Enhancing IaaS Consolidation with Resource Oversubscription

Pierre Jacquet

Enhancing IaaS Consolidation with Resource Oversubscription

Améliorer la sur-allocation des ressources pour une meilleure consolidation des IaaS

***Pierre* JACQUET**

École Doctorale MADIS
Centre Inria de l'Université de Lille
CRIStAL — Spirals & Stack research teams

Thèse présentée et soutenue le 19/07/2024

pour obtenir le grade de docteur en informatique

Directeurs

Thomas LEDOUX	Professeur	IMT Atlantique
Romain ROUVOY	Professeur	Université de Lille

Rapporteurs

Pascal FELBER	Professeur	Université de Neuchâtel
Gaël THOMAS	Directeur de recherche	Inria

Examinateurs

Laurent LEFEVRE	Chargé de recherche	Inria, ENS Lyon
Anne-Cécile ORGERIE	Directrice de recherche	CNRS (présidente du jury)

Résumé

En migrant sa charge de travail vers des centres de données plus grands, le numérique a pu améliorer son efficacité énergétique. La consommation liée à l'augmentation des usages a ainsi été atténuée par de nettes améliorations de l'infrastructure mutualisée (appelée communément *Cloud Computing*), ce qui est visible via des indicateurs tels que le *Power Usage Efficiency* (PUE).

L'infrastructure n'est cependant pas le seul point à optimiser. Le serveur en lui-même, et les tâches qu'il exécute, reste un axe important de la recherche. Le taux d'usage est notamment particulièrement étudié, car sa valeur relativement faible représente un gain potentiel non-négligeable. Ainsi, d'un point de vue énergétique (consommation) et matériel (coût environnemental et financier), l'utilisation d'un serveur chargé à 100% est préférable à celle de 3 serveurs chargés à 30%. Je propose donc d'étudier ces taux d'usages au travers de quatre contributions complémentaires :

1. **La création d'expériences contrôlées réalistes dans un contexte *Infrastructure-as-a-Service* (IAAS)**. Alors que les plateformes supportant les infrastructures Cloud sont particulièrement étudiées, la génération de charges de travail réalistes est primordiale. Chaque Cloud provider ayant ses propres caractéristiques (distribution de tailles de *Virtual Machine*s (VMs), taux d'usage individuels), nous proposons un outil permettant de générer ces charges réalistes.

2. **L'amélioration du calcul du taux de surréservation individuel des serveurs**. En tenant en compte de la stabilité individuelle des serveurs, il est possible d'affiner le calcul de ce taux sans causer de violations supplémentaires.

3. **L'introduction d'un nouveau paradigme de surréservation**. En démontrant tout d'abord que les vCPUs des VMs ne sont pas uniformément utilisés en conditions réelles, nous exposons aux VMs des cœurs de différentes puissances (car surréservés à différents niveaux) et démontrons que ce paradigme peut améliorer les performances globales.

4. **La complémentarité des taux de surréservation pour réduire les ressources non-allouées**. La comparaison des VMs dites premiums et des VMs surréservées permet d'identifier qu'elles tendent à saturer différemment les ressources de leurs hôtes. En les hébergeant sur les mêmes serveurs, il est ainsi possible de bénéficier de synergies, et de réduire jusqu'à 9.6% la taille du parc.

Abstract

By migrating its workload to larger *Data Center*s (DCs), the digital realm has been able to improve its energy efficiency. The consumption due to the increase in usage has thus been mitigated by significant improvements in shared infrastructure (commonly referred to as *Cloud Computing*), which is evident through indicators such as *Power Usage Efficiency* (PUE).

However, infrastructure is not the sole point of optimization. The server itself, and the tasks it executes, remain an important focus of research. Usage rate, in particular, is closely studied because its relatively low value also represents a considerable potential gain. Thus, from both an energy (consumption) and material (environmental and financial cost) standpoint, the use of a server loaded at 100% is preferable to that of 3 servers loaded at 30%. I propose to examine these usage rates along four complementary contributions:

1. **The creation of realistic controlled experiments in an *Infrastructure-as-a-Service* (IAAS) context**. While platforms supporting Cloud infrastructures are extensively studied, generating realistic workloads is crucial. As each Cloud provider has its characteristics (distribution of *Virtual Machine* (VM) sizes and individual usage rates), we propose a tool to generate these workloads.

2. **The calculation of individual server oversubscription ratio**. By considering the individual stability of servers, it is possible to fine-tune the calculation of this ratio without causing additional violations.

3. **The introduction of a new oversubscription paradigm**. By first demonstrating that VM vCPUs are not uniformly used in a real-world context, we expose to VMs cores of different powers (by oversubscribing them to different amounts) and demonstrate that this paradigm can improve overall performance.

4. **The complementarity of oversubscription techniques to reduce unallocated resources**. Comparing so-called premium VMs and oversubscribed VMs identifies that they tend to saturate their hosts' resources differently. By hosting them on the same servers, it is thus possible to benefit from synergies and reduce the number of servers by up to 9.6%.

Acknowledgements

Although a thesis is written in the first person, many people have contributed directly or indirectly to its completion. First, I must thank the jury members, Pascal Felber, Gaël Thomas, Laurent Lefevre, and Anne-Cécile Orgerie, for their time spent evaluating my work and for the quality of our exchanges during the defense. I warmly thank my two thesis supervisors, Thomas Ledoux and Romain Rouvoy, for all our discussions, for their trust, and their encouragement over these three years.

I also thank the members of the *FrugalCloud*[1] challenge, an alliance between Inria and OVHcloud forming the framework of this thesis, with whom I have also regularly exchanged ideas. Among them, I would like to personally thank Germain Masse and Charles Vaillancourt, without their help, the exploitation of production data would not have been possible.

My daily life was also punctuated by numerous discussions (both technical and informal) with the two research teams to which I was affiliated. I thank the members of the Inria *Spirals* and *Stack* teams for these good moments.

Moreover, I benefited from the unconditional support of my family, my friends, and my partner throughout this demanding adventure. Thank you all.

Finally, I must personally thank the team's coffee machine for its admittedly bad but free coffee, and therefore unlimited during certain long evenings.

[1] https://graal.ens-lyon.fr/frugalcloud/

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Motivation

1.1.1 Context

Human activities have significantly harmed the environment, causing a sharp decline in biodiversity over a relatively short geological time frame. This decline is evident through various indicators: the global tree population has halved since the advent of human civilization [1], terrestrial vertebrate populations have decreased by 70% since 1970 [2], and Germany has witnessed a 67% reduction in insect biomass since 2008 [3]. The reasons behind this widespread decline are complex and include factors like agricultural expansion, pesticide use, habitat loss, pollution, and indirect causes such as climate change.

As part of this biodiversity, humans themselves are also affected. Environmental degradation impacts our health, as seen in reduced life expectancies due to air pollution [4], the contamination of rainwater rendering it undrinkable [5], and approximately 30% of the global population being exposed to lethal heat for more than 20 days annually [6].

When examining the rapid pace of these developments, a notable connection can be drawn to global energy production. As asserted by [7], the availability of greater energy resources correlates with increased harvesting capabilities and production (GDP) output. Additionally, *Information and Communication technologies* (ICT) play a significant role in this acceleration by facilitating world-scale communication, enabling complex supply and production chains, providing access to data, etc. [8].

While major changes—i.e., related to the usage—must come from society, research still has a role to play by both describing the problem to actors, as well as reducing the impact of existing infrastructures [9]. In this thesis, I focus on the environmental impact of *Data Center*s (DCs), which is estimated to account for 18% to 41% of the carbon footprint of ICT [10].

1.1.2 Problem statement

Over the past decade, improvements in ICT energy efficiency have been achieved by gradually shifting workloads to larger DCs, where virtualization is extensively used to manage resources rented to clients. I refer to this environment as the Cloud Computing. The workload consolidation allowed by virtualization, along with infrastructure improvements, helps alleviate some of the significant increases in ICT usage [11]. However, there is a natural limit to infrastructure optimization, indicated by a theoretical *Power Usage Efficiency* (PUE) value of 1, which some DCs are approaching [12].

