SLACKVM: Packing Virtual Machines in Oversubscribed Cloud Infrastructures

Pierre Jacquet, Thomas Ledoux, Romain Rouvoy

SLACKVM: Packing Virtual Machines in Oversubscribed Cloud Infrastructures

Pierre JACQUET
Inria, Univ. Lille, CNRS,
UMR 9189 CRIStAL, France
pierre.a.jacquet@inria.fr

Thomas LEDOUX
IMT Atlantique, Inria, CNRS,
UMR 6004 LS2N, France
thomas.ledoux@inria.fr

Romain ROUVOY
Univ. Lille, Inria, CNRS,
UMR 9189 CRIStAL, France
romain.rouvoy@inria.fr

***Abstract*—Cloud providers generally expose a large catalog of *Virtual Machine*s (VMs) offers, some being categorized as *premium*—guaranteeing dedicated resources—and others being hosted in oversubscribed environments, where virtual resources can exceed the physical capabilities of *Physical Machine*s (PMs). The latter strategy is often employed to increase platform utilization, as hosted VMs are unlikely to fully utilize all their allocated resources simultaneously [1]. However, managing multiple oversubscribed VM levels introduces an additional layer of complexity for Cloud providers, often leading them to provision isolated clusters of PMs for each category of offers.**

In this paper, we introduce SLACKVM, a novel Cloud shared architecture wherein VMs from various oversubscription levels coexist on the same cluster of PMs. In particular, we demonstrate that oversubscription levels can be complementary, meaning they do not saturate the same resource components. By leveraging this complementarity, Cloud providers can couple multiple levels to better consolidate VMs offers onto PMs, and reduce the size of their clusters by up to 9.6%. This resource savings results in both an operational cost reduction and a reduced ecological footprint for Cloud infrastructures, with a limited impact on the *Quality of Service* (QoS).

I. INTRODUCTION

Over the past decade, there have been notable improvements in the power efficiency of *Data Center*s (DCs). However, following the Jevons paradox [2], the increasing demand for data processing and storage has risen at an even greater rate, resulting in a continued upward trajectory in power consumption [3].

As some DCs are close to their maximum efficiency in terms of *Power Usage Efficiency* (PUE) [4], research has been shifting from infrastructure-level optimizations to software-level optimizations. A key area of concern for Cloud providers remains the low resource usage per *Physical Machine* (PM) [5], [6], [7]. In particular, consolidating *Virtual Machine* (VM) workloads onto fewer PMs can significantly reduce the carbon footprint of DCs, lower their power consumption, and decrease their operational costs.

While improving resource usage is an active topic in the system community, the latest proposals only rely on the introduction of new kinds of workloads to "fill the gaps" of resources left by VMs, instead of reducing the existing cluster size, which may be associated to a rebound effect. Typically, unallocated PM resources may be leveraged by *spot* VM [8], [9], [10], while resources unused by tenants may be used by *harvest* VM [11]. *Disaggregated* VMs were also recently proposed to leverage resource fragmentation [12], [13]. While these proposals improve the resource usage of a server, they also share the particularity to be only suitable for a given type of workload [14], as they lack guarantees on availability or performance. In this paper, our approach does not aim to "fill the gaps" with alternative workloads. Instead, it advocates for a method to prevent the occurrence of gaps in the first instance.

We consider this objective more convenient for a Cloud provider, as it maintains the use of long-living generic VMs instead of introducing highly specific ones. In other words, we aim to enhance PM packing with a focus on long-term services rather than small, ephemeral tasks (*e.g.*, FaaS computing [15]). It avoids reliance on a complex equilibrium notion between infrastructure gaps and client demands [16]. Furthermore, this approach effectively reduces the *Infrastructure-as-a-Service* (IAAS) PM cluster size, contributing to an improvement in the carbon footprint of ICT, which is known to grow at a fast pace [3].

Oversubscription (also known as *overcommitment*)[1] is commonly adopted to increase resource usage. This paper demonstrates that oversubscription can also be harnessed to reduce unallocated resources, further contributing to the efficiency of DC operations.

Although Cloud providers are used to managing VMs oversubscribed at different levels, these are currently hosted by distinct clusters, with each PM adhering to at most a single oversubscription ratio. Since CPU and memory oversubscription levels do not occur in the same order of magnitude, we demonstrate that different oversubscription levels may saturate different types of physical resources. By combining different oversubscription levels, we, therefore, illustrate the potential to leverage their complementary nature, thereby reducing the number of PMs required to handle a IAAS workload.

Our contribution, named SLACKVM, comprises a local agent, demonstrating how resources can be segregated among distinct groups of VMs, and a novel metric to improve Cloud score-based schedulers with complementary packing considerations. We assess our approach on both a physical platform and a simulator, evaluating performance and noticing

[1] a strategy commonly employed by Cloud providers to rent more virtual resources than physically available—assuming that customers do not simultaneously utilize all the resources allocated to them.

significant gains at scale. Specifically, our results demonstrate that, by appropriately combining oversubscription levels, Cloud providers can save up to 9.6% in terms of the number of required PMs, which is a noticeable gain at the scale of a Cloud provider.

In the remainder of this paper, we report on related work (Section II), explain why oversubscription co-hosting is important (Section III), present an overview of SLACKVM architecture (Section IV), before diving into its design (Section V & Section VI). We evaluate our approach (Section VII) before concluding on this work (Section VIII).

II. Related work

The state of the art related to our contribution covers both contributions in the areas of *resource oversubscription* (Section II-A) and *VM scheduling* (Section II-B).

A. Resource oversubscription

Most hypervisors enable oversubscription by allowing the sum of all allocated virtual resources to exceed the PM capabilities [17], [18], [19]. An overload situation occurs when a VM workload effectively requires more resources than available from the PM. On a CPU overload, VMs cannot use their allocated time slices, which may lead to *Service-Level Agreement* (SLA) violations. To avoid such performance degradation, oversubscription is usually limited at a certain level, quantified as a ratio between the number of exposed virtual resources and the available physical resources. For example, a CPU oversubscription of 2:1 refers to the availability of 2 *virtual CPU*s (vCPUs) per physical core.

Some cluster managers, such as OPENSTACK [20] and Borg [21], can limit the oversubscription level using a single static value for the whole cluster of PMs. The oversubscription level can also be defined per PM. This ratio can be static, by taking into account individual PMs performance [22], or dynamic, relying on the effective usage of computing resources. In that case, oversubscription level computation depends on a predicted peak usage [1], [23], as new deployments must be performed on resources seen as available in the long run. Peak prediction may be computed using VM percentile [24] or standard deviation [1].

Consequently, each PM maintains a single oversubscription level, which is determined by the cluster configuration or the PM configuration. Although this PM ratio can be dynamically adjusted over time in some approaches, it still represents a single—albeit dynamic—level. However, Cloud providers commonly offer a range of VMs with varying performance guarantees, depending on their pricing tiers. This results in the support of multiple oversubscription levels. As PMs support a single oversubscription level, this requires provisioning dedicated clusters for each specific offer. We argue that this approach is impractical and misses packing opportunities between complementary workloads. In our approach, we instead improve oversubscription granularity beyond a single server by introducing multiple oversubscription levels in a single PM, using segregated resource pools.

B. VM scheduling

The selection of an appropriate PM for a given VM deployment is often framed as a *Vector Bin Packing Problem* [25]. In this problem, a set of VMs with known resource requirements must be allocated to PMs with known resource capacity. The objective is to fulfill the VMs demands while minimizing the number of required PMs. Bin packing problems are NP-hard [26] and, over the years, numerous heuristics have been proposed to tackle VM scheduling cases [27].

