Coach: Exploiting Temporal Patterns for All-Resource Oversubscription in Cloud Platforms

Benjamin Reidys*
University of Illinois
Urbana-Champaign, USA

Pantea Zardoshti
Microsoft
Redmond, USA

Íñigo Goiri
Microsoft
Redmond, USA

Celine Irvene
Microsoft
Redmond, USA

Daniel S. Berger
Microsoft, Redmond, USA
University of Washington, Seattle, USA

Haoran Ma*
University of California-Los Angeles, USA

Kapil Arya
Microsoft
Redmond, USA

Eli Cortez
Microsoft
Redmond, USA

Taylor Stark
Microsoft
Redmond, USA

Eugene Bak
Microsoft
Redmond, USA

Mehmet Iyigun
Microsoft
Redmond, USA

Stanko Novaković*
Google
Mountain View, USA

Lisa Hsu*
Meta
Menlo Park, USA

Karel Trueba
Microsoft
Redmond, USA

Abhisek Pan
Microsoft
Redmond, USA

Chetan Bansal
Microsoft
Redmond, USA

Saravan Rajmohan
Microsoft
Redmond, USA

Jian Huang
University of Illinois
Urbana-Champaign, USA

Ricardo Bianchini
Microsoft
Redmond, USA

Abstract

Cloud platforms remain underutilized despite multiple proposals to improve their utilization (*e.g.*, disaggregation, harvesting, and oversubscription). Our characterization of the resource utilization of virtual machines (VMs) in Azure reveals that, while CPU is the main underutilized resource, we need to provide a solution to manage all resources holistically. We also observe that many VMs exhibit complementary temporal patterns, which can be leveraged to improve the oversubscription of underutilized resources.

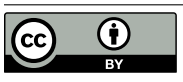
Based on these insights, we propose Coach: a system that exploits temporal patterns for all-resource oversubscription in cloud platforms. Coach uses long-term predictions and an efficient VM scheduling policy to exploit temporally complementary patterns. We introduce a new general-purpose VM type, called CoachVM, where we partition each resource allocation into a guaranteed and an oversubscribed portion. Coach monitors the oversubscribed resources to detect contention and mitigate any potential performance degradation. We focus on memory management, which is particularly challenging due to memory's sensitivity to contention and the overhead required to reassign it between CoachVMs. Our experiments show that Coach enables platforms to host up to ~26% more VMs with minimal performance degradation.

CCS Concepts: • **Computer systems organization → Cloud computing**.

Keywords: Cloud Computing, Memory Oversubscription, Temporal Patterns, Resource Management

ACM Reference Format:
Benjamin Reidys, Pantea Zardoshti, Íñigo Goiri, Celine Irvene, Daniel Berger, Haoran Ma, Kapil Arya, Eli Cortez, Taylor Stark, Eugene Bak, Mehmet Iyigun, Stanko Novaković, Lisa Hsu, Karel Trueba, Abhisek Pan, Chetan Bansal, Saravan Rajmohan, Jian Huang, and Ricardo Bianchini. 2025. Coach: Exploiting Temporal Patterns for All-Resource Oversubscription in Cloud Platforms. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS '25), March 30–April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3669940.3707226

*Benjamin Reidys and Haoran Ma interned at Microsoft. Stanko Novaković and Lisa Hsu were at Microsoft when they contributed to this work.



ASPLOS '25, March 30–April 3, 2025, Rotterdam, Netherlands

ACM ISBN 979-8-4007-0698-1/25/03.
https://doi.org/10.1145/3669940.3707226

1 Introduction

Motivation. Cloud platforms such as Microsoft Azure, Amazon Web Services (AWS), and Google Cloud Platform (GCP) offer compute resources (*e.g.*, CPU, memory, and network) as virtual machines (VMs). To meet the ever-increasing performance requirements of users, cloud providers are pressured to offer their services efficiently. However, achieving optimal efficiency remains challenging, and resource utilization in cloud platforms is often low [13, 19, 26, 61, 74, 95].

There are three common causes of low resource utilization in cloud platforms. First, platforms leave *unallocated* resources for future VM allocations to ensure an optimal experience for customer workloads (*e.g.*, rapid scale-out, high availability, and reliability) and for platform management (*e.g.*, datacenter tax [48, 82]). Prior work characterized unallocated resources and proposed solutions to minimize them, including Spot, Burstable, and Harvest VMs [3, 8, 9, 22, 32, 73, 85].

Second, *stranded* resources are unallocated but cannot be used to allocate new VMs because another resource on the server is fully allocated. Prior work focused on memory stranding [54] and proposed mitigating it using disaggregation [54, 105]. While disaggregation is promising for memory, it is unavailable for other resources (*e.g.*, CPU and network).

Third, *underutilized* resources are allocated but not always used by the workload on the VM. Prior work addressed this issue by using Harvest VMs, which can borrow underutilized resources from colocated VMs [3, 32, 73]. Unfortunately, users must modify workloads on Harvest VMs to account for evictions and dynamic resource allocations. Alternatively, using oversubscription can reduce underutilization by allocating fewer resources and multiplexing them between VMs on demand [25, 35, 40, 51, 58, 79, 84, 86, 93, 94, 97, 101, 110]. However, it has not been applied holistically to cloud VMs.

To understand the potential of oversubscription for cloud VMs, we study the resource utilization of over one million opaque VMs in Azure. We observe: (1) large and long-running VMs consume the most resources; (2) while CPU is usually the most underutilized resource [25, 46, 99], oversubscribing CPU can move the bottleneck for new VM allocations to other resources (*e.g.*, memory and network), making holistic management of all resources essential; (3) many VMs exhibit complementary temporal patterns (*e.g.*, some have peak utilization at noon while others peak at night); and (4) these patterns are predictable, Cloud platforms can leverage these patterns to help colocate oversubscribed VMs [23, 83].

Challenges. Cloud users run highly heterogeneous workloads on opaque VMs. Platforms must uphold strict service level objectives (SLOs) despite limited insights into diverse workload characteristics (*e.g.*, tail latency sensitivity) and restricted telemetry visibility (*e.g.*, CPU utilization). The virtualization abstraction introduces further challenges. The resource management granularity is typically coarser compared to that of processes or containers, which have been the focus of many prior works [29, 53, 100]. Oversubscription should also be compatible with optimizations and platform management techniques. Accordingly, we need to provide a transparent, safe, and resilient solution.

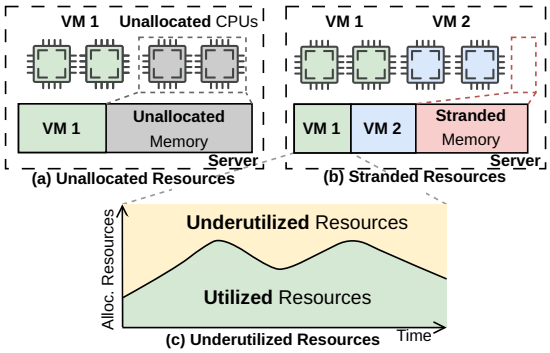


Figure 1. Examples of the causes of low resource utilization.

Our work. We propose Coach: a system to oversubscribe all VM resources. Coach leverages temporal patterns in resource utilization to predict which VMs will have higher resource demands at specific times of the day. Our scheduling policy identifies VMs with complementary resource utilization patterns for colocation. Our policy takes a holistic approach to considering all resources. Coach uses a new general-purpose VM type, called *CoachVM*, where each resource is partitioned into a guaranteed and an oversubscribed portion. The guaranteed portion is always allocated to the VM to maximize performance. In contrast, the oversubscribed portion is allocated on demand from an oversubscribed pool to maximize resource savings. Although we oversubscribe all resources, we focus on memory, one of the most sensitive and challenging, due to its non-fungibility. Coach manages server resources and uses reactive and proactive mitigations to minimize the potential performance degradation caused by contention. CoachVMs can be opt-in and discounted to compensate for the risk of performance degradation.

Results. We evaluate Coach using real workloads and production traces and quantify the trade-off between the resources saved and the risk of performance degradation. We demonstrate that exploiting temporal patterns enables hosting up to ~26% more VMs with minimal platform overhead and VM performance degradation.

Summary. We make the following main contributions:

- Characterize the resource utilization of VMs in Azure, focusing on utilization over time and opportunities for oversubscription.
- Propose Coach to oversubscribe all resources in cloud platforms and exploit temporal patterns at scale.
- Introduce a new oversubscribed VM type, called CoachVM, to ensure VM performance and maximize resource savings without requiring users to modify their workloads.
- Quantify the trade-off of oversubscription between its savings and its potential impact on workload performance.

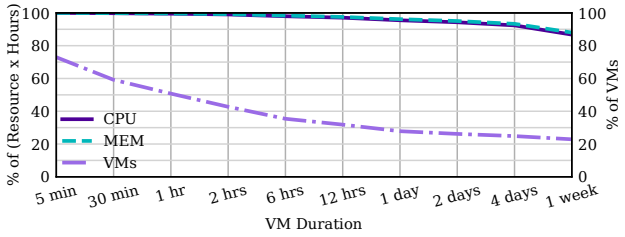


Figure 2. Percentage of resource×hours consumed by VMs lasting longer than VM duration (left) and the percentage of VMs lasting more than VM duration (right).

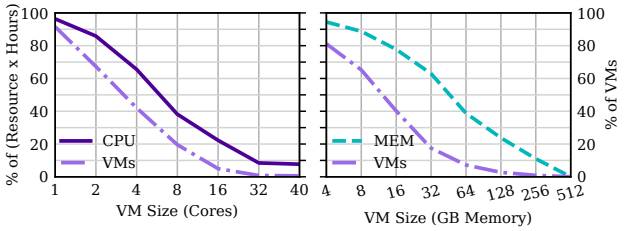


Figure 3. Resource×hours and number of VMs consumed by VMs larger than a size (cores and memory).

2 Opportunity for oversubscription

Low resource utilization is ubiquitous in cloud platforms [13, 19, 35, 74, 95, 101]. It can be caused by resources that are: (a) *unallocated*, if they are unsold or reserved by the platform [3, 25, 32, 110]; (b) *stranded*, if they are unallocated but cannot be allocated due to the lack of other resources in the server [54]; and (c) *underutilized*, if they are allocated to a VM, but not used [46, 69, 93, 99]. Figure 1 shows an example of each cause of low resource utilization. We study all three causes, focusing on their temporal patterns and the opportunities for oversubscription.

Methodology. We collected traces for two weeks in May 2024 of over one million opaque VMs from a subset of servers across ten popular clusters in seven Azure regions [25]. The traces cover thousands of servers from four hardware generations, including Intel and AMD processors. For each VM, we record the allocation and deallocation times, resource allocation, server on which it runs, and maximum resource utilization for CPU, memory, network, and storage, respectively. These utilization data are captured at 5-minute intervals (the default setting for long-term storage). Using 5-minute intervals establishes a lower bound for underutilization, as we use the maximum utilization in each interval.

2.1 Characterizing allocated resources

We study the characteristics of VMs that reserve the most *resource*×*hours* (*i.e.*, allocated resources weighted by time), as oversubscribing these VMs can yield the greatest benefit.

How long are resources allocated? Figure 2 shows that VMs lasting more than one day consume ~96% of allocated cores×hours, despite accounting for only 28% of the VMs. This is also the case for memory (96% of GB×hours), as most VMs have a similar ratio of memory to cores. Network and storage show the same patterns. These findings are consistent with those reported for CPU in prior work [25].

This observation shows an important distinction between the number of VMs and the resources they consume over time. For example, sixty 32GB VMs lasting one minute and one 32GB VM lasting one hour both consume 32GB×hours. Given this, we focus on VMs that last longer than one day.

What is the size of the VMs? Figure 3 shows that larger VMs use more resources. The median VM in our study has 4 cores[1] and less than 16GB of memory, which is larger than indicated by prior work [25], where most allocated VMs had less than 2 cores and 4GB. We observe the same distinction between the number of VMs and resources consumed as above. For example, VMs with 32GB or more consume over 60% of GB×hours, despite representing only ~20% of VMs. Therefore, a solution targeting low resource utilization should account for both longer-running and larger VMs.

