SweetspotVM: Oversubscribing CPU without Sacrificing VM Performance

Pierre Jacquet, Thomas Ledoux, Romain Rouvoy

SweetspotVM: Oversubscribing CPU without Sacrificing VM Performance

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

Thomas Ledoux
Inria, IMT Atlantique,
Nantes, France
thomas.ledoux@inria.fr

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

***Abstract*—The adoption of computing resources oversubscription in cloud environments is conventionally limited to a restricted subset of *Virtual Machine*s (VMs) within the providers' offerings, primarily driven by performance considerations. So far, VMs schedulers mostly implement all-or-nothing oversubscription strategies, wherein all VM resources are either oversubscribed or remain unaltered. While the former strategy offers higher consolidation rates, the latter delivers better performance guarantees.**

In this paper, we conducted an empirical study of the individual usage of *virtual CPU*s (vCPUs) in the OVHCloud production environment and we demonstrate that, as they are not uniformly utilized, the current holistic approach may not be appropriate. Based on these observations, we introduce a novel approach, named SweetspotVM, where oversubscription ratios are applied at the granularity of individual vCPU, instead of the whole VMs. This novel paradigm unlocks a more flexible oversubscription management strategy, pinning oversubscription ratios per vCPU within VMs. We present a prototype of SweetspotVM to illustrate the feasibility of accommodating multiple oversubscription levels within a single host and assigning them to individual vCPU.

We assess the viability of our approach on a physical platform, demonstrating the possibility of dividing the cost of hosting VMs by 3, while maintaining the VMs performance at the level of non-oversubscribed platforms. We, therefore, believe that SweetspotVM opens new avenues to boost the consolidation of VMs on a reduced number of servers, with positive impacts on the environmental footprint of cloud computing.

I. Introduction

Addressing the challenge of underutilized resources in cloud data centers remains a significant concern [1], [2], [3], aiming to reduce both costs and ecological footprints of these virtual platforms. The consolidation of workloads onto a smaller set of *Physical Machine*s (PMs) improves efficiency, considering the non-linear relationship between PM power consumption and workload [1], [4]. This consolidation also contributes to erasing the manufacturing footprint due to unnecessary components.

Various strategies are currently employed to increase resource utilization, ranging from aggressive harvesting mechanisms [5], [6] to more passive approaches based on sharing [7]. Oversubscription, also referred to as *overallocation* or *overcommitment*,[1] is frequently implemented by cloud providers. However, the universal adoption of oversubscription is not privileged, as many cloud clients prioritize the performance and reliability of their allocated resources.

Within virtualization and cloud computing, *workers* play a pivotal role in managing virtual resources, including *virtual CPU*s (vCPUs) exposed to *Virtual Machine*s (VMs) and containers. These resources are scheduled on the underlying infrastructure in a manner that accommodates the inherent heterogeneity of hosting PMs. Consequently, VMs are designed to interact with resources that appear to be uniform, with all the intricacies of hardware heterogeneity handled and managed by the host machine. In the context of oversubscription, this uniformity implies that a VM is either entirely oversubscribed or not oversubscribed at all. This dichotomous choice often leads to unused resources, as oversubscription is typically applied only to low pricing-tier VMs.

Paradoxically, beyond the VM scope, heterogeneity in resources has become the *de facto* industry standard. Performance heterogeneity within processor cores is now commonplace in contemporary computer systems. *Simultaneous Multithreading* (SMT), initially introduced in 1995 [8], has gained extensive adoption within x86 architectures, introducing performance variability among CPU cores, based on concurrent thread utilization. Furthermore, architectural designs incorporating CPU cores with distinct frequency ranges have become increasingly prevalent. This trend is exemplified by the big.LITTLE architecture developed by ARM and, more recently, by Intel's 12th generation processors, which integrate a combination of Performance and Energy cores to achieve diverse performance objectives.

Consequently, processes are commonly scheduled in conjunction with manufacturer-specific drivers to facilitate the allocation of time slices based on variations in hardware performance. Therefore, we propose exposing vCPUs with various performance levels to VMs.

This paper introduces per-vCPU performance variations using a novel oversubscription paradigm. Instead of managing oversubscription at the granularity of a VM, we demonstrate that the vCPUs of a VM can be oversubscribed individually to different levels, enabling a more flexible management of resources at large. This innovative approach provides the capability to offer a share of resource guarantees to a VM—i.e., oversubscribing to a 1:1 ratio—while concurrently sharing

[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 allocated resources.

other resources (oversubscribing to an n:1 ratio with $n > 1$) across various oversubscription levels.

In the remainder of this paper, we first motivate the need for an oversubscription paradigm closer to the actual usage (cf. Section II) based on an empirical analysis of OVHCLOUD production environment, one of the largest European cloud operators [9]. We, then, detail how oversubscription can be implemented at the vCPU granularity (cf. Section III). We evaluate our prototype in Section IV using realistic *Infrastructure-as-a-Service* (IAAS) workloads and report on our ability to selectively guarantee computing resources. Finally, we discuss related work (cf. Section V) before concluding in Section VI on this work and its perspectives.

II. MOTIVATION

In this section, we motivate the need for another oversubscription paradigm that better fits to individual vCPU usage.

A. Not all vCPUs are equally used

The cloud is characterized by its heterogeneous workload, covering VMs hosting storage-oriented services, batch processes, or interactive applications. IAAS customers have the flexibility to configure the level of computing power, typically indicated by a number of vCPUs to provision, based on their workload type and anticipated demand (e.g., peak requests per second on a website).

Cloud providers commonly analyze the utilization of initial resource allocations through VM CPU usage, a metric often included in the cloud datasets shared with the research community [10], [11], [12]. However, global VM usage does not allow for the differentiation of various workload situations. For instance, a VM configured with 4 vCPUs and utilizing 25% of its CPU time may concentrate its workload on a unique vCPU in a CPU-intensive single-threaded context, or evenly distribute it across all vCPUs in a fully multi-threaded workload.

