Providing SLOs for Resource-Harvesting VMs in Cloud Platforms

Pradeep Ambati Íñigo Goiri Felipe Frujeri Alper Gun
Ke Wang Brian Dolan Brian Corell Sekhar Pasupuleti
Thomas Moscibroda Sameh Elnikety Marcus Fontoura Ricardo Bianchini *

Microsoft Azure and Microsoft Research

Abstract

Cloud providers rent the resources they do not allocate as evictable virtual machines (VMs), like spot instances. In this paper, we first characterize the unallocated resources in Microsoft Azure, and show that they are plenty but may vary widely over time and across servers. Based on the characterization, we propose a new class of VM, called Harvest VM, to harvest and monetize the unallocated resources. A Harvest VM is more flexible and efficient than a spot instance, because it grows and shrinks according to the amount of unallocated resources at its underlying server; it is only evicted/killed when the provider needs its minimum set of resources. Next, we create models that predict the availability of the unallocated resources for Harvest VM deployments. Based on these predictions, we provide Service Level Objectives (SLOs) for the survival rate (*e.g.*, 65% of the Harvest VMs will survive more than a week) and the average number of cores that can be harvested. Our short-term predictions have an average error under 2% and less than 6% for longer terms. We also extend a popular cluster scheduling framework to leverage the harvested resources. Using our SLOs and framework, we can offset the rare evictions with extra harvested cores and achieve the same computational power as regular-priority VMs, but at 91% lower cost. Finally, we outline lessons and results from running Harvest VMs and our framework in production.

1 Introduction

Motivation. Cloud providers usually rent their resources to customers as Infrastructure as a Service (IaaS) VMs. When deployed, each VM consumes a fixed amount of resources from the server where it lands. Customers can keep their VMs for seconds or years [16] and may request more VMs over time. Thus, providers need to provide the illusion of perfectly elastic resources (*e.g.*, by reserving demand growth buffers) while operating the infrastructure with high availability (*e.g.*, by transparently handling hardware failures). For these reasons, they need to leave unallocated capacity.

To monetize this unallocated capacity, providers offer VMs with relaxed SLOs at discounted prices. Specifically, they offer low-priority evictable VMs, often called spot VMs [1, 8, 14]. These VMs are evicted if their resources are needed by regular-priority (or simply regular) on-demand VMs. Thus, evictable VMs are ideal for customers to run batch jobs or other workloads that can tolerate evictions, at very low cost.

Unfortunately, an evictable VM cannot consume all the unallocated resources of a server unless it fits perfectly in it. Even if it does, a large evictable VM will be promptly evicted whenever even a single resource is needed by a newly arriving regular VM. Multiple small evictable VMs can allocate the same amount of resources but will add overhead to operate more VMs. In addition, their larger number of evictions introduce VM re-creation and application re-initialization overheads that may even cause unavailability.

Given these limitations of existing evictable VMs, we argue that there should be a new class of evictable VMs able to *dynamically* and *flexibly* harvest all the unallocated resources of any server on which they land.

Our work. We first characterize the unallocated resources in Microsoft Azure. The characterization shows that there is potential for harvesting these resources, but they fluctuate over time and their availability is heterogeneous across servers and clusters. The characterization unearths the dynamics of the unallocated resources over multiple time durations.

Next, we propose a new class of evictable VM, called Harvest VM, as a novel way to monetize unallocated resources. A Harvest VM has a minimum size in terms of its physical resources, but it dynamically receives more or fewer physical resources beyond this minimum, depending on the amount of unallocated resources at its underlying server. A Harvest VM is only evicted if its minimum size is needed for a regular VM. In this paper, we focus on harvesting CPU cores.

Provisioning applications to run on harvested resources is challenging. However, we can predict the availability and amount of the unallocated resources in the datacenter. We use these predictions to provide SLOs for Harvest VM deployments. The SLO specifies the probability for a Harvest VM to survive for a certain period and how many resources it will get on average. For example, if a customer wants to create 100 VMs, the SLO may indicate that 90% of them

*Ambati is affiliated with the Univ. of Massachusetts Amherst, but was at Microsoft Research during this work. Gun and Wang are now with Google.

will survive for more than 1 day, with an average of 10 cores. The provider does not monitor or actively seek to meet each individual SLO; instead, we retrain our prediction models frequently and provide our SLO as a statistical estimate [12]. As such, our SLOs can be considered predictions or estimates over large numbers of Harvest VMs, rather than guarantees.

Renting unallocated resources is cheap, but requires applications to manage the evictions. In addition, with Harvest VMs, the amount of resources backing each VM can vary. Harvest VMs are most useful when the applications they run can adapt to the number of available resources. For example, many applications use thread pools and can naturally adapt their parallelism. Others can schedule more load on larger VMs. The provider can hide these complexities by using Harvest VMs to create cheap SaaS (Software-as-a-Service), PaaS (Platform-as-a-Service), and FaaS (Function-as-a-Service) offerings. In fact, Harvest VMs are ideal for cluster scheduling (*e.g.*, Apache YARN [37], Kubernetes [22]) and serverless (*e.g.*, AWS Lambda [32], Azure Functions [4]) frameworks. These frameworks can schedule more tasks/functions on a Harvest VM that has grown to use more physical cores, and stop scheduling tasks/functions on one that has lost physical cores. To demonstrate how to adapt these frameworks, we build Harvest Hadoop to schedule computation (*e.g.*, data-processing, machine learning training) on harvested resources.

Our evaluation shows that we accurately predict the unallocated resources and provide SLOs. We predict the survival rate of a VM for 1 hour with an average error under 2% and lower than 6% for longer terms. We also predict the additional cores that can be harvested within a fraction of a core on average. Our SLOs and framework allow us to run Hadoop workloads on Harvest VMs at 91% lower cost to the customer than regular VMs, by offsetting the rare evictions with additional harvested cores. Compared to standard evictable VMs, the cost savings can reach 47%. Finally, we discuss lessons and results from deploying Harvest VMs and Harvest Hadoop in production to run internal workloads in Azure.

Summary. Our contributions are:

- We characterize the unallocated resources of a large cloud.
- We propose Harvest VMs to harvest unallocated resources.
- We build predictors for the availability of unallocated resources and provide a new SLO for these resources.
- We build Harvest Hadoop, a cluster scheduling framework to leverage Harvest VMs.
- We discuss lessons and results from our production deployment of Harvest VMs and Harvest Hadoop.

2 Background and related work

Deploying VMs. Each VM deployment targets a geographical region, which is partitioned into clusters of servers that have the same hardware. Each region may have a different number of clusters and hardware mix. A region-level scheduler decides which VMs go to which clusters based on several factors (*e.g.*, hardware required, maintenance tasks, available capacity) [19]. These factors can cause clusters to have different VM loads, even in the same region. Then, a cluster-level scheduler decides which server in the cluster will run each VM. When a VM is assigned to a server, a server-level agent creates the VM and manages its lifecycle.

Evictable VMs. Providers sell their excess capacity at discounted prices as evictable VMs [1, 8, 14]. These VMs are evicted/killed when the provider needs the capacity (*e.g.*, due to a spike in the number of on-demand VMs). Providers notify the VMs before they evict them: GCP and Azure provide a 30-second warning, whereas AWS gives 2 minutes.

Variable-resource VMs. Sharma *et al.* [33] recently proposed Deflatable VMs, which change *virtual* resources dynamically (via hot-plugging/unplugging), and a multi-level resource reclamation approach for explicitly adapting applications, operating systems, and hypervisors to the available resources. They also combined reclamation with deflation-aware VM scheduling. We believe that expecting the whole stack to adapt is unrealistic in practice. Instead, we favor simplicity and maintainability for *production deployment*: (1) we minimize the changes to the cloud platform, so deploying Harvest VMs is no different than deploying any other VM, and the VM scheduler is unaware that Harvest VMs grow and shrink; (2) we do not change the number of virtual cores, and instead transparently vary the number of physical cores.

A more aggressive VM design could harvest the unallocated cores *and* any allocated cores that are temporarily idle. This is out of the scope of this paper. Instead, we focus on the usability of core-harvesting VMs (aggressive or otherwise) in practice with SLOs and software for them. Our SLOs can be extended for aggressive harvesting, whereas Harvest Hadoop can be used directly.

Like a Harvest VM, a burstable VM [7, 13] has a fixed number of virtual cores and receives a minimum number of physical cores. However, it is only allowed to burst (*i.e.*, receive additional physical cores) up to its maximum size, after accumulating enough "credits" by staying below a predefined core utilization. A Harvest VM differs in that (1) it harvests as many cores as are unallocated for as long as they remain so, *i.e.* there is no concept of credit; and (2) it is evictable. These characteristics mean that providing SLOs for Harvest VMs is also quite different than for burstable VMs.

Resource harvesting. Other approaches to resource harvesting have either focused on running batch workloads on idle machines (*e.g.* [25, 26]) or co-locating batch workloads with latency-sensitive services on bare-metal servers (*e.g.* [23, 27, 38, 39, 46, 47]). In contrast, we focus on a virtualized infrastructure where physical resources are reserved for the VMs that allocate them (as is the norm in the public cloud), and predict the availability and dynamics of the unallocated resources to produce SLOs.

Characterization and SLOs. To indirectly characterize the unallocated resources at cloud providers, prior work [2, 9, 31,

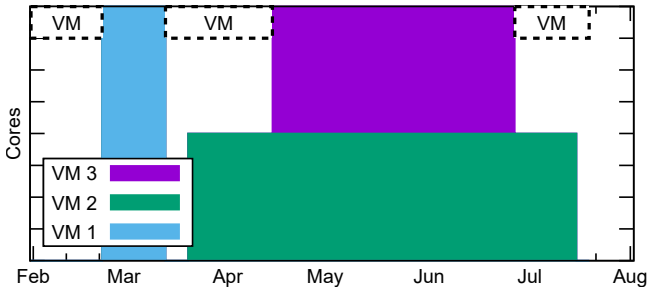

Figure 1: Allocation of VMs on a server, including hypothetical VMs (dashed) that consume unallocated resources.