Cloud schedulers, such as OPENSTACK Nova (containers and VMs) [28], Azure Protean (VMs) [29], Borg (Linux containers) and Kubernetes (containers) [21], follow a similar deployment process. The PM selection involves filtering candidates from the cluster based on hard constraints (that must be respected), such as resource availability for the deployment request. Additionally, candidates are scored based on soft constraints (that should be respected if possible), including resource usage considerations like CPU, RAM, and I/O. Final selection is made upon this score [28], [29]. In this paper, we focus on extending this score-based selection by introducing a new metric capturing the complementarity of deployment within already hosted VMs in a PM. This metric is introduced by the concept of hybrid oversubscription levels on a single PM.

III. Cloud resource balance

In this section, we discuss how oversubscription levels impact the packing of underlying PMs. We start by describing the VM resources allocation (Section III-A) and then, we compare it to PM configurations to identify bottlenecks (Section III-B).

A. Cloud allocations

VM configurations commonly adhere to the convention of proposing power-of-2 values. Among the prevalent CPU configurations for VMs are those with 1 vCPU, 2 vCPUs, and 4 vCPUs [30]. While hypervisor constraints do not inherently preclude the proposition of intermediate sizes, this practice is commonly adopted to facilitate VM packing [31]. Essentially, when focusing solely on the CPU dimension, this approach facilitates the efficient packing of PM, enabling the use of strategies, such as *First-Fit* scheduling.

In practice, achieving perfect allocation on a PM—i.e., having all its resources utilized to 100%—is unlikely. VMs allocation hosted on a given PM are often either CPU-bound, resulting in underutilized memory, or memory-bound, leading to underutilized CPU [32], [33]. Other types of resources, such as networks, are less likely to limit deployments [34] and are not considered in this paper.

To further dive into this issue, we analyzed the VM size distribution reported in [30] to compute the average VM size for both Azure and OVHcloud infrastructures (cf. Table I). This first illustrates than allocations can significantly differ between Cloud providers.

However, in an oversubscribed environment, resources allocation may deviate from the deployment request. We are now interested in identifying which server resource is depleted

TABLE I
AVERAGE VCPU & VRAM REQUESTS PER VM ($\overline{vCPU}$ & $\overline{vRAM}$)

Dataset	$\overline{vCPU}$	$\overline{vRAM}$
Microsoft Azure	2.25 vCPUs per VM	4.8 GB per VM
OVHcloud	3.24 vCPUs per VM	10.05 GB per VM

first under different oversubscription policies. To achieve this, we can compare the *Memory per Core* (M/C) ratio of hosted VMs to the PM configurations [35]. When these ratios do not align, one resource will typically be exhausted before the other, resulting in stranded resources. We begin by reporting on the M/C ratio of allocated resources.

Without oversubscription (1:1), we directly compute the M/C ratio from Table I, by dividing the average VM memory quantity by the average VM CPU request as, in this context, only one vCPU is proposed per physical core.

The 2:1 oversubscription level refers to the allocation of 2 vCPUs per CPU core. This allocation scheme reduces physical CPUs being allocated, while maintaining a similar amount of DRAM. Consequently, the M/C ratio is increased.

Table II reports on the M/C ratio per Cloud provider, across three different levels of CPU oversubscription.

TABLE II
M/C RATIO OF OVERSUBSCRIBED VMS (IN PROVISIONED GB/CORE)

Oversubscription levels	1:1	2:1	3:1
Microsoft Azure	2.1	3.0	4.5
OVHcloud	3.1	3.9	5.8

For oversubscribed environments, computations were conducted under two hypotheses. First, the catalog size was assumed to be more limited. For example, OVHcloud does not offer oversubscribed VMs with a capacity exceeding 8 GB. In our estimations of the M/C ratio for oversubscribed VMs, the average vCPU and vRAM deployments sizes were recomputed from the VM size distributions where VM having more than 8 GB were excluded. While this approximation may not be perfectly accurate, we contend that it is sufficient for identifying overarching trends.

Second, memory was not oversubscribed. In practice, some providers may opt to oversubscribe DRAM to a limited extent compared to what can be achieved with CPU oversubscription [36], [37], resulting in similar variations in the M/C ratio.[2]

B. Cloud resources collapse differently

While IAAS workload M/C ratio may evolve [35], each server of the cluster reports on a fixed M/C ratio obtained from its hardware configuration. For example, a PM with 64 cores and 256 GB of RAM will expose a static M/C ratio of 4 GB per core. We refer to this hardware ratio as its *target ratio*, since aligning hosted VMs allocation to this ratio would lead to an optimal allocation of hardware resources.

[2]For instance, OPENSTACK's default oversubscription ratios are 16:1 for CPU and 1.5:1 for DRAM [20]

a) Identifying the limiting factor: Comparing both VMs and PMs ratios, therefore, serves as a method to identify Cloud bottlenecks. When the M/C ratio of hosted VMs is higher than the PM, a host will face memory limitations, resulting in wasted CPU capacity. Conversely, if the M/C ratio of VMs is lower than the PM, a host will primarily saturate its CPU resources, leading to underutilized memory capacity.

With PMs operating at a M/C ratio of 2 GB per core, all the workloads outlined in Table II experience memory saturation, as the minimal VMs ratio is higher (2.1 GB on Azure 1:1 level).

Nonetheless, we contend that a 4 GB per core ratio is a more accurate representation of the PMs provisioned by Cloud providers. In this scenario, typical bounds for the Azure dataset are estimated as follows:

- 1:1 is highly *CPU*-bounded ($2.1 < 4$),
- 2:1 is *CPU*-bounded ($3.0 < 4$),
- 3:1 is slightly *memory*-bounded ($4.5 > 4$).

However, in the context of OVHcloud, which typically involves larger deployments, biases are different:

- 1:1 is slightly *CPU*-bounded ($3.1 < 4$),
- 2:1 is balanced ($3.9 \approx 4$),
- 3:1 is highly *memory*-bounded ($5.8 > 4$).

b) Resolving the limiting factor: In this context, improving VM packing can be achieved in different manners.

Only proposing VMs respecting a given M/C ratio cannot be optimal, as customers may prefer CPU- or memory-intensive workloads based on their requirements.

Determining the optimal oversubscription level to tune the hosted M/C ratio can be an objective, but it is worth noting that estimating this optimal level may also be unrealistic. This is because non-oversubscribed VMs continue to be offered by Cloud providers, as a significant share of their customers favor performance over resource efficiency in their Cloud deployments.

Another objective to consider is the adjustment of hardware configurations to closely match the workload ratio demands. However, achieving such an alignment is also unrealistic, due to the associated costs for Cloud providers. In practice, Cloud providers typically employ heterogeneous hardware, occasionally prioritizing the extension of a PM lifespan rather than consistently refreshing all the configurations at a fixed pace.

Therefore, our research is focused on fine-tuning the hosted VMs M/C ratio, by co-locating multiple oversubscription levels, to approximate the PM's specific resource ratio. It leverages the synergy between workloads that exhibit diverse resource requirements, such as the combination of a CPU-bound workload, which is typically encountered in a low oversubscribed environment, with a memory-bound workload, commonly observed in highly oversubscribed environments. By packing VMs from multiple oversubscription levels, it becomes possible to effectively "avoid the gaps"—hence maximizing the utilization of PMs while reducing the number of PM required to host a given workload.