2.2 Characterizing stranded resources

Cloud platforms can improve their packing of VMs into servers by aligning the resource ratios (*e.g.*, GB/core) of VMs with the server hardware. However, with the explosion of VM configurations [21, 24, 87, 106] (*e.g.*, 5 resource ratios, 9 sizes, 6 generations, and 4 specialized types in Azure [10]), platforms may need to allocate VMs to servers with misaligned ratios. For example, a server with hardware configured for general-purpose VMs (*e.g.*, 4GB/core) may receive allocations for memory-optimized VMs (*e.g.*, 16GB/core). Figure 1b shows an example in which the cores are fully allocated, leaving memory stranded. Prior work [54] focused on stranded memory, while we focus on the stranding of all types of resources and the implications for oversubscription.

How much are resources stranded? To study stranding, we place hypothetical VMs of the most typical VM configuration (*i.e.*, 4GB/core) [4] on each server in our trace until one resource is exhausted and no further VMs can be placed. The remaining unallocated resources are considered stranded. We repeat this calculation for each timestamp in the trace.

No Oversub in Figure 4 shows that CPU is the least stranded resource, with only 8% stranding on average. Memory, network, and SSD have 18%, 29%, and 54% stranding, respectively. Our analysis indicates that stranding may be more severe than previously reported [54] and includes all types of resources.

Is stranding consistent across clusters? Figure 5 shows the percentage of time each resource is the bottleneck for new VM allocations on a server (*i.e.*, the cause of stranding) across all clusters. No Oversub shows that the most common

[1]We normalize hyperthreaded and non-hyperthreaded vCPUs to "cores".

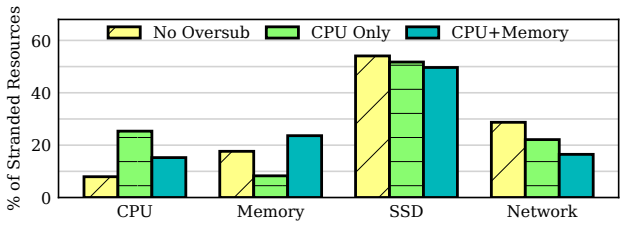


Figure 4. Average stranding for different resource types with varying levels of hypothetical oversubscription.

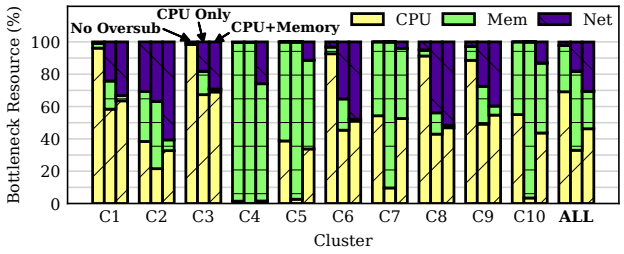


Figure 5. Percentage of time each resource is the bottleneck.

bottleneck is CPU, then memory, and finally, network. This is inversely proportional to the amount of stranding shown in Figure 4, as the bottleneck resource is fully allocated (*i.e.* not stranded) on the server. We omit SSD since it causes stranding less than 1% of the time in all configurations.

We observe significant variation between clusters. C1 is almost exclusively bottlenecked by CPU, C4 by memory, and C2 is divided between CPU, memory, and network. This is because different clusters have different hardware configurations. For example, servers in C4 have less memory relative to cores/network than the other clusters. Therefore, we need to account for the diverse configurations across servers.

What if we oversubscribe? Oversubscribing the bottleneck resource could unlock stranded resources for allocation. Figure 4 shows the hypothetical impact of oversubscribing CPU (and memory) on stranding. We compute this by placing hypothetical VMs, as before, except that we also use underutilized CPU (and memory) resources to allocate these VMs. For CPU Only, stranding increases for CPU to 25% and decreases for memory, SSD, and network to 8%, 52%, and 22%, respectively. CPU stranding increases because some previously underutilized cores (*i.e.*, allocated but unused) are now unallocated but bottlenecked by another resource. We confirm this in Figure 5, where the bottleneck shifts from CPU (69% to 33%) to memory (29% to 49%) and network (2% to 18%). We observe a similar trend for CPU+Mem: stranding for CPU, storage, and network decreases to 15%, 50%, and 16%, respectively, while it increases to 24% for memory. Meanwhile, the bottleneck shifts from memory (49% to 23%). In all cases, the utilization of each resource improves. This motivates the need to oversubscribe resources holistically.

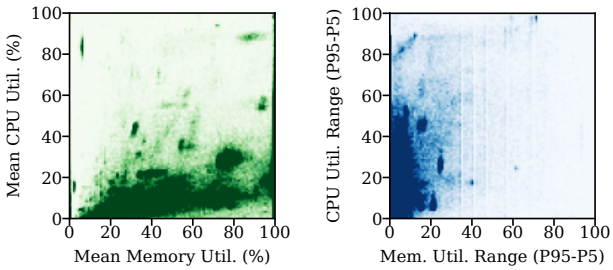


Figure 6. Correlation between CPU and memory for average utilization and utilization range across all VMs.

2.3 Characterizing underutilized resources

Users may buy VMs with more resources than necessary because of variable load, performance sensitivity [63, 99], or a mismatch with available options [2, 106]. For example, Figure 1c shows a VM with an additional buffer to absorb workload bursts. Alternatively, a user may need 6 cores and 12GB of memory, but the closest VM has 8 cores and 16GB.

Prior work characterized the underutilization of cloud resources, including CPU [3, 25, 46, 69, 93, 99, 110], memory [32, 88, 94, 96], SSD [73, 97, 110], and power [51, 92]. We study the underutilization, correlations across different types of resources, temporal patterns, and implications for oversubscription. We focus on VMs lasting over one day.

What is the average utilization? Figure 6 shows the correlation between CPU and memory utilization. The left half indicates that most VMs have an average CPU utilization below 50%, which is consistent with prior works [25, 35, 58, 69, 84, 99], while there is greater diversity in the average memory utilization. VMs with high CPU utilization also tend to have higher memory utilization. The network and storage behavior resembles that of CPU.

Does utilization vary? We define the utilization range as the difference between utilizations (*e.g.*, P95-P5) over the lifetime of a VM. The right half of Figure 6 shows that the range for CPU often reaches 60%, while the memory is within 30%, indicating that CPU utilization fluctuates more than that of memory. In addition, 50% of VMs have a memory range less than 10%, and only 10% of VMs have a range exceeding 50%. While VMs may have unique memory utilization, it typically fluctuates within narrow bounds. The utilization patterns of network and storage resemble those of memory.

Is there a relation between average and range? We also measure the correlation between average utilization and range. VMs with higher average CPU utilization tend to have a higher range. Conversely, memory has a shorter range (less than 30%) across all average utilizations.

Are there busier times in a day? Prior work classified the utilization patterns of bare-metal servers as periodic, constant, or unpredictable [110]. Our work identifies peak times during the day (*e.g.*, higher utilization every day at

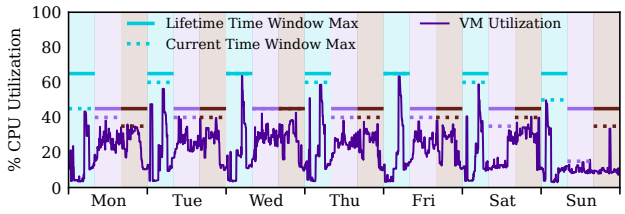


Figure 7. CPU utilization for a VM over a week split into three daily time windows: 0-8hr, 8-16hr, and 16-24hr.

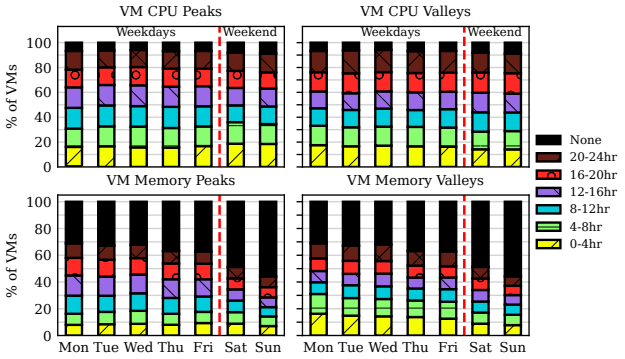


Figure 8. VMs with a peak/valley in each of the six 4-hour time windows for one cluster.

noon). Figure 7 shows the CPU utilization of a VM with consistent daily peaks over a week. We divide each day into three 8-hour windows: 0-8hr, 8-16hr, and 16-24hr. For each time window, we show its peak utilization (current time window max) and its peak across the seven days (lifetime time window max), rounded to 5% buckets (*e.g.*, 17.3⟶20.0%). In the first window (0-8hr), the utilization is primarily under 10% but has spikes up to 65%. The remainder of the day (8-24hr) uses ~40% CPU. The current max is typically similar across days and close to the lifetime max.

We first characterize the peak times for long-running VMs. A VM has a peak (and valley) in a given day if the difference between the maximum utilization in different time windows that day is at least 5%. Any time window with a maximum utilization equal to the maximum (or minimum) across all time windows that day is counted as a peak (or valley). Accordingly, a VM can have multiple peaks and valleys per day in different time windows. Figure 8 shows the percentage of VMs with peaks (and valleys) in each of six 4-hour time windows (*i.e.*, 0-4hr, 4-8hr,…), and those without peaks (None). For each day, we normalize the total VMs with a peak (or valley) in each time window against the total VMs with a peak (or valley) that day. Both CPU peaks and valleys are evenly distributed across the six time windows. Less than 10% of VMs have no CPU peaks/valleys (*i.e.*, their utilization is within a 5% bucket). Nearly 70% of VMs have memory peaks/valleys evenly distributed over time. This indicates that we could exploit these peaks and valleys by placing VMs

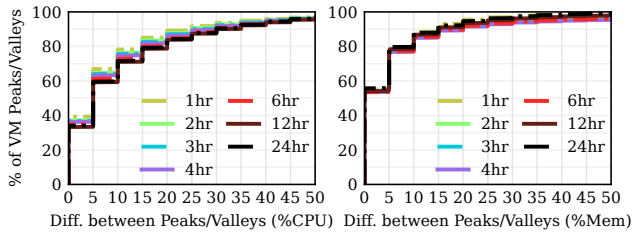


Figure 9. Difference in peak/valley utilization in consecutive days for different time window lengths.

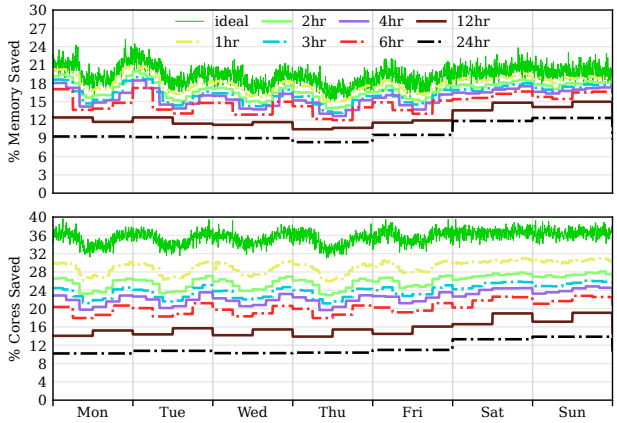


Figure 10. Potential savings for memory and CPU using time windows of multiple lengths for one cluster.

that peak in specific time windows alongside VMs that have a valley at the same time.

Is the behavior consistent over time? Figure 9 shows the variation in peak and valley utilization in consecutive days. With 4×6-hour windows, 80% of VMs have a utilization difference of at most 20% for CPU and at most 5% for memory. Overall, most VMs have consistent peaks and valleys, indicating that such patterns could be exploited over time.

Are patterns complementary? Figure 10 shows the percentage of allocated resources we can save in a representative cluster by packing VMs using their maximum utilization in each time window. We compute the resources saved as the difference between oversubscription using these patterns (the maximum utilization in each time window) and overlooking them (the VM's lifetime max). For example, if a VM has a max utilization of 75%, but its time windows have 30%, 75%, and 55% utilization, we could save 45%, 0%, and 20%, respectively. We show the average savings across all VMs.