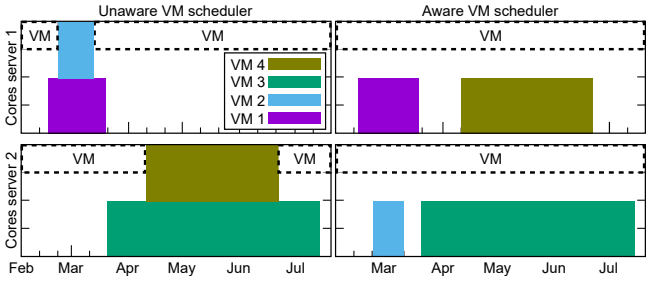

Figure 2: VM allocations on two servers in our characterization (left) and when the VM scheduler is aware of VMs consuming unallocated capacity (right).

34] has analyzed publicly available traces of EC2 spot prices. Using the traces, they tried to model the availability of spot instances. In contrast, we use actual resource allocation data from the entire Azure server fleet to characterize the resources more accurately and comprehensively.

From the perspective of the provider, Carvalho *et al.* [12] characterized the reclaimable resources in 6 Google clusters. They aggregated the cluster-wide resources and predicted their availability for long-term (6-month) SLOs. They did not consider VM evictions or how the reclaimable resources vary at each server. However, the majority of VMs live less than 1 day and get deployed in relatively small groups [16]. Hence, we quantify the unallocated resources *per server* at a *fine time granularity*. Moreover, our SLOs quantify VM survival rates and average numbers of cores over horizons as short as 1 hour.

3 Characterizing unallocated resources

In this section, we characterize the potential for resource harvesting and the dynamics of the unallocated capacity in Azure. The characterization is affected by the Azure VM scheduler [19]. However, the scheduler behaves similarly to those of other providers [38] by tightly packing VMs while ensuring that it can find big enough holes for large VMs.

Methodology. We analyze the resource allocation in Azure from February to October 2019. The data we present does not include confidential metrics, such as number of servers or percentage of unallocated resources. However, the trends we illustrate are enough for the purposes of this paper.

We compute the allocated resources in each server based on the regular VMs running over time, *i.e.* we exclude resources that have been allocated to existing evictable VMs. We account for the main resources (*i.e.*, cores, memory, storage, and network bandwidth) for both the VMs and the servers. We then check if we could allocate in each server a hypothetical evictable VM of a minimum size, for how long, and how many unallocated resources it could potentially get.

In more than 80% of cases where we could not allocate the hypothetical VM, the scarcest resource (*i.e.*, the one that prevents the allocation) is cores. This is not surprising as Azure matches its hardware and VM sizes to have a single dominant resource and simplify capacity management. In the vast majority of remaining cases, disk space is the constraint. Thus, if we can find unallocated cores at a server, the other resources will most likely be unallocated as well.

Figure 1 shows an example server that runs 3 VMs over six months with the allocation of cores on the Y-axis. In early February, there are no VMs allocated to the server so we can run a 1-core hypothetical VM (dashed box) during that time. In late February, *VM 1* starts and takes the full server so we cannot run any other VM. Once *VM 1* finishes, the server becomes empty so we can run another hypothetical VM. *VM 2* starts in late March but it only takes half of the server, so we can keep running the hypothetical VM until *VM3* starts. In this period, we could place 3 hypothetical VMs with an average lifetime of almost one month.

This figure shows the hypothetical VMs with a fixed size but there are plenty of additional unallocated cores still left in the server. For example, when the hypothetical VM can run, at least half of the cores are unallocated.

Our characterization is *pessimistic* in that the unallocated resources are actually *more stable* in practice. For example, our characterization may find the scenario on the left side of Figure 2, which shows two servers with real VMs and hypothetical VMs. However, if the VM scheduler were to actually allocate VMs to consume the unallocated resources, it could allocate the real VMs differently to avoid evicting the hypothetical VMs as on the right side of the figure.

Temporal patterns. A key aspect to quantify is how long we could run a hypothetical evictable VM to consume unallocated resources in each server. Figure 3 shows how many servers could host a 1-core VM with 16GB of memory and 200GB of disk for a given time (*e.g.*, 1 hour, 1 day) in a popular region. We do not list the actual numbers of servers on the Y-axis for confidentiality reasons. Considering 1 hour into the future, we can see a daily pattern where there are more unallocated resources at night. For 1 day, we can see a weekly pattern and how weekends have substantially more unallocated resources. Once we consider the next week, the temporal pattern is not as clear. Overall, the longer horizon numbers show a decrease in unallocated resources over time. These data show that it is important to account for the time of day and day of the week (at least implicitly) when predicting the unallocated resources, especially for shorter periods.

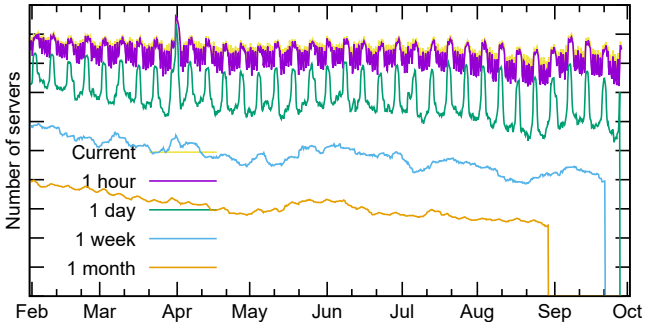

Figure 3: #servers with 1 unallocated core in a region.

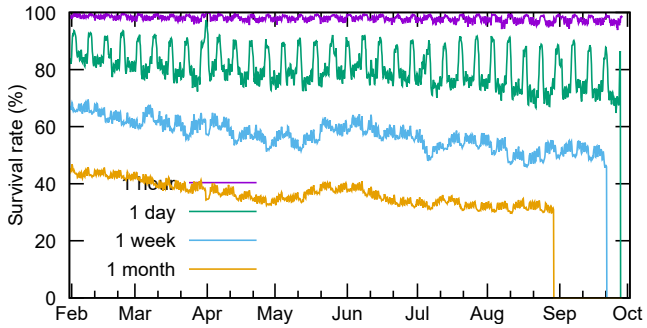

Figure 4: Survival rate with 1 unallocated core in a region.

From these numbers, we can compute the survival rate, *i.e.* the percentage of these evictable VMs that would survive for a given time (*e.g.*, dividing the "1-hour" values by the corresponding "Current" values computes the percentage of VMs that would survive for 1 hour). Figure 4 shows this survival rate over time. For example, it shows that in April, an average of roughly 60% of the 1-core evictable VMs (at most one per server) would survive for one week.

Cluster behaviors. As we discuss in Section 2, clusters may behave differently even within a region. Figure 5 shows how many servers could host a 1-core evictable VM in one specific cluster in the same region as Figure 3. In both late May and early June, the number of allocated VMs increased substantially, each time leaving less unallocated capacity. This shows that the amount of unallocated resources can change drastically over time. There are multiple reasons for such an effect, but in this case it was due to a shift in load across clusters, driven by the higher level across-clusters scheduler. These results show that we must consider each cluster individually when predicting the available unallocated resources.

Aggregating across all regions. So far, we have discussed servers in 1 region. Now, we discuss aggregate data over all regions. First, we consider the average durations over which at least 1 core is unallocated at each server. Over all regions, most servers can host a 1-core evictable VM for at least 1 hour on average. This number drops by 40% for 1 day and by another 40% for 1 month. As expected, fewer servers have at least 1 unallocated core for long periods (*e.g.*, 1 month) than short ones (*e.g.*, 1 hour). Moreover, even when servers have the same overall amount of unallocated capacity over time

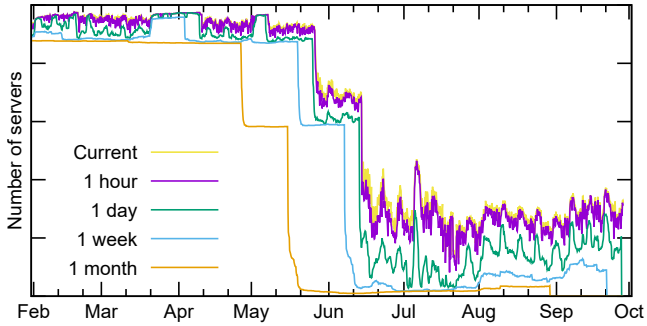

Figure 5: #servers with 1 unallocated core in a cluster.

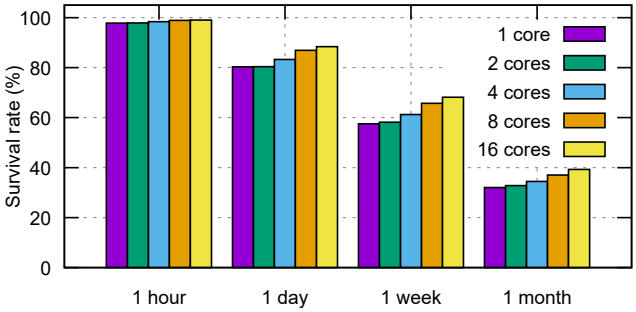

Figure 6: Survival rate of deployable evictable VMs as a function of lifetime and minimum size.

(measured in core×hours), they may be able to host widely different numbers of evictable VMs: servers that tend to have short periods with unallocated cores can host many (short-lived) evictable VMs, whereas those that tend to have long periods with unallocated cores host fewer (long-lived) VMs.

Next, we consider the average survival rate of the deployable 1-core VMs (at most one per server), again aggregating across all regions. The purple bars in Figure 6 plot the average survival rate for all deployable 1-core VMs for 1 hour, 1 day, 1 week, and 1 month. These four bars compute the average of the curves in Figure 4 but for all regions. Almost 100% of the VMs would survive for 1 hour, but only 80% of them would survive for of 1 day and 32% would survive for 1 month.