To the best of our knowledge, the individual usage of VM vCPUs has not been previously studied. We, therefore, conducted such an analysis using exploitation traces from an production-scale IAAS environment operated by OVHCLOUD, a major cloud provider.[2] This analysis results from the monitoring of an OPENSTACK computing platform over a 1-week window, where the CPU time for each vCPU associated with each VM was recorded at 5-minute intervals. We consider a premium offer that delivers dedicated resources—i.e., no oversubscription is implemented (1:1)—thereby precluding the examination of contention situations.

Figure 1 first reports on the distribution of provisioned VM configurations on the left-hand side. The right-hand side of the Sankey diagram highlights how these VM configurations map to the CPU that are effectively provisioned by the PMs upon deployment. While the smallest VM configurations seem to be prevalent in IAAS platforms (62 % of the VM configurations include at most 2 vCPUs), as previously acknowledged by [10], the share of provisioned CPU highlights that the largest VM configurations are the ones which consume most of the computing resources exposed by the PMs, with 76 % of the CPUs being provisioned by VMs requesting at least 4 vCPUs.

[2]https://www.ovhcloud.com/

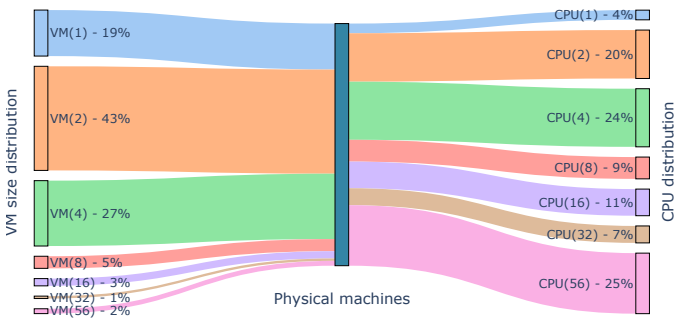

Fig. 1. Mapping the distributions of VM sizes to the physical CPUs provisioned by the OVHCLOUD infrastructure

Beyond this first observation, we further dive into the effective usage of individual vCPUs by the different VM configurations. To do so, we derive an utilization metric, and we order the vCPUs from the most utilized (designated as vCPU0) to the least utilized (designated as vCPU$(n-1)$, where n is the VM size). It is essential to note that during the 5-minute aggregation period, the guest scheduler of each VM has the freedom to relocate processes from one vCPU to another. This allocation may be based, for instance, on the *Completely Fair Scheduler* (CFS) queue calibration mechanism [13]. The substantial differences observed in vCPU time after a 5-minute interval underscore the significance of these variations, indicating that the workload could not be uniformly distributed across all VM resources during this time frame.

However, optimal performances on a given platform are typically obtained for a CPU charge below 100% due, among others, to SMT and cache contention. To account for it, a vCPU is labeled as *active* even if it does not consume 100% of the associated window CPU time. To perform a sensibility analysis of this threshold, we examined 3 values to label a vCPU as active: 1%, 10%, and 30%.

Our results are plotted as *Cumulative Distributed Functions* (CDFs) in Figure 2. For each graph, the Y-axis represents the share of VMs, while the X-axis denotes the proportion of time during which the vCPU can be deemed active, based on the considered threshold (indicated in the caption). For instance, when focusing on the last CDF (lower right) for VMs with 16 vCPUs and an activity threshold of 30%, the comparison is as follows: for 80% of the VMs (Y-axis), vCPU0 (indicated by the red line) is considered as active 100% of the time (X-axis) with the intersection at point A. In contrast, the least used vCPU, namely vCPU15, is considered as active less than 50% of the time (intersection at point B).