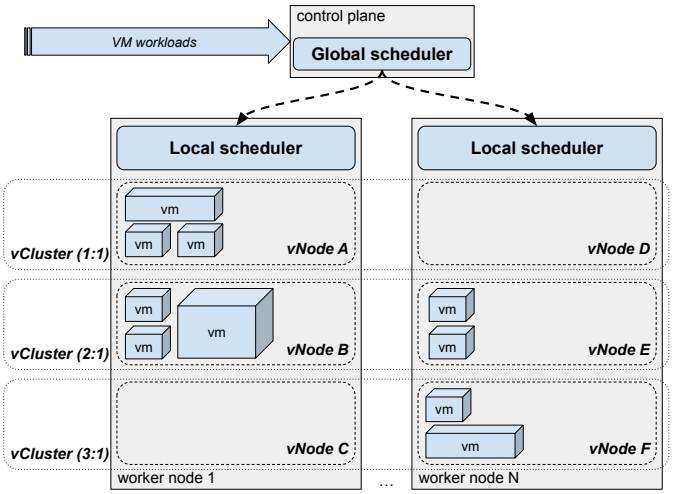

Fig. 1. Overview of SLACKVM partitioning

IV. SLACKVM OVERVIEW

A Cloud scheduler architecture consists of two key components [27], [38]. The first one is a *global scheduler*, hosted by the Cloud control plane, which handles incoming VM deployment requests and selects the most suitable PM for deployment. It typically achieves this by communicating with an agent located on each worker node—the PM—referred to as the *local scheduler*, to gather information about the PMs' current state.

Once a PM is selected as the target, the VM deployment request is forwarded to the *local scheduler*. The *local scheduler* generally assumes responsibility for tasks, such as creating a disk image, invoking the hypervisor to initiate the VM, and, in some cases, determining how resources are allocated among the VMs, possibly utilizing features like *cgroups* for resource management.

We propose to extend the capabilities of state-of-the-art Cloud schedulers by enhancing their *local scheduler* functionality to manage different oversubscription levels. Under SLACKVM architecture, the *local scheduler* segregates a PM's resources into *vNodes*, where each *vNode* represents a group of exclusive physical resources. As depicted in Figure 1, each oversubscription level on a single PM utilizes a separate *vNode*. The collection of *vNodes* referring to the same oversubscription level is referred to as a *vCluster*.

In our context, a VM is deployed on a vNode rather than an entire PM. This vNode represents a smaller share of a PM's resources. Interestingly, in SLACKVM, the size of a vNode is dynamically adjusted upon a VM deployment, depending on the resource request and its oversubscription level. We delve into how our local scheduler manages vNodes in Section V.

The selection of the most appropriate vNode inside a *vCluster* is performed by the *global scheduler*, which may leverage this context to improve PM packing. We explore the adaptation of Cloud scheduling for *vClusters* in Section VI.

V. LOCAL SCHEDULER

A traditional *local scheduler* is tasked with overseeing the management of an individual PM within the system. Its responsibilities encompass responding to requests from the *global scheduler* regarding the PM state, which includes resource utilization, the number of hosted VMs, and other relevant parameters. Additionally, the *local scheduler* is responsible for coordinating the deployment and removal of VMs on the PM by translating these actions into hypervisor-related operations. Finally, it plays a critical role during VMs lifetime by determining how resources are shared.

In contrast to other implementations, our *local scheduler* employs a resource partitioning approach, where resources are segregated into distinct resource partitions referred to as vNode. Each vNode can host a set of VMs at a given oversubscription level. Consequently, in our context, hosting a VM within a vNode entails allocating and pinning it to the resources managed by that vNode.

Determining the optimal distribution of vNodes, along with their respective sizes can pose a complex challenge, as the variability in VM offerings over time and across different Cloud providers can be substantial. Instead of computing a static distribution, we prefer to harness the dynamic capabilities inherent to our vNodes. The size of a specific vNode is dynamically adjusted, based on the arrival and departure of hosted VMs.

Deploying a VM on a vNode is achieved by adjusting the vNode's size allocation to meet the new requirements. This involves first selecting the appropriate cores to add to the existing resource collection (as discussed in Section V-A), before extending the pinning of all hosted VMs in that vNode to the new range. Conversely, when a VM departs, it may free up resources from the existing allocation. We also discuss requirements in workload variability in Section V-B

A. Topology-driven resizing of vNodes

Modern PM processor topologies can be intricate. Cores within a processor may lack common cache levels with other cores due to segmented last-level cache (as observed in EPYC architectures) or the presence of multiple sockets on the PM. SLACKVM allocates vNodes to report on a configuration that resembles a CPU model with fewer cores. This is done both to improve isolation and to leverage existing Linux OS scheduling mechanisms effectively.

Favouring resource isolation: Since each vNode is associated with a distinct oversubscription level, they must be isolated. At the CPU level, this is achieved by avoiding the sharing of low-cache levels between vNodes.

Ideally, we allocate each vNode to a separate physical socket, as this provides the best isolation on the same PM [39]. However, when the number of vNodes exceeds the number of sockets, or when the size of a vNode exceeds a single socket, multiple vNodes must be hosted on the same socket. In such cases, we carefully examine cache levels being shared between cores. In a setting with n cache levels, we first attempt to guarantee isolation between cores at the n^{th} level. If not

feasible, we proceed to the $(n-1)^{th}$ level and so on, until reaching $n = 1$.

Leveraging Linux scheduler: Scheduling of processes to physical cores relies on *Earliest Eligible Virtual Deadline First* (EEVDF), the Linux scheduler. This task is complex and benefits from ongoing development by the Linux community and processor vendors through dedicated drivers. Although SLACKVM restricts the usage of some processes to a limited range of cores, it does not go beyond this constraint. We only consider resources through collections of physical cores. The responsibility of selecting the most suitable core from the specified vNode is left to the standard Linux scheduler, hence benefiting from state-of-the-art scheduling optimizations.

Cores that belong to the same vNode have typically a low level of cache in common, which mirrors a traditional CPU topology. Therefore, if the PM implements an asymmetric load mechanism, such as *Intel Turbo Boost Max Technology* (ITMT), specifically designed to handle this type of topology, it will interact in synergy with our core pinning strategy.

Exposing a virtual topology: We introduce a core distance metric extending the *Non-Uniform Memory Access* (NUMA) distance [40]. This extended metric incorporates an assessment of the shared cache levels to provide a more complete evaluation of core proximity. Linux system exposes an ID for each core to identify the cache zone. We collect this information and we compute distances between each core, as described in Algorithm 1. While the incremental value is arbitrary (line 6), we chose it to be in the same order of magnitude as the current NUMA distance notion [40].

Algorithm 1 Distance computation between two cores

Input: $core_0$, $core_1$, $height$
Output: $distance$
1: $distance \leftarrow 0$
2: **for all** $level \in 0..height$ **do**
3: **if** CACHE$(level, core_0)$ == CACHE$(level, core_1)$ **then**
4: **return** $distance$
5: **end if**
6: $distance \leftarrow distance + 10$
7: **end for**
8: **return** $distance +$ NUMA-DISTANCE$(core_0, core_1)$

The computed distance between cores is what allows our local scheduler to be generic when pinning cores on heterogeneous physical machines. When extending a vNode, we choose additional cores that are closest in terms of cache level to the current allocation of the vNode, enabling a gradual integration of sibling cores. Conversely, when creating a vNode, initial cores are selected from the farthest ones compared to existing vNodes.

While frequent changes in the pinning strategy may lead to decreased performance due to more context switches, it is important to note that these changes occur only when a VM is being deployed or destroyed. Such events do not happen at a significant frequency in the realm of CPU operations.

B. Leveraging workloads diversity in vNodes

In a conventional cluster, a PM only becomes oversubscribed when the quantity of allocated virtual resources exceeds the PM's configuration. This occurs independently of the oversubscription level, which serves as an upper limit that may not always be reached. Given that this approach introduces diversity among VMs, leading to variations in CPU utilization before they directly compete for time-slices, it becomes essential to provide mitigation strategies when oversubscribing a smaller set of VMs within a vNode.