Minimum unallocated cores. These results quantify the survival rate of 1-core evictable VMs. However, many servers have more unallocated cores than 1. For example, only 55% of the servers have 4 unallocated cores (*i.e.*, capable of hosting a 4-core evictable VM) for at least 1 hour on average.

Figure 6 also plots the average survival rate of other minimum sizes (at most one VM per server). Larger deployed evictable VMs tend to survive longer than smaller ones, even though they are less likely to find a server where to run. For example, 88% of the 16-core VMs survive for 1 day or longer, but only 80% of the 2-core VMs survive for that long. This effect is due to the cluster-level scheduler trying to pack new VMs tightly in servers that are already closer to being full.

Additional unallocated cores. The results above consider evictable VMs that consume a minimum number of unallocated cores. However, as shown in Figure 1, there are many

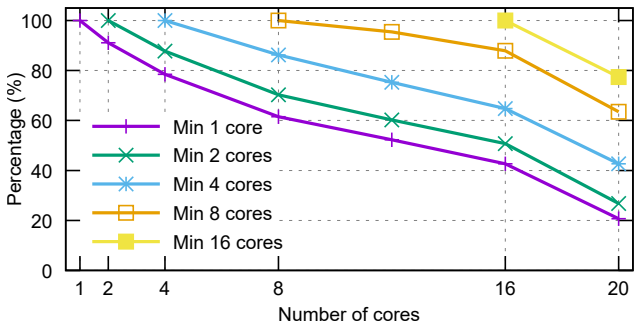

Figure 7: Percentage of deployable evictable VMs that could have received more cores, for each minimum size.

periods when there are additional unallocated resources in the server. Figure 7 shows the percentage of evictable VMs of each minimum size that could potentially have been as large as 1, 2, 4, 8, 16, and 20 cores. For example, 78% of 1-core VMs could have gotten 4 or more cores, and 85% of the 4-core evictable VMs could have gotten 8 or more cores. These results illustrate that (1) a large percentage of the (more numerous) small VMs could have been much larger; and (2) a large percentage of the (less numerous) large VMs could have been even larger. However, allocating a larger evictable VM on a server increases the chance that it will be evicted when the additional cores are needed for higher priority VMs.

Another important consideration is how stable the set of additional cores is, *i.e.* how quickly the set changes due to core allocations/deallocations. We find that 94% of these state changes last for more than 1 second, 90% of them last for more than 5 seconds, 50% of them last for more than 10 minutes, and 10% of them last more than 3 hours. Clearly, the set of additional cores is stable enough that they could be effectively harvested and used by applications.

Multiple VMs per server. So far, we have discussed deploying at most one evictable VM in each server. However, the results above show that there are often enough unallocated resources for more VMs and the amount of these resources varies over time. Under these conditions, the provider can maximize the amount of unallocated resources it monetizes via evictable VMs with as many 1-core VMs as will fit in each server at each point in time. Unfortunately, a larger number of VMs per server increases management (more evictions) and resource (more copies of the guest OS) overheads. The key problem is that *standard evictable VMs are not the ideal abstraction to maximize the use of unallocated resources while keeping overheads down.*

High-level takeaways. Our characterization shows that:

1. There are many unallocated resources that can be harvested. However, they fluctuate significantly over time. There are plenty of unallocated resources for a short time but many fewer for longer periods.
2. These resources are not evenly distributed across clusters. A cluster's allocation may also change drastically over time.
3. The available unallocated resources vary substantially depending on amount (minimum size) and duration. Smaller minimum sizes are more widely available but they do not survive as long as larger minimum sizes.
4. There are many additional unallocated resources in each server beyond this minimum size that can be harvested. The additional resources vary over time at a fairly coarse granularity, but trying to harvest them with standard evictable VMs could cause many evictions and waste resources.

4 Harvest Virtual Machines

Section 3 shows that there are plenty of unallocated resources that can be harvested, while takeaway #4 suggests that doing so with standard evictable VMs is not ideal. Thus, we propose a new class of evictable VM, called *Harvest VM*, that dynamically grows and shrinks to harvest as many unallocated resources as available on the server where it runs. With Harvest VMs, we maximize the resource harvesting at each server, while keeping evictions and overheads down.

Overview. Users select a minimum size for each Harvest VM. A Harvest VM starts with as many unallocated physical resources as are available in its host server, but grows and shrinks dynamically after that. For example, a Harvest VM may have 4 physical cores as its minimum size. At server selection time, this VM is assigned to a server that has at least 4 unallocated cores. Say this server has 20 cores. At creation time, the Harvest VM would be created with 20 *virtual* cores and would receive an initial number of *physical* cores equal to the number of unallocated cores in the server (at least 4 cores, of course). During its lifetime, the Harvest VM will grow (*i.e.*, receive more physical resources) when a co-located regular VM terminates and shrink (*i.e.*, lose physical resources) when a new regular VM lands on the same server. Since the Harvest VM changes size only when other VMs arrive/terminate, these changes occur fairly infrequently (Section 3). As a Harvest VM has lower priority, it is evicted/killed if the cloud platform needs its minimum size for a regular VM.

As an example, Figure 8 shows a server with 8 physical cores that hosts 2 regular VMs with 2 cores each. At t_0, a Harvest VM with a minimum size of 2 cores lands on the server. As there are unallocated cores, the Harvest VM grows to 4 cores. At t_1, VM 2 finishes and the Harvest VM grows to 6 cores. At t_2, VM 3 with 4 cores lands on the server and the Harvest VM shrinks to 2 cores (its minimum size). At t_4, VM 4 with 2 cores lands on the server, causing the Harvest VM to be evicted as it would have to shrink below its minimum size.

Production implementation in Azure. We create a new family of hyperthreaded Harvest VMs that users can select from. The family defines VM types with a minimum size of 1, 2, or 4 cores (*i.e.*, 2, 4, and 8 hyperthreads, respectively). The smallest Harvest VM has a minimum of 1 core, 16GB of memory, 200GB of disk with 3k IOPS, and 1Gbps of network bandwidth. The resources for the larger sizes scale proportionally to the number of minimum physical cores.

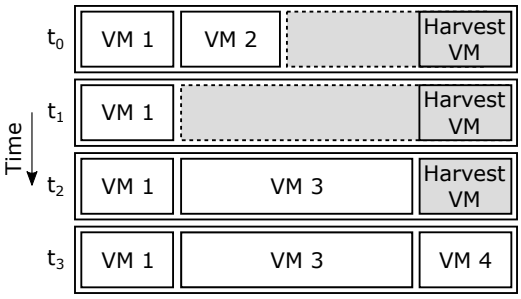

Figure 8: Harvest VM dynamically changing sizes over time.

Our current implementation only harvests physical cores; the other resources stay fixed during the Harvest VMs' lifetimes. The Harvest VMs can grow to use all physical cores of the server, as this fits nicely our current production uses (Section 6), which can consume as many cores as are available. For simplicity, the implementation does not allow more than one Harvest VM per server. We discuss upcoming changes to this design in Section 8.

Users can deploy many Harvest VMs to a region at the same time. The provider deploys the Harvest VMs in the same way that it deploys any other VM. Ultimately, each Harvest VM is scheduled onto a server by a cluster-level VM scheduler. On each server, Azure runs the Hyper-V hypervisor [29] and an agent responsible for managing VMs locally, including VM creation, termination, and physical core reassignment across VMs. The agent uses hypercalls for assigning a Harvest VM to a group of cores and capping the amount of CPU time the group receives. To prevent cache interference between a Harvest VM and the co-located regular VMs, the agent constrains the Harvest VM to a subset of cache ways of the last-level cache, using cache allocation technology [15].

The changes in the number of physical cores are not directly visible by the Harvest VM, as its number of virtual cores does not change. However, the application or scheduling framework running on a Harvest VM may want to take advantage of any harvested cores. Thus, we expose the number of currently assigned physical cores to the Harvest VM via the KVP mechanism of Hyper-V [30]. Applications or frameworks can use this information to adapt their behaviors. For example, a scheduling framework can assign more tasks to a Harvest VM that has just received more cores.

The scheduler may evict a Harvest VM (1) when it needs the minimum resources for a regular VM, or (2) proactively to avoid the eviction latency when it expects that its minimum resources will be needed soon. In either case, the scheduler informs the Harvest VM about the upcoming eviction, and gives it 30 seconds to shutdown cleanly. At deployment time, users can specify whether they want another Harvest VM to be created (on a different server) to replace an evicted one.

Comparison to standard evictable VMs. Unlike evictable (*e.g.*, spot) VMs, Harvest VMs are only evicted when the provider needs their minimum resources for higher priority VMs. In addition, Harvest VMs harvest additional unallocated cores from the servers that host them. In Figure 8, using evictable VMs to harvest those additional cores would have caused them all to be killed at t_2, whereas the Harvest VM shrinks and avoids the high eviction overhead. Due to the additional harvested cores, it takes many more evictable VMs to harvest as many cores as Harvest VMs, implying higher management and resource overheads. In Section 7.5, we show that evictable VMs also imply higher costs to users.

Using Harvest VMs. Harvest VMs are most useful when workloads can gracefully adapt to evictions and a time-varying number of physical cores. First, workloads must be able to continue operating correctly after VM evictions. An eviction is similar to a server failure, so all practical distributed applications are already capable of handling them. Embarrassingly parallel applications handle these failures even more easily. Regardless of application type, users often want new (evictable) VMs to be created to replace evicted VMs, and cloud platforms already provide this functionality. However, as VM re-creation and application re-configuration are expensive, users can make informed decisions about their Harvest VM deployments using our SLOs.