We first observe that the vCPU0 line can be distinguished

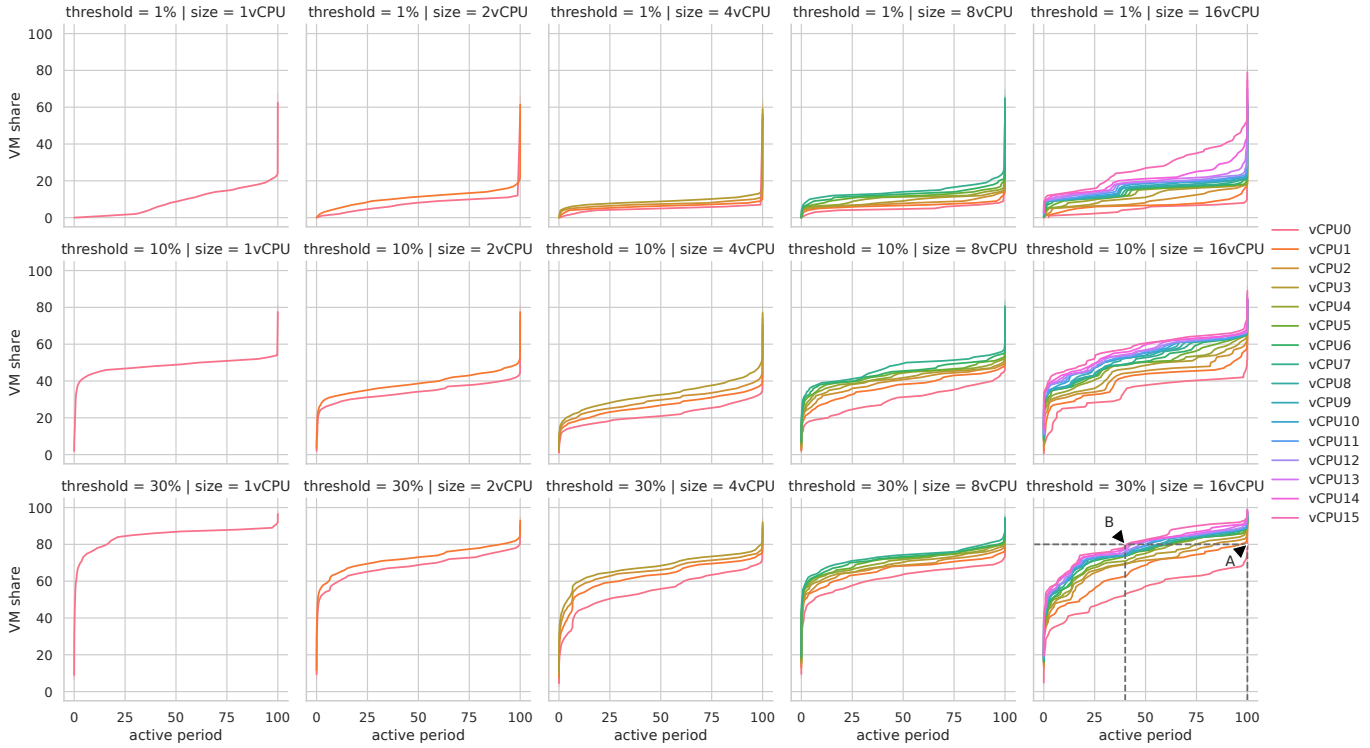

Fig. 2. CDF of individual vCPU utilization ratios of various VMs profiles hosted by OVHCLOUD.

in most of the graphs, having an activity threshold higher than 1%, indicating that during a significant proportion of the time, the workload is mono-threaded. This highlights that not all the IAAS workloads are multi-threaded, and not every multi-threaded workload leverages all the vCPUs provisioned by a VM.

Then, one can observe that larger VMs tend to exhibit a non-uniform usage of their vCPUs. Except vCPU0, 2 vCPUs with close indices (such as vCPU1 and vCPU2) typically report a low usage shift. As this shift is cumulative, it becomes all the more pronounced between the least used and the most used vCPUs, especially in larger VM sizes.

Single-core VMs tend to be less used, indicating that they may be preferred by clients for non-CPU-oriented workloads. However, less-used cores in larger VMs are close to the usage observed in single-core VMs. The count of unused resources is amplified in the case of bigger VMs as they individually have more cores matching this low-usage pattern. This leads us to conclude that the larger the VM, the more resources are wasted by the cloud infrastructure.

We can, therefore, conclude that *i)* the largest VMs are provisioning most of the computing resources, and *ii)* these provisioned resources fail to be fairly consumed, hence leading to wasted resources.

While one could expect the cloud customers to size their VM appropriately, there are many reasons that can explain the provisioning of large VMs with a non-uniform resource usage pattern, among which the lack of predictability of resource usage or the over-provisioning of resources to address potential usage peaks. Cloud providers, therefore, have to cope with this issue and find an alternative to reduce the non-negligible wastes of computing resources imposed by the number of provisioned vCPU that are not effectively used.

In the following sections, SWEETSPOTVM introduces the principle of vertical oversubscription as a solution to mitigate performance requirements and resource utilization, aiming to reconcile both dimensions.

B. Introducing vertical oversubscription

The oversubscription of cloud resources remains a non-trivial exercise as a compromise has to be made between leveraging under-utilized capacity and performance, by avoiding *Service-Level Agreement* (SLA) violations. This is challenging as the specific cloud context implies hosting heterogeneous types of workloads, with almost no direct information on individual workloads, given that cloud providers are operating VMs as black-boxes.

Each PM of a given cloud context is singular. Tuning an oversubscription ratio at the cluster level (e.g., the factor applied to all the PMs), while trying to avoid SLA violations on edge situations, leads to configure pessimistic ratios, hence missing resource optimization opportunities.

The mapping of virtual resources of a VM to the physical resources of a PM can be implemented in different ways. Specifically, the oversubscription scope plays a critical role when accounting for exceptional situations—i.e., those likely

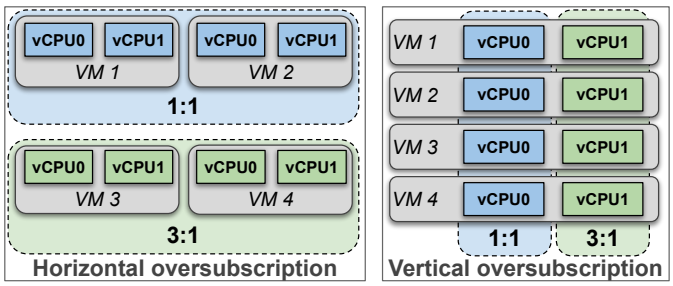

Fig. 3. Transitioning from horizontal CPU oversubscription to vertical oversubscription as implemented by SWEETSPOTVM

to provoke SLA violations. Transitioning from a cluster scope to a server scope, which involves defining a ratio of exposed virtual resources per server, enables the consideration of both hardware heterogeneity [14] and workload heterogeneity [10], [15]. We believe that this shift in scopes has the potential to further yield more optimistic ratios. In particular, to better reflect individual usages, the oversubscription computation scope needs to be extended beyond the per-server limit. In opposition to the current oversubscription paradigm, which treats all resources equally in a *horizontal* manner, as depicted in Figure 3, we advocate for an orthogonal approach, hence qualified as *vertical* oversubscription. In this paradigm, the consumers of the PM resources are considered to be the vCPUs, in contrast to the VMs, thereby lowering the scope granularity.

Similarly to hardware architectures with cores operating at different maximum clock frequencies (e.g., big.LITTLE processor architectures), this leverages oversubscription ratios to introduce different levels of performance within a VM. The *vertical oversubscription* paradigm allows, for example, a cloud provider to guarantee performances on a restricted set of cores (associated with low or even no oversubscription), while also mutualizing resources on others (associated with higher oversubscription ratios).

III. IMPLEMENTATION DETAILS

Conventional horizontal oversubscription paradigm exhibits resource allocation at a host-based granularity, wherein the pool of resources associated with a particular PM is uniformly distributed among all vCPUs relying on the Linux scheduler for sharing time slices.