With a single 24-hour window, we save ~8% of memory and ~8% of CPU. Using 4×6hr windows, we save ~15% of memory and ~20% of CPU. Multiplexing 5-min windows (ideal) saves ~18% memory and ~34% CPU.

Figure 11 summarizes the potential resource savings across all 10 clusters as a violin plot. We observe that colocating VMs with complementary patterns can consistently result in

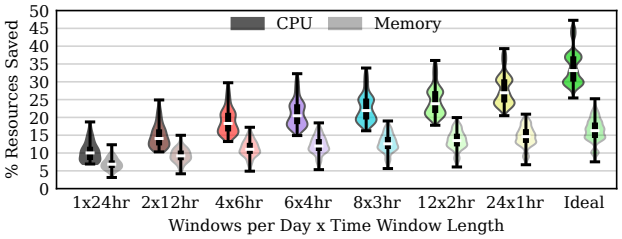
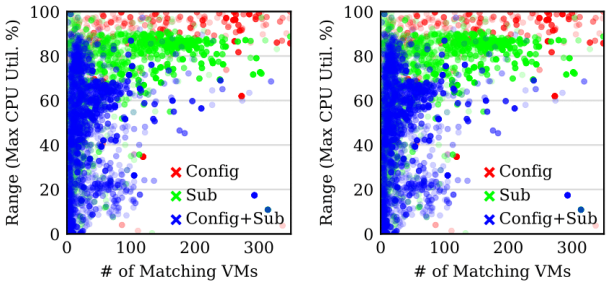


Figure 11. Summary of potential savings using different time windows across all clusters. The distribution is shown as a colored violin, median by white lines, interquartile range (P75/P25) by black rectangles, and max/min by black lines. Darker/lighter violins are for CPU/memory, respectively.



(a) CPU. **(b)** Memory.

Figure 12. Per VM correlation between number of previous VMs of the same group and their utilization range. We use 3 groups: subscription, VM configuration, and a combination.

significant savings across clusters. The savings increase with the number of windows but start plateauing with 6×4-hour. We can typically save more CPU than memory.

Are new VMs similar to old VMs? We analyze whether existing VMs can be grouped to use their aggregated resource utilization patterns to predict the patterns of future VMs. For each VM in the second week of our trace, we analyze the utilization of a group of similar VMs in the first week based on three groupings of similarity: VMs from the same (1) customer subscription [25], (2) VM configuration, and (3) subscription and VM configuration. Other features (*e.g.*, VM name, guest OS version, or creation time) were less relevant. For each VM, Figure 12 shows the number of matching VMs from the same group (*e.g.*, subscription) and the range of their maximum resource utilization. For example, a VM with 10 prior VMs from the same subscription whose peak CPU utilizations were within a range of 10% is plotted as $(10, 10)$. Ideally, we want many matching VMs (*e.g.*, >50) with low ranges (*e.g.*, <10%).

Grouping by VM configuration, the median VM has many previous VMs (over 2,000), but their utilization range is high (nearly 100 for memory). Grouping by subscription, the median VM has fewer previous VMs (over 120), and their utilization range is smaller (under 70 for memory). Grouping by both, the median VM has the fewest previous VMs (40) with

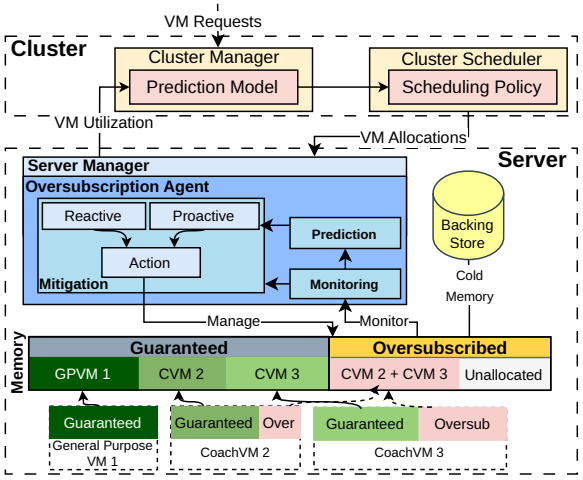


Figure 13. Design overview of Coach.

the smallest range (only 31 for memory). To determine predictability, we compare the maximum utilization of each VM with the average peak of its prior VMs. With subscription and VM configuration, memory shows better predictability (over 70% of VMs within 10% of the average peak utilization) compared to CPU (70% of VMs within 20% of the average peak). Overall, most VMs have sufficient historical patterns that can aid in predicting future resource utilization.

3 Temporal pattern-based oversubscription

Based on Section 2, there is a significant opportunity to leverage complementary temporal utilization patterns for all resources due to their predictability. Traditional resource oversubscription solutions [14, 46, 93, 96, 97] aimed to reduce underutilization in cloud platforms but did not exploit these patterns. We propose Coach: a system to oversubscribe virtualized cloud platforms that leverages complementary temporal utilization patterns for all resources at scale.

3.1 Coach overview

We present the design overview of Coach in Figure 13, including the common workflow to create and operate oversubscribed VMs. It comprises a logically centralized cluster management layer and a local component for each server.

Cluster management. When creating an oversubscribed VM [23, 83], the *cluster manager* converts the request (*e.g.*, 4 cores and 16GB of memory) into resource requirements and oversubscription rates. It uses a *prediction model* to decide the oversubscription rates for each resource in each time window (*e.g.*, oversubscribe memory by 30% during the day and by 20% at night). The *cluster manager* sends these rates to the *cluster scheduler*, which uses them to assign the VM to a server and sends the request to the selected *server manager*.

Server management. The local *oversubscription agent* manages the resources on each server and adjusts the guaranteed

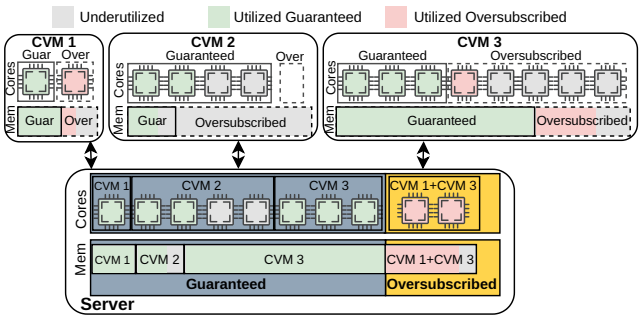


Figure 14. Three CoachVMs in a server showing the guaranteed and oversubscribed CPU and memory.

and oversubscribed resources whenever a VM is allocated or deallocated. The agent consists of three components: (1) *monitoring* to collect utilization data; (2) *prediction*, to predict future utilization; and (3) *mitigation* to detect contention and take local (*e.g.*, reassign resources) and global (*e.g.*, migrate VMs) remediation actions. The *oversubscription agent* periodically sends the utilization data to the *cluster manager* to improve prediction accuracy.

Customer adoption. Ideally, customers can seamlessly transfer as many workloads as possible to Coach. To achieve this, we set the following additional design goals:

- **G1: Minimize customer burden.** Coach should be transparent to the workloads on the VM to allow customers to deploy unmodified workloads.
- **G2: Minimize workload interference.** Coach should minimize any negative impact on VM performance to maintain existing SLOs as much as possible.

To achieve G1, Coach addresses the virtualization-specific challenges through CoachVMs (Section 3.2) without requiring workload modifications from users. For G2, Coach oversubscribes conservatively despite opportunities to save additional resources, thereby minimizing the chance of contention (Section 3.3). By default, Coach eschews techniques that rely on awareness of the workloads running in the VM. However, such techniques can be integrated if desired.

3.2 CoachVMs: Oversubscription for cloud VMs

Unlike containers, which are typically short-lived and share the host kernel, VMs face additional challenges for oversubscription. First, VMs have more limited telemetry visibility (*i.e.*, they are opaque to the platform) and coarser resource granularity, complicating resource management. Second, oversubscribed VMs must remain compatible with existing optimizations and platform management techniques (*e.g.*, device assignment, live migration, and host updates).

To support oversubscription, we introduce a new general-purpose VM type called *CoachVM (CVM)*. It has a guaranteed portion of each resource for reliable performance and receives the remaining allocation on-demand from an oversubscribed pool for savings. All resources are managed transparently to the VM, preserving its general-purpose nature. This approach enables customers to run any guest OS and unmodified workloads without burdensome application changes, eliminating barriers to widespread adoption (G1).

Table 1. Common fungible and non-fungible resources and the mechanism used to share them across VMs.

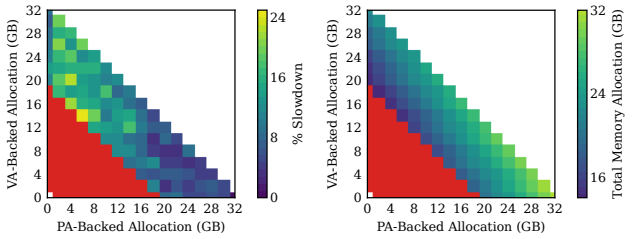
Resource	Fungible	Mechanism
CPU	✓	CPU groups
Memory space	✗	PA/VA portions, VA-backing
Memory bandwidth	✓	Shares, reservations, caps
Network bandwidth	✓	Shares, reservations, caps
Accelerated network	✗	SR-IOV
Storage bandwidth	✓	Shares, reservations, caps
Local storage space	✗	Disk partitions, DDA, SR-IOV
Remote storage space	✓	Cache size and network bandwidth
GPU	✗	DDA, SR-IOV
Power	✓	Frequency and power caps

Guaranteed and oversubscribed. CoachVM resources are divided into *guaranteed* (always allocated to the CVM to ensure performance) and *oversubscribed* (shared across CVMs to save resources). Figure 14 shows how Coach may allocate resources to three CVMs: CVM1 with 2 cores and 8GB of memory, CVM2 with 4 and 16GB, and CVM3 with 8 and 32GB. Coach guarantees 8 cores and 26GB of memory (CVM1: 1 and 4GB, CVM2: 4 and 4GB, and CVM3: 3 and 18GB). The remaining 6 cores and 30GB of memory are oversubscribed and backed by only 2 cores and 16GB. In this way, Coach can fit VMs with 14 cores and 56GB of memory into a server with 10 cores and 36GB (~30% oversubscription rate).

Fungible and non-fungible. Certain resources are harder to share across CVMs when oversubscribed. We consider resources that can be quickly reassigned between VMs as fungible [75]. For example, we can easily multiplex CPU and network bandwidth across multiple VMs. In contrast, virtual memory pages are assigned to specific physical pages and need to be paged out before the physical page can be reassigned. We consider these resources as non-fungible. Table 1 summarizes the fungibility of common resources.

For fungible resources, we assign multiple VMs to the same resource and let the hypervisor quickly reassign them. However, we must assign non-fungible resources carefully. In all cases, we must monitor and mitigate potential contention.

Table 1 lists the mechanisms we use to manage common resources. For example, we use CPU groups to assign a subset of cores statically and oversubscribe the rest. For the rest of the paper, we focus on *memory space*, as it is non-fungible and one of the most challenging to oversubscribe. The general techniques discussed can be applied to other resources. We omit a thorough discussion of CPU as it has been extensively covered in prior work [25, 99].



(a) Performance slowdown. **(b)** Memory allocation.

Figure 15. Trade-off for PA/VA-backing with a 32GB CoachVM that runs a workload with a working set of 18GB. In (b), we back 70% of VA memory with physical memory.

Memory oversubscription. The memory space of VMs can be physically addressed (PA) or virtually addressed (VA). The hypervisor allocates PA memory at VM creation time and statically maps it to the guest physical addresses (GPA). It uses huge pages (1GB) to reduce TLB overheads. Accessing PA pages provides high performance, but their static allocation limits flexibility, as they cannot be easily reassigned. Therefore, we use PA memory for the guaranteed portion.

The hypervisor manages VA memory using 1GB pages, which can be allocated/mapped to specific VMs at a smaller granularity on demand. It can be dynamically backed by a smaller amount of physical memory, using a backing store (*i.e.*, disk) if it is insufficient. Unused VA pages can be *trimmed*/unmapped, and *paged in* from the backing store when accessed. Since VA pages may be unmapped, accessing VA memory may result in page faults, which is slower than accessing PA memory. Despite the risk of reduced performance, we use VA memory for the oversubscribed portion because it can be (de)allocated on demand. Coach maps the oversubscribed (*i.e.*, VA) portion to the VM's GPA as a NUMA node with no cores (zNUMA) [54], which funnels accesses to the guaranteed (*i.e.*, PA) portion without guest changes.