When considering unsustainable resources, a natural objective is to limit as much as possible their use. Products derived from mining are inherently unsustainable because they depend on resources available in finite, non-renewable, quantities. Mitigating the social and environmental impacts of mining can also be challenging. As DCs heavily rely on mined resources, an ongoing objective is to contain their size, specifically the number of *Physical Machine*s (PMs) they use. Additionally, DCs consume a significant amount of power, constituting approximately 1.7% of global energy consumption [11]. In certain countries, such as Ireland, the proportion of energy consumed can be substantially higher, reaching up to 18% of the national consumption.

Going beyond general advice regarding software optimization to more Cloud-oriented recommendations on how to limit the power consumption of Cloud users is challenging due to the virtualized infrastructure. While software impacts hardware component power consumption, its direct consumption is highly variable in a Cloud context, notably due to heterogeneity in the hardware, the kernel, and the collocated workloads.

To illustrate this effect, let us focus on the first aspect: hardware components. I propose to study whether Cloud providers should advise their clients to order small *Virtual Machine*s (VMs) (*consolidating* the workload on a few cores) or larger ones (*spreading* the workload on multiple cores). Let us consider two hardware platforms, one based on an Intel processor and the other one based on AMD, both having the same order of magnitude in their maximum power consumption and the default scaling governor options (in charge of managing core frequencies).

Under the first hypothesis (*consolidating*), we generate CPU load by launching processes that use 100% of a core's time. Under the second hypothesis (*spreading*), we achieve the same CPU load target by launching five times more processes, each using only 20% of a core's time. On the Intel architecture, one can observe that both hypotheses lead to similar power consumption as depicted in Figure 1.1. Paradoxically, in the AMD architecture, consolidating the workload increases energy consumption, as observed in the figure. The delta can be substantial, with a maximum of $200W$ under a 40% CPU overall usage, which is the order of magnitude of current Cloud PM usage [13]. This may be linked to limited support for C-states from AMD (therefore not benefiting from the sleeping phases of unused cores) and more aggressive power consumption under P-states.

In this thesis, I choose to shift from the client perspective to the Cloud provider perspective to be closer to hardware resources. Cloud providers operate clusters of PM that must be distributed between clients. To this end, virtualization may be seen as a way to distribute server resources into smaller,

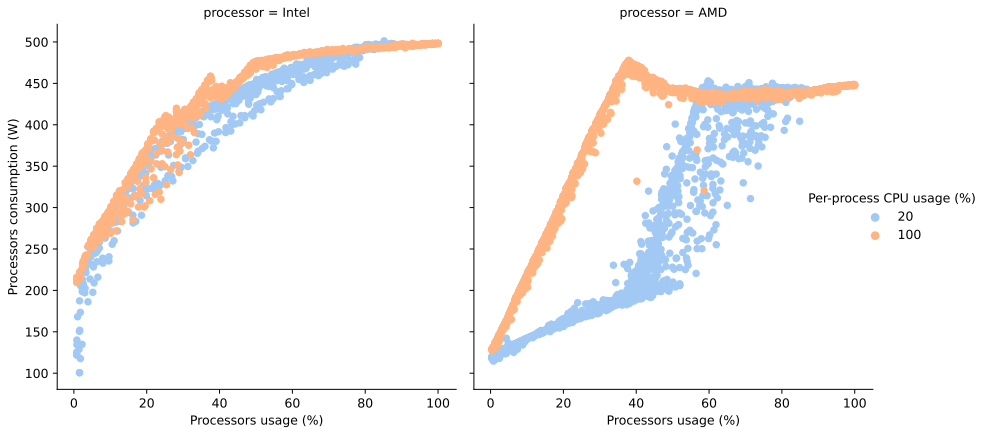

Fig. 1.1 Power consumption of Intel & AMD servers according to core's consolidation strategies

and more practical, units. However, server components are not all distributed in the same manner. Some components have to be *de facto* shared, like the motherboard. Others, like CPUs and memory, can be divided (e.g. core for the first ones, pages for the second one), allowing to split them among clients instead of sharing them.

This thesis aims to **minimize the number of servers** required to operate *Infrastructure-as-a-Service* (IAAS) workloads. To achieve this, effective resource sharing is essential, even for resources that can be partitioned.

Clients typically size their deployments to handle peak loads, which are ephemeral by nature, resulting in over-provisioned scenarios most of the time. For example, in the context of Microsoft Azure, most VMs have an average CPU utilization below 20% [14], leading to a significant proportion of unused resources. To estimate overall idle resources, these *unused* resources must be added to non-allocated resources. 40% of physical machines (PMs) have at least 20% of *unallocated* CPU [15]. This results in DCs operating under low utilization. Idle resources can be seen as *wasted resources*, as failing to exploit them necessitates the provisioning of more servers than theoretically required.

One common way to share resources is to place multiple clients into the same computing units, accounting for the fact that not all the clients need all their resources all of the time. This leads to the practice of offering more virtual resources than are physically available, a technique known as *oversubscription* (or *overcommitment*) [16]. This technique must answer a simple yet challenging question: how many additional virtual resources should be offered? This question is implicitly linked to underlying ones:

1. How can IAAS systems be effectively assessed under realistic operational conditions?
2. Is there room for enhancement in existing oversubscription computation methodologies?
3. Could the oversubscription paradigm in IAAS environments be reimagined?

While reducing the ICT impact requires ongoing efforts from our community, we hope that the following chapters will present interesting leads on how IAAS workloads can be studied and their hosting needs reduced using oversubscription.

1.2 Contributions

This thesis is articulated around four distinct contributions. The first contribution addresses the need for realistic experiments, providing a foundation for the empirical approach used in the subsequent contributions. The second contribution enhances the current oversubscription paradigm by fine-tuning its ratio. The third and fourth contributions explore innovative applications of the oversubscription paradigm.

1.2.1 Improving IaaS experiments using realistic users' behavior

Cloud infrastructures are large-scale and complex platforms designed to host a wide diversity of applications and workloads. Given these complexity and scale factors, simulators and benchmarks are broadly adopted *in vitro* to study their behaviors, prototype new software components and heuristics, and evaluate their effective performances.

However, both state-of-the-art simulations and benchmarks may suffer from a representativeness problem, as the reported results can vary depending on their input workloads. For example, an IAAS platform aims to host VMs, whose characteristics (resource configurations, workload intensity, arrival/departure rate, etc.) can greatly differ depending on Cloud providers and public/private deployments. Addressing this IAAS representativeness thus requires Cloud providers to share production-scale datasets, which might be considered sensitive. Moreover, simulations and benchmarks require a specific experiment scenario that cannot be easily generated from Cloud providers' characteristics.

To address these issues, I introduce CLOUDFACTORY, an IAAS workload generator. This contribution is first composed of a library that can be used by Cloud providers to share IAAS statistics, instead of raw datasets. Then, a generator is introduced to produce realistic VM workloads that match these statistics. CLOUDFACTORY is made available as open-source software that can be adopted by Cloud providers and researchers to foster the evaluation of new contributions.

As an example, I perform an analysis on scheduling choices for different IAAS workload intensities of two different Cloud providers: Microsoft Azure and Chameleon. I also report on OVHcloud statistics computed from CLOUDFACTORY and compare them to other Cloud providers.

1.2.2 Computing oversubscription ratios under stability consideration

Despite continuous improvements, Cloud physical resources remain underused, hence severely impacting the efficiency of these infrastructures at large. To overcome this inefficiency, IAAS providers can compensate for oversized VMs by offering more virtual resources than are physically

available on a host. However, this technique may hinder performances when a statically-defined oversubscription ratio results in resource contention of hosted VMs.