In contrast, our vertical oversubscription paradigm assumes a non-uniform distribution of physical resources among provisioned vCPUs. Specifically, some premium vCPUs may be allocated to dedicated physical resources, while others may be subject to oversubscription. In this section, we explore the coexistence of various oversubscription ratios on a single PM.

A. Local scheduler

A cloud scheduling architecture can be summarized as two main software components [16], [17].

The first one is a *global scheduler*, also known as the 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 PM, referred to as the *local scheduler*, to gather information about the PMs's 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 provisioning 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.

In SWEETSPOTVM, the capabilities of the *local scheduler* are expanded to include the management of multiple oversubscription ratios. The PM resources are logically separated through distinct pools of vCPUs. Each pool is associated with a given oversubscription ratios and is size can be adjusted dynamically, depending on hosted VMs.

A pool comprising n cores can support a maximum of n vCPUs in the absence of oversubscription. At a 2:1 oversubscription ratio, $2n$ vCPUs can be accommodated, and this applies to any oversubscription ratio. The logical segregation mechanism is further discussed in Section III-B.

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

B. Segregate physical cores

The non-uniform distribution of resources between vCPUs necessitates the utilization of distinct resource pools, each characterized by a specific oversubscription ratio. This allocation is achieved by implementing a shared CPU affinity policy, wherein vCPUs are affixed to a common set of cores on the PM.

We identified and addressed two main challenges. Firstly, the performance and isolation of PM core selection are intricately linked with the PM topology, as established in existing literature [8]. However, the applicability of our heuristics across diverse architectures is imperative, given the heterogeneous configurations prevalent in cloud data centers [14] Secondly, the size of the vCPU pools needs to be dynamically adjusted to cope with unforeseen IAAS workload (covering VMs of various sizes).

1) Generic core selection: Contemporary PMs exhibit intricate processor topologies, potentially featuring heterogeneous distribution of cache levels or multiple sockets, making the segregation process not trivial.

The process of selecting an appropriate core for a given task usually relies on both the Linux scheduler and manufacturers' drivers responsible for managing features, such as C-States and P-States. Manufacturers' drivers may take decisions, such as loading a specific core to harness Turbo-boost capabilities or distributing a workload to optimize cache resource utilization.

These decisions are contingent upon the hardware configuration and the chosen scaling governor.

Directly pinning processes to specific cores would circumvent the involvement of these components, posing a significant threat to the universality of our strategy. In practical implementation, instead of selecting individual cores, we opt for the selection of a range of cores, even in the case of premium vCPUs.

The selection of cores is undertaken with the objective of closely aligning with the characteristics of the processor topology, hence leveraging the optimizations of manufacturer driver capabilities. When expanding a pool of physical resources to accommodate a newly provisioned vCPU, the selection of cores is predicated by a distance metric.

The computation of the distance between two cores depends on their degree of shared cache, where cores with a lower level of shared cache, such as in a SMT topology, are deemed closer than cores with no shared cache. Note that configurations where the last level of cache is not be universally shared are common, such as in a multi-socket architectures or with AMD EPYC processors.

Algorithm 1 Distance (Δ) computation between 2 cores

Input: $core_0$, $core_1$
Output: Δ

```
Δ ←0
for <cachelevels> do
    if LEVEL(core_0) == LEVEL(core_1) then
        return Δ
    end if
    Δ ← Δ + 10
end for
return Δ + NUMA-DISTANCE(core_0, core_1)
```

To account for it, we introduce a core distance metric extending the *Non-Uniform Memory Access* (NUMA) distance [18]. This extended metric incorporates an assessment of the shared cache levels to provide a more complete evaluation of core proximity. Linux system exposes for each core and cache level an ID to identify the cache zone. We retrieve this data and compute distances between each core, as described in Algorithm 1. While the incremental value is arbitrary, we chose it to be in the same order of magnitude as the current NUMA distance notion.

The selection of a core closer to the existing pool ensures that the newly added physical resources share cache levels, resembling the characteristics associated with a socket of the same architecture but with a reduced core count. Consequently, in an SMT topology, cores within the same physical unit are assigned to a common pool.

From a process scheduling perspective, Linux engages sibling cores only when all physical cores are in use, to mitigate performance degradation. In comparison to a scenario where each non-oversubscribed vCPU is assigned to a single personal physical core, we rather considered them as a group and pin them to a range of physical CPUs. If the number of vCPUs matched the number of physical CPUs associated, the non-oversubscribed status is sustained, and the approach let the CFS scheduler manage the distribution of work. This approach maintains the drivers and scheduler behavior, thereby averting performance deterioration due to SMT (as vCPUs do not consistently exceed 50%) while judiciously employing it during peak demands.

2) Pool resizing: A VM with n vCPUs may have a maximum of n different oversubscriptions levels (one for each vCPU). The allocation of a VM to each pool of resources is, therefore, dependent on its size. Some VM may have a vCPU allocated to a given pool while others do not or, some VM may have more vCPUs associated with a given pool than others.

The dynamic nature of VMs encourages a dynamic pool size. Rather than defining the size of each pool statically, the allocation is changed upon each deployments. This flexibility accommodates the uncertainty associated with the number of VM creation requests, enabling our system to efficiently allocate resources in response to varying workloads.

A VM deployment is implemented as the assignment of its vCPUs to the associated vCPU pools. If a pool is not sufficient to provision a new vCPU, this pool can automatically grow by selecting the closest unallocated core from its current configuration. In practice, this implies changing the pinning of VMs having at least one vCPU in the considered pool to accommodate the new range. While frequent changes in core pinning can potentially introduce performance overhead due to increased context switches, it is important to note that, in our specific context, such changes are infrequent occurrences. They only occur when a VM is being deployed or decommissioned. These VM deployment and decommissioning events do not happen at a high frequency within the time scale of CPU operations.

If a pool size extension fails due to a lack of available resources, the VM deployment is rejected by the local scheduler. Furthermore, VM departures from the system do change the allocation to accommodate future deployments.

