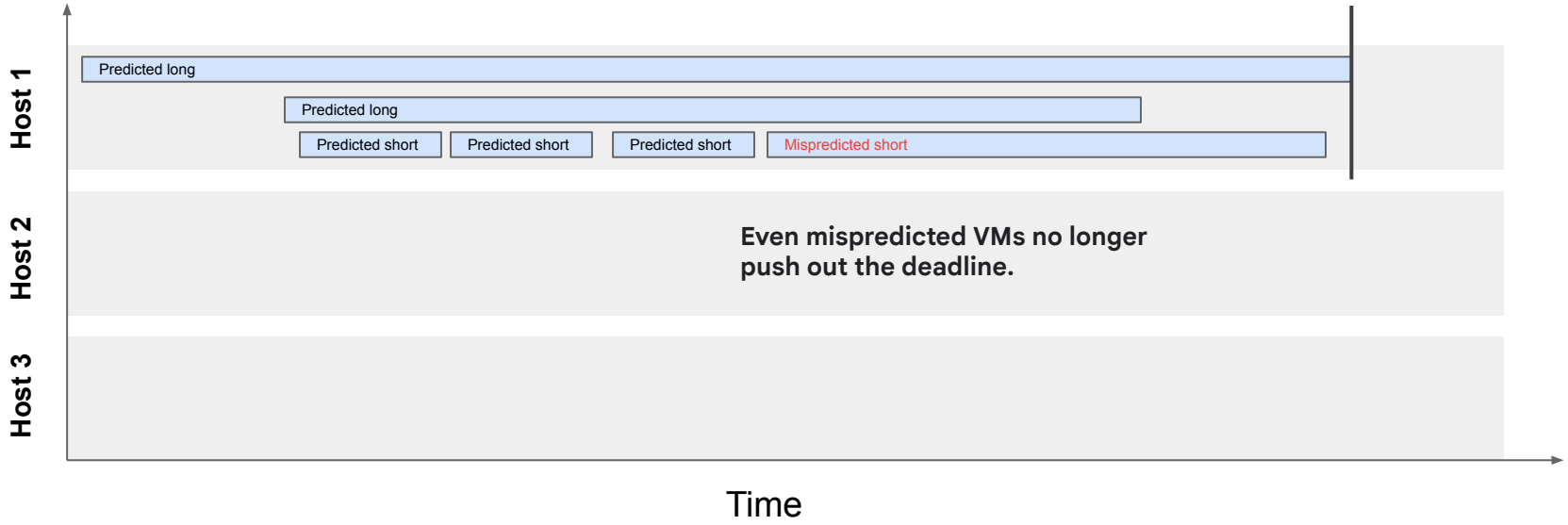


Key Idea

Fill gaps between long-lived VMs with much shorter-lived VMs.

Algorithm 2: LAVA

LAVA: Inspired by LLAMA algorithm in C++ memory allocation, fill gaps between long VMs with short ones.



PART II
Production Experience
Deploying lifetime-aware scheduling in Google Cloud

Model Choice & Integration

Prediction is **on the critical path** for VM scheduling.

- **Model**: Gradient boosted trees,
  - balances accuracy & performance
  - part of the scheduler binary, *not on other servers*.
- Median latency is **9 us** (780x reduction vs. MLSys'23).

Safety & Correctness

Model is bundled in system binary and updated ~monthly.