Therefore, instead of setting a static and cluster-wide ratio, this contribution studies how a greedy increase in the oversubscription ratio per PM and resources type can preserve performance goals. Keeping performance unchanged allows this contribution to be more realistically adopted by production-scale IAAS infrastructures. This contribution, named SCROOGEVM, leverages the detection of PM stability to carefully increase the associated oversubscription ratios. Based on metrics shared by public Cloud providers, I investigate the impact of resource oversubscription on performance degradation. Subsequently, I conduct a comparative analysis of SCROOGEVM with state-of-the-art oversubscription computations. The results demonstrate that this approach outperforms existing methods by leveraging the presence of long-lasting VMs while avoiding live migration penalties and performance impacts for stakeholders.

1.2.3 Introducing per-vCPU oversubscription

The adoption of computing resources oversubscription in Cloud environments is conventionally limited to a restricted subset of 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 contribution, I conducted an empirical study of the individual usage of *virtual CPU*s (vCPUs) in the OVHcloud production environment and demonstrated that, as they are not uniformly utilized, the current global approach (where a VM resources are either entirely oversubscribed or not oversubscribed at all) may not be appropriate. Based on these observations, I 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.

I assess the viability of this 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.

1.2.4 Balancing complementary oversubscription levels

Cloud providers generally expose a catalog of various 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 PMs. The latter strategy is often employed to increase platform utilization, as hosted VMs are unlikely to fully utilize all their allocated resources simultaneously [17]. However, managing multiple oversubscribed VM levels introduces complexity for Cloud providers, often leading them to provision dedicated clusters of PMs for each category of offers.

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

1.2.5 Overview of contributions related to oversubscription

A summary of contributions related to oversubscription is presented in Table 1.1. While oversubscription inherently reduces the number of unused resources, SCROOGEVM and SWEETSPOTVM address performance degradation using different approaches. Additionally, SLACKVM demonstrates that oversubscription can also effectively reduce unallocated resources.

Table 1.1 Contributions related to oversubscription

Contribution	Address unused resources	Address unallocated resources	Address performance degradation
SCROOGEVM	Yes	No	Ratio fine-tuning
SWEETSPOTVM	Yes	No	Multiple ratio
SLACKVM	Yes	Yes	No

1.3 List of Scientific Publications

Parts of this thesis are adapted from the following publications:

1. 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)*, (Boston, United States), pp. 81–91, IEEE, 2023.
   Available at: https://hal.science/hal-04168667

2. P. Jacquet, T. Ledoux, and R. Rouvoy, "ScroogeVM: Boosting Cloud Resource Utilization with Dynamic Oversubscription," *IEEE Transactions on Sustainable Computing (TSUSC)*, 2024.
   Available at: https://hal.science/hal-04466538

3. P. Jacquet, T. Ledoux, and R. Rouvoy, "SweetspotVM: Oversubscribing CPU without Sacrificing VM Performance," in *24th IEEE/ACM international Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, (Philadelphia, United States), IEEE, 2024.
   Available at: https://hal.science/hal-04454043

4. P. Jacquet, T. Ledoux, and R. Rouvoy, "SlackVM: Packing Virtual Machines in Oversubscribed Cloud Infrastructures," in *26th IEEE International Conference on Cluster Computing (CLUSTER)*, (Kobe, Japan), IEEE, 2024.
   Available at: https://hal.science/hal-04636648

1.4 Other contributions

This thesis has resulted in the creation of a tool and multiple prototypes. The complete source code for all artifacts has been made available. The list of artifacts is as follows:

- CLOUDFACTORY: This tool is designed to generate realistic IAAS workloads for experiments.. https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://github.com/jacquetpi/cloudfactory
- SCROOGEVM: A scheduler that regulates per-server oversubscription ratios by evaluating the quiescent state of resources https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://github.com/jacquetpi/scroogevm
- SWEETSPOTVM: A local scheduler that exposes cores with varying performance levels to VMs https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://github.com/jacquetpi/sweetspotvm
- SLACKVM: A local scheduler that accommodates VMs with different oversubscription ratios on a single server https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://github.com/jacquetpi/slackvm

In complement, a press article has been written and published online:

- P. Jacquet, T. Ledoux, and R. Rouvoy, "La chasse au gaspillage dans le cloud et les data centers," *The Conversation, France*, 2023.
  Available at: https://theconversation.com/la-chasse-au-gaspillage-dans-le-Cloud-et-les-data-centers-196669

1.5 Outline

This thesis is structured as follows. I first discuss the background and related work to this thesis in Chapter 2. Chapter 3 elaborates on the generation of realistic IAAS experiments. This involves considering client behaviors, including selecting VM sizes and resource consumption patterns, to evaluate IAAS architecture choices from the perspective of Cloud providers in a controlled environment. In Chapter 4, I advocate for oversubscription ratios being computed closer to the individual usage of resources by evaluating the quiescent state of servers. In Chapter 5, I report on the individual vCPUs usage being made in real-world scenarios. Given the heterogeneity in vCPU usage, the proposal involves exposing cores of varying performance levels to VMs. Chapter 6 evaluates the limiting resources for additional deployments in IAAS environments. It identifies that the limiting resource depends on the level of oversubscription and suggests hosting differently oversubscribed VMs in the

same cluster of PMs to improve packing. Lastly, Chapter 7 concludes this thesis by summarizing the contributions and proposing research perspectives.

Chapter 2

Background

In this chapter, I focus on IAAS resource usage-related work. I begin by discussing relevant IAAS datasets in Section 2.1. In Section 2.2, I delve into the generation of IAAS workloads for controlled experiments, as this thesis heavily relies on empirical approaches. In Section 2.3, I examine the behavior of generic schedulers. Finally, in Section 2.4, I explore methods to improve packing efficiency beyond scheduling.

2.1 IaaS context

IAAS services are widely adopted hosting solutions where the Cloud provider oversees the hardware infrastructure and provides clients with VM, leaving them responsible for managing the software stack, including the operating system, middleware, and applications. In 2017, Azure released a dataset containing cluster traces to facilitate the study and understanding of these environments' characteristics [23]. It includes, amongst others, VM memory and vCPU configuration options distribution, arrival rate patterns, VM lifespan, and CPU usage daily pattern. This dataset stands as a valuable contribution, given its status as one of the only available public IAAS datasets from a large Cloud provider.

These traces underline that resource usage is not optimal. The average CPU usage of VM is notably low, with 60% of VMs exhibiting an average CPU utilization below 20%. This is also confirmed by data from privately operated Cloud environments, with Twitter scheduler Mesos reporting a similar order of magnitude on the CPU and an average memory usage below 50% [24]. It's worth noting that a significant proportion of these usage patterns are stable and not periodic, which contradicts the assumption that most Cloud-hosted applications are interactive [25].

In addition to considering the usage of allocated resources within VMs, it is also important to recognize that a notable portion of resources also remains unallocated within a cluster. Azure's findings indicate that at least 20% of CPU resources are unallocated on more than half of the servers [15]. Furthermore, the percentage of unallocated memory increases to 40%. It's worth noting that this

unallocated CPU and memory may not necessarily occur on the same servers, resulting in scattered resources that are unusable by traditional deployments.

In this thesis, the term "Cloud workload" refers to the usage patterns of clients on the platform, encompassing actions such as VM deployment (with client-specified sizes), VM resource utilization, and VM deletion. Cloud workloads are dependent on clients' behaviors and may significantly differ from one provider to another due to their specific contexts. For instance, research-oriented Cloud operator CHAMELEON [26] published 6-month traces in 2020 where half of their deployments imply VMs having at least 4 vCPUs, while Azure reports that less than 2 vCPUs VMs are required in 80% of their deployments. This discrepancy notably impacts server packing strategies and may consequently result in increased levels of unallocated resources within the context of Chameleon.