C. Pool heterogeneity requirements

In a n:1 oversubscription scenario, a cloud provider guarantees that no more than n vCPUs can contend for a single physical core. However, oversubscription relies on workload heterogeneity and the assumption that unused resources by some VMs can be utilized by others.

Consequently, a VM should not be oversubscribed with itself, as this mislead the guest into expecting a certain level of CPU availability that is impossible to receive in practice. The introduction of oversubscription with 2 VMs can also pose a significant risk of performance degradation. This risk diminishes when more VMs are being provisioned, as the probability of all VMs simultaneously reaching their peak usage diminishes.

In an horizontal setting, while each PM may have an oversubscription objective, its resources are only effectively oversubscribed when the number of virtual resources provisioned exceeds its configuration. This guarantees a certain

heterogeneity in the workload. In our vertical context, oversubscription may occur earlier, as we limit the resources available for use by vCPUs. For example, a subset of physical resources may be oversubscribed before all cores are allocated.

To mitigate the risks of contention associated with oversubscribed contexts, we increase workload heterogeneity when possible. In practice, it is possible to allocate different oversubscription levels of VMs to the same set of resources provided that they adhere to the conditions imposed by the lowest oversubscription level within the VM set. In simpler terms, a vCPU with a 2:1 oversubscription level may coexist with a vCPU having a 3:1 oversubscription level, but 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 vCPU is "upgraded", it may be strategically employed to enhance workload heterogeneity temporarily, if there are some unallocated resources to leverage. 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 vCPUs when feasible, effectively leveraging all resources that remain unallocated by the current hosted VMs. Upon deployments, we also prevent VMs from being oversubscribed with themselves by verifying than the pool size is greater than the requested virtual resources.

D. Oversubscription templates

In a vertical oversubscription scenario, the vCPUs of a given VM may be oversubscribed to different ratios. This is achieved using what we term an *oversubscription template*.

A template is a configuration specifying an oversubscription ratio for each vCPU index (or range of vCPUs). While any vCPU can be oversubscribed to any positive amount, we believe that using progressive oversubscription ratios is a good practice. The VM should be aware of the performance of its individual cores to leverage the most of our approach. A general rule stating "*the lower the vCPU index is, the better the performance*" emphasizes that.

Oversubscription templates are configurable and may be changed by cloud providers to match the specificities of their workloads.

IV. Empirical evaluation

In this section, we discuss how our vertical oversubscription paradigm was evaluated.

A. On core priority

While a VM workload may not use more than the equivalent of one core at a given time, its workload may be spread through all its vCPUs due to the CFS behavior. However, in a vertically oversubscribed scenario, performance obtained by a VM is improved if its workloads foster its least-oversubscribed cores.

This can be done in different manners, such as pinning inside the VM the workload of interest. More generic approaches imply to develop a specific Linux scheduler and/or a driver.

In our evaluation, we used a third approach based on a root daemon running on each VM. The daemon, composed of around 150 Python lines of code, computes the CPU usage each 200 ms. The minimal number of cores required to run this load is deducted and applied by deactivating unnecessary VMs cores. Deactivation is executed in decreasing core index order, deactivating the farthest index cores as they are the least powerful ones in our template. This consolidation strategy aims to concentrate the workload on the most powerful cores, with vCPU0 exhibiting the highest performance. If more than 80% of the activated cores are used, a new one is permitted, allowing to handle an increasing intensity in the workload.

This approach implies a certain latency before activating all cores, however, we found it to be acceptable for the size of VMs considered. Activating all cores in a VM composed of 8 vCPUs is performed in a maximum of $7 \times 0.2 = 1.4$ seconds (assuming we start in the worst case from having only vCPU0 activated).

B. On workload generation

Our proposed solution was tested on a physical platform to assess VM performances.

The input is generated from customer traces, encompassing actions, such as VM creation, VM usage, and VM deletion. This compilation of customer activities is collectively referred to as the "workload".

Ensuring the inclusion of realistic workloads was paramount in our context, given that oversubscription relies on a heterogeneous usage of resources by customers. To achieve this, we chose to utilize CloudFactory [19] for workload generation. This decision was motivated by the tool's capability to generate workloads that match statistics in terms of VM usage patterns. The workload was composed of an increasing number of VMs, each having 8 vCPU, overall matching the CPU usage observed in Azure context [10].

The considered VMs hosted two distinct types of applications. Firstly, there was a micro-services architecture, known as *Social Network*, derived from the DeathStarBench [20]. The response times of these micro-services were continuously monitored and utilized as a proxy for the individual performance of the respective VMs. While being dynamic through time, the applied input ranged between 10 to 1,000 requests per second. Secondly, the StressNG load test was applied to a second set of VMs. This load test facilitated the precise load on CPU resources for each VM, contributing to an overall realistic context on the given host. The load of each VM was also dynamic and could change between 0% and 100% of resources used every 90 seconds.

C. Experimental IaaS platform

For our example, we used the PM described in Table I as a worker node. Of the 256 cores, 20 were kept for the