Second, applications must be able to leverage additional cores and degrade gracefully when cores are removed. To do so, applications can check the number of currently assigned physical cores and adapt accordingly. Core re-assignments are much cheaper than VM re-creation and re-configuration, so applications can more easily handle them. For example, the application may create (destroy) software threads when more (fewer) cores are available or have a thread pool where work can wait for cores. Despite their lower overhead, users can use our SLOs to know how many cores to expect per Harvest VM, so they can provision enough threads and VMs.

Still, providers may decide that Harvest VMs are not ideal as an IaaS offering. Instead, they can use them to implement cheaper SaaS, PaaS, or FaaS offerings. In fact, our current Azure deployment uses Harvest VMs to implement a core-harvesting version of Hadoop.

Privacy/confidentiality. On individual servers, Harvest VMs reveal the VM arrival and departure events. However, they do not threaten the confidentiality of the cloud platform's resource utilization, as long as determined (and well-funded) users are not allowed to deploy Harvest VMs to most servers. To avoid this, the provider can simply establish an overall quota of Harvest VMs in each region. The privacy of the workloads is also protected, as Harvest VMs do not reveal any info about (1) co-located regular VMs to the users of Harvest VMs, or (2) their workloads to the provider or co-located regular VM users. In addition, using Harvest VMs for SaaS, PaaS, or FaaS adds an extra software layer that further reduces the chance of leaking sensitive information.

Pricing and deployment cost. A detailed pricing discussion is beyond our scope. Instead, we assume that users pay (in $\$/(core \times hours)$) the same for their Harvest VM minimum size as a standard evictable VM of equal size (evictable VMs

are already heavily discounted compared to regular VMs), and get a further discount on any additional cores beyond the minimum (billing for these cores can be per-use or per-allocation to the Harvest VM). This pricing scheme is beneficial for both users, who can rent resources cheaply, and the provider, who can aggressively monetize its unallocated capacity.

To compute the cost to the user of a deployment of multiple Harvest VMs, we need to consider evictions. An eviction forces the re-creation of the VM at another server, which takes the time to instantiate the VM and restore the application. This results in a loss in useful compute power (measured in $core \times hours$). Thus, the average cost per useful core hour (in $\$/(core \times hours)$) of a deployment is:

$$\frac{minsize\ core\ hrs \times price + additional\ core\ hrs \times \alpha \times price}{minsize\ core\ hrs + additional\ core\ hrs - recovery\ core\ hrs}$$

where *minsize core hrs* is the total core hours for the VMs' minimum size, *additional core hrs* is the total number of cores hours harvested beyond the minimum size, α is the extra discount the provider offers on the additional cores ($\alpha = 0$ means those cores are free and $\alpha = 1$ means they cost the same as the minimum size cores), and *recovery core hrs* is the total amount of core hours spent recovering from evictions.

Harvesting other unallocated resources. Our current implementation only harvests cores. Many workloads can use additional cores (*e.g.*, ML training and most data analytics) with stable needs for other resources. Yet, harvesting other resources would make Harvest VMs more broadly beneficial, so we are building prototypes for harvesting some of them.

Harvesting network and disk bandwidth are similar to core harvesting (they are all compressible resources). Current hypervisors manage bandwidth limits and set them up when starting each VM. To harvest these resources, the server agent can dynamically change the limits. For applications or frameworks to be aware of changes, we expose these values to the Harvest VM using our existing mechanisms.

Harvesting memory is more challenging. Current hypervisors support dynamically changing the memory assigned to a VM. When adding new memory, this shows as hot-plugged memory in the VM. When removing memory, the guest OS uses memory ballooning to make some part of it unavailable. This may trigger swapping in the Harvest VM and the applications/frameworks should be aware. If the VM cannot free up memory, the operation may crash (or ungracefully evict) the VM. Other works discuss similar approaches [33].

For disk space, VMs usually mount a virtual disk (VHD) for data. A naive option would be to extend and shrink the VHD. Extending a VHD can be done while it is mounted, but shrinking it requires unmounting and compressing. Another option would be to add and remove full VHDs depending on the disk space available in the server. Both approaches are intrusive and require applications/frameworks to be aware.

5 Providing SLOs for Harvest VMs

Our characterization showed that the amount of unallocated resources to run Harvest VMs varies over time, in terms of temporal patterns (*e.g.*, daily and weekly) and across-cluster behavior changes (*e.g.*, shift in load across clusters). Moreover, the VM scheduling dynamics produce numerous smaller sets of unallocated resources that survive shorter times, and fewer larger sets that survive longer. These factors make it difficult for users to provision the right minimum size and number of Harvest VMs.

To ease this task, we predict the survival rate of the Harvest VMs and the amount of resources they are likely to receive on average, and provide these predictions to users in the form of an SLO. The SLO is a best-effort statistical estimate as in prior work [12], so the provider should retrain the prediction models frequently (*e.g.*, every day). The provider need not monitor or actively try to enforce each SLO individually, which would be impractical. Nevertheless, the SLO enables applications beyond just batch workloads to use Harvest VMs, as long as they can tolerate the occasional eviction and the core reassignments (Section 6).

Our predictions leverage machine learning (ML) models and features we can collect in production.

User input and SLO definition. The user must first inform her desired number of Harvest VMs (*e.g.*, 100), minimum size (*e.g.*, 2 physical cores), and region. Based on these requirements, we provide an SLO for the survival rate and the number of additional cores for a set of predefined time horizons: 1 hour, 1 day, 1 week, and 1 month. For example, the survival rate SLO for each horizon can be: 60% of the Harvest VMs will likely survive at least 1 hour, 40% will likely survive at least 1 day, 25% will likely survive at least 1 week, and 15% will likely survive at least 1 month. We also provide confidence intervals (*e.g.*, between 55% and 70% will last 1 hour with 95% confidence).

For each horizon, our SLO also estimates the average number of additional cores. For example, the Harvest VMs will likely receive an average of 5-7 cores with 95% confidence for the first hour, 8-11 cores over the first day, etc.

If the SLO is not acceptable, users can change the number and/or minimum size of the Harvest VMs they request. If no SLO is acceptable after multiple tries, users may opt for a mix of regular and Harvest VMs or select a different region. Once the Harvest VMs are running, users can check for updated SLOs, which become more accurate over time. Based on this updated information, they can adapt their deployments.

ML models and features. To provide the SLO, we use ML models to predict the survival rates and average sizes for each time horizon. After experimenting with multiple modeling approaches, we settled on Random Forest regressors [10].

The features we use in our models are as follows.

Cluster characteristics: This includes (1) number of servers, (2) number of racks, (3) generation of the hardware (including

their sizes), and (4) total resources (*e.g.*, cores and memory) in the cluster. Clusters with similar characteristics (*e.g.*, same type of servers) are likely to have similar behaviors. This is useful for new clusters without much historical data or clusters that have not seen particular conditions (*e.g.*, high allocation).
Cluster name: The identifier for the cluster helps improve the prediction accuracy for a specific cluster. This complements the cluster characteristics and still allows learning from the historical data from similar clusters.
Total resources allocated: This includes the total number of cores and memory (*e.g.*, in GBs) currently allocated to regular VMs in the cluster. Together with the cluster characteristics, we can compute the allocation percentage.
Number of VMs: This is the total number of VMs currently running in the cluster. The ratio between resources allocated and the number of VMs gives insights on how large the VMs in the cluster are. This is particularly useful to estimate the sets of unallocated resources in the cluster for Harvest VMs.
Auto Regressive: These are previous time series values of the outputs we want to forecast. This feature is especially useful because, as our characterization shows, past values are a reasonable indicator of the current values. Each output will use values for different past periods. For example, if we are predicting the survival rate for Harvest VMs in 1 day, this would include the evictions we actually saw in the last day.
Moving Average: This is similar to the Auto Regressive feature, but it smooths the past values using averages. We use multiple periods for the averages (*e.g.*, 1 hour, 1 day). This feature is useful to filter out peaks and reduce noise.
ML training and inference. We can *train* our models using data from Harvest VMs that ran in production in the past. This data includes the aspects that we want to predict (*e.g.*, how long the VM lasted for), the characteristics of the Harvest VM (*e.g.*, a minimum of 2 cores), and the state of the cluster at each point in time. However, as Harvest VMs have not run in production long enough, we use traces from production as our training data in this paper (Section 7.1).

At model *inference* time (*i.e.*, an SLO needs to be shown to a user), we first check which cluster in the desired region would potentially host the Harvest VMs that are being requested, and use the cluster characteristics and name for the cluster as input features for the inference. If the Harvest VM deployment is to be split across multiple clusters, we then predict for each one independently.
Discarded features. Other features' impact on prediction quality was small or even detrimental. Some of them are:
Number of VMs of each type: A VM type defines the number of cores, memory size, if it has GPUs, etc. There are hundreds of types and the model cannot make sense of them. Some features we use (*e.g.*, total number of VMs and cores allocated) are proxies and enable our models to infer this data concisely.
Date/time: These features were used in [36]. We do not include them, as features like the total number of VMs already carry implicit temporal patterns (*e.g.*, weekdays vs weekends).

Predicting standard evictable VM survivability. Our survival rate predictions can be directly applied to standard evictable VMs. In fact, we are working on a simpler version of our model to provide survival rate predictions for evictable VMs in production. The uses for these predictions are similar to the ones for Harvest VMs.

6 Harvest Hadoop

Cluster scheduling frameworks, such as Apache YARN [37] or Kubernetes [22], are good targets for Harvest VMs. A large number of applications already run on them, and they can be adapted to use Harvest VMs transparently to applications. These frameworks are built to handle server/VM failures, so they can be easily extended to manage evictions. Applications built for these frameworks, like Spark [44], also manage the straggler tasks that might result from an eviction. Moreover, these frameworks can be modified to schedule more tasks on a Harvest VM that has grown to use more cores, and stop scheduling new tasks on a Harvest VM that has lost cores. To demonstrate how to adapt these frameworks, we build *Harvest Hadoop* to schedule computation on harvested resources.
Harvest Hadoop architecture. Harvest Hadoop is an extension to the Hadoop [3] ecosystem. Hadoop includes the YARN cluster scheduler [37], which enables running many applications (*e.g.*, Spark [44], Flink [11]) to leverage harvested resources. It also includes the Hadoop Distributed File System (HDFS), which is optimized for large data files.