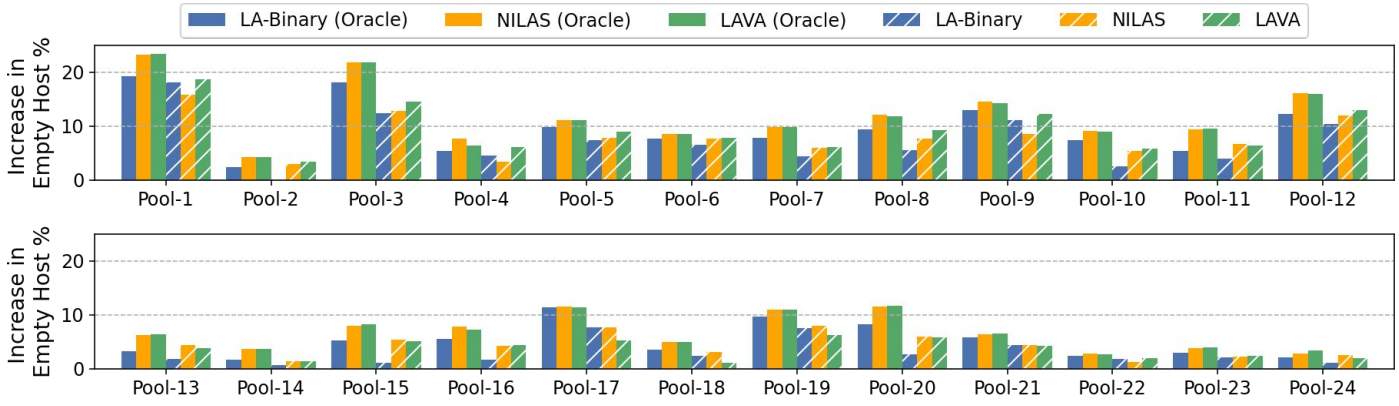
- **Offline training** with **backtesting**.
- Model is **tested** and **rolled-out gradually**.
- **Monitoring and Alerting** catch regressions.
- We use **Pilots** with causal analysis and A/B testing for verifying correctness and effectiveness.

PART III
Evaluation

Evaluating our approach in production and simulation

Simulations

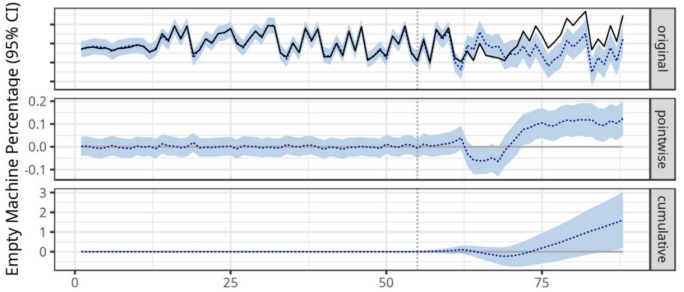
Using a validated, highly accurate simulator based on production code & traces over 2 weeks, we show that **NILAS and LAVA outperform LA by 1.1pp and 1.5pp empty hosts and up to 5.3pp and 5.7pp**, respectively (1% of our fleet is a large amount).



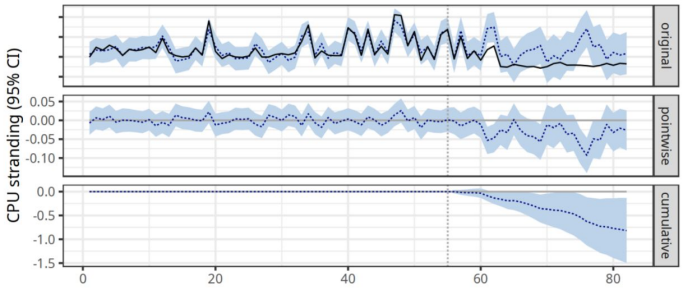
Production Deployment

NILAS has been **running in production for almost a year**. With **2-10pp empty machine improvement**, and up to **2-3% stranding reduction**.

Resource consumption from our approach is **negligible**.



(a) Empty hosts.