In a n:1 oversubscription scenario, a Cloud provider guarantees that no more than n vCPUs can contend for a single physical core. Given that the EEVDF mechanism equitably shares CPU time-slices among processes, a straightforward approach is to allocate only VMs with the same premium level policy to the same set of resources.

However, it is possible to allocate different oversubscription levels of VMs to the same set of resources—i.e., vNode—provided that they adhere to the conditions imposed by the lowest oversubscription level within the VM set. In simpler terms, a VM with a 2:1 oversubscription level may coexist with VM belonging to a 3:1 oversubscription level, if and only if the set of physical resources still complies with the 2:1 ratio (as the "no more than 2 vCPUs per physical core" condition satisfies the "no more than 3 vCPUs per physical core" condition).

While this approach increases the allocated resources, as the 3:1 overcommitted VM is "upgraded", it may be strategically employed to enhance workload heterogeneity, temporarily. Alternatively, remediation mechanisms, like those involving *cgroups* are feasible, but they may be considered at odds with the oversubscription principle, which aims to distribute the pool of resources equally among all consumers.

Hence, our strategy relies on the pooling of oversubscribed vNodes when feasible, effectively leveraging all resources that remain unallocated by the non-oversubscribed vNode on the same PM to enhance workload heterogeneity.

VI. GLOBAL SCHEDULER INCENTIVE

Instead of proposing a new IAAS scheduler, we focus in this section on how packing can be improved by extending current scheduler mechanisms. The PM selection process is contingent upon the predefined objectives set forth by Cloud providers. Objectives are treated by control planes using a scoring system to pick the most appropriate PM for a given VM deployment [21], [41]. We introduce a new metric in existing scoring mechanisms to enhance VMs' complementarity. This new metric leverages our unique context, where a PM can be oversubscribed simultaneously to multiple levels, to improve server packing.

A vCluster is an abstraction of a set of vNodes of a given oversubscription objective. It behaves similarly to a traditional cluster of PMs: receiving a VM deployment request, interrogating its pool of candidate hosts, and selecting the most appropriate one. The difference comes from the dynamic capabilities of its hosts—i.e., the *vNodes*. We deploy VMs on

vNodes while trying to maintain the M/C ratio of the set of hosted VMs close to the M/C ratio of the hosting server.

Algorithm 2 Progress towards *target ratio* computation

Input: $configPM$, $allocPM$, vm
Output: $progress$
1: $targetRatio \leftarrow \frac{\text{CONFIGPM}(mem)}{\text{CONFIGPM}(cpu)}$
2: **if** $allocPM(cpu) > 0$ **then**
3: $\quad currentRatio \leftarrow \frac{\text{ALLOCPM}(mem)}{\text{ALLOCPM}(cpu)}$
4: $\quad nextRatio \leftarrow \frac{\text{ALLOCPM}(mem)+\text{VM}(mem)}{\text{ALLOCPM}(cpu)+\text{VM}(cpu)}$
5: **else**
6: $\quad currentRatio \leftarrow targetRatio$
7: $\quad nextRatio \leftarrow \frac{\text{VM}(mem)}{\text{VM}(cpu)}$
8: **end if**
9: $current\Delta \leftarrow |currentRatio - targetRatio|$
10: $next\Delta \leftarrow |nextRatio - targetRatio|$
11: $progress \leftarrow current\Delta - next\Delta$
12: **if** $progress < 0$ **then**
13: $\quad factor \leftarrow 1 + \frac{\text{ALLOCPM}(cpu)}{\text{CONFIGPM}(cpu)}$
14: $\quad progress \leftarrow progress \times factor$
15: **end if**
16: **return** $progress$

A PM has an inherent constant M/C ratio due to its configuration, but the M/C ratio associated with its workload is subject to dynamic variations. From an intuitive standpoint, when allocated VMs emphasizes CPU allocation compared to their PM configuration, it becomes desirable to prioritize memory-intensive deployments on that PM. This approach aims at preventing resource saturation before fully allocating all the available dimensions. Our methodology is rooted in this consideration.

Algorithm 2 is, therefore, designed to compute a progress indicator aimed at assessing whether a PM would move closer to its target M/C ratio if a candidate VM were to be deployed on it. To achieve this, the algorithm first calculates the *target ratio*, based on the PM configuration (line 1) and, then, compares it with two distinct workload ratios. The first one is derived from the PM current set of VMs (line 3), while the second one considers the potential addition of the candidate VM (line 4). Subsequently, both of these workload ratios are compared to the optimal resource ratio (lines 9–10). The algorithm, then, determines if the deployment of the new VM would bring the PM closer to its target resource ratio or not (line 11).

In the subsequent selection process, the PM having the highest progress score in the cluster can be prioritize. If a candidate deployment would shift the workload ratio away from the *target ratio* resulting in a negative progress score, the PM under consideration is therefore typically not selected. However, there are scenarios where the progress score may be negative for all PMs, such as when dealing with a large, unbalanced VM deployment. In such cases, our preference is to deploy the considered VM on a PM with a lighter workload, as this improves our chances of counterbalancing the bias later on. This is why lines 12 to 15 factor in the negative score of the PM by considering its current resource allocation.

A PM that does not host any VM is regarded as having an ideal ratio, as indicated in line 6 of the algorithm. This implies a preference for consolidating existing hosting PMs before considering idle ones for new deployments. The rationale behind this approach is that a PM with an ongoing workload will typically exhibit an allocation ratio diverging from its *target ratio*, thereby making deployments more appealing to it.

Allocations considered in this algorithm are based on PM resource usages. Oversubscribed vNodes are considered through the PM allocation, and not, for example, the sum of exposed vCPUs. This approach enables our algorithm to accommodate all possible oversubscription levels.

Furthermore, the algorithm computes the *target ratio* on an individual PM basis, thereby accommodating variations in hardware settings within a given cluster. This consideration allows for the optimization of resource allocation tailored to the specific characteristics of each PM.

VII. Empirical evaluation

Our proposed solution was tested on both a physical platform, as detailed in Section VII-A, and a simulator, as described in Section VII-B.

The input for both platforms is generated by customer traces, encompassing actions, such as VM creation, VM usage, and VM deletion. This collection of client activities is collectively referred to as the "workload". Ensuring the inclusion of realistic workloads was important in our context, as we highlighted in Section III, where the distribution of typical VMs sizes from Cloud providers has a noticeable impact on the M/C ratio.

For both of our platforms, we opted to employ CloudFactory [30] as a workload generator. This tool facilitates the generation of a dynamic set of VMs that align with a Cloud provider context, considering both the distribution of VM sizes and the CPU usage of VMs. We extended the generator to incorporate our oversubscription considerations. These modifications enabled our version of the generator to create VM oversubscribed across multiple levels, with proportions specified during the generation process. The impact of the shares among oversubscription levels is subsequently discussed in our evaluation.

A. Evaluation in the wild

We now turn our attention to presenting an example of the operational behavior of our *local scheduler* in the context of a physical platform. Prior research has extensively examined the performance implications of pinned resources [39], [42], [43]. Our focus is on comparing the performance of our strategy with the baseline scenario, where a PM hosts VMs at a single oversubscription level without pinning considerations. Additionally, we aim to evaluate our ability to isolate VMs from distinct vNodes.

TABLE III
HARDWARE SETTINGS OF OUR IAAS WORKER

Processor	AMD EPYC 7662 64-cores $\times 2$
Total threads	2×64 cores $\times$ 2 hyperthreads $= 256$
Memory	1 TB
***Memory per Core* (M/C)**	$1,000/256 = 4$
Operating System	Linux Redhat 8.9
Virtualization Platform	QEMU & KVM 7.1

1) Physical experimentation settings: In our experiments, we used the PM described in Table III. The hardware settings of this worker include 256 threads and 1 TB of memory, resulting in a M/C ratio of 4 GB per thread.