A key goal for Harvest Hadoop was to minimize the number of intrusive changes to YARN and HDFS, so that our system would be simple and practical, and our changes could be more easily contributed to open-source Hadoop. With this in mind, we design Harvest Hadoop with the following main features:
- It executes the YARN and HDFS master processes (called Resource Manager and Name Node, respectively) on regular VMs, as it is expensive to manage the failure of the masters;
- It executes the YARN and HDFS worker processes (called Node Manager and Data Node, respectively) on Harvest VMs;
- It uses storage within each Harvest VM simply as a cache of remote data (from the provider's highly available storage service), as evictions do not leave enough time for fully decommissioning a storage server; and
- It introduces a Harvest VM Manager (HVM Manager) that monitors the number of resources currently available to its Harvest VM and the informs the master processes. The master processes act accordingly at the next heartbeat.

We have contributed all the needed code changes to open-source Hadoop 3.3.0 [40,41]. Figure 9 illustrates the architecture, showing a server that has two regular VMs and a Harvest VM consuming all the resources not used by the regular VMs.
Managing evictions. We leverage the decommissioning feature in YARN [42]. When the provider notifies the Harvest VM that it will be evicted, the HVM Manager notifies the YARN Resource Manager (RM) to kill the containers in that

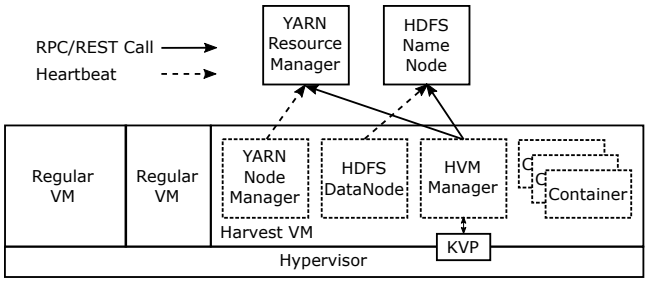

Figure 9: Architecture of Harvest Hadoop.

VM. If the worker gets evicted before killing all the containers, the RM will handle this as a failure and re-schedule the containers. Otherwise, the applications can decide whether they want to re-schedule any of their killed containers.

Evictions are more intrusive for data storage. We leverage the decommissioning mechanisms available in HDFS [20]. The HVM manager catches the eviction notification from the provider and tells the HDFS Name Node to start decommissioning the corresponding Data Node. As the advance notice to evict a VM is usually short (*e.g.*, 30 seconds), there is little time to decommission a full data node. This is the reason why we only cache remote replicas in Harvest VMs, as we mention above. HDFS replicates the cached files in other Harvest VMs, and uses a write-back policy for them [21].

At Harvest Hadoop deployment time, the user specifies (as an auto-scaler option) whether she wants her Harvest VM deployment to be replenished by the cloud platform to its original number of Harvest VMs when an eviction occurs.

Managing core reassignments. To adapt the scheduling, we leverage the resource updates in the existing heartbeats to the YARN RM. Zhang *et al.* took a similar approach in the bare-metal scenario [47]. The HVM manager periodically checks the number of cores assigned to its Harvest VM, and it notifies the RM if the number has changed.

If the Harvest VM gets more cores, the RM can now assign more containers to the VM. If the VM shrinks, the scheduler can: (1) kill some containers and let the application handle it as a failure, (2) run the containers in a deprived mode and wait until the application terminates them, or (3) notify the application to free up some containers.

Our current implementation uses a combination of the three options. The RM first selects the containers that should be killed based on their priorities and whether they are opportunistic [43]. Then, it notifies the applications in case they can terminate the containers. After a grace period (30 seconds), if the cores are still not enough, it will terminate the containers. This period allows graceful termination and can correct for the number of cores increasing again.

Harvesting other resources. We also modify Hadoop to be aware of the VMs' memory allocation, so it will work out-of-the-box when Harvest VMs become capable of harvesting memory. When the Harvest VM gets more memory, Harvest Hadoop can just deploy more containers to it. However, when the Harvest VM shrinks, we cannot run in deprived mode, unless the VM allows swapping to disk. For this reason, we keep a buffer of unused unallocated memory. The HVM manager notifies the Harvest VM when memory from this buffer is allocated, so that it can free up some of its own memory. The HVM manager kills the Harvest VM if the buffer is exhausted, *i.e.* the Harvest VM cannot release memory fast enough.

Hadoop does not need changes to benefit from harvested network and disk bandwidth, as applications can automatically use any additional bandwidth that becomes available.

7 Evaluation

7.1 Methodology

Our evaluation focuses on two sets of results. First, we assess the benefits of Harvest VMs and the accuracy of our SLOs. Ideally, we would do this assessment based on real production data. However, our SLOs are not in production yet. Moreover, the set of conditions under which we can evaluate our SLOs is limited with the production deployments of Harvest VMs. For these reasons, we use a *validated* simulator and production VM data for 25 clusters for our SLO evaluation.

Second, we explore the *real implementation of Harvest VMs and Harvest Hadoop in the provider's production infrastructure*. We use two large clusters: one running internal production VM workloads, and another running VM workloads that stress the cluster.

Simulator. We use Azure's own cluster simulator, which executes the real VM scheduler [19] code in assigning VMs to physical servers. Thus, the simulator closely mimics the constraints and preferences in the real scheduler. We feed the simulator with production VM arrival traces, and add Harvest VMs continuously to fill the cluster (*i.e.*, no new Harvest VMs can be allocated). We run the simulations in real Harvest VMs, *i.e.* each harvested core allows us to run one more (single-threaded) simulation in parallel.

We validate the simulator by comparing the number of Harvest VMs that can be created in a real cluster (based on logs of VM assignments to servers) and in simulation (replaying the corresponding VM arrival trace). Figure 10 shows this validation for Harvest VMs of 3 minimum sizes over 1 week, where dashed lines represent the real executions and solid lines represent simulated executions. The curves match closely because the simulator mimics the packing per server accurately. We also validate using longer periods and other clusters, and find an absolute average error of just 3%.

As inputs for our simulations, we use VM arrival data from 25 randomly selected clusters across 14 regions, including relatively small satellite regions. The data was collected from December 1st 2019 to March 1st 2020. We process the VM arrival data to generate the allocation state of each server every 10 minutes. The clusters exhibit a wide range of behaviors in terms of how highly allocated they are on average and how

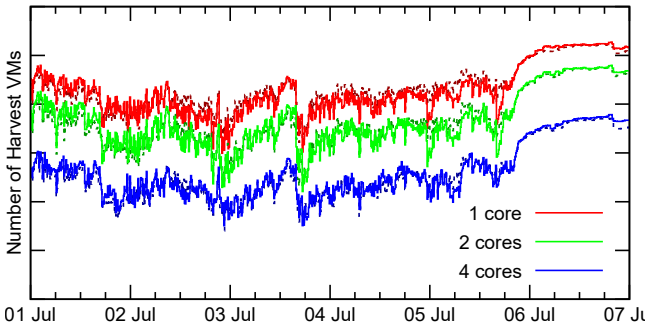

Figure 10: Simulation validation over 1 week. Solid curves are actual data; dashed curves are simulated.

stable the total allocated resources are over time. The number of servers per cluster ranges from several hundred to several thousand. There is no correlation between cluster size and allocation percentage or stability. When training our models, we use data from December 1^{st} 2019 to January 15^{th} 2020. We use the other 45 days for evaluating predictions.

Real experiments. For our real experiments, we use the Harvest Hadoop implementation we describe in Section 6. We configure the provider's deployment system to replace any Harvest VMs that get evicted, so that the overall number of Harvest VMs stays fixed during our experiments.

We use two large clusters, which we call *private* and *canary*. The private cluster has over 1700 servers and runs VMs that implement a production key-value store. The VM load is fairly stable over time. We create Harvest VMs in this cluster and attach them to a Harvest-Hadoop-based production data analytics and ML training system. We have deployed Harvest VMs to other private clusters as well, but selected this one for our results because it has been the most extensively used.

The canary cluster has around 650 servers and runs a synthetic VM load that stresses the provider's production infrastructure. This cluster is in the top percentile in terms of VM creations and terminations, and produces many resource allocation changes and evictions. For our experiments with this cluster, we create full Hadoop clusters. Each cluster consists of 3 Name Nodes and 2 Resource Managers (which run on regular VMs) and Harvest VMs that we scale on demand. For coordination, we also deploy a 5-node ZooKeeper 3.6.0 stamp in the same regular VMs. We run synthetic jobs, including MapReduce (*e.g.*, TeraGen and TeraSort) and Spark.

7.2 Benefits of Harvest VMs

We start the evaluation by assessing the benefits of Harvest VMs over standard evictable VMs in terms of numbers of VMs and evictions. Our comparison simulates the 25 production clusters in two scenarios: one in which we consume all the clusters' unallocated resources using evictable VMs, and another where we consume them using Harvest VMs. We place as many evictable 1-core VMs as will fit; larger sizes would not consume many unallocated resources. In the Harvest VMs scenario, 1 core is the minimum size and we only place one Harvest VM per server. In both scenarios, each VM has 16GB of memory and 200GB of disk.