Depending on the *working set* of the VM (*i.e.*, active pages in the GPA), we can adjust the PA/VA ratio. For example, if the working set is typically below 16GB for a 32GB VM, Coach can allocate 16GB of PA memory and back the 16GB of VA memory with 8GB (25% oversubscription). This maximizes performance with the PA portion and resource savings with the VA portion. However, selecting the optimal PA/VA ratio is crucial for balancing both goals.

Impact of PA/VA ratio on performance. Figure 15a shows the performance impact on an unmodified memory-sensitive application with a working set of 18GB, running on a 32GB VM while varying the size of the PA and VA portions.

The point with 32GB PA and 0GB VA (bottom right) is the baseline performance of a fully PA-backed VM (*i.e.*, 0% performance slowdown). The white area represents invalid configurations (*i.e.*, with more memory than the 32GB VM size or with no memory). The red area represents configurations where the VM suffers unacceptable performance degradation due to continuous paging to disk. The bottom right areas show minimal performance degradation. This is because we can leverage the NUMA policies of unmodified guest OSes to transparently deprioritize the use of the VA portion with zNUMA. When we allocate less than 16GB of PA, we observe greater slowdowns.

Impact of PA/VA ratio on memory savings. Figure 15b shows the total amount of allocated memory for the same 32GB VM with various sizes for the PA and VA portions. We back only 70% of the VA portion (based on the expected temporal-pattern multiplexing) and the entire PA portion with physical memory. The fully PA-backed VM saves no memory, while a VM with 16GB PA and 16GB VA (backed by 12GB) saves 4GB. By combining the performance impact in Figure 15a and the savings in Figure 15b, we can quantify the trade-offs when deciding the PA/VA ratio.

Direct access to oversubscribed memory. To expose resources such as GPUs, NVMe SSDs, and accelerated networking to VMs with high performance, cloud platforms use techniques like direct device assignment (DDA) [64] (or device pass-through [52]) and single-root input/output virtualization (SR-IOV) [65, 98], which rely on direct memory access (DMA). In VMs with oversubscribed memory, some parts of the GPA space may not be mapped to actual physical memory. When DMA requests attempt to access these unmapped memory regions, it can lead to I/O failures. We discuss two solutions to address this issue.

Hardware support. Devices with Address Translation Services/Page Request Interface (ATS/PRI) [44] can handle these intercepts. ATS enables the device to use the second-level address translation table (SLAT) to translate GPAs and store translations locally. PRI enables the device to handle SLAT failures (due to invalid pages) by requesting the host to fault the address as the CPU would. Therefore, these devices can access memory that could result in invalid translations.

Guest enlightenments. Most devices do not yet support ATS/PRI. In such cases, we need to ensure that any memory that may be accessed by a device is in the SLAT (*i.e.*, has a valid mapping). To address this, we introduce guest enlightenments (*i.e.*, paravirtualization) that explicitly exchange memory ranges for I/O with the guest OS at boot time. The host and guest then prevent moving or invalidating these ranges to avoid invalid translations. CVMs are also compatible with other existing guest enlightenments [5].

Compatibility with platform management. Large-scale cloud platforms perform management operations to optimize VM packing and handle software/hardware maintenance. To seamlessly deploy Coach at scale, CVMs–particularly their VA memory–must be compatible with these operations.

Live migration. Cloud platforms use live migration to reduce fragmentation and perform maintenance [76, 77]. Live migration already supports migrating PA-backed memory without modification. For the VA-backed portion, we must first page in any trimmed cold memory. However, since copying cold memory happens during the pre-copy phase of migration [76, 77], this does not extend VM downtime.

Host updates. Cloud platforms may need to update the host OS and reboot servers when there are critical security vulnerabilities [1, 77]. To perform these host OS updates, one could live migrate the VMs out of the server and reboot. However, the overhead of doing this for every server in a data center is excessive. To avoid this, platforms use VM preserving host updates, which temporarily pause the VMs and restore them after the host update [77]. This host update preserves the VM memory through reboots. This is simple for the PA-backed portion as it is directly mapped to the GPA. In contrast, the VA portion relies on host OS memory management and involves complex data structures that are harder to persist across upgrades. We incur this necessary complexity to persist these complex structures with negligible overhead.

3.3 Utilization time windows

When placing a VM, platforms use a single allocation for the entire VM lifetime [37]. To exploit complementary patterns, Coach divides the VM utilization into time windows. Figure 7 shows an example using three 8-hour windows per day.

Predicting utilization. To decide the oversubscription rate for CoachVMs, we use a *prediction model* that predicts the percentile (*e.g.*, P95) utilization for each resource in each time window. The model uses VM- and customer-specific features. VM-specific features include the VM configuration, the weekday of allocation, and offering (PaaS vs. IaaS). Intuitively, these inputs capture utilization behavior observed across customers. For example, utilization tends to be higher in VMs allocated on weekdays and IaaS VMs. Customer-specific inputs capture different utilization behavior between customers. We use the subscription type (*e.g.*, internal production vs. test) and the history of resource utilization of previous VMs in that customer subscription. The existing platform telemetry already collects all these inputs in the background, requiring no user input.

We use a random forest regressor [81] that predicts utilization in 5% buckets (*e.g.*, at most 65%). Random forest is well-suited for predicting VM utilization due to its effectiveness with categorical variables, as shown in prior work [3, 54]. Among similar predictors (*e.g.*, XGBoost [104] and LightGBM [57]), we choose random forest because it tends to be less sensitive to overfitting. This can improve robustness and reduce the likelihood of underpredictions (Section 4.2), minimizing the risk of worst-case contention (G2).

To collect the training data, we aggregate the utilization data for each VM. If there is insufficient data to predict a VM, we conservatively do not oversubscribe it. We generate these predictions in the background to minimize overheads on the critical path of VM allocation.

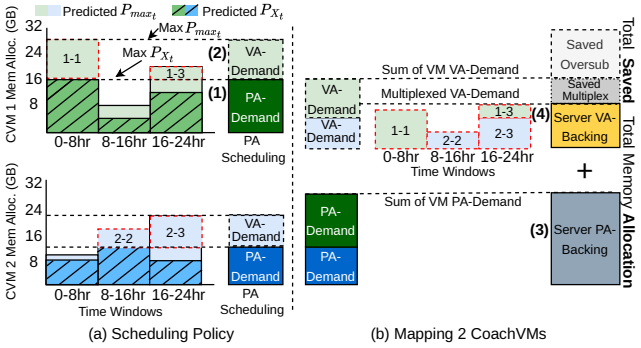


Figure 16. Memory allocation and mapping between time windows and PA/VA-backing. Two 32GB VM with 3 time windows map to 44GB (28GB PA and 16GB VA-backed). Numbers in parentheses refer to the formula that computes them.

VM scheduling policy. Once Coach decides the oversubscription ratio for the CoachVM, it picks which server to host the VM. Traditional VM schedulers solve this bin-packing problem using heuristics that account for the availability of each resource [17, 37, 67, 80]. They use a vector with the VM requirements (*e.g.*, {8 cores, 32GB memory, 300GB SSD, 10Gbps}) and check if it fits within the available server resources (*e.g.*, {16 cores, 24GB memory, 500GB SSD, 20Gbps}). For example, they cannot place this VM on this server due to insufficient memory.

Scheduling time-windows. Instead, Coach considers the predicted utilization of each resource for each time window. For fungible resources like CPU, we can simply use a vector of the predicted utilization for each time window. For an 8 core CoachVM with three time windows, we may predict 2 cores for 0-8hr, 6 for 8-16hr, and 4 for 16-24hr and try to place it in a server with 4 cores available for 0-8hr, 6 for 8-16hr, and 8 for 16-24hr. Represented as vectors, we have {2, 6, 4} ≤ {4, 6, 8}, so the CoachVM fits.

Non-fungible resources. However, this approach does not consider if the resources can be reassigned easily (fungibility). Specifically, the PA portion of the memory space cannot be easily adjusted at runtime (*i.e.*, non-fungible). To maximize performance, we would allocate PA memory for the VM's peak utilization across all time windows and not multiplex VMs with complementary patterns. Conversely, using VA for the entire allocation to fully exploit temporal patterns may reduce performance. Our approach is to balance these considerations. We maximize performance by allocating enough PA to satisfy the VM's working set a majority (*e.g.*, 95%) of the time. We maximize multiplexing by allocating the remainder (*i.e.*, up to the peak utilization) with VA.

Figure 16a shows the VM scheduling for memory for two 32GB CoachVMs with three time windows in a 48GB server. For each window, we predict the maximum memory utilization (the total PA+VA-backed working set) and a percentile (*e.g.*, P95) of the utilization for the guaranteed (PA-backed) portion. The difference between the maximum and percentile represents the potential amount of VA-backed memory.

Initially, the scheduler ensures that the predicted maximum for each time window does not exceed the server's PA+VA-backed capacity: $\{28, 8, 22\}+\{10, 18, 24\} \leq \{48, 48, 48\}$. Next, since the PA portion is static across windows, the scheduler verifies the server has sufficient memory for the PA-backed portions: 16+12<48. Combining these checks into vectors, we have: $\{28, 8, 22, 16\}+\{10, 18, 24, 12\} \leq \{48, 48, 48, 48\}$. With this approach, the scheduler considers the number of windows plus one (for the max) for each resource, instead of just one, which incurs negligible overhead.

Mapping time windows to CVMs. Once Coach assigns the CoachVM to a server, we need to map the predicted utilizations to guaranteed and oversubscribed resources. Figure 16b shows how we map the memory space for the CVMs in Figure 16a. Coach allocates the maximum percentile prediction across all time windows for the guaranteed (*i.e.*, PA) portion. For the oversubscribed (*i.e.*, VA) portion, one simple approach would be to allocate the sum of each VM's VA-demand. However, this overlooks the fungibility of VA-backed memory and allocates excess memory. Instead, we multiplex the VA-demand of each VM in each time window to further save memory by allocating the maximum multiplexed VA-demand. When the prediction component identifies potential changes in the resource requirements of a VM (*e.g.*, across time windows), it notifies the mitigation component to reassign resources between CoachVMs. Note that when assigning resources, we consider locality and relations between resources (*e.g.*, NUMA). The server manager stores the VA-demand in each time window for each VM. It recomputes the multiplexed demand when it (de)allocates VMs and adjusts the oversubscribed portion accordingly.

Formulation. $P_{\max_t}$ and P_{X_t} are the maximum and PX (*e.g.*, P95) percentile for time window t, respectively:

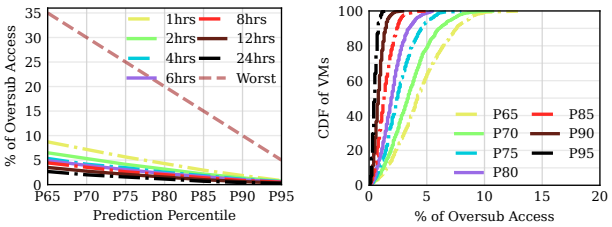
$$\forall i \in VM, \quad \text{PA\_demand VM}_i = \max_{t \in TW} (P_{X_t}) \tag{1}$$

$$\forall t \in TW, \quad \text{VA\_demand VM}_{i,t} = \max(0,\ P_{\max_t} - \text{PA\_demand VM}_i) \tag{2}$$

$$\text{Guaranteed memory} = \sum_{i \in \text{VM}} \text{PA\_demand VM}_i \tag{3}$$

$$\text{Oversubscribed memory} = \max_{t \in TW} \left(\sum_{i \in \text{VM}} \text{VA\_demand VM}_{i,t} \right) \tag{4}$$

Choosing a prediction percentile. We can navigate the trade-off between resource savings and potential performance impact by adjusting the prediction percentile. For example, using P95 might risk 5% of the accesses with lower performance but save 30% of memory.



(a) Oversubscribed accesses varying prediction percentile and time window.

(b) CDF of oversubscribed access percentage for 4hrs varying the prediction percentile.