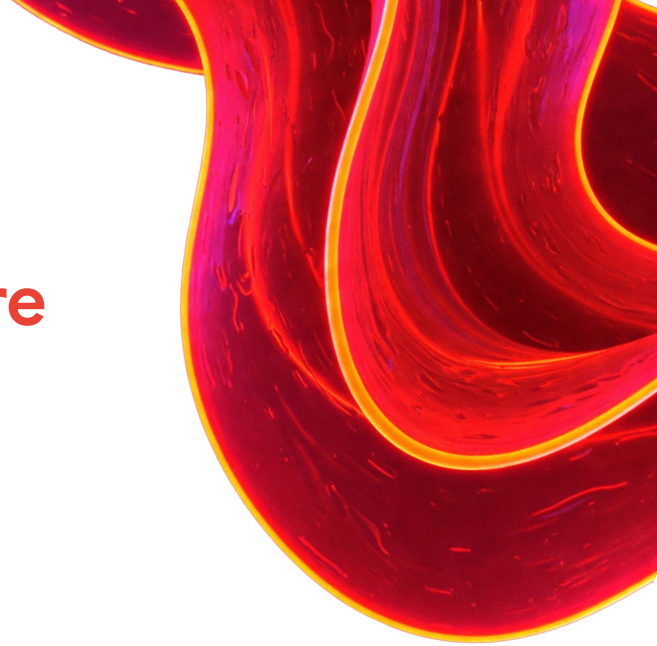


(b) CPU stranding.

Ablation studies and other details in the paper.

Conclusion

Summary



We **deployed our novel lifetime-aware scheduling techniques in Google's production data centers**

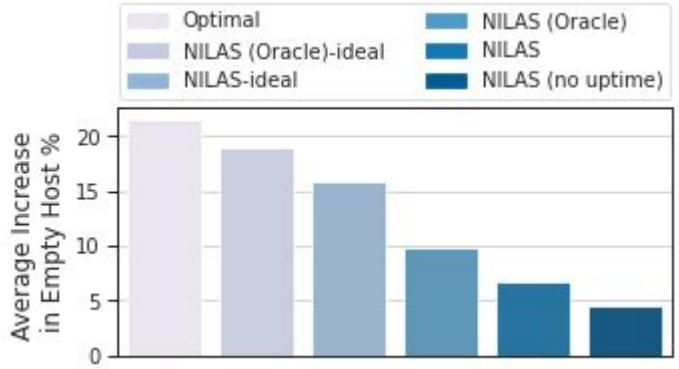
- We show the benefits of **correcting mispredictions**, using distribution-based models.
- We introduce **new algorithms** that group long and short-lived VMs together.
- We **significantly improve SOTA performance**.

Thank you! Any questions?

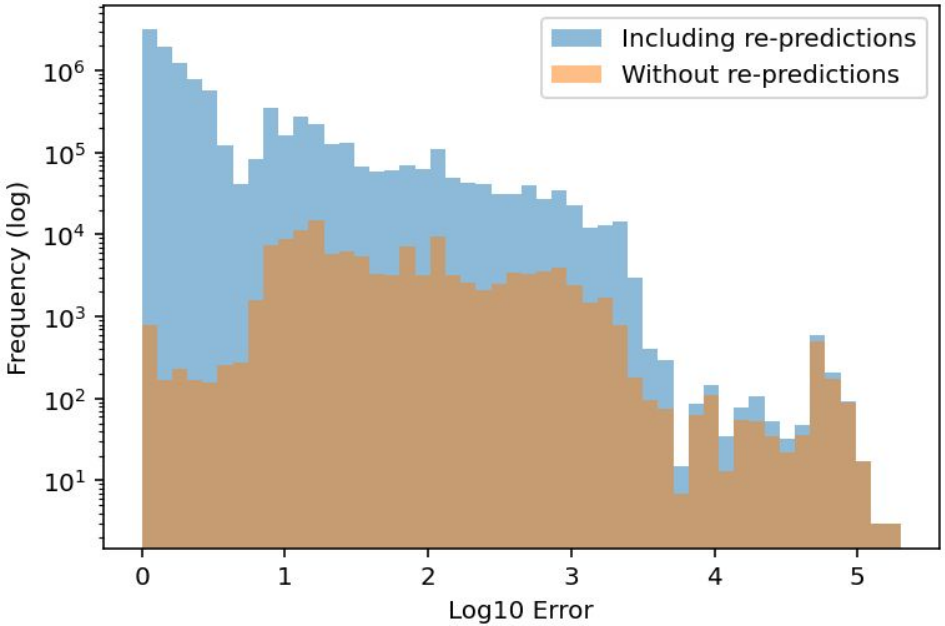
Jianheng Ling, Pratik Worah, Yawen Wang, Yunchuan Kong, Chunlei Wang, Clifford Stein, Diwakar Gupta, Jason Behmer, Logan A. Bush, Prakash Ramanan, Rajesh Kumar, Thomas Chestna, Yajing Liu, Ying Liu, Ye Zhao, Kathryn S. McKinley, Meeyoung Park, Martin Maas