Besides IAAS services, other services offer a broader management scope. *Platform-as-a-Service* (PAAS) and *Container-as-a-Service* (CAAS) computing, for instance, extend their management to include the guest operating system. In this thesis, these services are seen as a specialization of the IAAS paradigm, often internally built upon it. Taking the more specific context of co-located ephemeral jobs, traces from Alibaba [27] and Google Borg [28] can be studied. Despite also reporting under-optimal allocations (e.g., Alibaba's PMs utilize on average 40% of CPU and 60% of memory [29]), generalizing their characteristics to a broader context is challenging due to the typically short lifetimes of tasks in their workloads compared to VMs.

2.2 IaaS experiments

In the pursuit of enhancing the usage of Cloud platforms, experimentation plays a vital role in assessing hypotheses, evaluating system performance, and simulating potential gains. In the context of Cloud computing, state-of-the-art benchmarks typically rely on the combination of two key components. First is the *System under Test* (SUT), which is derived from production environments, and can either be a copy of an existing system, considered representative of real ones, or a simulator. Second is the input workload, generally described as the usage pattern submitted to the SUT. I now discuss common SUT studied in a Cloud context, while input workload is later discussed.

2.2.1 Platforms used for experiments

When evaluating Cloud-based systems, researchers may adopt two complementary approaches. The first involves *simulations* to synthesize various system configurations and workloads, thereby avoiding the cost of resource-intensive and time-consuming real-world experiments. The second approach involves *benchmarks*, in which real-world workloads are applied to a system under test, which may be either a generic system or a copy of an existing one. Unfortunately, both of these approaches often face a representativeness challenge, where injected workloads must be as faithful as possible to real-world conditions. Cloud metrics reported by practitioners are a key insight in this regard.

This section outlines the state-of-the-art for Cloud simulators, *System under Test* (SUT) benchmarks, workload representativeness, as well as relevant Cloud datasets.

Cloud simulators

CLOUDSIM [30] is an extensible software framework to simulate a Cloud-based infrastructure. Its flexible nature enables users to create customizable platforms under various workload scenarios, such as VM (Virtual Machine) distribution, arrival rate, CPU usage, etc. CLOUDSIMPLUS [31] was introduced as a fork, enhancing its capabilities. Crossref[1] records indicate that the original framework is widely adopted by the research community, with over $2,700$ citations.

While other Cloud DC simulators exist, like closed-source MDCSIM [32], simulators expect developers to encode synthetic deployment scenarios rather than providing guidelines on realistic Cloud workloads. For instance, the MDCSIM validation protocol relied on a custom scenario to compare its resource utilization to a physical infrastructure hosting a web-oriented application under a specific workload. Similarly, GREENCLOUD and CLOUDSIM used specific evaluation prototypes as examples.

SIMGRID [33] is a more generic solution that can be applied to the Cloud via its S4U interface. It can run a VM-based scenario and evaluate scheduling decisions on physical resources. VMPLACES framework [34] leverages it to inject realistic VM usage. However, the framework does not consider VM lifespan and arrival rate.

Prototype under test

Application under Test *Hosted applications* are the smallest SUT unit in a Cloud experimentation [35]. Metrics of interest include response time, request rates, errors, etc. [36].

Due to the wide adoption of Cloud infrastructures, many types of applications can be hosted. A non-exhaustive list of legacy applications includes relational database structures proposed by TPC [37], CPU-intensive tasks, such as those provided by SPEC CPU [38], mail server [39], and others.

Additionally, web-oriented architectures are also frequently considered and tested using various projects, such as the OLIO monolithic application [40] and other microservice architectures implemented in the DEATHSTARBENCH [41]. Key-value stores are also widely deployed and can be evaluated using the YCSB benchmark [42].

VM under Test Assessing VM performance for a given application is beneficial to Cloud clients. In this configuration, the application and the workload generator may not be co-located, to evaluate the platform-induced latency.

The choice between hosting options can be made by evaluating the balance between performance and cost for various VM sizes and providers. Work targeting providers benchmark includes

[1] https://www.crossref.org/

MOSAIC [43, 44], CLOUDCRAWLER [45] framework, and C-METER [46] where the monitored application can be configured. Other approaches compare application metrics on provided benchmarks [40, 47, 48].

Platform under Test Cloud platform consists of physical servers, known as *nodes*. These nodes can be organized as *clusters*, which are groups of servers used for a specific Cloud workload, such as hosting VM with the same premium policy. Conducting experiments at the scale of a node can provide significant benefits for both providers and researchers, as it can evaluate various platform configurations and architecture choices. In the context of multiple VMs, the node represents the smallest SUT where deployments can be studied. In this context, a deployment is defined as the provisioning of a VM on a node according to its requested resources.

SPECVIRT benchmark [49] offers a means of measuring end-to-end performances, including hardware, virtualization platform, and guest operating system on various application types. While being behind a paywall, they also do not target workload representativeness. They use an increasing workload until *Quality of Services* (QoS) is violated.

Another available option is the VMMARK benchmark, provided by VMWare [50]. This benchmark deploys simultaneously 6 VMs hosting different workloads: mail server, java server, idle workload, web server, database server, and a file server before evaluating their performance under stress. They do not seem to take into account larger deployments, VM lifespan, departure, and arrival rates.

The management complexity associated with studying multiple Cloud nodes has led to the development of dedicated frameworks. A prominent example is CBTOOL [35], which has become the *de facto* standard for describing an IAAS scenario through a given number of nodes, VM, and usage parameters. Interpreted scenarios can be applied to both private clusters and public providers. Single-node experiments are also an option.

2.2.2 Input used for experiments

Cloud-like workloads applied to SUT aim to be as representative as possible. Two strategies are typically used: *replaying recorded traces* or *using models*.

Replaying traces [51, 52] involves resubmitting production requests, which provides a comprehensive representation of the workload, but may not be available in all cases.

Alternatively, workload models can be used to represent an input workload behavior. Basic usage may rely on a targeted rate (requests per second), while other generators include patterns and burst mechanisms [53].

Workload representativeness is typically focused on the application or VM level. To the best of our knowledge, few previous works provide workloads at the node or cluster scale, even if they introduce additional metrics, such as VM arrival rate, VM lifespan, used resources, etc. One exception is [54], which implements a specific workload scenario on CBTOOL, although the adopted parameters are not publicly communicated due to a paywall.

2.3 IaaS scheduling

Reducing the cluster size involves adapting how resources are allocated and shared between deployment units. In an IaaS-like environment, resource management occurs at two levels. First, I discuss PM selection during VM deployment. Then, I detail how resources are scheduled within a PM after deployment.

2.3.1 Orchestrators

Cloud clusters are commonly composed of thousands of servers, with non-uniform configurations. Selecting an appropriate host for a given VM task is the responsibility of a component, called an *orchestrator*, in most distributed system architecture. By extension, this is also the component refusing to deploy more workload on a given host when its allocation is considered full.

The selection of an appropriate PM for a given VM deployment is often framed as a *Vector Bin Packing Problem* [55]. 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 [56] and, over the years, numerous heuristics have been proposed to tackle VM scheduling cases [57]. While *First Fit Decreasing* and *Best Fit Decreasing* algorithms may be used to address them, Cloud providers must however account for more parameters than just the resources request.

State-of-the-art Cloud orchestrators typically use a score-based mechanism to select the most appropriate PM. I focus on schedulers used at large scale (>1000 nodes) hosting generic workloads (not tailored to a single framework).

Google Borg

Google's internal workloads are managed by a monolithic orchestrator called Borg [58]. Both production and non-production workloads are co-located in the same cluster of machines. The scheduler is a centralized application that manages both long-running workloads and more ephemeral jobs. Borg utilizes a very fine-grained resource reservation system, using units such as milli-cores and bytes.

Jobs are hosted without virtualization, while tasks are isolated from one another using cgroups functionalities, making them akin to containers.