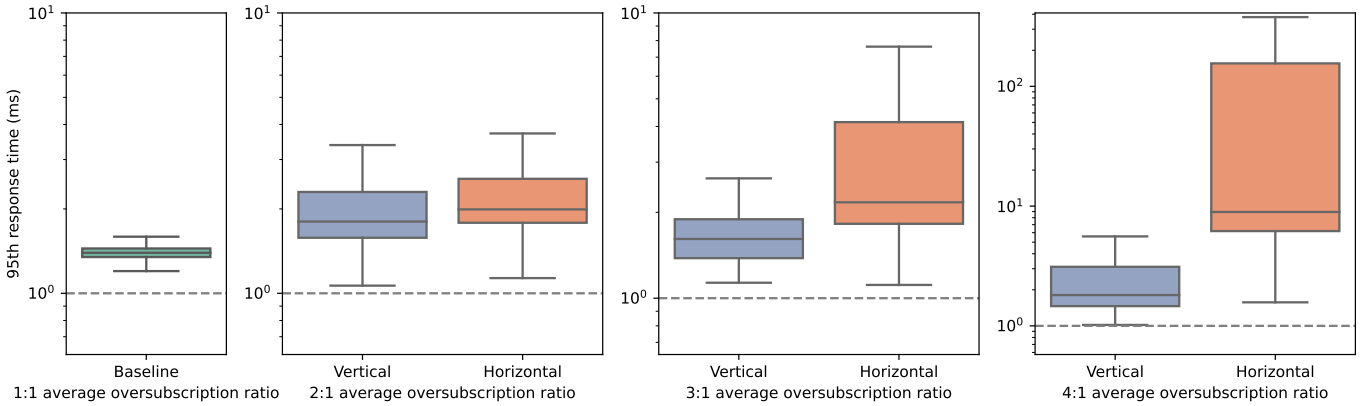

Fig. 4. DEATHSTARBENCH *social network* response time on different oversubscription scenarios

monitoring and components parts, leading to 232 usable cores. Memory was not a limiting factor in all the experiments reported afterward.

TABLE I
HARDWARE SETTINGS OF OUR IAAS WORKER NODE

Processor	AMD EPYC 7662 64-cores $\times 2$
Total threads	2×64 cores $\times$ 2 hyperthreads $= 256$
Memory	1 TB
OS	Linux Redhat 8.6
Hypervisor	QEMU & KVM 7.1

D. Experimental results

We applied the same workload in different contexts. Initially, as a baseline, we executed the workload without considering oversubscription, thereby limiting the virtual resources to the amount proposed by the platform. This configuration is used to set our ground truth as being the optimal performances that can be obtained from our experimental testbed.

The evaluation and comparison of the considered oversubscription techniques was conducted in both horizontal (single ratio) and vertical (multiple ratios) approaches. Specifically, we measured performances under platforms with average oversubscription ratios of 2:1, 3:1, and 4:1. In the horizontal approach, the targeted ratio was uniformly applied to all the provisioned vCPUs. In the vertical approach, the oversubscription template was chosen to have, on average, the same ratio as the one targeted by the horizontal configuration. For example, with a 3.0 oversubscription ratio, 1 vCPU was dedicated for each VM (referred to as vCPU0), one was oversubscribed to a 1.5:1 ratio (vCPU1), and all others were oversubscribed to a 6:1 ratio (vCPUn, for n within 2 to 7). For a workload composed of VMs with 8 vCPUs, this led to an average oversubscription ratio of $r = \frac{vCPU}{CPU} = \frac{8}{(1/1)+(1/1.5)+(6/6)} = 3.0$.

Other templates used by SWEETSPOTVM are described in Table II. While these choices are arbitrary and can be customized by the cloud provider, we selected templates that dedicate part of the resources—i.e., no oversubscription on vCPU0—while oversubscribing others more aggressively to match the ratio. This type of template allows us to illustrate the heterogeneous usage being made of vCPUs.

TABLE II
OVERSUBSCRIPTION TEMPLATES CONSIDERED IN OUR EXPERIMENTS

Oversubscription target	vCPU0 ratio	vCPU1 ratio	vCPU2–7 ratio
2:1	1:1	1.5:1	2.6:1
3:1	1:1	1.5:1	6.0:1
4:1	1:1	1.5:1	16.0:1

The performance of each context, evaluated through the 95^{th} response time of each exposed service, is visualized in Figure 4. Unsurprisingly, the baseline without any oversubscription exposes a good response time, as no vCPU is competing for resources nor SMT is required (as the overall CPU usage remains below 50%).

Under a 2:1 oversubscription template, where the number of hosted VMs is doubled, both vertical and horizontal approaches perform similarly. There is no significant performance degradation (please note the logarithmic scale) with the traditional (horizontal) approach, indicating that pinning vCPUs differently does not notably improve performance.

Under a 3:1 oversubscription template, host resources are still not fully utilized. However, performance decreases significantly with the traditional approach, while our vertical oversubscription template keeps maintaining VM performances closer to their optimal values, succeeding hosting more VMs—i.e., $3\times$ more than the baseline.

The performance gain is even more significant with the 4:1 oversubscription template. In this situation, the horizontal approach leads to an overloaded CPU, and fewer time slices being attributed to each vCPU (note that the Y-axis scale had to be adjusted accordingly). Using a vertical approach, the contention is limited to only a subset of the vCPUs of the VM, and the vCPU0 keeps exhibiting good performances, compared to

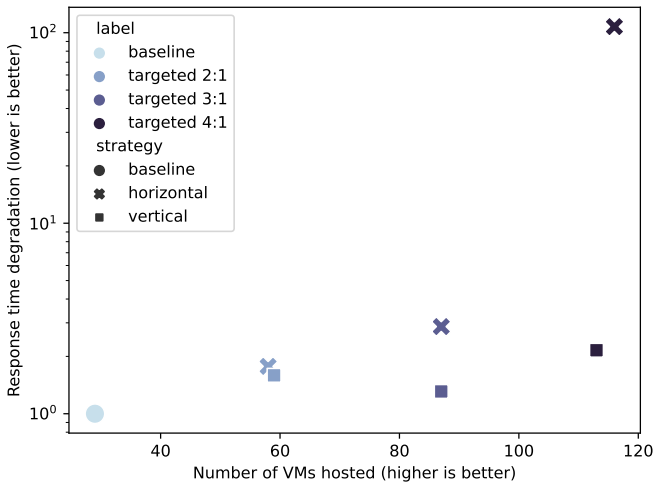

Fig. 5. Performance degradation in response time of the *social network app* of DEATHSTARBENCH (as multiple of the baseline)