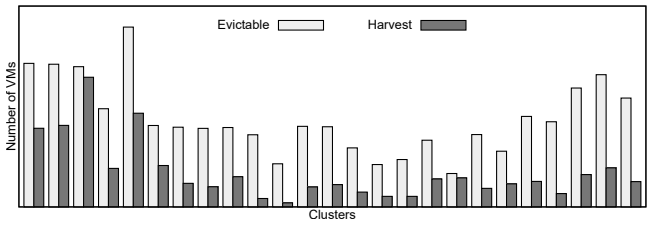

Figure 11: Number of VMs required to consume the unallocated resources of 25 production clusters.

Figure 11 shows the number of VMs required to consume the unallocared resources with evictable and Harvest VMs for each cluster. Across all clusters, we need between 8% and 10.7× (3.7× on average) more evictable VMs than Harvest VMs. The number of evictions of evictable VMs is also much higher by $\sim$ 3.6× on average. These results quantify our earlier observation that standard evictable VMs incur higher management and resource overheads than Harvest VMs.

7.3 Accuracy of SLOs for Harvest VMs

The accuracy of our SLOs hinges on our ability to accurately predict survival rates and average numbers of harvested cores. We start our evaluation with a detailed analysis of prediction accuracy for a few sample clusters, and then offer a global view of all clusters. The last part of the section evaluates our ML model and studies its sensitivity to multiple parameters.

Detailed analysis. Let us first consider the accuracy of our survival rate SLOs. For a cluster with fairly stable load, the graphs on the left of Figure 12 show the number of Harvest VMs with a minimum size of 4 cores that can be created (top), those that would survive 1 day (middle), and their survival rate after 1 day (bottom) over time. Each graph shows the actual and predicted values (with 95% confidence intervals), as well as the corresponding absolute errors. We plot predictions and errors every 10 minutes, given the actual cluster state at those times. For example, if the actual value is 100 and the prediction is between 90 and 120 with 95% confidence, the absolute error is 0%. Instead, if the actual value is 60, the error is -33% (*i.e.*, (60-90)/90) and the absolute error is 33%. The vertical line at January 15^{th} marks the split between the training and test datasets. The graph on the right shows the CDF of the errors comparing the actual survival rates to our predictions with and without 95% confidence intervals, during the test period. These would be the error distributions of our 1-day survival rate SLO.

The top graph shows that our predictions for the number of VMs that can be created are very accurate, even though the training data is almost a flat line and there are substantial variations after January 15^{th}. Our model recognizes that these behaviors are unknown and leverages the data from other

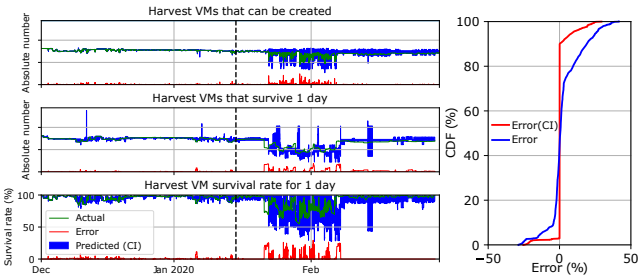

Figure 12: Predictions for Harvest VMs with 4-core minimum size and their survival rates after 1 day for a stable cluster.

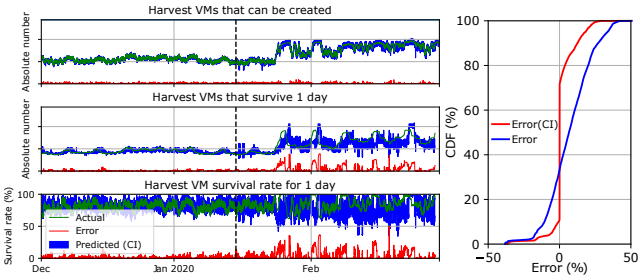

Figure 13: Predictions for Harvest VMs with 4-core minimum size and their survival rates after 1 day for the worst cluster.

clusters to provide a prediction that is not as precise. So, when those variations occur, the confidence intervals widen. The middle graph also shows very good accuracy when predicting the number of VMs that would survive after 1 day. Most importantly, the bottom graph and the CDFs to the right show that 80% of the absolute errors are very close to 0%, and 95% of them are lower than 15%, when compared to the predictions with confidence intervals. Errors are larger when comparing to exact predictions, but 90% of them are still lower than 20%. These results show very good accuracy for our SLO.

For comparison, we now study the cluster with the largest 99th-percentile errors in our dataset in Figure 13. Again, we assume 4-core Harvest VMs and 1 day survival. The top graph shows very low absolute errors, despite the significant change in behavior after January 15th. In contrast, the middle and bottom graphs show significant errors at times, despite the wider confidence intervals. The CDFs show the distribution of the errors in our SLO prediction. In this case, 85% of the predictions are within 10% and 95% within 20%. Thus, even in the worst cluster, our SLO would still provide valuable guidelines (*e.g.*, an actual survival rate of 85% while we predicted 65%).

We now study the accuracy of our predictions of average number of harvested cores. Figure 14 shows these predictions for Harvest VMs with 1 (top), 2 (middle), and 4 (bottom) minimum cores in another cluster with significant unallocated capacity. The horizontal lines show the minimum and maximum numbers of cores for each VM. The figure shows that a Harvest VM with a minimum size of 1 core gets between 6 and 14 cores in the cluster. During the test phase, the prediction accuracy is very good, showing that our average cores SLO would be accurate for this cluster. In addition, we can see

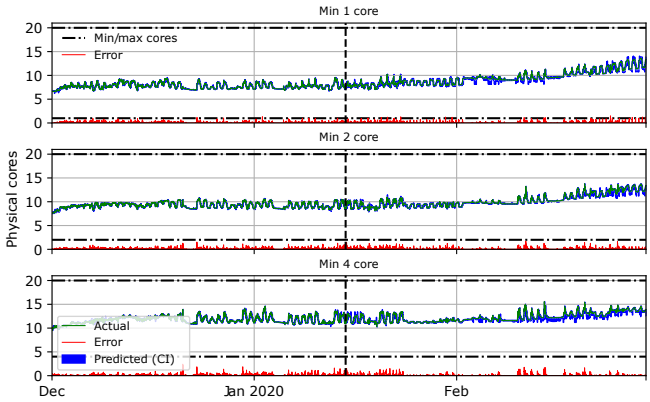

Figure 14: Prediction of unallocated cores in one cluster.

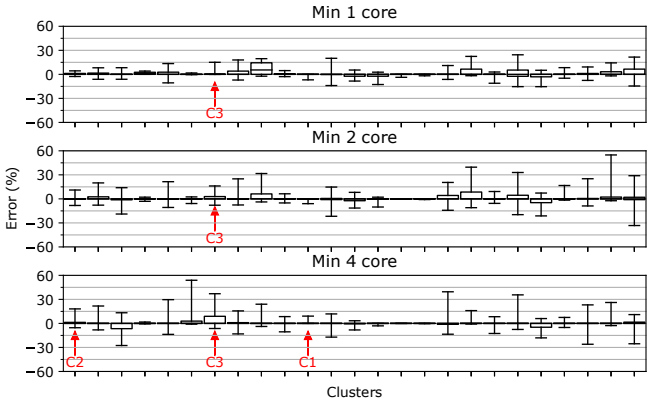

Figure 15: Prediction errors for the 1-day survival rate, as a function of minimum size and cluster.

that the larger Harvest VMs get slightly more cores overall.

Results for all clusters. The results above illustrate the accuracy of our predictions for individual clusters. We now turn to results for all clusters. Figure 15 plots the errors (in boxplot format) when predicting 1-day survival rate with 95% confidence, for each minimum size and cluster. Each box ranges between the first and third error quartiles, with the line representing the mean error, whereas the whiskers extend out to the 2.5th and 97.5th percentiles. The clusters marked C1, C2, and C3 are those from Figures 12, 13, and 14, respectively. The vast majority of mean errors are around 0% and the bulk of the errors are lower than 20% for most clusters.

Figure 16 plots the error distribution of the survival rate without confidence intervals, for each horizon and minimum size, aggregated across all clusters. As expected, short-term predictions (*i.e.*, current, 1-hour) have the lowest errors. The short-term results show that >90% of the predictions have no errors and the worst predictions have an error under 15%. Interestingly, long-term predictions (*i.e.*, 1-month) tend to be more accurate than medium-term ones (*i.e.*, 1-day, 1-week). The reason is that small load changes have a larger impact when predicting medium-term survival, whereas they often get smoothed out in the long term.

The figure also shows that errors are balanced and there

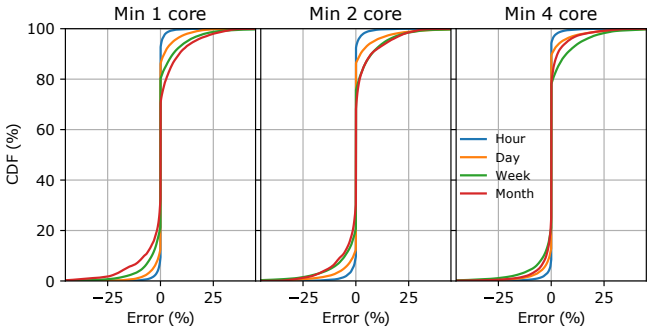

Figure 16: Prediction errors for time horizon and min size.

are not many more overpredictions than underpredictions (or vice-versa). When we average all the absolute errors, both current and 1-hour survival predictions exhibit errors lower than 2%. The 1-day and 1-week average absolute errors are roughly 6%, whereas the 1-month average absolute error is just under 4%.

Our predictions of the average number of cores available to Harvest VMs are even more accurate: the average error is smaller than 2.3% (<0.9% considering the confidence interval). When predicting the median and the 75^{th} percentile numbers of cores, the errors are below 4.1% (<1.5% considering the confidence interval). Even though our current Harvest VM implementation does not harvest memory, targeting our model to predict the memory available for harvesting also produces accurate results: the average absolute error is smaller than 1.5% (<0.5% considering the confidence interval).