The scheduler asynchronously fetches a queue containing submitted jobs and scans it from high to low priority, with a round-robin scheme within each priority level to ensure fairness across users and prevent head-of-line blocking. A job deployment first involves the finding of a list of suitable PMs, matching those with sufficient available resources and meeting potentially specified constraints. The selection of suitable PM is determined by applying a list of filters. Each filter tests a specific property, resulting in a boolean value that determines whether a host is accepted or rejected based on hard criteria.

A score is then computed for each suitable PM, taking into account various factors such as the preemption of existing services, prioritizing jobs of different priority on PMs, and the pre-existing installation of software dependencies for the given job (soft criteria). While details on the score computation are not provided, it is described as a spreading-oriented strategy aimed at avoiding scattered resources caused by one computing resource being fully allocated before others.

To speed up the PM selection, only a random sample of PM is chosen for the scoring step. Additionally, Borg caches computed scores until the state of the PM changes (e.g., a job ends). Finally, when submitting identical jobs, only one score is computed.

Due to its traces being published since 2009, Borg's scheduler is widely studied by the community.

Azure Protean

Azure employs a scheduler named Protean [59] to orchestrate services in its Cloud offering. It also features a filtering and scoring mechanism. However, since Azure DCs consist of clusters composed of homogeneous PMs, the filtering and scoring mechanism is divided into two steps, initially focusing on selecting suitable clusters. The cluster filtering process considers factors such as hardware features (e.g., GPUs), while the scoring process for the suitable clusters takes load considerations into account.

The top n clusters are then retained, and the list of PMs within these clusters undergoes a second round of filtering and scoring at the PM scale under distinct rules. This already deduced subset is the random sample of Borg.

Cached results of rule computations are stored for each computed machine/VM-type tuple to expedite subsequent scheduling tasks. When multiple similar VM types are deployed within a single client deployment (a tenant), a single score is computed.

In 2020, there were approximately one hundred filtering and scoring rules. On average, the overall scheduling decision takes around 20 milliseconds.

OpenStack Nova

OpenStack Nova is an open-source orchestrator capable of managing bare-metal servers, VMs, and containers. When selecting a physical machine (PM), Nova employs a filtered and weighted scheduler, as described in [60].

Given a VM deployment, the default PM filtering criteria encompass factors such as adequate RAM, disk, CPU resources, desired hardware specifications, geographical location, and the types of instances already hosted. This filtering list can be customized to meet specific requirements. Additionally, the scheduler maintains a cached list of available hosts, which is periodically updated, to expedite the selection process.

The selection of a PM is based on an associated weight assigned to each suitable PM. Default weights include the quantity of available RAM (leading to spreading the workload) and light I/O workload. To reduce the likelihood of returning the same PM for multiple simultaneous deployments, a small random factor is introduced. Among the top n best PMs, one is selected randomly.

Each time the scheduler selects a host, it virtually consumes resources on it, and the weights for subsequent selections are adjusted accordingly.

Alibaba Fuxi

Alibaba uses another monolithic scheduler known as Fuxi [61] to manage its distributed workloads. Distributed frameworks (such as Hadoop and MPI) request resources using ScheduleUnit, which represents CPU and memory quantities. These requests may be partially accepted by the scheduler based on cluster resource availability. If not all requested ScheduleUnits are allocated, pending resources are placed in a queue.

When resources are deallocated from other jobs, the orchestrator checks the waiting queue and selects a pending workload based on its priority and waiting duration. Server preferences can be specified in the ScheduleUnits and are internally managed by having a per-server queue and a per-rack queue, organized through a tree structure under the cluster queue.

Apache Mesos

The Mesos scheduling mechanism [62] also has the objective of hosting multiple distributed frameworks. It allows concurrent usage of multiple schedulers (one per framework). A centralized resource allocator monitors the available resources in the infrastructure and offers them to the framework scheduler, which then selects a subset in the offer based on its needs and constraints.

This architecture, referred to as a two-level scheduler, aims to keep Mesos' complexity low, as it does not need to know any characteristics of the frameworks' needs. Once a framework scheduler accepts a resource offer from the Mesos master, a framework executor is instantiated on the appropriate slave with the requested resources (CPU and memory). Isolation between executors on slaves is primarily based on Linux containers.

YARN

Similarly to Mesos, YARN is also a two-level scheduler that allows multiple distributed frameworks to use a single cluster [63]. However, instead of using its offer-based mechanism, YARN enables more direct communication between the cluster manager and its framework managers.

Using a tick-based approach, applications are expected to periodically send a request for their resource needs to the centralized cluster manager. If the cluster manager can fulfill this request based on availability and scheduling policies, resources are allocated through a lease mechanism. These resources are then parsed by the application on a later call and used for its job deployment through a container. The application is then expected to update its request for resources on the next tick. Scheduling policies in YARN can be adapted but may include fairness (ensuring every application gets a fairly equal share of resources), capacity (managing multiple queues, each having a specific fraction of cluster resources), or a *First-In First-Out* (FIFO) queue.

Omega

Omega [64] also aims to host multiple distributed frameworks. However, unlike Mesos and YARN, it does not rely on a two-level architecture. Instead, Omega exposes resources to framework schedulers as a view of the cluster state. This view is frequently updated, and any framework scheduler can claim part of the available resources in the cluster. Conflicts are managed using a transaction mechanism on claims.

Shared-state schedulers like Omega can improve utilization compared to two-level schedulers such as Mesos, which only expose a partial view of resources, but may encounter "shadow" resources problems (resources free between updates on the view) that are not negligible [65].

Swarm

Docker Swarm serves as an orchestrator for managing Docker containers. Its scheduler module operates in two steps [66].

Initially, a filtering step identifies a list of suitable servers for deployment. Subsequently, the scheduler selects the least loaded server using the default strategy (known as the spreading strategy). Additionally, two alternative strategies are available: binpack (which prioritizes nodes with fewer available resources) and random selection. Its behavior is kept simple, without caching mechanisms, as it is not intended for use in production environments.

Kubernetes

Kubernetes (K8s) is often viewed as the evolution of Google Borg [58]. Resources are managed through scheduling units called pods, which can contain one or more containers.

Its architecture can be considered centralized, with a master node receiving pod requests and transferring them to the scheduler (a pod running on the master), which is responsible for selecting an appropriate host.

The scheduling process in Kubernetes involves several steps. Firstly, a filtering step selects nodes that meet the pod's resource requirements and precise labels (e.g., port availability, disk types) using predicates. Next, candidates are assigned a score based on various criteria. Examples of criteria include balancing resource usage between nodes, distributing pods belonging to the same service among different hosts, and favoring nodes that match a pod affinity list. By default, Kubernetes uses a "least used node" criterion to assign scores. Finally, the node with the highest score is selected to host the pod.

Summary

As described in Table 2.1, heterogeneity in scheduling mechanisms is primarily observed in distributed frameworks, such as Fuxi, Mesos, YARN, and Omega. However, the lack of guarantees on resource allocation makes these frameworks less suitable for Cloud computing, where clients expect their

Table 2.1 Considered orchestrators

Orchestrator	Scope	Selection mechanism
Google Borg	Containers	Score-based scheduler
Azure Protean	VMs	Score-based scheduler
OpenStack Nova	baremetal and VMs	Score-based scheduler
Apache Mesos	Containers	Two-level offer-based scheduler
Alibaba Fuxi	Containers	Two-level request-based scheduler
YARN	Containers	Two-level request-based scheduler
Omega	Containers	Shared-state scheduling
Swarm	Containers	Score-based scheduler
Kubernetes	Containers	Score-based scheduler

requests to be fully met, with lower needs for dynamic scaling. In the Cloud context, score-based mechanisms are the de facto standard due to their simplicity and customization capabilities.

On dynamic approaches

While scheduling has been primarily discussed in this section from the static initial PM selection, it's worth noting that VMs can also be migrated during their lifetime. However, large-scale VM migrations are not commonly performed. For example, Google primarily considers them only to mitigate both hardware and software downtime [67]. According to [68], VM migrations lead to slowdowns and require additional computing resources to perform the copy, leading to an interest in avoiding them as much as possible.