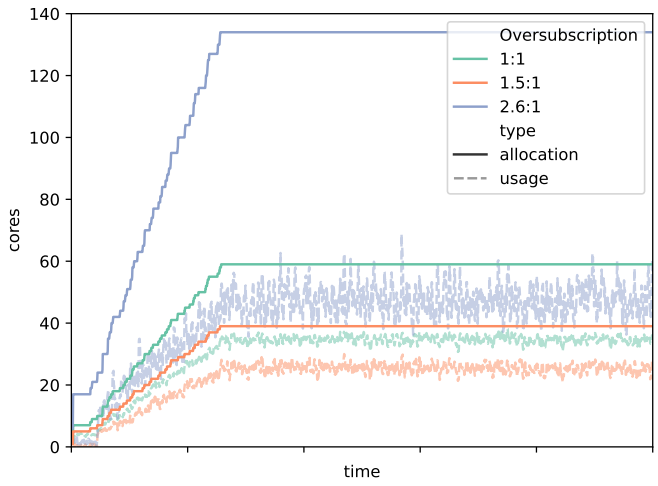

Fig. 6. Evolution of resources allocation and usage per oversubscription level for a SWEETSPOTVM template targeting 2:1

the horizontal oversubscription. This demonstrates that vertical oversubscription can be adopted to mitigate the effects of an overloaded situation.

As such, oversubscription can be used to introduce different levels of performance on different cores. When VMs uses their most powerful cores (the less shared ones) in priority, performances can be close to the optimal.

The VM count is emphasized in Figure 5. Our strategy does host the same number of vCPUs and, therefore, the same number of VMs. When the average oversubscription achieved is not an integer, small differences may appear, which may be mitigated by changing the oversubscription templates. In this example, our average oversubscription was slightly above the 2:1 target (leading to one additional deployment), while being slightly below the 4:1 target (leading to 3 fewer deployments). Notably, one should note that the quantity of memory (vRAM)—and its potential oversubscription—is not affected, compared to a horizontal oversubscription mechanism, as the number of VMs hosted is similar for the same targeted oversubscription ratio. The performance degradation is expressed here in the form of the multiple of the 95^{th} response time of the baseline. In the 2:1 oversubscription scenario, the degradation is reduced by 10% (from 1.8 times the baseline to 1.6), by half in the 3:1 oversubscription scenario (from 2.9 times the baseline to 1.3), and by a factor of 50 in the 4:1 situation (from 107 times the baseline to 2.1 times). While variability may be observed due to other resources (such as the network), the performance degradation is clearly mitigated by SWEETSPOTVM compared to state-of-the-art strategies enforcing horizontal oversubscription.

By dedicating CPUs to the VMs and oversubscribing others more aggressively, our scheduler succeeds in preserving performances by considering the heterogeneous usage made by VMs of their vCPUs. Specifically, with the last oversubscription ratio having no contention, $r = 3.0$, our approach closely aligns with the performance of the non-oversubscribed scenario, while allowing the deployment of $3\times$ more VMs in that case.

The allocation size of the different oversubscription levels targeting an average oversubscription of 2:1 is depicted in Figure 6. The premium subset, dedicated to all vCPU0, provisions 59 physical cores under our workload. Its usage remains low compared to its allocation size, avoiding any concurrency issues (at most 40 cores, 68% of the provisioned resources). The second subset, dedicated to vCPU1, reports on the same number of vCPUs attributed. However, as vCPU0 concentrates most of the workload, the vCPU1 subset exhibits a lower core usage, with only 31 physical cores being effectively used (21% less than the 40 cores). The last subset has a much larger number of hosted vCPUs, as each VM allocates 6 vCPUs to this oversubscription ratio. Its 136 attributed cores were used, with a peak of up to 51% of the provisioned resources.

The performance obtained from the VM perspective depends on the contention observed in the pool of host resources considered. In this example, while the 2:1 is the most oversubscribed one, no contention is observed (as the size of the allocation is far greater than its usage), leading to good performance even on the least powerful VM vCPUs.

E. On the provisioning of small VMs

Our paper focused on the use case of relatively large VMs, due to both their proportion in the count of provisioned vCPUs and their tendencies to have non-uniform usage patterns between their individual vCPUs, as exemplified in Section II. Nonetheless, nothing prevents SWEETSPOTVM from hosting smaller VM configurations in conjunctions with larger VMs as it uses to be case in production-scale IAAS platforms. In particular, SWEETSPOTVM can accommodate smaller VMs by allocating their VMs to non-oversubscribed pools, hence

preserving their quality of service, based on the observations we drawn from Figure 2.

V. Related work

A. Resource oversubscription

Most hypervisors enable oversubscription by allowing the sum of all allocated virtual resources to exceed the PM capabilities [21], [22], [23]. To avoid SLA violations, oversubscription is usually limited at a certain level, quantified as a ratio between the number of virtual resources provided and the available physical resources.

To the best of our knowledge, this ratio is only applied horizontally (albeit homogeneously through all vCPUs) by state-of-the-art cluster managers. For instance, OpenStack [24] and Borg [25] restrict the oversubscription of resources by utilizing a single static value for the entire cluster PMs.

The horizontal oversubscription was also proposed to be defined per PM. It may be static, by taking into account individual PM performance [14]. Alternatively, it can be dynamic, relying on predicted peak usage [15], as new deployments must be performed on resources seen as available in the long run. Peak prediction may be computed using VM percentile [10] or standard deviation [15]. However, none of these approaches dive into the actual usage workloads within each VM, which may not utilize all their vCPUs homogeneously. Each PM maintains a single oversubscription level, determined by its configuration or the cluster-scale configuration.

In this paper, we proposed a vertical oversubscription paradigm, where each vCPU may have an individual oversubscription level. This approach allows cloud providers to better fit consumer requirements (by considering the vCPUs rather than the VM), achieving more optimistic oversubscription ratios while guaranteeing a minimum amount of available resources.