Prediction adaptability. During this work, our model has been exposed to three versions of the VM scheduler that changed the allocation behaviors over time. We periodically re-train our model to capture these new behaviors. In addition, the auto-regressive features are especially good at adjusting to such changes. We have also evaluated our predictions with multiple hardware generations (our results are for the two most popular ones) and the model is able to adjust to them.

In summary, our predictions for both survival rate and average number of cores are *accurate and robust* for a wide range of cluster characteristics and behaviors.

Impact of the ML model. For comparison against our Random Forest model [24], we evaluated a Multi-Layer Perceptron (MLP) [35], Gradient Boosting [17], Exponential Smoothing (ETS) [18], and ARIMA [45]. MLP is a type of neural network. It achieves the closest results to our model, but we had to explore multiple combinations in the numbers of layers and neurons per layer. Figure 17 shows a comparison between the prediction errors of the two models' survival rates (left) and number of cores (right). The predictions are slightly worse using MLP and it does not provide confidence intervals. Training times are not a concern for large cloud providers. For context, one Random Forest training session typically takes around 16 hours on one Harvest VM, using 45 days of data from all 25 clusters. A similar MLP training session takes much longer, but we did not try using GPUs for

Figure 17: Avg absolute errors for Random Forest and MLP.

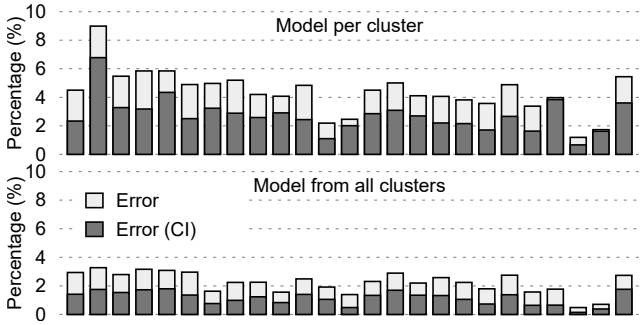

Figure 18: Avg absolute errors using two model types.

it. Prediction times are negligible in both cases.

The other models do not compare as well. Random Forests outperformed Gradient Boosting in both accuracy and performance. ETS and ARIMA are well-behaved for certain clusters and provide confidence intervals. However, they cannot incorporate other features (*e.g.*, cluster characteristics) that improve predictions in unseen situations. In contrast, Random Forests can easily use data from other clusters when the cluster starts to behave differently from its past (*e.g.*, the load increase in Figure 13). Nevertheless, our approach does incorporate the auto-regressive and moving average aspects of ARIMA.

Impact of the features. Our model uses 31 features. Both SHAP analysis [28] and a feature importance algorithm indicate that the auto-regressive features are the most relevant. However, the other features do improve prediction quality, especially when the cluster starts to behave differently. The features that we discarded do not improve our predictions.

Impact of global modeling. We can see the benefit of using data from other clusters by comparing prediction errors from 25 per-cluster models vs a single model trained with data from all 25 clusters. Figure 18 shows this comparison for survival rates. The top graph shows the errors of the independent models and the bottom one the errors for the single model. The bottom graph shows lower errors in every case.

7.4 Harvest VMs and Harvest Hadoop

As we mention in Section 7.1, we experiment with Harvest VMs and Harvest Hadoop in the Azure's production infrastructure, using two clusters called private and canary.

Private cluster. We have been running Harvest VMs and Harvest Hadoop in this cluster in production for more than 6 months. The organic regular-VM workload is a key-value

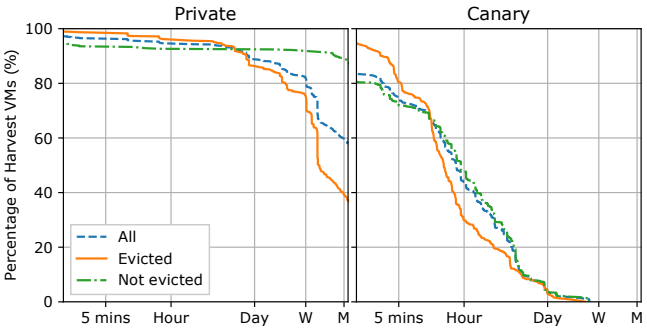

Figure 19: Time to evict Harvest VMs in the two clusters.

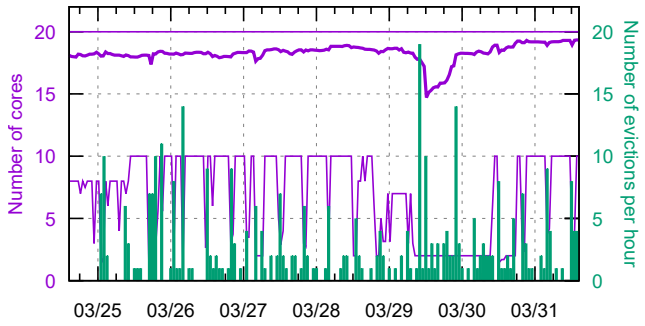

Figure 20: Minimum, maximum, and average number of harvested cores and evictions over one week.

store that uses full-server VMs but leaves many other servers empty. Our Harvest VMs (mininum 2 cores) run on these servers and execute Hadoop-based ML and data analytics jobs. As the Harvest VMs run alone, they grow to consume all the cores on their host servers. The left side of Figure 19 shows their survival statistics over 3 months. The "Not Evicted" VMs are those that terminated, instead of being evicted. Roughly 95% of the Harvest VMs that are evicted survive one hour or more, and roughly 40% survive for one month. These durations match the production jobs nicely: this cluster ran 105k tasks in a month, 90% of them ran for less than 10 minutes, 95% less than 1 hour, and only 1% longer than 6 hours. Interestingly, roughly 90% of the not evicted Harvest VMs last for more than a month. Over time, the organic load has increased and we now have capacity for around 450 Harvest VMs.

Canary cluster. To stress Harvest Hadoop, we create full deployments in the canary cluster, whose organic workload also seeks to stress the platform and varies significantly. Each deployment includes 100 workers in Harvest VMs of minimum size 2, and executes various Hadoop benchmarks. The results over 3 months appear on the right side of Figure 19. The constantly varying organic load and our stress-benchmarking result in only roughly 30% of the Harvest VMs that are evicted surviving one hour or longer. Even the ones that are not evicted are very short, and terminate before they are evicted.

To see these behaviors in more detail, we let a 100-VM setup run for over a week, continuously executing the TeraGen and Terasort benchmarks with 1000 map tasks. From these experiments, Figure 20 shows the minimum, maximum, and average number of cores the Harvest VMs got over the week. Most of the time, the Harvest VMs got over 18 cores on average. On March 29^{th} there was a surge in load and the average dropped to 15 cores. During this time, there was at least one Harvest VM with only 2 cores. The figure also shows the number of evictions per hour. During the load surge, there were 19 evictions but in 30% of the hours there were no evictions and in 75% there were 2 or fewer. After every eviction, the auto-scaler replaced the evicted Harvest VM with a new one trying to keep 100 workers at all times. For most of the VM re-creations, accesses to the remote storage service were not needed to re-hydrate the cache, because the data could come from other cached replicas; the exceptions were the 19 evictions during the load surge, which actually lost all cached replicas of certain files.

7.5 Cost comparisons

To compute the cost savings that Harvest VMs can accrue, compared to standard evictable and regular VMs, we can use the formula from Section 4. To use it, we need to instantiate the time to recover from evictions in core×hours (*recovery core hrs*), the prices per core×hour (*price* and α), and the number of minimum (*minsize core hrs*) and additional (*additional core hrs*) Harvest VM core×hours. The number of cores used by regular and standard evictable VMs as equivalent to the minimum size of Harvest VMs.

We compute the recovery time for each evicted VM from the experiments with the canary cluster. Specifically, an eviction forces the re-creation of the VM at another server, which takes roughly 30 seconds. In addition, Harvest Hadoop needs to re-create its workers, which takes a minimum of 2 minutes and 5 minutes at the 90^{th} percentile. The breakdown for the common case is 1 minute to get all the binaries (*e.g.*, Java, Hadoop, Docker, libraries), 30 seconds to setup and install dependencies, around 10 seconds to setup the environment (including security packages, compliance, and firewall setup), and around 10 seconds to start the services (DataNode and NodeManager) and heartbeat. Some stages are prone to long tails, which usually occur when creating a few hundred Harvest VMs at the same time. For this analysis, we assume that each eviction causes 5.5 minutes of recovery time.

To instantiate the prices, we use the amounts that Azure charges for the VMs from which we derive the Harvest VM resource quantities. Specifically, we instantiate the prices as \$0.126/(core×hour) for a regular VM and \$0.019/(core×hour) for a standard evictable VM [5, 6]. We use the latter price for the minimum size of the Harvest VMs as well. By default, we assume that the discount for additional Harvest VM cores (beyond the minimum size) over the evictable core price is 50%, *i.e.* $\alpha = 0.5$. Below, we discuss other values as well. As Harvest Hadoop can use as many cores as Harvest VMs give it, it is immaterial whether the provider charges for additional cores per-use or per-allocation.

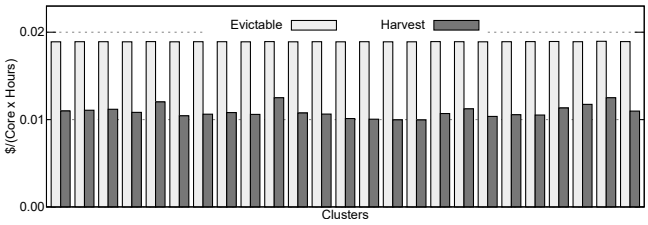

Figure 21: Costs when consuming the unallocated resources of 25 production clusters.

Using data from the simulations in Section 7.2 (where we consume as much of the unallocated resources as possible), we combine the recovery time above with the average survival rates for the 25 clusters and compute an average per-VM recovery overhead of only 0.13%. Again for the 25 clusters, we find that Harvest VMs receive an average of 7.2 additional cores beyond the 1-core minimum.