Figure 17. Packing versus performance trade-off between PA and VA-backed memory for different time window lengths.

To choose the percentile, we estimate this trade-off using the VM traces from our study. Figure 17 shows the expected number of VA accesses based on the utilization percentile and the time window length, assuming each VM uniformly accesses its utilized memory. Figure 17a shows that the VA accesses are much fewer than the prediction percentile for all time window lengths (Worst), due to rounding up the PA-allocation to 5% buckets. For lower percentiles, the time window length is more important. Finer-grained windows exploit more temporal patterns but risk additional accesses to oversubscribed memory. Figure 17b shows the percentage of VMs with less than a percentage of VA accesses using a 4-hour window. For example, 99% of VMs have below 5% VA accesses when predicting the P80 utilization.

Figure 17b helps estimate additional accesses to oversubscribed memory that may result from under-allocating the guaranteed portion. For example, allocating the P75 instead of P80 risks ~1% more accesses. Coach's scheduling policy is robust against such mispredictions. Under-predictions only lead to under-allocation if they under-predict the maximum across all time windows, which may require multiple under-predictions. Conversely, an over-prediction in a single time window can result in over-allocation. While this may reduce savings, it is acceptable, as we prioritize protecting workload performance (G2).

Coach configuration. We configure Coach based on the performance vs. packing trade-off described above and time window length vs. packing trade-off in Figure 11. To minimize the potential impact on VM workloads (G2), we use the P95 utilization prediction and six 4-hour time windows. This ensures oversubscribed resources are used at most 5% of the time, though much less in practice. We conservatively round allocations up to 5% buckets and the resource management granularity (*e.g.*, 1GB for memory).

3.4 Monitoring and mitigating contention

Resource contention. Oversubscribing resources introduces the risk of contention as the total utilized VM resources

may exceed the available capacity. This can degrade the performance of VM workloads. For CPU contention, VMs may need to wait for a specific core or run on other cores, which can hurt cache locality. For memory contention, the hypervisor may page memory out, leading to increased disk I/O and latency. Coach effectively reduces the potential impact of contention on VM workloads (G2) by *monitoring* and *predicting* contention and *mitigating* it when it occurs. Coach uses generic metrics (*e.g.*, CPU utilization) to monitor for contention without user input, but can be extended to include user metrics for further optimization (*e.g.*, tail latency) [42].

Monitoring resource utilization. The *monitoring* component periodically (every 20 seconds in our implementation) tracks resource utilization and contention metrics (*e.g.*, CPU wait time and page read/write operations). To detect potential contention, it uses thresholds (*e.g.*, >0.1% CPU wait time at >20% CPU utilization) computed using historical data at scale and correlated to performance incidents. It notifies the *mitigation* component to trigger reactive mitigations whenever it detects contention. We find that 20 seconds works well for memory, which spikes gradually. For CPU, its fungibility can help reduce the impact of short spikes, allowing us to reduce the monitoring frequency. The *monitoring* component sends this data to the *prediction* and *mitigation* components.

Predicting contention. The local *prediction* component uses real-time and historical data to anticipate potential resource contention. It uses a two-level prediction, an exponential weighted moving average (EWMA) [43] that predicts the utilization for the next 20 seconds, and a long short-term memory network (LSTM) [70, 107] for the next 5 minutes. EWMA is effective because resource behavior tends to be stable for short periods, while the LSTM better captures workload trends over time. If the prediction exceeds the contention threshold, it notifies the *mitigation* component to trigger proactive mitigations. If Coach underpredicts the utilization, it relies on the *monitoring* component to trigger reactive mitigations.

Mitigating contention. To prevent performance degradation, the *mitigation* component triggers *reactive* and *proactive* mitigations when signaled by the *monitoring* and *prediction* components, respectively.

Local mitigations. For CPU, the local *mitigation agent* first readjusts the CPU groups to meet actual demand. Given that CPU is fungible, VMs under contention can borrow guaranteed idle cores from other VMs. If necessary, Coach may increase the CPU frequency (if possible). For memory, the agent first trims cold pages and, if necessary, requests the *server manager* to add unallocated memory to the oversubscribed portion. It uses similar techniques for other resources.

Global mitigations. When local mitigations are insufficient, the *server manager* may ask the *cluster manager* to evict lower-priority VMs (*e.g.*, Spot VMs) or trigger live migration. Coach decides which VM to migrate based on the potential to remedy contention (*e.g.*, busier VMs cause more contention) and overhead (*e.g.*, larger VMs require longer migration times). As most resources are consumed by long-running VMs, Coach may migrate VMs whose resource requirements have changed. However, migration is the last option as it is the most expensive.

3.5 Production considerations

Managing VMs vs. containers. Unlike containers, VMs present additional challenges for cloud platforms due to their coarser resource granularity and opacity. We developed the CoachVM to address these transparently, requiring no workload or OS modifications. Unlike short-lived, auto-scaled containers, VMs typically have more stable and longer temporal patterns. Coach leverages these patterns for efficient VM scheduling and uses hypervisor-level metrics to transparently monitor and mitigate the impact of oversubscription.

Maintainability and simplicity. We consider the complexity of maintaining and operating Coach. While more sophisticated learning-based approaches could offer marginal improvements, their decisions are often harder to interpret, making them impractical [28, 59, 60]. Earlier versions of Coach used more complex algorithms but later shifted to higher-level concepts (*e.g.*, time windows) that operators can more easily understand when troubleshooting issues.

Modularity and extensibility. Coach accounts for components at multiple levels of the stack (*e.g.*, hardware, hypervisor, OS, and management agents), each with distinct rollout and development cycles. While agents can be developed iteratively, hardware requires multi-year cycles. Therefore, Coach is extensible and considers backward compatibility (*e.g.*, hardware without ATS/PRI and legacy VM types).

Staged rollout. We initially oversubscribed fungible resources (*e.g.*, CPU of certain first-party workloads [25] and power via capping [51]) at lower rates. But, other resources like memory quickly become the bottleneck (Section 2.2). As confidence builds (*e.g.*, in mitigation effectiveness), we will oversubscribe them and increase the oversubscription rate.

Customer awareness. Coach simplifies adoption (G1) and minimizes the impact on customers' workloads (G2). However, users often prefer to know if they are oversubscribed. Thus, Coach can be exposed to users as an opt-in feature at a discount and/or restricted to first-party VMs. Our study of internal workloads showed that over 13% were not user-facing, and nearly 25% were delay-tolerant, characteristics that are favorable for oversubscription [68].

3.6 Coach implementation

We build Coach based on the system for static CPU oversubscription described in the Resource Central paper [25].

CoachVM. We extend our existing VM management code to initialize the guaranteed and oversubscribed portions for each resource. We use the mechanisms described in Table 1, which are already available in Windows Server 2025 [12] and Hyper-V. For memory, we initialize the virtual NUMA topology at VM boot time. All these mechanisms are transparent and do not require any modifications to the guest.

Utilization time windows. We extend Resource Central [25] to predict the utilization for each resource for each time window (*e.g.*, time bins of 4 hours) using existing 5-minute resource utilization telemetry. We add this as a new resource to our rule-based VM allocator [37]. We extend the local VM manager to store the resource requirements for each CVM in each time window and perform multiplexing across CVMs.

Monitoring and mitigating contention. We also extend the monitoring, prediction, and mitigation components in the local VM manager (Section 3.4) to support new resources. The monitoring component runs every 20 seconds and monitors existing OS performance counters [102]. It tracks memory utilization by monitoring the VM page accesses.

The prediction component produces short-term predictions every 20 seconds and longer-term ones for the next five minutes using the data from the monitoring component. The EWMA is updated in each 20-second window with the preceding resource utilization using $\alpha = 0.5$. The LSTM uses the maximum and average utilization in the five previous 5-minute windows as input and is also updated online. As LSTMs require more input data, we train the model for 24 hours before using its predictions.

The mitigation component reuses existing OS capabilities to trim and flush cold memory and resize the oversubscribed memory partition [6]. After paging in cold memory, we reuse the existing Hyper-V live migration [7]. Similarly, we make no changes to the process of killing low-priority VMs.

4 Evaluation

Our evaluation shows that Coach: (1) introduces minimal performance degradation to workloads running in CoachVMs; (2) enables platforms to host up to 26% more VMs; (3) alleviates worst-case resource contention through mitigations; and (4) introduces minimal overhead to the platform.

4.1 Experimental setup

We evaluate the impact of Coach on VM performance and estimate the overhead it introduces by running VMs on a real production server. We also assess the effectiveness of Coach's scheduling policy and examine performance violations at scale through simulations.

VM workloads. We use unmodified applications representing common cloud workloads, summarized in Table 2. They run on VMs with unmodified Windows Server 2019 [11] and Linux Ubuntu 22.04 [18]. Each workload measures performance with a different *key metric*. Cache, Database, KV-Store, and Microservices are workloads with real-time requirements and their key metric is the P99 tail latency. The remaining workloads have no strict real-time requirements and their key metrics are either run time or throughput.

Table 2. Evaluated cloud workloads.

Workload	Description	Key metric
Cache	Memcached read/writes [62].	Tail Latency
Database	Queries on a SQL database [54].	Tail Latency
Big Data	Sorting with TeraSort [38].	Run Time
Web	3-tier web application [91].	Throughput
KV-Store	Querying a KV-store [54].	Tail Latency
Graph	Computing pagerank [39].	Run Time
Microservices	Social network [33].	Tail Latency
LLM-FT	BERT LLM fine-tuning [30].	Run Time
Video Conf	Video conference application [54].	Throughput

Server. We use a production server with two NUMA nodes and 160 hyper-threaded Intel CPU cores running at 2.3GHz and 512GB of DRAM. The page file is placed on a Dell P5600 NVMe SSD [27]. We reserve 2 cores and 4GB of memory to run Coach. For each experiment, we ensure isolation by using CPU groups and placing the PA/VA memory into separate memory partitions.

Simulator. Before fully deploying Coach, which takes multiple years, we assess its benefits at scale using simulations. This enables us to evaluate multiple scheduling policies using identical VM traces on clusters with diverse hardware configurations without the need for more complex testing methods (*e.g.*, A/B testing or input mirroring).

Our simulator assigns VMs to servers by executing the real production VM scheduler code [37] on the production VM traces (Section 2). We extended the simulator to support long-term predictions and time-window-based scheduling, as described in Section 3.3. It is validated and closely mimics the constraints and preferences in the real scheduler. Based on the VM placements of the simulator, we simulate the resource utilization for each server using the 5-minute data and estimate the contention.

4.2 CoachVM performance

Coach minimizes the performance degradation from oversubscription. To demonstrate this, we execute the workloads in Table 2 with four VM configurations: Gpvm, which is fully guaranteed with PA; Ovm, which is fully oversubscribed with VA; Cvm, which uses Coach to generate the PA/VA split; and Cvm-Floor, which emulates an under-allocation (by 1GB). We run each experiment five times and create a new VM for each run. We report the median for the *key metric* of the workload (Table 2), with error bars for the max/min.

Workload performance. Figure 18 shows the performance slowdown for each workload, normalized to the median of

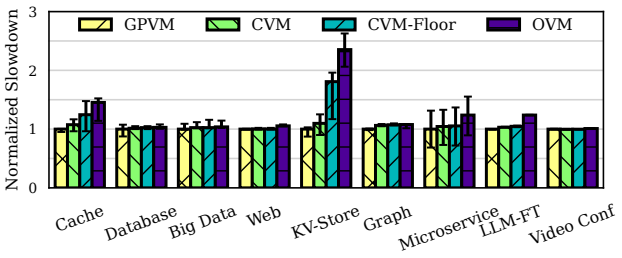


Figure 18. Performance of cloud workloads using various VM configurations.

Gpvm. Microservice, Cache, and KV-Store are the most sensitive to oversubscription, with their performance degrading by up to 2.35× in the worst case (0.41 vs. 0.96ms) with Ovm. These workloads access memory in the critical path that may need to be allocated on demand, which degrades their tail latency.