We adopted the Azure VM size distribution as a reference and created a progressively escalating workload until the PM capacity was reached. Regarding the workload of each individual VM, the CPU usage patterns obtained from CLOUDFACTORY were translated into application loads. Among the VMs, 10% were set to idle, 60% underwent a CPU benchmark using `stress-ng` [44], and the remaining VMs were composed of interactive applications. Specifically, we selected the *social network application*, a micro-service architecture from the DEATHSTARBENCH [45], and continuously monitored their response time under varying requests per second objectives generated with `wrk2`. These response times served as a proxy of VMs performance.

We considered three distinct oversubscription levels: 1:1, 2:1, and 3:1. In the baseline scenario, the three oversubscription levels are hosted separately. Our PM can host 131 VMs without oversubscription, 271 VMs at 2:1, or 356 VMs oversubscribed at 3:1.

Under the SLACKVM scenario, the three oversubscription levels are hosted concurrently, each accounting for about one-third. Out of the total of 220 VMs, 70 were premium (1:1), 76 were 2:1, and 74 were 3:1. The *social network applications* were deployed on all 3 vNodes to assess performance isolation.

Please note that both the number of oversubscription levels and the maximum level of 3:1 in both scenarios were arbitrary choices used as a proof of concept, but their value can be adjusted according to hardware configurations and workloads of Cloud providers. Our *local scheduler* does not impose a limit on the considered oversubscription levels, and can host more vNodes with more oversubscribed VMs, especially for non-interactive workloads, like storage and batch processing. As such, it can be configured by Cloud providers to align with their specific context. In our scenarios, the 3:1 oversubscription level was selected as it is the last whole level being suitable for interactive workloads within the Azure CPU usage distribution (higher levels introduce a pronounced time slices contention).

Our local scheduler is implemented in Python and interfaces with the hypervisor using the `libvirt` library. It has been tested with QEMU/KVM as the hypervisor of choice due to its native support for dynamic CPU pinning changes.

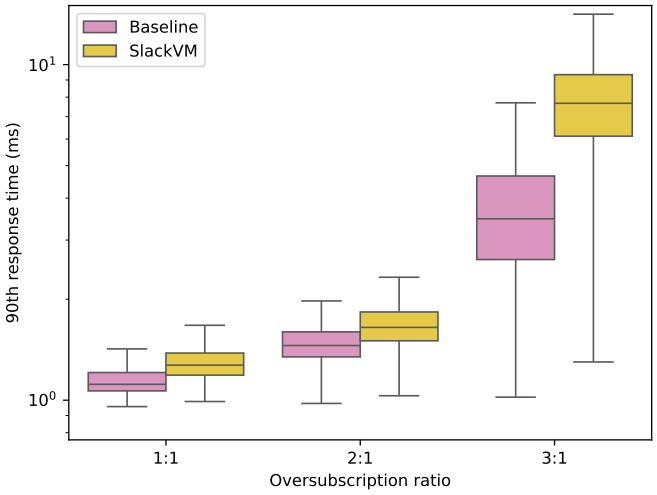

Fig. 2. Comparison of 90^{th} percentile response times for the DEATHSTARBENCH *Social network app* (log-scale Y axis)

2) Performance results: Performances between SLACKVM and the dedicated clusters were compared through the 90^{th} response times that we measured in our empirical experiments. Observed median values are reported in Table IV, while the performance distribution per oversubscription ratio can be visualized in Figure 2.

TABLE IV
PERFORMANCE COMPARISON BY THE MEDIAN OF THE 90^{th} RESPONSE TIMES MEASURED

Oversubscription levels	Baseline (ms)	SLACKVM (ms)
1:1	1.16	1.27 ($\times$1.09)
2:1	1.46	1.65 ($\times$1.13)
3:1	3.47	7.67 ($\times$2.21)

As expected, the response time depends on the applied oversubscription level. In the baseline scenario, isolation between VMs with different oversubscription levels is achieved using different PMs. However, we demonstrate with SLACKVM that performance can still be isolated within a single PM. In our experiment, all three oversubscription levels were hosted concurrently on a single PM. The core pinning mechanism based on cache-level distinction was efficient in maintaining distinct levels of performance while keeping the performance overhead low on the most critical workloads (the ones associated with low levels of oversubscription).

While thread oversubscription may decrease performance, as reported by [46], the vNodes performance overhead is primarily due to the heterogeneity between cores. In a classic setting, CPU scheduler mechanisms do not exploit *Simultaneous Multithreading* (SMT) capabilities until cache-level groups are fully loaded. However, in scenarios where the allocation of available cores is constrained, such as highly oversubscribed environments, the operating system may trigger SMT capabilities "earlier", due to limited workload spreading possibilities.

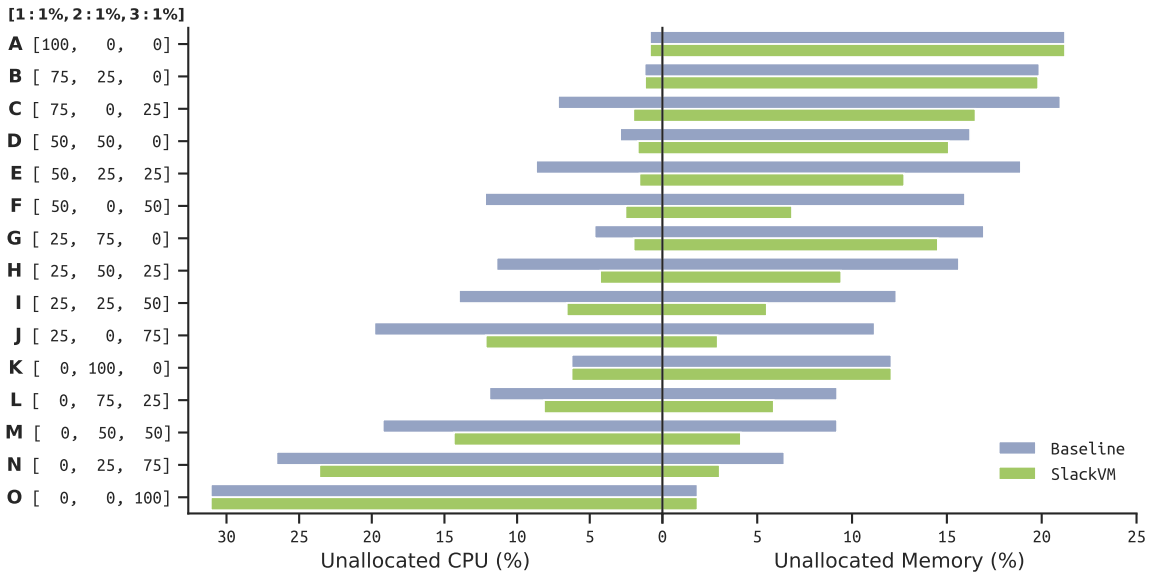

Fig. 3. Comparison of unallocated resource ratios between *dedicated clusters* (baseline) and SLACKVM when considering the OVHcloud setups

Interestingly, this performance penalty remains limited in non-oversubscribed environments.

On the one hand, one can observe that the heuristics used by SLACKVM affects the oversubscribed VMs in priority, which are—by design—less prone to enforcing performance guarantees with strict *Service-Level Objective*s (SLOs). On the other hand, the least oversubscribed VMs are preserved from performance degradation (less than 10% for 90^{th} percentile), hence maintaining their relevance as part of a premium offer. Given these observations, we are now interested in further investigating the influence of the distribution of oversubscription levels on the savings that can be achieved in terms of cluster size, which is a key indicator for Cloud providers interested in optimizing their *Return on Investment* (RoI).