With these parameters instantiated, we compute the cost per useful core of Harvest VMs in the 25 clusters to range from 34% to 47% cheaper than standard evictable VMs. On average, Harvest VMs are 42% cheaper at \$0.011/(core×hour). Figure 21 illustrates these costs. The fact that evictable VMs suffer many more evictions is a minor factor in these savings, since survival times are much larger than recovery times for both VM classes. Instead, the key reason for the large savings is the additional cores that can be harvested at discounted prices. When those cores are priced the same as the minimum size ($\alpha = 1$), there are almost no savings. When they are free ($\alpha = 0$), Harvest VMs cost \$0.003/(core×hour) on average across the clusters, *i.e.* a savings of 84%.

Compared to filling the unallocated capacity of the 25 clusters with 1-core regular VMs, Harvest VMs are 91% cheaper on average for $\alpha = 0.5$. Here, the heavily discounted nature of evictable cores dominates. Lower prices for the additional cores increase these savings, whereas charging the same price as for evictable/minimum cores lowers the savings to 85%.

8 Lessons from production deployment

Adapting applications and fast adoption. We initially thought that the main users of Harvest VMs would be those who could deploy lots of evictable VMs. However, after discussing with internal teams, we soon realized that their workloads could not benefit from additional cores without modification. This made adoption harder, despite the much lower price of Harvest VMs. Fortunately, many large users at the provider rely on the Hadoop stack, so we devised Harvest Hadoop. These users then immediately and transparently adopted Harvest VMs. We are now starting to adapt a FaaS platform and Kubernetes for Harvest VMs.

Harvesting without evictions. Other potential users were concerned about experiencing frequent evictions. They were the motivation for our SLOs. Still, some would prefer not to have any evictions. For them, we are considering regular-priority Harvest VMs, which still have a (non-evictable) minimum size but can grow. For these VMs, the discount will apply only to the cores used beyond the minimum size.

Unbalanced Harvest VMs. Our current implementation only harvests cores, which may lead to VMs that cannot use some cores because their memory becomes too small. For example, some production VMs harvest 20 cores with only a fixed 16GB of memory. This imbalance is fine for some workloads but not others. Based on this, we started prototyping harvesting of unallocated memory. Another option is to define Harvest VM types with larger (fixed) memories, but that would make it harder to place them. The other resources have not posed imbalance problems so far.

Multiple Harvest VMs per server. Our implementation allows one Harvest VM per server because this works well for our initial (Hadoop) customers. However, to address the imbalance above and enable workloads that have less parallelism per VM, we are implementing the ability for users to specify a maximum size for each Harvest VM, and the fair sharing of a server's unallocated cores across multiple Harvest VMs. Our models easily extrapolate to having multiple of them per server. We need to add the maximum size of each Harvest VM and the number of Harvest VMs in the cluster as features.

New VM family. We initially limited Harvest VMs to a few pre-defined sizes (Section 4). However, some users needed VMs with faster disks or a different hardware generation. So, we had to create new types. For this reason, we plan to make harvesting a feature that can be enabled for most VM types, instead of being a separate family.

Impact on regular VM creation times. Our initial implementation of core reassignments had the unexpected side-effect that regular VM creation could be slowed down significantly on servers that were already hosting many VMs. The problem only became noticeable when we started testing in the canary cluster. Fixing it involved using a different API to the hypervisor and made the overhead negligible.

9 Conclusion

In this paper, we first characterized the unallocated resources of a large cloud provider. We then proposed to dynamically harvest the unallocated resources using Harvest VMs. To provide SLOs for these resources, we built an accurate ML-based predictor for VM survival rates and average number of cores. To demonstrate the use of Harvest VMs, we built a cluster scheduling framework called Harvest Hadoop. Finally, we discussed the lessons and results from our production deployment of Harvest VMs and Harvest Hadoop in Azure.

Acknowledgements

We thank Rebecca Isaacs, our shepherd, the anonymous reviewers, David Irwin, and Stanko Novakovic for their many helpful comments and suggestions.

References

[1] Amazon Elastic Compute Cloud. Amazon EC2 Spot Instances, 2019. https://aws.amazon.com/ec2/spot/.

[2] Pradeep Ambati and David Irwin. Optimizing the Cost of Executing Mixed Interactive and Batch Workloads on Transient VMs. In *SIGMETRICS*, 2019.

[3] Apache. Apache Hadoop, 2020. https://hadoop.apache.org/.

[4] Microsoft Azure. Azure Functions. https://azure.microsoft.com/en-us/services/functions/.

[5] Microsoft Azure. Ev3 and Esv3-series. https://docs.microsoft.com/en-us/azure/virtual-machines/ev3-esv3-series.

[6] Microsoft Azure. Pricing Calculator. https://azure.microsoft.com/en-us/pricing/calculator/.

[7] Microsoft Azure. Introducing B-Series, Our New Burstable VM Size, 2019. https://azure.microsoft.com/en-us/blog/introducing-b-series-our-new-burstable-vm-size/.

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

[9] O. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing Amazon EC2 Spot Instance Pricing. *ACM Transactions on Economics and Computation (TEAC)*, 1(3), 2013.

[10] Leo Breiman. Random Forests. *Machine learning*, 45(1):5–32, 2001.

[11] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache Flink: Stream and Batch Processing in a Single Engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

[12] Marcus Carvalho, Walfredo Cirne, Francisco Brasileiro, and John Wilkes. Long-Term SLOs for Reclaimed Cloud Computing Resources. In *SoCC*, 2014.

[13] Amazon Elastic Compute Cloud. Burstable Performance Instances, 2019. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-performance-instances.html.

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

[15] Intel Corp. Intel® CAT. https://software.intel.com/content/www/us/en/develop/articles/introduction-to-cache-allocation-technology.html.

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

[17] Jerome H Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics*, pages 1189–1232, 2001.

[18] Everette S Gardner Jr. Exponential Smoothing: The State of the Art—Part II. *International Journal of Forecasting*, 22(4):637–666, 2006.

[19] 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.

[20] Apache Hadoop HDFS. HDFS DataNode Admin Guide, 2020. https://hadoop.apache.org/docs/current3/hadoop-project-dist/hadoop-hdfs/HdfsDataNodeAdminGuide.html.

[21] Apache Hadoop HDFS. HDFS Provided Storage, 2020. https://hadoop.apache.org/docs/current3/hadoop-project-dist/hadoop-hdfs/HdfsProvidedStorage.html.

[22] Kubernetes. Production-Grade Container Orchestration, 2020. https://kubernetes.io/.

[23] Jacob Leverich and Christos Kozyrakis. Reconciling High Server Utilization and Sub-Millisecond Quality-of-Service. In *EuroSys*, 2014.

[24] Andy Liaw, Matthew Wiener, et al. Classification and Regression by Random Forest. *R News*, 2(3):18–22, 2002.

[25] Heshan Lin, Xiaosong Ma, Jeremy Archuleta, Wu-Chun Feng, Mark Gardner, and Zhe Zhang. MOON: MapReduce On Opportunistic eNvironments. In *HPDC*, 2010.

[26] Michael J Litzkow, Miron Livny, and Matt W Mutka. Condor-A Hunter of Idle Workstations. In *ICDCS*, 1988.

[27] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. Heracles: Improving Resource Efficiency at Scale. In *ISCA*, 2015.

[28] Scott M Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions. In *NIPS*, 2017.

[29] Microsoft. Hyper-V Technology Overview, 2016. https://docs.microsoft.com/en-us/windows-server/virtualization/hyper-v/hyper-v-technology-overview.

[30] Microsoft. Hyper-V Integration Services, 2019. https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/integration-services.

[31] X. Ouyang, D. Irwin, and P. Shenoy. SpotLight: An Information Service for the Cloud. In *ICDCS*, 2016.

[32] Amazon Web Services. AWS Lambda. https://aws.amazon.com/lambda/.

[33] Prateek Sharma, Ahmed Ali-Edlin, and Prashant Shenoy. Resource Deflation: A New Approach For Transient Resource Reclamation. In *EuroSys*, 2019.

[34] Supreeth Shastri, Amr Rizk, and David Irwin. Transient Guarantees: Maximizing the Value of Idle Cloud Capacity. In *SuperComputing*, 2016.

[35] Bruce W Suter. The Multilayer Perceptron as an Approximation to a Bayes Optimal Discriminant Function. *IEEE Transactions on Neural Networks*, 1(4):291, 1990.

[36] Sean J Taylor and Benjamin Letham. Forecasting at Scale. *The American Statistician*, 72(1):37–45, 2018.

[37] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache Hadoop YARN: Yet Another Resource Negotiator. In *SoCC*, 2013.

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

[39] Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. Bubble-flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers. In *ISCA*, 2013.

[40] Apache Hadoop YARN. Dynamic Resource Configuration. https://issues.apache.org/jira/browse/YARN-999.

[41] Apache Hadoop YARN. In case of long running tasks, reduce node resource should balloon out resource quickly by calling preemption API and suspending running task. https://issues.apache.org/jira/browse/YARN-999.

[42] Apache Hadoop YARN. Graceful Decommission of YARN Nodes, 2020. https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/GracefulDecommission.html.

[43] Apache Hadoop YARN. Opportunistic Containers, 2020. https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/OpportunisticContainers.html.

[44] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster Computing with Working Sets. In *HotCloud*, 2010.

[45] G Peter Zhang. Time Series Forecasting Using a Hybrid ARIMA and Neural Network Model. *Neurocomputing*, 50:159–175, 2003.

[46] Xiao Zhang, Eric Tune, Robert Hagmann, Rohit Jnagal, Vrigo Gokhale, and John Wilkes. CPI2: CPU Performance Isolation for Shared Compute Clusters. In *EuroSys*, 2013.

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