The conservative allocations strategy from Coach and memory funneling with zNUMA prevent worst-case degradation (at most 10%, from 0.41 to 0.45ms), as shown by Cvm. The other real-time workloads (Cache, Database, and Microservice) have 7% (6.32 vs. 6.77ms), 2% (40 vs. 41ms), and 4% (2.71 vs. 2.83ms) tail latency slowdown, respectively. This demonstrates that Coach can support even sensitive workloads with real-time requirements.

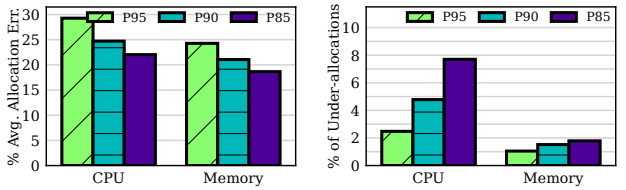
Among workloads using other key metrics, Llm-ft is the most sensitive (1.24× worse: 3.7 vs. 4.5 mins) because it has the largest working set and frequently allocates/deallocates memory for each training iteration. The limited memory reuse and frequent turnover stress the lower TLB reach and on-demand allocation, reducing performance. The remaining workloads experience at most 6% slowdown with Cvm.

Performance with under-allocations. Under-allocating the guaranteed portion can result in 1.8× performance degradation (0.41ms vs. 0.74ms), as shown by Cvm-Floor. KV-Store and Cache are more sensitive to under-allocation than other workloads. Since their working sets are smaller, the oversubscribed portion receives a larger fraction of the total memory accesses. The remaining workloads experience at most an 8% slowdown with CVM-Floor. However, Coach's scheduling policy is robust against under-allocations (Section 3.3), and rarely under-allocates VMs (Section 4.3).

4.3 Impact of time window scheduling

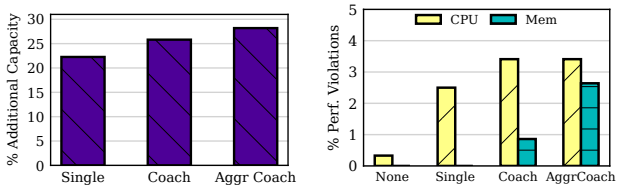
Coach effectively predicts oversubscription rates to maximize available capacity and minimize resource contention.

Prediction accuracy. We evaluate the effectiveness of Coach's predictions for the cluster-level, time-window-based VM scheduling policy in avoiding potential performance degradation from under-allocations while maximizing resource savings. Figure 19a shows the over-allocation error using P95, P90, and P85 prediction percentiles. The average error



(a) Over-allocation. **(b)** Under-allocations.

Figure 19. Effectiveness of Coach's long-term predictions for different prediction percentiles.



(a) Additional capacity normalized against None. **(b)** Performance violations.

Figure 20. Impact of various oversubscription policies.

is 23-30% for CPU and 19-24% for memory. The positive error is the additional resources that could have been saved compared with the ideal VM allocation. As we decrease the prediction percentile, the error decreases.

Figure 19b shows the under-allocations when allocating fewer resources than the ideal allocation. Memory has few under-allocations (1-2%), while CPU has a slightly higher percentage (3-8%). Predicting CPU is harder due to its higher fluctuations, but it is also more fungible, which reduces the impact of under-allocations. Our scheduling policy helps mask the impact of individual under-predictions, as only an under-prediction reducing the maximum across all time windows results in an under-allocation. Multiplexing CoachVMs reduces this further. This demonstrates that Coach effectively prioritizes minimized performance degradation (*i.e.*, fewer under-allocations) for maximized resource savings (*i.e.*, fewer over-allocations).

Packing. We evaluate the cluster-level savings brought by Coach's time-window-based VM scheduling policy. Figure 20a shows the additional sellable capacity (*i.e.*, additional VMs that can be hosted) generated by different oversubscription policies. We simulate four policies. (1) No oversubscription (None), which allocates all the VM's requested resources. (2) Single oversubscription rate per VM (Single), which predicts a static oversubscription rate for each resource. This represents a baseline similar to the state-of-the-art [25, 86, 93, 94]. (3) Time-window-based oversubscription per VM (Coach). (4) An aggressive Coach (Aggr Coach), which uses a P50 prediction percentile. Single increases capacity by 22% compared to None. Coach provides an additional 16% capacity over Single and, Aggr Coach adds 9% more capacity over Coach. Coach also reduces the number of required servers by 44% through improved consolidation of VMs onto servers.

Performance. We quantify how effectively Coach's scheduling policy minimizes the resource contention from oversubscription. CPU contention occurs when demand exceeds 50% of the server capacity, while memory contention occurs when memory accesses result in page faults. Figure 20b summarizes the contention for each policy. Single adds 2% CPU contention and no memory contention. Coach increases CPU contention by 1% while resulting in less than 1% memory violations. Aggr Coach introduces an additional 2% memory violations. Overall, Coach's predictions and scheduling policy successfully leverage complementary temporal patterns to improve utilization with minimal impact.

4.4 Mitigating contention

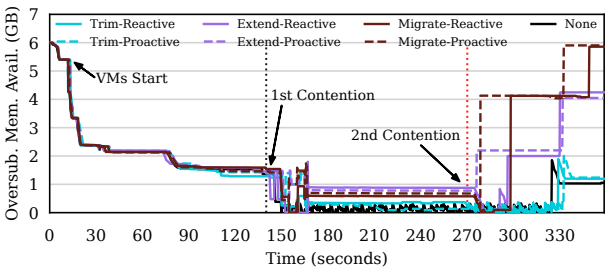
As oversubscription may still introduce some contention, we evaluate how effectively Coach identifies and mitigates it to minimize performance degradation. We focus on memory in this subsection due to its sensitivity to contention.

Predicting contention. The EWMA has low error due to the overall stability of memory utilization, with an average error under 4% for over 85% of VMs. The LSTM better captures historical utilization patterns to predict utilization and achieves only 2% average error for 95% of VMs. The LSTM more accurately predicts VMs with dynamic but predictable patterns where the EWMA can show significant error. The combination of these two predictors allows Coach to effectively predict contention and trigger proactive mitigations.

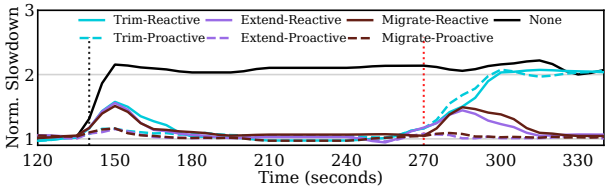
Mitigation policies. To demonstrate the effectiveness of our mitigation policies in minimizing performance degradation, we evaluate six policies against a baseline with no mitigation (None). Trim only implements trimming. If no cold memory is available for trimming, Extend may expand the oversubscribed memory pool with unallocated memory, and Migrate migrates a VM to free resources. The Reactive variations trigger mitigation only after the monitoring component (Section 3.4) detects contention, while the Proactive variations use the prediction component to proactively trigger mitigation.

Effectiveness of mitigation policies on memory. To demonstrate the effectiveness of Coach's mitigation policies, we evaluate their impact on memory contention and workload performance. We use the two most memory-sensitive workloads, Cache and KV-Store, to show the worst-case scenario. We colocate them with a CVM running Video Conf, which uses more memory than predicted, causing contention twice. Each workload runs on an 8GB CVM. The working set for both Cache and KV-Store is ~4GB, and they run on CVMs with 3GB-PA (and 5GB-VA). The working set of Video Conf is 5GB, but it runs on a CVM with 1GB-PA (and 7GB-VA). We allocate an initial 6GB to the oversubscribed pool to back the 17GB of VA in the CVMs.

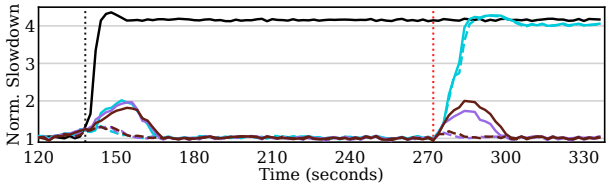
Figure 21a shows the available memory in the oversubscribed portion (*i.e.*, VA-backed). The first contention starts



(a) Available VA-backed memory during contention.



(b) Zoom in on Cache performance during contention.



(c) Zoom in on KV-Store performance during contention.

Figure 21. Comparison of Coach's mitigation policies during two memory contentions.

at 135 seconds, when Video Conf consumes more memory than initially predicted. For the first contention, VMs have enough cold memory that can be trimmed. The second contention starts at 255 seconds, when Video Conf increases its working set, exceeding the amount of cold memory available for trimming. The first contention demonstrates the effectiveness of trimming, while the second demonstrates the effectiveness of the other mitigation measures.

None frequently pages out memory that is paged in later and fails to recover from contention. Trim identifies large portions of cold memory in advance to reduce the number of trimming operations, and resolves the first contention quickly. As the other policies also trim, they exhibit similar performance during the first contention but differ during the second. Trim cannot recover from the second contention because there is insufficient cold memory. Extend quickly mitigates the second contention by expanding the oversubscribed pool with unallocated memory from the server. Migrate takes longer than Extend to resolve it as the memory cannot be reclaimed until Video Conf is migrated. Across all policies, Proactive triggers mitigations earlier and resolves contention faster than Reactive.

Impact of mitigation policies on performance. Figures 21b and 21c show the performance of the VMs during contention. Contention degrades performance by up to 4.3×, while our

proactive policies reduce this overhead to only 1.3× by reducing the duration and intensity of the contention. This demonstrates that Coach's mitigation policies effectively minimize the performance degradation from contention. The other workloads experience less degradation.

4.5 Coach platform overheads

Coach minimizes overhead by reusing and extending existing systems in Azure when possible. We demonstrate this by profiling the overhead Coach introduces to cloud platforms.

Predicting utilization time windows. To train the model for 6 time windows with about one million VMs, Coach requires under 100MB of data and 121 seconds for daily offline training. The model consumes 186MB of memory.

Scheduling overheads. From the simulations, the additional six dimensions for bin-packing each resource introduce less than 1ms to the scheduling time of a VM.

CoachVMs. In Section 4.2, the worst case page fault count for Cvm is less than 15% of the ones for the Ovm. In addition, the oversubscribed portion requires tracking accesses for trimming. This requires 8MB of memory for a typical 32GB VM. Tracking every 20 seconds requires 2 additional hyperthreaded cores (~1.25% overhead). We offset this overhead by saving 16% through CPU oversubscription (15% net savings).

Predicting local contention. Each local predictor requires 25KB of memory and 0.86ms for each training/inference cycle (every 5 minutes). Even at peak load with >100 VMs on a single server, these overheads are easily absorbed by the existing cores reserved for server management.

Mitigation measures. Coach supports trimming and extending the resource pool to help alleviate memory contention. We can achieve a trim bandwidth of 1.1GB/s. Extending the oversubscribed pool achieves higher bandwidth at 15.7GB/s as it does not require writing cold memory to the backing store.

5 Related work

Resource oversubscription. Numerous studies have explored resource oversubscription [14, 31, 46, 51, 69, 76, 84, 90, 101] and cloud providers already offer static oversubscription [25, 45, 58, 78, 83, 86, 93, 94, 96, 110, 110]. Several studies have focused on optimizing individual resource utilization in data centers, including CPU [46, 66, 69, 90], memory [58, 86, 93, 94, 101], power [15, 34, 41, 50, 51, 55, 72, 79, 103], and storage [73, 97]. Coach is the first work to target all resources in a virtualized environment while exploiting complementary temporal patterns to improve utilization.

Oversubscribed containers. Borg [94, 96] manages cluster scheduling and allocation, optimizing resource use through job oversubscription. Google's Autopilot [78] configures task concurrency and CPU/memory limits using machine learning and historical data to reduce slack and task failures. Twine [36, 93] orchestrates containers across servers using dynamic machine partitioning, preventing capacity stranding and allowing CPU or memory oversubscription upon user request. Twine SRM adjusts job tasks based on historical data. Coach introduces the CoachVM to overcome the additional challenges for all-resource oversubscription in virtualized environments (*e.g.*, opaqueness, live migration, direct device assignment, and host updates).

Oversubscribed VMs. EC2 [86] offers VMs with flexible resource allocation and oversubscription capabilities, leveraging a shared Linux kernel to optimize CPU and memory utilization. Coach employs the new CoachVM, which has a guaranteed and oversubscribed portion of each resource. This enables platforms to ensure performance while exploiting complementary temporal patterns to save additional resources without requiring users to modify their workloads.