B. Evaluation at scale

Evaluating IAAS schedulers is known to be challenging, due to the lack of detailed information on current solutions used in production [47]. Cloud schedulers compute a fitting score for suitable PMs based on hundreds of undisclosed rules [21], [24], [28], [29]. We choose to focus on improving packing efficiency. We evaluate our newly introduced metric, which indicates progress toward the perfect M/C ratio and its impact on reducing cluster size. First-fist scheduling serves as our baseline to evaluate it. This scheduling strategy is commonly employed to assess packing efficiency [47], [48], as it fills existing servers before considering new ones for deployments [25], [48]. In practice, Cloud providers may guide workload packing by adjusting the weight of our metric in their scoring mechanism, alongside their others criteria.

SPOTVMS, HARVESTVMS, and disaggregated VMs serve as complementary strategies when focusing on DC usage metrics, by filling infrastructure gaps, but our evaluation objective is to assess our capability to prevent these gaps from occurring initially.

1) Simulated experimentation settings: We also implemented SLACKVM in the CLOUDSIMPLUS simulator [49], which is a derivative of CLOUDSIM [50]. Specifically, we created a particular worker type, utilizing our *local scheduler* heuristics to accommodate VMs from various oversubscription levels. Additionally, we implemented a *global scheduler* responsible for selecting the most suitable host based on the highest progress score, therefore improving the M/C ratio.

Regarding the distribution of VM configurations, we adopted the provided specifications from OVHcloud and Azure Cloud providers. As Cloud providers do not disclose the share of VMs oversubscribed at each level, we conducted tests with different distributions.

The established protocol generated a workload involving a target of 500 VMs for each Cloud provider, exploring various oversubscription level distributions. We simulated DC workloads over the course of a week, adhering to arrival and departure rates of VMs. For each workload, a CLOUDSIMPLUS simulation was initiated, starting from an empty cluster and progressively increased until the minimal number of PMs was determined. Each PM within the cluster offered 32 cores and 128 GB of memory, resulting in a M/C ratio of 4 GB per core. To account for the typical largeness of Cloud workloads, we express gains in percentage values, as our approach scales with the cluster size.

2) Results at scale: The gains can be quantified in two ways: in terms of stranded resources avoided and in terms of avoided PMs (due to a better packing).

a) On the reduction of stranded resources: Figure 3 compares the share of unallocated resources for various distributions of oversubscription levels. These distributions are ordered from the least oversubscribed (distribution A, includ-

ing only VMs without oversubscription—i.e., at 1:1) to the most oversubscribed (distribution O, fully composed of VMs oversubscribed at 3:1). In low-oversubscribed environments, characterized by a CPU bottleneck, one can assess that there is a high proportion of unallocated memory. Nevertheless, as the ratio of oversubscribed resources increases in the distributions, a notable shift takes place, leading to an excess of unallocated CPU resources in the most oversubscribed environments. This is attributed to memory bottlenecks.

Figure 3 highlights the resource allocation biases in the DC using the baseline *First-Fit* scheduling. These biases are a combination of the individual limiting factor within each cluster, weighted by their significance in the overall worker distribution. By adopting SLACKVM, the amount of unallocated CPU and memory resources in the DC is reduced for a large majority of the explored distributions. Thanks to the pooling principle of SLACKVM, significant gains are observed when there is a substantial share of both CPU and memory unused in a given distribution. In the baseline approach, these unused resource types are typically on distinct clusters, but when combined with SLACKVM, they have the potential to facilitate additional deployments.

This simulation can also be used by Cloud providers to study the effects of the oversubscription level parameters on the potential gains they can expect, depending on the characteristics of their IAAS workloads.

Given our dependence on the sequence of arriving VMs, it is important to acknowledge that the unallocated resource shares are not reduced to the theoretical minimum of 0%. Our progress towards a balanced M/C ratio aimed at enhancing PM packing (even in the context of heterogeneous hardware configurations), but it does not guarantee that a PM allocates all of its resources (CPU and memory) when deploying the last VM. Considering live migration to further balance the packing of our vNodes is let as a future work.

b) On the reduction of the cluster size: Beyond resource allocation, we also study the PM gains achieved from the distributions involving the above 3 oversubscription levels, as depicted in Figure 4. The x-axis represents the share of 1:1 VMs, the y-axis reflects the ratio of 2:1 VMs, while the ratio of 3:1 VMs results from the intersection of both axes (as the complementary value to reach 100%). In each cell, the figure reports on the percentage of PM saved using SLACKVM. Reported savings are contingent upon the interplay of resource limits at each oversubscription level within the infrastructure.

In scenarios where all oversubscription levels tend to saturate the same resource—i.e., CPU or memory—the gains are generally modest. Considering a M/C ratio of 4, only the workload associated with 3:1 VMs is memory-bound, while others are either CPU-bound or exhibit a balanced resource utilization. Consequently, the gains remain limited in scenarios where no 3:1 VMs are deployed, as observed in distributions A, B, D, G, and K in Figure 3, as well as in the values reported along the diagonal in Figure 4.

However, gains may still be observable due to a "threshold effect", inherent in mechanisms similar to *First-Fit* scheduling.

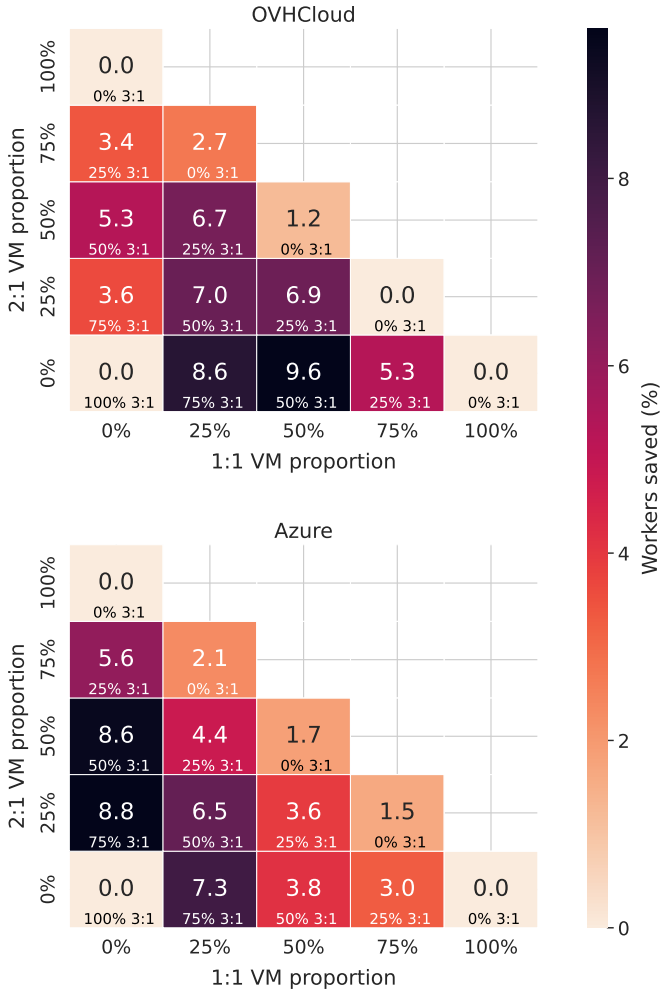

Fig. 4. SLACKVM gains in terms of PM (%) for various oversubscription distributions (the 3:1 VM distribution corresponds to the 100 complement of the other two distributions)

A *First-Fit* scheduling strategy typically consolidates workloads on the first $i-1$ PMs before considering deployment on PM_i, resulting in lower utilization of PM_i. In the case of isolated clusters for each oversubscription level, this results in one additional PM_i per cluster, which is subsequently consolidated when considering our vClusters in SLACKVM. This consolidation leads to a maximum gain of $n-1$ PMs, where n represents the number of oversubscription levels. This type of gain is considered marginal, as it does not scale with the number of VMs under consideration.