2.3.2 Host internal scheduling

Beyond cluster orchestration, computation is also managed locally, inside a PM, involving a local scheduler. A PM is composed of resources that need to be shared among software applications.

Modern x86 server architectures typically feature pairs of cores that share the same functional units, such as *Arithmetic Logic unit* (ALU) and *Floating-point unit* (FPU) thanks to hyper-threading capabilities, also known as *Simultaneous Multithreading* (SMT) [69]. Data access is organized through different levels of cache: the lower the cache level, the faster it is, and the higher the cache level, the larger it is. *Last Level Cache* (LLC) is therefore the largest. Cores within the same pair share all their cache levels. In the common (*Non-Uniform Memory Access* (NUMA)) architecture, a pair of cores belong to a given NUMA node. While cores belonging to a given NUMA typically share the LLC with other cores, it may not be shared between all cores, especially on large EPYC architectures. A server may have multiple NUMA nodes, each having access to its specific main memory resources.

Modern CPUs are sophisticated pieces of hardware, with flexible performance capabilities. Each core may adapt its operating frequency using *Dynamic Voltage and Frequency Scaling* (DVFS) to either decrease its energy consumption (using C-state mechanism) or enhance performance (using P-state mechanism).

Non-x86 architectures, such as ARM and RISC-V, also exist in DCs, although they remain relatively marginal and were not explored in this thesis.

I now briefly discuss how modern OS schedulers operate to handle this hardware complexity. The software operates using multiple instances, known as processes. Each process utilizes at least one thread, which is responsible for executing instructions. The scheduler acts as a bridge between the computing resources—i.e., the cores—and the threads, determining which thread should be allocated CPU time. In this thesis, I concentrate on the Linux scheduler because of its widespread usage in DC management.

EEVDF-based scheduler The *Earliest Eligible Virtual Deadline First* (EEVDF) scheduler [70], introduced in Linux with kernel version 6.6, is the default scheduler since 2023. EEVDF belongs to a scheduling class referred to as fair scheduling. Its primary objective is to allocate a proportional share of CPU time among processes.

It adopts a distributed architecture where each core is assigned an individual queue. Each thread in the queue is assigned an individual time slice reference accounting for the number of threads in the queue and the thread priority. When a thread's running time slice expires, the scheduler compares the thread's initial time allocation to the duration it received. The difference between these values, known as lag, is stored in a tree structure for efficient access and manipulation.

With a positive lag, the evicted thread did not receive completely its timeshare. When a new thread needs to be selected to run, either due to an expired time slice or the current thread relinquishing its core, only threads with a positive lag (0 or higher) are considered eligible for execution. Threads affected by negative lag become eligible again after a specified time, referred to as the eligible time. Among the eligible threads, the thread having the earliest virtual deadline (calculated as the reference time slice plus the eligible time) is chosen to run.

This behavior differs from the previous default Linux scheduler, *Completely Fair Scheduler* (CFS), which elected thread was based on the shortest *vruntime* (last usage being made by the thread weighted by its priority). The priority, expressed by the *nice* value, indicates how long time slices should be attributed compared to other threads. However, a piece of missing information was how frequent time slices should be affected (reflecting latency requirements), expressed by the newer *latency-nice* metric. *latency-nice* is easier to take into account with EEVDF, by reducing the next allocated time slice and therefore its virtual deadline. It's important to note that while this adjustment affects the fragmentation of CPU time, the overall duration of CPU time allocated remains unchanged.

Both CFS and EEVDF per-core queue architectures imply balancing mechanisms to handle asymmetric loads. Firstly, at thread creation, the scheduler typically places the thread on the least busy core. Secondly, when a core becomes idle (i.e., its queue is empty), it will "steal" threads from other cores' queues, starting with the closest ones. Lastly, the load of each queue is periodically assessed, and if a certain threshold is exceeded between domains (with a lower threshold for SMT cores compared to cores in different NUMA nodes), load balancing is triggered.

Other schedulers While EEVDF serves as the default scheduler in Linux, prioritizing fairness for general-purpose software, Linux also offers other scheduling policies, particularly for real-time applications where fairness is not the primary concern. Threads can change their scheduling policy through a system call, typically requiring root permissions in most systems.

In practice, Linux determines a priority order for scheduling policies. When a thread needs to be scheduled, it interrogates policies in the following order:

1. Deadline Under this scheduling class, Linux uses the *Earliest Deadline First* (EDF) policy [71] to manage periodic processes with real-time requirements. Each process must specify an estimation of its duration, a deadline, and a window period. For each window, the process must be executed once before the deadline relative to the start of the window. When a task is submitted, the scheduler performs an admittance test to verify the feasibility of the requirements with other threads.

2. Real-time Under this scheduling class, Linux checks if a thread asks for a FIFO scheduling (*SCHED_FIFO*) or a *Round-Robin* (RR) scheduling (*SCHED_RR*). With the FIFO, the thread having the highest priority is executed until it releases the CPU or until a higher-priority thread arrives in the queue. This means that CPU blocking is possible. With RR scheduling, the FIFO principle is extended with time slices consideration. When a time slice is expired, the process is moved to the end of the queue.

3. Fair Under this class, referred to as *SCHED_OTHER* in the manual, EEVDF is used. In practice, most workloads rely on this category.

4. Idle The tasks in this policy will only be run if no other thread is runnable.

2.4 Improve packing beyond orchestration

While a cluster orchestrator's role may be to minimize the number of unallocated resources on a cluster, the packing problem does not by itself fully leverage all server resources. Real-world scenarios demonstrate that resources assigned to clients remain underutilized, rendering packing mechanisms reliant solely on VM size inadequate.

While under-utilization of Cloud platforms has a significant operational cost for Cloud providers, low resource utilization also corresponds to the least-energy efficient range of operation for a PM [72]. Therefore, even if energy proportionality kept improving over the last decade [73, 74], resource under-utilization inevitably imposes higher power consumption and hardware costs [14].

In the literature, several elements contribute to achieving greater usage of IAAS platforms. Let's delve into these contributing factors.

2.4.1 Evictable VMs

The use of ephemeral workloads may be adopted to leverage resources seen as available for a short period of time. Spot VMs (also referred to as *preemptible* VMs) are VMs that may be terminated due to providers' requirements [75–77]. These VMs are typically of small size to be deployed on servers with spare resources. They are considered lower priority compared to regular VMs and may be interrupted and relocated elsewhere if their resources are needed for a conventional deployment. Users are typically given advance notice of impending interruptions, albeit with varying delay times (from seconds to minutes) depending on the provider's policies. Spot VMs are offered at a reduced price compared to other VMs due to their transient nature and are only suitable for fault-tolerant workloads.

2.4.2 Harvesting VMs

Spot VMs having a dynamic size were also proposed and used in Azure's context [78]. This type of instance—referred to as Harvesting VMs—can retrieve resources deemed unused from conventional deployments. Again, no guarantees are given on their allocation as its size may be adjusted at any time to avoid perturbing other VMs. Resources considered include CPU [78], memory [15], and storage [79]. These instances are particularly suitable for FaaS computing, where most computing tasks are lower than 30 seconds, due to the advance notice given by providers before an eviction, allowing to finish the function execution in most cases [80].

SmartHarvest [81] is based on the evaluation of available CPU time from the premium VMs in the very near future. It does so by retrieving a small set of statistics related to CPU usage from the premium VM used as a feature to train a supervised classification algorithm. The classification, used to attribute the number of cores required (unit-based) is cost-sensitive, meaning that under-predictions are penalized by construction to avoid them. The online learner is composed of linear models to be lightweight, allowing evaluation in the order of 10ms windows. Once the core needed for premium VMs in the next time window is predicted, the predicted unused resources are attributed to a so-called ElasticVM.