Workload-aware scheduling. Researchers leveraged learning techniques to optimize resource efficiency while ensuring SLOs [16, 20, 42, 51, 71, 88, 89, 108, 109] . Some studies utilized historical information about the services for intelligent task scheduling and data placement [40, 47, 58, 93, 110], while others concentrated on optimizing server and VM utilization, improving scalability and fault tolerance [25, 49, 56, 92]. Coach uses generic metrics to predict and leverage patterns in resource utilization. In addition, it uses these metrics to predict and proactively mitigate contention, ensuring efficient utilization of resources and maximized workload performance without requiring workload awareness.

6 Conclusion

We introduced Coach, a system to improve resource utilization in cloud platforms by leveraging temporal patterns in VM workloads. Our comprehensive characterization of resource utilization revealed that VMs often exhibit complementary patterns, which Coach exploits to increase oversubscription without compromising performance.

Coach's time-window-based predictive scheduling policy enables cloud platforms to significantly reduce low resource utilization. We introduce the CoachVM to address the challenges of oversubscription in virtualized environments. By considering all resources, Coach provides a holistic solution that safely increases resource oversubscription, enabling cloud platforms to host up to 26% more VMs.

Acknowledgements

We thank the anonymous reviewers and our shepherd, Michael Swift, for their valuable feedback and constructive suggestions that helped improve this paper. We thank Arup Roy, Patrick Payne, Milos Kralj, and the entire Core OS team at Microsoft Azure for their help. Benjamin Reidys and Jian Huang were partially supported by NSF grant CCF-1919044 and NSF CAREER Award CNS-2144796.

References

[1] Nael Abu-Ghazaleh, Dmitry Ponomarev, and Dmitry Evtyushkin. How the spectre and meltdown hacks really worked. *IEEE Spectrum*, 56(3):42–49, 2019.

[2] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. CherryPick: Adaptively unearthing the best cloud configurations for big data analytics. In *NSDI*, 2017.

[3] Pradeep Ambati, Íñigo Goiri, Felipe Frujeri, Alper Gun, Ke Wang, Brian Dolan, Brian Corell, Sekhar Pasupuleti, Thomas Moscibroda, Sameh Elnikety, Marcus Fontoura, and Ricardo Bianchini. Providing SLOs for Resource-Harvesting VMs in Cloud Platforms. In *OSDI*, 2020.

[4] Azure. Azure D-Series. https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/general-purpose/d-family, 2024.

[5] Azure. Hyper-V Architecture. https://learn.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/hyper-v-architecture, 2024.

[6] Azure. Hyper-V Memory Performance. https://learn.microsoft.com/en-us/windows-server/administration/performance-tuning/role/hyper-v-server/memory-performance, 2024.

[7] Azure. Live Migration Overview. https://learn.microsoft.com/en-us/windows-server/virtualization/hyper-v/manage/live-migration-overview, 2024.

[8] Microsoft Azure. Introducing B-Series, our new burstable VM size. https://azure.microsoft.com/en-us/blog/introducing-b-series-our-new-burstable-vm-size/, 2017.

[9] Microsoft Azure. Azure Spot Virtual Machines. https://azure.microsoft.com/en-us/pricing/spot, 2020.

[10] Microsoft Azure. Sizes for virtual machines in Azure. https://learn.microsoft.com/en-us/azure/virtual-machines/sizes, 2024.

[11] Microsoft Azure. Windows Server 2019. https://www.microsoft.com/en-us/evalcenter/download-windows-server-2019, 2024.

[12] Microsoft Azure. Windows Server 2025 Preview. https://www.microsoft.com/en-us/evalcenter/evaluate-windows-server-2025, 2024.

[13] Luis Andre Barroso and Jimmy Clidaras. *The datacenter as a computer: An introduction to the design of warehouse-scale machines*. 2022.

[14] Salman A. Baset, Long Wang, and Chunqiang Tang. Towards an Understanding of Oversubscription in Cloud. In *Hot-ICE*, 2012.

[15] Arka A Bhattacharya, David Culler, Aman Kansal, Sriram Govindan, and Sriram Sankar. The need for speed and stability in data center power capping. *Sustainable Computing: Informatics and Systems*, 3(3):183–193, 2013.

[16] Ricardo Bianchini, Marcus Fontoura, Eli Cortez, Anand Bonde, Alexandre Muzio, Ana-Maria Constantin, Thomas Moscibroda, Gabriel Magalhaes, Girish Bablani, and Mark Russinovich. Toward ML-centric Cloud Platforms. *Communications of the ACM*, 2020.

[17] Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. Apollo: Scalable and coordinated scheduling for Cloud-Scale computing. In *OSDI*, 2014.

[18] Canonical. Ubuntu 22.04.4 LTS (Jammy Jellyfish). https://www.releases.ubuntu.com/22.04/, 2022.

[19] Marcus Carvalho, Walfredo Cirne, Francisco Brasileiro, and John Wilkes. Long-term SLOs for reclaimed cloud computing resources. In *SoCC*, 2014.

[20] Liuhua Chen and Haiying Shen. Consolidating complementary VMs with spatial/temporal-awareness in cloud datacenters. In *INFOCOM*, 2014.

[21] Alibaba Cloud. Sizes for virtual machines in Alibaba Cloud. https://www.alibabacloud.com/help/en/ecs/user-guide/overview-of-instance-families, 2024.

[22] Google Cloud. Preemptible VM Instances. https://cloud.google.com/compute/docs/instances/preemptible, 2020.

[23] Google Cloud. Overcommitting CPUs on sole-tenant VMs. https://cloud.google.com/compute/docs/nodes/overcommitting-cpus-sole-tenant-vms, 2023.

[24] Google Cloud. Sizes for virtual machines in Google Cloud. https://cloud.google.com/compute/docs/machine-resource, 2024.

[25] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In *SOSP*, 2017.

[26] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-efficient and QoS-aware cluster management. In *ASPLOS*, 2014.

[27] Dell. Dell P5600. https://dl.dell.com/manuals/all-products/esuprt_data_center_infra_int/esuprt_data_center_infra_storage_adapters/dell-poweredge-exp-fsh-nvme-pcie-ssd_deployment-guide4_en-us.pdf, 2020.

[28] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *ASPLOS*, 2016.

[29] Padmapriya Duraisamy, Wei Xu, Scott Hare, Ravi Rajwar, David Culler, Zhiyi Xu, Jianing Fan, Christopher Kennelly, Bill McCloskey, Danijela Mijailovic, Brian Morris, Chiranjit Mukherjee, Jingliang Ren, Greg Thelen, Paul Turner, Carlos Villavieja, Parthasarathy Ranganathan, and Amin Vahdat. Towards an Adaptable Systems Architecture for Memory Tiering at Warehouse-Scale. In *ASPLOS*, 2023.

[30] Hugging Face. Fine-tune a pretrained model. https://huggingface.co/docs/transformers/en/training, 2024.

[31] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *NSDI*, 2018.

[32] Alexander Fuerst, Stanko Novaković, Íñigo Goiri, Gohar Irfan Chaudhry, Prateek Sharma, Kapil Arya, Kevin Broas, Eugene Bak, Mehmet Iyigun, and Ricardo Bianchini. Memory-Harvesting VMs in Cloud Platforms. In *ASPLOS*, 2022.

[33] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Yuan He, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Rick Lin, Zhongling Liu, Jake Padilla, and Christina Delimitrou. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems. In *ASPLOS*, 2019.

[34] Sriram Govindan, Jeonghwan Choi, Bhuvan Urgaonkar, Anand Sivasubramaniam, and Andrea Baldini. Statistical profiling-based techniques for effective power provisioning in data centers. In *EuroSys*, 2009.

[35] Jing Guo, Zihao Chang, Sa Wang, Haiyang Ding, Yihui Feng, Liang Mao, and Yungang Bao. Who Limits the Resource Efficiency of My Datacenter: An Analysis of Alibaba Datacenter Traces. In *IWQoS*, 2019.

[36] Nishant Gupta, Iyswarya Narayanan, Shivam Handa, Sayak Chakraborti, Pankit Thapar, Baohua Shan, Ariel Rao, Yuanlai Liu, Pengyuan Wang, Yuqing Wu, Qingyi Gao, Chris Chao-Chun Cheng, Sihan You, Louis Huang, Jingyuan Fan, Kenny Yu, Kevin Lin, Tengfei Mu, Parth Malani, Haiying Wang, Trey Lu, and Peter Zhang. Dynamic idle resource leasing to safely oversubscribe capacity at meta. In *Proceedings of the 2024 ACM Symposium on Cloud Computing (SoCC'24)*, 2024.

[37] Ori Hadary, Luke Marshall, Ishai Menache, Abhisek Pan, Esaias E Greeff, David Dion, Star Dorminey, Shailesh Joshi, Yang Chen, Mark Russinovich, and Thomas Moscibroda. Protean: VM Allocation Service at Scale. In *OSDI*, 2020.

[38] Hadoop. Hadoop TeraSort. https://hadoop.apache.org/docs/current/api/org/apache/hadoop/examples/terasort/package-summary.html, 2024.

[39] HiBench. HiBench PageRank. https://github.com/Intel-bigdata/HiBench, 2024.

[40] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for Fine-Grained resource sharing in the data center. In *NSDI*, 2011.

[41] Chang-Hong Hsu, Qingyuan Deng, Jason Mars, and Lingjia Tang. SmoothOperator: Reducing power fragmentation and improving power utilization in large-scale datacenters. In *ASPLOS*, 2018.

[42] Lexiang Huang, Anjaly Parayil, Jue Zhang, Xiaoting Qin, Chetan Bansal, Jovan Stojkovic, Pantea Zardoshti, Pulkit Misra, Eli Cortez, Raphael Ghelman, Íñigo Goiri, Saravan Rajmohan, Jim Kleewein, Rodrigo Fonseca, Timothy Zhu, and Ricardo Bianchini. Workload Intelligence: Punching Holes Through the Cloud Abstraction. 2024.

[43] Armann Ingolfsson and Emanuel Sachs. Stability and sensitivity of an EWMA controller. *Journal of Quality Technology*, 25(4):271–287, 1993.

[44] Intel. Address Translation Services (ATS). https://www.intel.com/content/www/us/en/docs/programmable/683686/20-4/address-translation-services-ats.html, 2024.

[45] Michael Isard. Autopilot: Automatic Data Center Management. *ACM SIGOPS Operating Systems Review*, 41(2):60–67, 2007.

[46] Seyedali Jokar Jandaghi, Kaveh Mahdaviani, Amirhossein Mirhosseini, Sameh Elnikety, Cristiana Amza, and Bianca Schroeder. AUDIBLE: A Convolution-Based Resource Allocator for Oversubscribing Burstable Virtual Machines. In *ASPLOS*, 2024.

[47] Karthik Kambatla, Vamsee Yarlagadda, Íñigo Goiri, and Ananth Grama. UBIS: Utilization-aware cluster scheduling. In *IPDPS*, 2018.

[48] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. Profiling a warehouse-scale computer. In *ISCA*, 2015.

[49] Hwanju Kim, Hyeontaek Lim, Jinkyu Jeong, Heeseung Jo, and Joonwon Lee. Task-aware virtual machine scheduling for I/O performance. In *VEE*, 2009.

[50] Vasileios Kontorinis, Liuyi Eric Zhang, Baris Aksanli, Jack Sampson, Houman Homayoun, Eddie Pettis, Dean M Tullsen, and Tajana Simunic Rosing. Managing Distributed UPS Energy for Effective Power Capping in Data Centers. In *ISCA*, 2012.

[51] Alok Gautam Kumbhare, Reza Azimi, Ioannis Manousakis, Anand Bonde, Felipe Frujeri, Nithish Mahalingam, Pulkit A. Misra, Seyyed Ahmad Javadi, Bianca Schroeder, Marcus Fontoura, and Ricardo Bianchini. Prediction-Based Power Oversubscription in Cloud Platforms. In *USENIX ATC*, 2021.

[52] KVM. How to assign devices with VT-d in KVM. https://www.linux-kvm.org/page/How_to_assign_devices_with_VT-d_in_KVM, 2024.