Nonetheless, when considering complementary oversubscription levels, the gains in terms of PMs being used can become substantial. For instance, in distribution F, where 50% of VMs operate without oversubscription—i.e., a 1:1 ratio—and the remaining 50% are oversubscribed at 3:1, there is a potential reduction of 9.6% in the number of PMs required when employing our vClusters in the context of OVHcloud distribution. In this specific scenario, dedicated clusters would

require the provisioning of 83 PMs (55 for the 1:1 cluster and 28 for the 3:1 cluster), whereas our approach required only 75 PMs overall. This highlights the significant PM utilization optimization achieved by SLACKVM through the use of our vClusters.

The observed gains are contingent on the distribution, as they can be perceived as the quantity of resources that would have been unallocated on the critical path, but are effectively collected by SLACKVM. If a cluster has the equivalent of n unallocated CPU configurations, and another cluster has the equivalent of m unallocated memory configurations, the hypothetical pooling gain will be the lesser of the two—i.e., the critical path is reduced to a minimum.

The OVHcloud environment achieves higher gains, primarily due to more balanced biases (compared to the considered M/C ratio) between its oversubscription levels. However, Azure can also realize significant gains (up to 8.8% workers saved), especially in distributions with a limited ratio of 1:1 VMs. The Azure 1:1 distribution is heavily biased towards CPU, which requires limiting its usage on the distribution to attain the highest gains, as Azure situation lacks a heavily memory-biased oversubscription level to counterbalance it.

Although Cloud providers do not have full control over the share of customers selecting oversubscribed or non-oversubscribed VM offers, they can still tune their appropriate oversubscription levels, based on their catalog and workload profiles. This customization can minimize the unallocated resources, allowing Cloud providers to derive maximum benefits from our approach.

The gains in terms of PM scheduling infrastructure, such as the elimination of the need for multiple OPENSTACK instances (so-called control planes), are not explicitly reported in our analysis but can be considered as an additional benefit of this approach.

Production-ready schedulers may therefore benefit from incorporating our M/C ratio progress score in the context of multi-oversubscribed PMs, complementing it with their existing scheduling rules. The exploration of potential compromises between these rules is left as a topic for future work.

VIII. CONCLUSION

In this paper, we have demonstrated that different oversubscription levels can saturate different physical resources. Building upon this insight, we explored the complementarity of oversubscription levels. We introduced SLACKVM, a IAAS architecture that can orchestrate heterogeneous oversubscription levels on the same PM and, consequently, within the same cluster of PMs.

On the global scheduling front, SLACKVM takes advantage of the complementarity between oversubscription levels by considering the individual hardware settings of each PM involved in a cluster—using a *Memory per Core* (M/C) indicator. In terms of local scheduling, SLACKVM effectively segregates physical resources by carefully analyzing the PM's hardware topology to isolate performance implications.

Physical experiments have shown that SLACKVM can effectively preserve both the performance of premium offers and isolation when compared to physical clusters. Our simulations have further illustrated the potential of SLACKVM at scale, with the ability to save up to 9.6% in terms of PM hosts within the Cloud. While this reduction in the number of PMs has a positive impact on the energy consumption and carbon footprint of the Cloud ecosystem, it also improves the return on hardware investments of Cloud providers.

We identified two perspectives for this work. This paper primarily focuses on accommodating diverse oversubscription levels for CPU resources within a single PM. However, our approach can be extended to cover other critical resources, including memory, disk, and network, provided effective partitioning or isolation can be achieved between different groups of VMs. In particular, the isolation of memory resources for distinct VMs, as exemplified in [51], represents a compelling area for further exploration and research. While our vNodes adopted static oversubscription levels, they could potentially benefit from dynamically computed levels. This dynamic approach has the potential to further enhance PM resource utilization and improve the performance of oversubscribed VMs. This resource allocation knob could be effectively used to tune the performances of hosted services according to agreed SLA.

SOFTWARE ARTEFACTS

Our *local* and *global schedulers* implementations are publicly available.[3] To encourage the reproduction of our results, our modified version of CLOUDSIMPLUS[4] and CLOUDFACTORY[5] are also publicly available online.

REFERENCES

[1] N. Bashir, N. Deng, K. Rzadca, D. Irwin, S. Kodak, and R. Jnagal, "Take it to the limit: Peak prediction-driven resource over-commitment in datacenters," in *16th European Conference on Computer Systems*, EuroSys'21, p. 556–573, ACM, 2021.

[2] R. York and J. A. McGee, "Understanding the jevons paradox," *Environmental Sociology*, vol. 2, no. 1, pp. 77–87, 2016.

[3] International Energy Agency, "Data centres and data transmission networks," 2021. Available at https://www.iea.org/energy-system/buildings/data-centres-and-data-transmission-networks#programmes.

[4] European Commission Joint Research Centre, "The eu code of conduct for data centres – towards more innovative, sustainable and secure data centre facilitie)," 2023. Available at https://joint-research-centre.ec.europa.eu/jrc-news-and-updates/eu-code-conduct-data-centres-towards-more-innovative-sustainable-and-secure-data-ce en.

[5] L. A. Barroso, J. Clidaras, and U. Hölzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Edition*. 2013.

[6] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and qos-aware cluster management," in *19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS'14, p. 127–144, ACM, 2014.

[7] C. Lu, K. Ye, G. Xu, C.-Z. Xu, and T. Bai, "Imbalance in the cloud: An analysis on alibaba cluster trace," in *IEEE International Conference on Big Data*, Big Data'17, pp. 2884–2892, 2017.

[3] https://anonymous.4open.science/r/VMSlack-C546/

[4] https://anonymous.4open.science/r/Cloudsimplus-VMSlack-7EE2/

[5] https://anonymous.4open.science/r/Cloudfactory-VMSlack-1B7A

[8] Amazon Elastic Compute Cloud, "Amazon ec2 spot instances," 2022. Available at https://aws.amazon.com/ec2/spot/.

[9] Microsoft Azure, "Azure spot virtual machines," 2020. Available at https://azure.microsoft.com/en-us/pricing/spot.

[10] Google Cloud Platform, "Preemptible vm instances," 2020. Available at https://cloud.google.com/compute/docs/instances/preemptible.

[11] Y. Wang, K. Arya, M. Kogias, M. Vanga, A. Bhandari, N. J. Yadwadkar, S. Sen, S. Elnikety, C. Kozyrakis, and R. Bianchini, "Smartharvest: Harvesting idle cpus safely and efficiently in the cloud," in *16th European Conference on Computer Systems*, EuroSys'21, p. 1–16, ACM, 2021.

[12] X. Jia, J. Zhang, B. Yu, X. Qian, Z. Qi, and H. Guan, "Giantvm: A novel distributed hypervisor for resource aggregation with dsm-aware optimizations," *ACM Trans. Archit. Code Optim.*, vol. 19, mar 2022.

[13] H.-R. Chuang, K. Manaouil, T. Xing, A. Barbalace, P. Olivier, B. Heerekar, and B. Ravindran, "Aggregate vm: Why reduce or evict vm's resources when you can borrow them from other nodes?," in *18th European Conference on Computer Systems*, EuroSys'23, p. 469–487, ACM, 2023.

[14] A. Fuerst, A. Ali-Eldin, P. Shenoy, and P. Sharma, "Cloud-scale vm-deflation for running interactive applications on transient servers," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '20, p. 53–64, ACM, 2020.

[15] Y. Zhang, Í. Goiri, G. I. Chaudhry, R. Fonseca, S. Elnikety, C. Delimitrou, and R. Bianchini, "Faster and cheaper serverless computing on harvested resources," in *28th ACM SIGOPS Symposium on Operating Systems Principles*, SOSP'21, pp. 724–739, 2021.