Memory harvesting mechanism from [15] focused on unallocated memory resources. The *hot-plug* and *hot-unplog* mechanisms are used at run-time to modify the guest's physical address space. While adding memory to a VM is straightforward, shrinking is performed by talking to a guest agent performing the hot-remove actions. Part of the unallocated is also unleveraged to reduce the deployment time of new VMs and NUMA spreading.

Disk harvesting was introduced as an extension to HarvestVM in [79] using programmable SSDs. By managing the internal SSD block attributions, BlockFlex predicts the SSD channel bandwidth and cell quantities associated with traditional VMs, monitors the unallocated resources, and tries to satisfy the projected needs of SpotVMs. Predictors are based on LSTM models and consider time windows of 3 minutes. Per channel blocks, allocation is performed through so-called *vSSD* (for premium VMs)

and *gSSD* (for harvesting-based VMs) managing the underlying hardware mapping complexity. On the host, the VM's virtual disk size is adjusted accordingly.

GPU harvesting was also explored in Cloud Gaming [82], a paradigm where players access GPU resources remotely to run games. By first reporting on low GPU usage being made by recent games, Azure proposes to harvest resources between frames computation. A game running at 60 frames per second results in a computation each 16.67ms during which not all resources may be fully utilized. The paper suggests using a notification of rendering completion, obtained by instrumenting graphic libraries, to determine the available time before the next frame computation. This time can then be used to schedule deep-learning tasks. Deep learning workloads typically consist of small training kernel times, usually less than 1 millisecond, which can be easily predicted, making them good candidates for collocation.

2.4.3 Disaggregated resources

Another prominent area of research focuses on the capability of a VM to utilize resources from multiple PMs. This method allows for the creation of larger VMs [83], and also addresses resource fragmentation by harnessing scattered resources within the cluster [68]. Disaggregated resources can occur at either the rack scale [84] where hardware is physically close to each other, or at a DC scale [85]. Access to disaggregated resources, relying on network connectivity, favors certain resources like storage for disaggregation due to their higher tolerance for latency in accessing data [84, 85]. Memory can also be a candidate [86, 87], with potential adaptation from the software stack running inside the VM [88, 89]. Memory also presents itself as a potential candidate for disaggregation [86, 87], with potential adaptations required in the software stack running within the VM [88, 89].

However, due to the overhead involved, disaggregated resources may not be suitable for all types of workloads [68]. A proposal was also made to "disaggregate" software by decoupling its code according to resource types [90], to facilitate resource scheduling within the cluster.

2.4.4 Oversubscription techniques

The oversubscription (or overcommitment) term refers to the possibility of deploying more virtual resources on a server than physically available. As previously described, Cloud clients tend to oversize their VM [23] to be able to handle load spikes, failover scenarios, and growth in demand. One consequence is that not all clients need all their resources all the time, leading to consolidation opportunities. Unlike other approaches described in this section, oversubscription allows consolidating the existing workload (composed of VMs in our context), instead of introducing new ones to "fill the gaps".

Software support for oversubscription

Exceeding the quantity of physical resources is achieved through various techniques, depending on the type of resources being considered.

CPU When considering the oversubscription of CPU resources, the number of vCPUs allocated to VMs can exceed the number of physical CPUs as each individual vCPU has its associated thread on a QEMU/KVM environment. The host scheduler (typically EEVDF) manages the sharing of CPU between vCPUs just like any traditional process. Increasing the CPU usage may lead to VM performance degradation due to competition on cache or time slice resources, but cannot kill a process by itself.

Memory Memory oversubscription leverages the virtual memory feature from the Linux memory management subsystem. Each process (including ones associated to hosted VMs) has a personal virtual address space. Virtual pages are only allocated in memory when accessed, a technique referred to as "demand paging". For a given VM, its host memory usage is defined by *Resident Set Size* (RSS), the amount of physical memory allocated can therefore be substantially inferior to its request. Additionally, the RSS can be reduced closer to the *Working Set Size* (WSS) (the amount of physical memory being actively used) using different techniques.

First, WSS estimation may rely on different techniques, such as [91], which periodically invalidates a set of pages to trap their access at the hypervisor level. Others rely on a ballooning mechanism, which inflates a driver in the VM to fill the memory until the VM starts using its swap partition [92]. Different techniques also attempt to estimate the optimal WSS when the VM is under-provisioned, using a dedicated cache [93, 94].

Once WSS is estimated, the ballooning driver can return to the hypervisor-occupied virtual page addresses on the VM for further usage within the host scope. In the context of virtualization, *hotplug* can also be used to adjust the memory capacity at runtime [95].

While dynamic memory management reduces (ballooning, hotplug) or increases (hotplug) the RSS of VM and other mechanisms can also impact its host memory consumption. For example, *Kernel Samepage Merging* (KSM) allows VMs to share common pages [96] with a daemon that periodically scans the allocated pages and merges the identical ones into a single read-only page. If a process updates this page, KSM duplicates it into its original form. ZRAM [97] can also be used to reduce a process RSS by creating a set of virtual disks. Pages written to these disks are compressed using a run-length encoding algorithm, trading CPU cycles for memory.

Others Other resources may be oversubscribed in a Cloud context, such as network and power. The network was not investigated in this thesis as I do not see it as a common bottleneck in Cloud infrastructure. Power was also not investigated as it is not a resource directly ordered by clients.

Oversubscription ratio

Most hypervisors enable oversubscription by allowing the sum of all allocated virtual resources to exceed the PM capabilities [98–100]. To avoid performance degradations, oversubscription is usually limited using a maximum ratio between the number of exposed virtual resources and the available physical resources. For example, a CPU oversubscription of 2:1 refers to the possibility of deploying a maximum of 2 vCPUs per physical core.

Table 2.2 Review of oversubscription ratio computations

Name	Scope	Type	References
Default	Cluster	static	[58, 101]
i-porter	PM	static	[59]
Resource Central	PM	dynamic	[14]
N-sigma	PM	dynamic	[17]
Maximum	PM	dynamic	[17]
DOA	PM	dynamic	[102]
CloudVamp	PM	dynamic	[103]

Defining this limiting ratio is a complex task, as it can be determined at different scales and based on various approaches. Table 2.2 provides a brief overview of oversubscription ratio computation techniques. The following paragraphs review these techniques based on the two scopes currently used to define an oversubscription ratio: the cluster level and the PM level (where dynamic computations are possible).

Oversubscription ratio at the cluster level Most IAAS cluster managers, like BORG [58] and OPENSTACK [101], propose to define the oversubscription ratio at the cluster level, using a value applied to all PMs. This approach has the advantage of simplicity, due to both, a single parameter to set, and to avoid the dependency on a monitoring infrastructure as the resource usage is not taken into account.

In practice, industrials may search for exhaustively a cluster oversubscription ratio that maximizes the used resources while trying not to violate the constraints [104], something that can be searched using a grid search approach [105].

In [106], the authors propose to consider the oversubscription ratio computation based on a chance-constraints problem. Chance constraints are a modeling tool used to take into account risks (in our context, a PM load reaches its full capacity) and constraints (this risk should be below a specified probability). While this risk is influenced by the PM size (the larger a PM is, the more pooled the risk is due to the variety of jobs), defining risk is seen as a convenient proxy to manipulate for Cloud providers.

Oversubscription ratio at the server level The oversubscription level can also be defined per PM either statically or dynamically.

Static sever ratio As a Cloud DC typically has different hardware generations [59], therefore having heterogeneous PM configuration, per-PM oversubscription ratio allows to account for their performances. In [107], authors adapted OpenStack Nova to be able to set a static ratio per PM, accounting for their virtualization capabilities (the more powerful a server is, the more oversubscribed it can be). The exact oversubscription computation is not discussed but is likely to have been manually tuned.

Dynamic server ratio: peak driven computation Per-PM oversubscription ratio also allows to account for a specific PM workload. In that case, oversubscription is assessed by studying internal metrics and is dynamically adjusted.