[53] Andres Lagar-Cavilla, Junwhan Ahn, Suleiman Souhlal, Neha Agarwal, Radoslaw Burny, Shakeel Butt, Jichuan Chang, Ashwin Chaugule, Nan Deng, Junaid Shahid, Greg Thelen, Kamil Adam Yurtsever, Yu Zhao, and Parthasarathy Ranganathan. Software-Defined Far Memory in Warehouse-Scale Computers. In *ASPLOS*, 2019.

[54] Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. In *ASPLOS*, 2023.

[55] Shaohong Li, Xi Wang, Faria Kalim, Xiao Zhang, Sangeetha Abdu Jyothi, Karan Grover, Vasileios Kontorinis, Nina Narodytska, Owolabi Legunsen, Sreekumar Kodakara, et al. Thunderbolt: Throughput-Optimized, Quality-of-Service-Aware Power Capping at Scale. In *OSDI*, 2020.

[56] Xiang Li, Peter Garraghan, Xiaohong Jiang, Zhaohui Wu, and Jie Xu. Holistic virtual machine scheduling in cloud datacenters towards minimizing total energy. *IEEE Transactions on parallel and distributed systems*, 29(6):1317–1331, 2017.

[57] LightGBM. LightGBM. https://github.com/microsoft/LightGBM, 2024.

[58] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. Heracles: Improving resource efficiency at scale. In *ISCA*, 2015.

[59] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource Management with Deep Reinforcement Learning. In *OSDI*, 2016.

[60] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning Scheduling Algorithms for Data Processing Clusters. In *SIGCOMM*, 2019.

[61] McKinsey and Company. Revolutionizing data center efficiency. *Uptime Institute Symp*, 2008.

[62] Memcached. Memcached. https://memcached.org/, 2024.

[63] Xiaoqiao Meng, Canturk Isci, Jeffrey Kephart, Li Zhang, Eric Bouillet, and Dimitrios Pendarakis. Efficient resource provisioning in compute clouds via VM multiplexing. In *ICAC*, 2010.

[64] Microsoft. Deploy graphics devices by using Discrete Device Assignment. https://learn.microsoft.com/en-us/windows-server/virtualization/hyper-v/deploy/deploying-graphics-devices-using-dda, 2024.

[65] Microsoft. Introduction to Single Root I/O Virtualization (SR-IOV). https://learn.microsoft.com/en-us/windows-hardware/drivers/network/single-root-i-o-virtualization--sr-iov-, 2024.

[66] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. Shenango: Achieving high CPU efficiency for latency-sensitive datacenter workloads. In *NSDI*, 2019.

[67] Kay Ousterhout, Patrick Wendell, Matei Zaharia, and Ion Stoica. Sparrow: Distributed, low latency scheduling. In *SOSP*, 2013.

[68] Anjaly Parayil, Jue Zhang, Xiaoting Qin, Íñigo Goiri, Lexiang Huang, Timothy Zhu, and Chetan Bansal. Towards cloud efficiency with large-scale workload characterization, 2024.

[69] Yajuan Peng, Shuang Chen, Yi Zhao, and Zhibin Yu. UFO: The Ultimate QoS-Aware Core Management for Virtualized and Oversubscribed Public Clouds. In *NSDI*, 2024.

[70] PyTorch. LSTM. https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html, 2024.

[71] Anshul Rai, Ranjita Bhagwan, and Saikat Guha. Generalized resource allocation for the cloud. In *SoCC*, 2012.

[72] Parthasarathy Ranganathan, Phil Leech, David Irwin, and Jeffrey Chase. Ensemble-level power management for dense blade servers. *ACM SIGARCH computer architecture news*, 34(2):66–77, 2006.

[73] Benjamin Reidys, Jinghan Sun, Anirudh Badam, Shadi Noghabi, and Jian Huang. BlockFlex: Enabling Storage Harvesting with Software-Defined Flash in Modern Cloud Platforms. In *OSDI*, 2022.

[74] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *SoCC*, 2012.

[75] Zhenyuan Ruan, Seo Jin Park, Marcos K Aguilera, Adam Belay, and Malte Schwarzkopf. Nu: Achieving Microsecond-Scale Resource Fungibility with Logical Processes. In *NSDI*, 2023.

[76] Adam Ruprecht, Danny Jones, Dmitry Shiraev, Greg Harmon, Maya Spivak, Michael Krebs, Miche Baker-Harvey, and Tyler Sanderson. VM Live Migration At Scale. In *VEE*, 2018.

[77] Mark Russinovich, Naga Govindaraju, Melur Raghuraman, David Hepkin, Jamie Schwartz, and Arun Kishan. Virtual Machine Preserving Host Updates for Zero Day Patching in Public Cloud. In *EuroSys*, 2021.

[78] Krzysztof Rzadca, Pawel Findeisen, Jacek Swiderski, Przemyslaw Zych, Przemyslaw Broniek, Jarek Kusmierek, Pawel Nowak, Beata Strack, Piotr Witusowski, Steven Hand, et al. Autopilot: Workload Autoscaling at Google. In *EuroSys*, 2020.

[79] Varun Sakalkar, Vasileios Kontorinis, David Landhuis, Shaohong Li, Darren De Ronde, Thomas Blooming, Anand Ramesh, James Kennedy, Christopher Malone, Jimmy Clidaras, et al. Data center power oversubscription with a medium voltage power plane and priority-aware capping. In *ASPLOS*, 2020.

[80] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. Omega: Flexible, scalable schedulers for large compute clusters. In *EuroSys*, 2013.

[81] scikit learn. RandomForestRegressor. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html, 2024.

[82] Korakit Seemakhupt, Brent E Stephens, Samira Khan, Sihang Liu, Hassan Wassel, Soheil Hassas Yeganeh, Alex C Snoeren, Arvind Krishnamurthy, David E Culler, and Henry M Levy. A Cloud-Scale Characterization of Remote Procedure Calls. In *SOSP*, 2023.

[83] Amazon Web Services. Amazon EC2 M7i and M7i-flex instances. https://aws.amazon.com/ec2/instance-types/m7i/, 2019.

[84] Amazon Web Services. How Amazon ECS manages CPU and memory resources. https://aws.amazon.com/blogs/containers/how-amazon-ecs-manages-cpu-and-memory-resources/, 2019.

[85] Amazon Web Services. Amazon Elastic Compute Cloud, Burstable Performance Instances. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-performance-instances.html, 2020.

[86] Amazon Web Services. Caspian: Cooperative oversubscription. https://reinvent.awsevents.com/on-demand/, 2023.

[87] Amazon Web Services. Sizes for virtual machines in AWS. https://aws.amazon.com/ec2/instance-types/, 2024.

[88] Haiying Shen and Liuhua Chen. CompVM: A complementary VM allocation mechanism for cloud systems. *Transactions On Networking*, 26(3), 2018.

[89] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. CloudScale: elastic resource scaling for multi-tenant cloud systems. In *SoCC*, 2011.

[90] Junjie Sheng, Lu Wang, Fangkai Yang, Bo Qiao, Hang Dong, Xiangfeng Wang, Bo Jin, Jun Wang, Si Qin, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. Learning Cooperative Oversubscription for Cloud by Chance-Constrained Multi-Agent Reinforcement Learning. In *WWW*, 2023.

[91] Spec. SpecJBB. https://www.spec.org/jbb2015/, 2024.

[92] Jovan Stojkovic, Pulkit Misra, Íñigo Goiri, Sam Whitlock, Esha Choukse, Mayukh Das, Chetan Bansal, Jason Lee, Zoey Sun, Haoran Qiu, Reed Zimmermann, Savyasachi Samal, Brijesh Warrier, Ashish Raniwala, and Ricardo Bianchini. SmartOClock: Workload- and Risk-Aware Overclocking in the Cloud. In *ISCA*, 2024.

[93] Chunqiang Tang, Kenny Yu, Kaushik Veeraraghavan, Jonathan Kaldor, Scott Michelson, Thawan Kooburat, Aravind Anbudurai, Matthew Clark, Kabir Gogia, Long Cheng, Ben Christensen, Alex Gartrell, Maxim Khutornenko, Sachin Kulkarni, Marcin Pawlowski, Tuomas Pelkonen, Andre Rodrigues, Rounak Tibrewal, Vaishnavi Venkatesan, and Peter Zhang. Twine: A Unified Cluster Management System for Shared Infrastructure. In *OSDI*, 2020.

[94] Muhammad Tirmazi, Adam Barker, Nan Deng, Md E Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. Borg: the Next Generation. In *EuroSys*, 2020.

[95] Arunchandar Vasan, Anand Sivasubramaniam, Vikrant Shimpi, T Sivabalan, and Rajesh Subbiah. Worth their watts? An empirical study of datacenter servers. In *HPCA*, 2010.

[96] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at Google with Borg. In *EuroSys*, 2015.

[97] VMware. Handling Datastorage Over-Subscription. https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.storage.doc/GUID-68C294FC-7612-4829-95B7-9791915C49AC.html, 2019.

[98] VMWare. Single Root I/O Virtualization (SR-IOV). https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.networking.doc/GUID-CC021803-30EA-444D-BCBE-618E0D836B9F.html, 2024.

[99] Yawen Wang, Kapil Arya, Marios Kogias, Manohar Vanga, Aditya Bhandari, Neeraja J. Yadwadkar, Siddhartha Sen, Sameh Elnikety, Christos Kozyrakis, and Ricardo Bianchini. SmartHarvest: Harvesting Idle CPUs Safely and Efficiently in the Cloud. In *EuroSys*, 2021.

[100] Johannes Weiner, Niket Agarwal, Dan Schatzberg, Leon Yang, Hao Wang, Blaise Sanouillet, Bikash Sharma, Tejun Heo, Mayank Jain, Chunqiang Tang, and Dimitrios Skarlatos. TMO: Transparent Memory Offloading in Datacenters. In *ASPLOS*, 2022.

[101] Dan Williams, Hani Jamjoom, Yew-Huey Liu, and Hakim Weatherspoon. Overdriver: Handling memory overload in an oversubscribed cloud. In *VEE*, 2011.

[102] Windows. Windows Performance Monitor. https://learn.microsoft.com/en-us/dynamics365/business-central/dev-itpro/administration/tools-monitor-performance-counters-and-events, 2024.

[103] Qiang Wu, Qingyuan Deng, Lakshmi Ganesh, Chang-Hong Hsu, Yun Jin, Sanjeev Kumar, Bin Li, Justin Meza, and Yee Jiun Song. Dynamo: Facebook's data center-wide power management system. In *ISCA*, 2016.

[104] XGBoost. XGBoost. https://xgboost.readthedocs.io/en/stable/, 2024.

[105] Chuhao Xu, Yiyu Liu, Zijun Li, Quan Chen, Han Zhao, Deze Zeng, Qian Peng, Xueqi Wu, Haifeng Zhao, Senbo Fu, and Minyi Guo. FaaSMem: Improving Memory Efficiency of Serverless Computing with Memory Pool Architecture. In *ASPLOS*, 2024.

[106] Neeraja J Yadwadkar, Bharath Hariharan, Joseph E Gonzalez, Burton Smith, and Randy H Katz. Selecting the best VM across multiple public clouds: A data-driven performance modeling approach. In *SoCC*, 2017.

[107] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: LSTM cells and network architectures. *Neural computation*, 31(7):1235–1270, 2019.

[108] Chaojie Zhang, Alok Gautam Kumbhare, Ioannis Manousakis, Deli Zhang, Pulkit A Misra, Rod Assis, Kyle Woolcock, Nithish Mahalingam, Brijesh Warrier, David Gauthier, et al. Flex: High-availability datacenters with zero reserved power. In *ISCA*, 2021.

[109] Di Zhang, Dong Dai, Youbiao He, Forrest Sheng Bao, and Bing Xie. RLScheduler: An Automated HPC Batch Job Scheduler Using Reinforcement Learning. In *SC*, 2020.

[110] Yunqi Zhang, George Prekas, Giovanni Matteo Fumarola, Marcus Fontoura, Íñigo Goiri, and Ricardo Bianchini. History-Based Harvesting of Spare Cycles and Storage in Large-Scale Datacenters. In *OSDI*, 2016.