[16] D. Movsowitz Davidow, O. Agmon Ben-Yehuda, and O. Dunkelman, "Deconstructing alibaba cloud's preemptible instance pricing," in *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '23, p. 253–265, ACM, 2023.

[17] Citrix, "Overcommiting pcpus on individual xenserver vms," 2018. Available at https://support.citrix.com/article/CTX236977/overcommiting-pcpus-on-individual-xenserver-vms.

[18] VMWare, "Cpu virtualization basics," 2019. Available at https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.resmgmt.doc/GUID-DFFA3A31-9EDD-4FD6-B65C-86E18644373E.html.

[19] Proxmox, "Proxmox ve administration guide," 2022. Available at https://pve.proxmox.com/pve-docs/pve-admin-guide.pdf.

[20] OpenStack, "overcommiting cpu and ram," 2022. Available at https://docs.openstack.org/arch-design/design-compute/design-compute-overcommit.html.

[21] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *10th European Conference on Computer Systems*, EuroSys'15, pp. 1–17, 2015.

[22] J. Wang, H. Zhang, Z. Xu, W. He, and Y. Guo, "A scheduling algorithm based on resource overcommitment in virtualization environments," in *1st IEEE International Conference on Computer Communication and the Internet*, ICCCI'16, pp. 439–443, 2016.

[23] P. Jacquet, T. Ledoux, and R. Rouvoy, "Scroogevm: Boosting cloud resource utilization with dynamic oversubscription," *IEEE Transactions on Sustainable Computing*, pp. 1–13, 2024.

[24] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *26th Symposium on Operating Systems Principles*, SOSP'17, p. 153–167, ACM, 2017.

[25] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder, "Heuristics for vector bin packing," *research. microsoft. com*, 2011.

[26] D. S. Hochba, "Approximation algorithms for np-hard problems," *ACM Sigact News*, vol. 28, no. 2, pp. 40–52, 1997.

[27] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management*, vol. 23, pp. 567–619, 2015.

[28] OpenStack, "Scheduling," 2019. Available at https://docs.openstack.org/mitaka/config-reference/compute/scheduler.html.

[29] O. Hadary, L. Marshall, I. Menache, A. Pan, E. E. Greeff, D. Dion, S. Dorminey, S. Joshi, Y. Chen, M. Russinovich, and T. Moscibroda, "Protean: VM allocation service at scale," in *14th USENIX Symposium on Operating Systems Design and Implementation*, OSDI'20, pp. 845–861, USENIX Association, Nov. 2020.

[30] P. Jacquet, T. Ledoux, and R. Rouvoy, "Cloudfactory: An open toolkit to generate production-like workloads for cloud infrastructures," in *11th IEEE International Conference on Cloud Engineering*, IC2E'23, 2023.

[31] AWS, "Aws re:invent 2017 deep dive on amazon ec2 instances, featuring performance optimization (cmp301)," 2017. Available at https://www.youtube.com/watch?v=mZy6E2I5Rek&t=815s.

[32] Q. Zhang, P. Bernstein, D. S. Berger, B. Chandramouli, B. T. Loo, and V. Liu, "Compucache: Remote computable caching using spot vms," in *Conference on Innovative Data Systems Research*, CIDR'22, January 2022.

[33] A. Fuerst, S. Novaković, I. n. Goiri, G. I. Chaudhry, P. Sharma, K. Arya, K. Broas, E. Bak, M. Iyigun, and R. Bianchini, "Memory-harvesting vms in cloud platforms," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '22, p. 583–594, ACM, 2022.

[34] P. Ambati, Í. Goiri, F. Frujeri, A. Gun, K. Wang, B. Dolan, B. Corell, S. Pasupuleti, T. Moscibroda, S. Elnikety, *et al.*, "Providing SLOs for Resource-Harvesting VMs in Cloud Platforms," in *14th USENIX Symposium on Operating Systems Design and Implementation*, OSDI'20, pp. 735–751, 2020.

[35] H. Li, D. S. Berger, S. Novakovic, L. Hsu, D. Ernst, P. Zardoshti, M. Shah, S. Rajadnya, S. Lee, I. Agarwal, M. D. Hill, M. Fontoura, and R. Bianchini, "Pond: Cxl-based memory pooling systems for cloud platforms," 2022.

[36] P. Sharma and P. Kulkarni, "Singleton: system-wide page deduplication in virtual environments," in *21st International Symposium on High-Performance Parallel and Distributed Computing*, HPDC'12, p. 15–26, ACM, 2012.

[37] D. Williams, H. Jamjoom, Y.-H. Liu, and H. Weatherspoon, "Overdriver: handling memory overload in an oversubscribed cloud," *SIGPLAN Not.*, vol. 46, p. 205–216, mar 2011.

[38] F. Wuhib, R. Stadler, and H. Lindgren, "Dynamic resource allocation with management objectives—implementation for an openstack cloud," in *8th International Conference on Network and Service Management (CNSM) – Workshop on Systems Virtualiztion Management (SVM)*, pp. 309–315, IEEE, 2012.

[39] D. Ghatrehsamani, C. Denninnart, J. Bacik, and M. Amini Salehi, "The art of cpu-pinning: Evaluating and improving the performance of virtualization and containerization platforms," in *49th International Conference on Parallel Processing*, ICPP'20, ACM, 2020.

[40] Linux Documentation, "Numa binding description," 2021. Available at https://www.kernel.org/doc/Documentation/devicetree/bindings/numa.txt.

[41] OpenStack, "Scheduling," 2019. Available at https://docs.openstack.org/mitaka/config-reference/compute/scheduler.html#weights.

[42] M. Badaroux, S. Miroddi, and F. Pétrot, "To pin or not to pin: Asserting the scalability of qemu parallel implementation," in *24th Euromicro Conference on Digital System Design*, DSD'21, pp. 238–245, IEEE, 2021.

[43] A. Podzimek, L. Bulej, L. Y. Chen, W. Binder, and P. Tuma, "Analyzing the impact of cpu pinning and partial cpu loads on performance and energy efficiency," in *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGrid'15, pp. 1–10, 2015.

[44] C. King, "stress-ng," 2024. Available at https://github.com/ColinIanKing/stress-ng/.

[45] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, Y. He, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvinsky, M. Espinosa, R. Lin, Z. Liu, J. Padilla, and C. Delimitrou, "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems," in *ASPLOS*, pp. 3–18, ACM, 2019.

[46] H. Huang, J. Rao, S. Wu, H. Jin, H. Jiang, H. Che, and X. Wu, "Towards exploiting cpu elasticity via efficient thread oversubscription," in *30th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC'21, p. 215–226, ACM, 2021.

[47] A. Pucher, E. Gul, R. Wolski, and C. Krintz, "Using trustworthy simulation to engineer cloud schedulers," in *IEEE International Conference on Cloud Engineering*, IC2E'15, pp. 256–265, 2015.

[48] T. Knauth and C. Fetzer, "Energy-aware scheduling for infrastructure clouds," in *4th IEEE International Conference on Cloud Computing Technology and Science*, pp. 58–65, 2012.

[49] M. C. Silva Filho, R. L. Oliveira, C. C. Monteiro, P. R. M. Inácio, and M. M. Freire, "Cloudsim plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, ex-

tensibility and correctness," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 400–406, 2017.

[50] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Softw. Pract. Exper.*, vol. 41, p. 23–50, jan 2011.

[51] S. Kim, H. Kim, J. Lee, and J. Jeong, "Group-based memory oversubscription for virtualized clouds," *Journal of Parallel and Distributed Computing*, vol. 74, no. 4, pp. 2241–2256, 2014.