Backup Slides

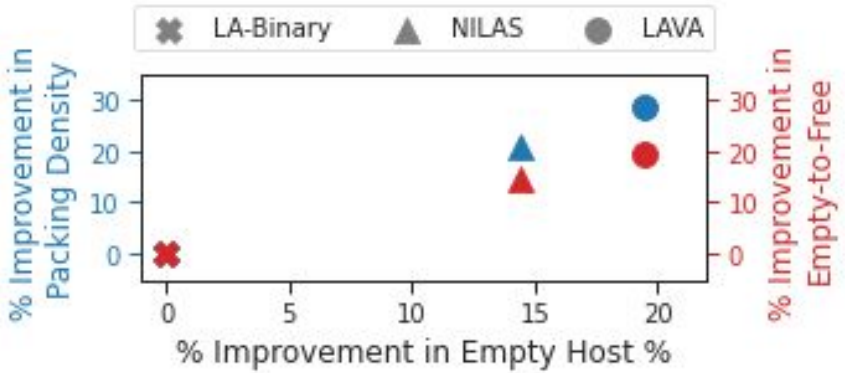
Optimal Upper Bound



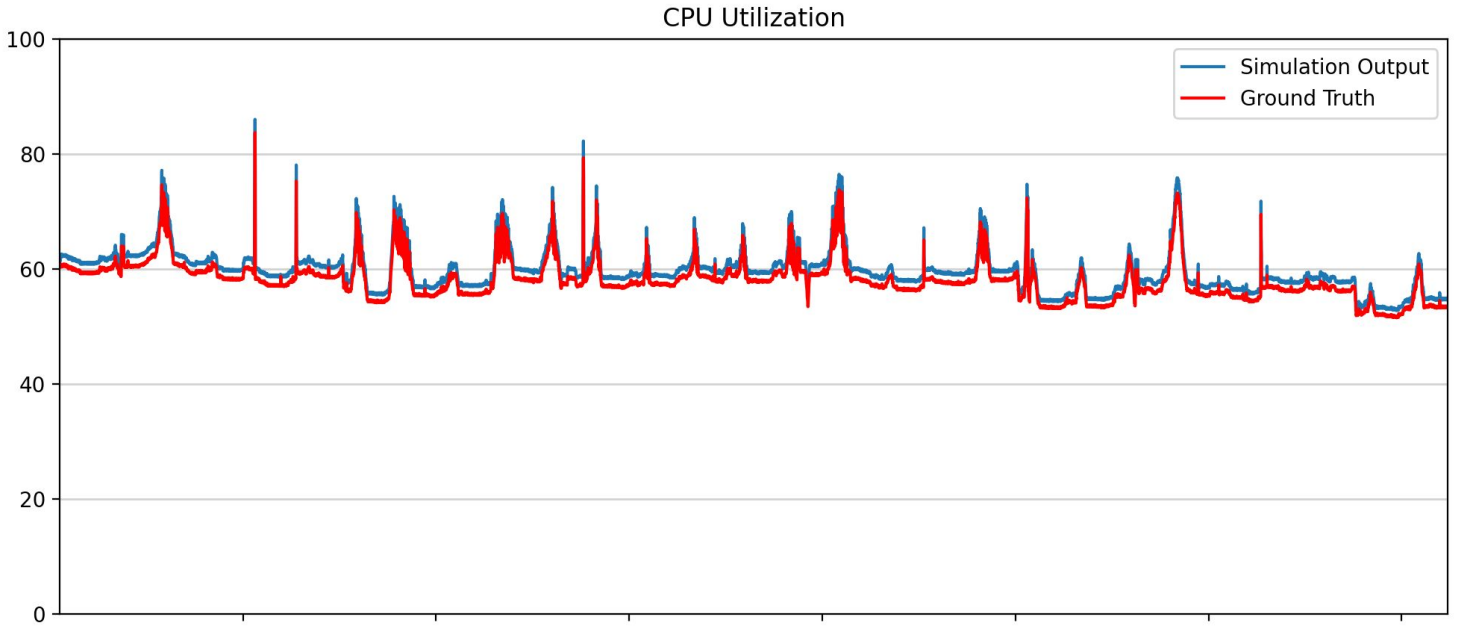
Model Accuracy & Reprediction



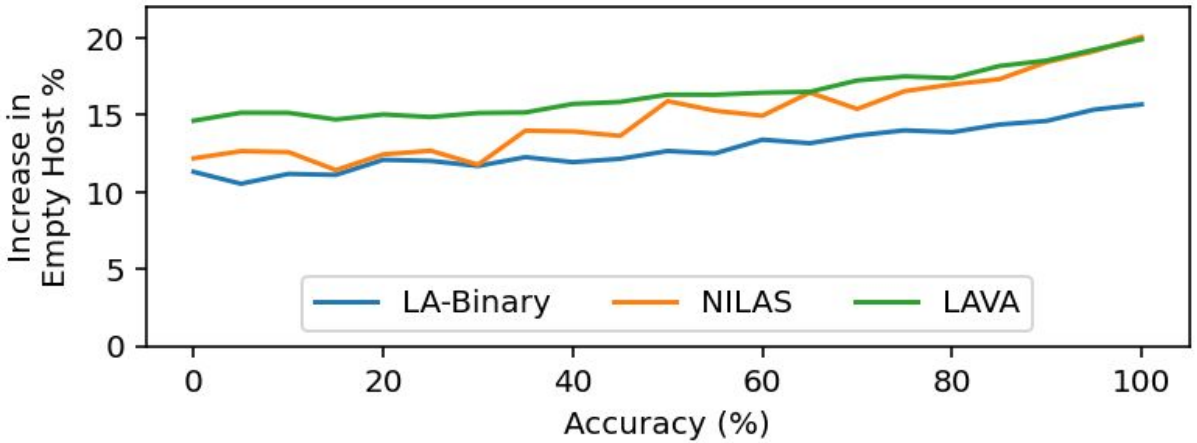
Bin Packing Metrics



Simulator Validation



Impact of Model Accuracy



Rollout Processes

Option 1: Roll out the model independently of the system.

- ✔ Allows updating model more often than the system.
- ✘ Might break verification assumptions.

Option 2: Roll out model with the system binary. **(Ours)**

- ✔ Can leverage existing rollout testing.
- ✘ Model might be stale when it reaches production.

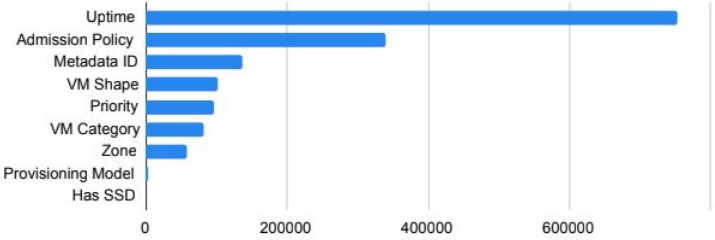
Model Choice & Integration

Prediction is **on the critical path** for VM scheduling.

- **Model**: Gradient boosted trees
  linked into the scheduler binary,
  not on other servers.
- Median latency is **9 us** (780x reduction vs. MLSys'23).

Explainability/Interpretability

- Model the problem in a way that it becomes **naturally interpretable** (e.g., predictor+algorithm recipe).
- Use explainable model libraries (e.g., decision trees).
- Use interpretability techniques (e.g., TCAV).



Feature importance for LAVA models, calculated by the gradient boosted tree library.

VM Live Migration

Reduce VM migrations for de-fragmentation & maintenance.

Key Idea

Migrate VMs in the order of their lifetime, starting from the longest-lived.

Details in the paper.