B. Performance heterogeneity

Performance heterogeneity for VMs has been previously investigated to improve energy proportionality [26], [27], [28], [29]. This is achieved by migrating VMs between PMs with different architectures depending on the VM workload. This approach is also horizontal, as the performance is adapted for all the provisioned vCPUs. In our proposal, performance heterogeneity does not arise from hardware differences, but rather from the applied oversubscription ratio, enabling a more flexible management and a better resource packing. Yet, SweetSpotVM offers a complementary approach to cope with hardware heterogeneity and to contribute to the implementation of energy proportionality in cloud infrastructure while minimizing the number of PMs required to host a set of VMs.

VI. Conclusion

In conclusion, this paper has introduced SweetSpotVM, a novel oversubscription paradigm that addresses per-vCPU performance variations, departing from the conventional approach of oversubscribing resources at the VM granularity. By demonstrating the feasibility of individually oversubscribing vCPU to different extents, we have introduced a more flexible resource management strategy for IaaS platforms supporting the cloud industry. In particular, this innovative approach allows cloud providers to propose resource guarantees to a VM on a 1:1 ratio, while concurrently reallocating other resources across potentially multiple oversubscription levels (n:1 ratio with $n > 1$).

Our contribution is implemented as a functional software prototype, leveraging cache level distance between cores to efficiently segregate multiple oversubscription levels. Our evaluation demonstrates that the performances achieved by a non-oversubscribed environment can be replicated in an oversubscribed context, allowing cloud providers to consider the generalization of oversubscription and therefore, drastically reducing the number of servers required to host their customer services and the associated workloads.

We foresee several perspectives for this work, notably on the applied oversubscription templates. While these templates are defined statically in this paper, we believe that they could also benefit from more dynamic tuning approaches, including techniques where the targeted template configuration is driven by performance objectives rather than a fixed ratio. Examining the impact of diverse oversubscription templates based on VM size or distinct premium tiers is also left as a prospect for future work.

Software artefacts

For the sake of reproducibility of the empirical results we shared in this paper, our software prototype is made publicly available.[3] In particular, we documented an offline mode to reproduce all the reported experiments. Furthermore, the daemon used in the experiments to prioritize the cores with lower indexes inside the VMs is also available.[4]

Acknowledgments

This work is supported by the "*FrugalCloud*" Inria and OVHcloud partnership. Additionally, this work also received partial support from the French government through the *Agence Nationale de la Recherche* (ANR) under the France 2030 program, including partial funding from the CARECloud (ANR-23-PECL-0003), DISTILLER (ANR-21-CE25-0022), and SeMaFoR (ANR-20-CE25-0017) grants.

References

[1] 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.

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

[3] C. Lu, K. Ye, G. Xu, C.-Z. Xu, and T. Bai, "Imbalance in the cloud: An analysis on alibaba cluster trace," in *2017 IEEE International Conference on Big Data (Big Data)*, pp. 2884–2892, 2017.

[3] https://github.com/jacquetpi/sweetspotvm

[4] https://github.com/jacquetpi/cpu-staker/

[4] J. Krzywda, A. Ali-Eldin, T. E. Carlson, P.-O. Östberg, and E. Elmroth, "Power-performance tradeoffs in data center servers: Dvfs, cpu pinning, horizontal, and vertical scaling," *Future Generation Computer Systems*, vol. 81, pp. 114–128, 2018.

[5] 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 *Proceedings of the Sixteenth European Conference on Computer Systems*, pp. 1–16, 2021.

[6] A. Fuerst, S. Novaković, Í. 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*, pp. 583–594, 2022.

[7] J. Chen, C. Cao, Y. Zhang, X. Ma, H. Zhou, and C. Yang, "Improving cluster resource efficiency with oversubscription," in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 01, pp. 144–153, 2018.

[8] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous multithreading: Maximizing on-chip parallelism," *SIGARCH Comput. Archit. News*, vol. 23, p. 392–403, may 1995.

[9] OVHcloud, "Who are we?," 2023.

[10] 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 *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, p. 153–167, ACM, 2017.

[11] Alibaba, "The alibaba clusterdata201708 trace data," 2018.

[12] Google, "Borg cluster traces," 2019.

[13] C. S. Wong, I. Tan, R. D. Kumari, and F. Wey, "Towards achieving fairness in the linux scheduler," *SIGOPS Oper. Syst. Rev.*, vol. 42, p. 34–43, jul 2008.

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

[15] 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 *Proceedings of the Sixteenth European Conference on Computer Systems*, EuroSys '21, p. 556–573, ACM, 2021.

[16] F. Wuhib, R. Stadler, and H. Lindgren, "Dynamic resource allocation with management objectives—implementation for an openstack cloud," in *2012 8th international conference on network and service management (cnsm) and 2012 workshop on systems virtualiztion management (svm)*, pp. 309–315, IEEE, 2012.

[17] 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.

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

[19] P. Jacquet, T. Ledoux, and R. Rouvoy, "Cloudfactory: An open toolkit to generate production-like workloads for cloud infrastructures," in *2023 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 81–91, 2023.

[20] 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.

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

[22] 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.

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

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

[25] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proceedings of the Tenth European Conference on Computer Systems*, pp. 1–17, 2015.

[26] G. Da Costa, "Heterogeneity: The key to achieve power-proportional computing," in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pp. 656–662, 2013.

[27] V. Villebonnet, G. Da Costa, L. Lefevre, J.-M. Pierson, and P. Stolf, ""big, medium, little": Reaching energy proportionality with heterogeneous computing scheduler," *Parallel Processing Letters*, vol. 25, no. 03, p. 1541006, 2015.

[28] V. Villebonnet, G. Da Costa, L. Lefevre, J.-M. Pierson, and P. Stolf, "Dynamically building energy proportional data centers with heterogeneous computing resources," in *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 217–220, 2016.

[29] I. Rocha, C. Göttel, P. Felber, M. Pasin, R. Rouvoy, and V. Schiavoni, "Heats: Heterogeneity-and energy-aware task-based scheduling," in *PDP*, pp. 400–405, IEEE, 2019.