The most common way to implement dynamic oversubscription is to first predict what resources will be used in the future, and consider the delta between this prediction and the PM configuration as available for new deployments. This prediction is usually pessimistic as VM tends to be long-running workloads (in the sense that their lifetime is not conditional to a specific process termination), resources must therefore be available in the long run to avoid violations.

Azure proposes to use VMs percentile to compute used resources [14]. Specifically, the 95th percentile is seen as appropriate to capture the peak usage of existing VMs. The PM future used resources are then viewed as the sum of the peak usage of all VMs. The per-VM predicted 95th percentile is determined using a machine learning-based approach that incorporates various metrics, including VM size, type, guest OS, lifetime, average CPU usage, and maximum 95th observed. Additionally, the candidate VM 95th percentile is estimated based on knowledge from similar deployments. If the confidence in this estimation is sufficiently high, it is used to assess the VM's fit; otherwise, full resource utilization is assumed. To prevent the host from reaching maximum usage (with only the 100% threshold studied), two different versions of the scheduler were implemented: one at the score computation stage, trying to avoid it as part of soft constraints, and the other at the PM selection stage, avoiding it as part of hard constraints. The evaluation primarily focused on scheduling failure, defined as the inability to find a suitable host, rather than workload consolidation. Simulations revealed that both strategies were effective in maintaining a low scheduling failure rate. The maximum oversubscription observed was 1.25:1 under relatively small servers (16 cores). However, it's worth noting that since the peak usage of all hosted VMs is unlikely to occur simultaneously, this method has been criticized in other papers as sub-optimal [17].

Bashir et al. justify that a Gaussian distribution usually captures well the variety of CPU usage observed on a Cloud PM and proposes to compute the peak derived from it [17]. In this approach, the peak is estimated from PM usage metrics instead of VM metrics. Specifically, they propose to compute it as $\overline{CPU} + N \times \sigma$, where $\overline{CPU}$ and σ capture the average and standard deviation of CPU usage, respectively. They refer to this technique as N-sigma. Their study, drawing upon Google Borg traces, recommends a configuration with $N = 5$, which can handle most worst-case scenarios. In their

evaluation, they introduced two complementary techniques. One is a max predictor, which selects the maximum value among the Azure approach, a static oversubscription, and N-sigma. Another is an oracle-based predictor, assuming perfect knowledge of future resource usage, used in their simulation as a baseline. The evaluation primarily focused on scheduling failure, due to their parameter selection. Both N-sigma and the maximum predictor were found to reduce failures.

Both approaches share a similar limitation. The adjustment of optimistic/pessimistic oversubscription computation using parameters (such as percentile for Azure or N for Bashir et al.) is challenging. Firstly, these parameters are difficult to define accurately. Secondly, they fail to capture the specificities of individual PMs, as they are applied at the cluster scale.

Dynamic server ratio: others computation The *Dynamic Oversubscription ratio Adjustment* (DOA) introduced in [102] takes a different approach as it is not driven by a peak prediction. DOA increases available resources by a fixed percentage (e.g., 10%) of PM configuration until a maximum resource usage is reached (e.g., 95%). If this threshold is reached, the available resources are reduced to a percentage of the PM configuration (e.g., 50%). The ratio is periodically recomputed. Compared to a static ratio or a random PM selection, the technique was found to limit failures more effectively. However, one main limitation of this approach is that it does not try to prevent host overload, focusing only on the mitigation aspect.

On dynamic memory oversubscription Per-server oversubscription ratio mostly targets CPU oversubscription. In previously cited works, only DOA claimed to also be suitable for memory. Two reasons may be found. First, heterogeneous hardware does not affect memory dimension as much as the CPU. Indeed, the number of pages requested from VMs is not affected by DRAM generation, in opposition to requested CPU time slices that depend on PM performance. Second, dynamic memory oversubscription (i.e., accounting for existing VM usage) implies cooperation between PM and VM as a page reserved by a VM cannot be released as long as it is in use [103].

CLOUDVAMP [108] monitors VM internal memory usage using an inside agent. Available resources are proposed to be computed as the quantity of unallocated memory plus the unused memory inside the VM. If a new deployment exceeds the number of unallocated resources, memory is retrieved from VMs using a balloon principle. [109] proposes to exceed this computation by adding the amount of memory retrieved across the network (weighted by a coefficient to penalize it). Both are coupled with a mitigation mechanism based on live migration as memory violations are more severe than CPU ones [103]. However, since unused resources are observed and not predicted, this may lead to more migrations than required, resulting in increased unavailability and resource usage.

On oversubscription adoption

Oversubscription has existed for as long as VMs have existed. In 1966, IBM introduced CP-67, one of the first commercial hypervisors, and described how VMs could share the single CPU of their mainframe system [110, 111]. CP-67 was said to be able to support 40 VMs on the System/360 model leading, with a bit of anachronism, to what today would be considered an impressive 40:1 oversubscription ratio. Memory oversubscription was also possible, thanks to a virtual memory address space.

Later on, the introduction of multiple cores in architectures (thanks to *Symmetric Multiprocessing* (SMP) and SMT) and the decline in hardware price [112] allowed systems to use virtualization without requiring oversubscription of computing resources. Despite being the very origin of virtualization, VMs orchestration is nowadays mostly performed without sharing cores.

While oversubscription is documented in DC production environments [113, 114], it remains limited in public Cloud infrastructures. In practice, Cloud platforms do not oversubscribe general usage VMs in their production environments [115, 59]. This may be attributed to performance [17] and security (side-channel attacks) [115] reasons.

Oversubscription is typically reserved for low-pricing VMs [116] and burst VMs [117, 118]. Burst VMs are small VMs adhering to a credit-based mechanism to track their CPU usage. On low usage phases, they accumulate credits that they can spend to handle peaks using either more performances or computing resources provided by the platform. This mechanism is used to ensure a peak-driven behavior on CPU usage, allowing Cloud providers to oversubscribe their resources [115]. Note that this behavior does not guarantee by itself that peaks will be asynchronous between VMs.

On oversubscribed platforms, a static ratio at the cluster scale remains a common technique [101, 119]. The broader adoption of dynamic oversubscription may be limited by the more complex infrastructure it requires, as it creates a dependency of the orchestrator on the monitoring infrastructure. Additionally, parameters from different approaches that are hard to define make it challenging to adjust the pessimistic degree of the computation when facing real conditions.

2.4.5 Summary of usage improvement techniques

Table 2.3 Classification of usage improvement techniques

Technique	Unallocated resources	Unused resources	Generic workload
Evictable VMs	Yes	No	No
Harvesting VMs	Yes	Some	No
Disaggregated resources	No	Yes	No
Oversubscription	No	Yes	Yes

A classification of the listed techniques can be found in Table 2.3. Since they do not target the same workloads or scope, these techniques are complementary to a Cloud provider's strategy to improve its resource usage. However, when the focus is on reducing the clusters of the existing workloads,

only oversubscription-based techniques are suitable, as they can maintain the same availability and performance guarantees for the workload if configured correctly.

As oversubscription is currently not widely applied beyond specialized VMs, improvements in current techniques are necessary. Oversubscription leverages opportunities inherent in the existing IAAS context: VMs are often underused, small in size, and hosted on increasingly larger PMs. Pooling their unused time, which is facilitated by a significant workload heterogeneity at the PM scale, offers an opportunity for better physical resource sharing. In a context where reducing the number of PMs is crucial, sharing physical resources through oversubscription is an effective way to minimize the impact of ICT.

Given that oversubscription may impact performance, its broader adoption requires careful consideration. One objective is to replicate the conditions observed in real platforms, where VMs typically utilize their resources partially and dynamically.

Chapter 3

Improving IaaS experiments using realistic users' behavior

Abstract: *In this contribution, I present a method for testing Cloud systems under realistic client behaviors. This method generates actions such as VM deployment, VM resource usage, and VM deletion that can mirror the usage patterns observed in actual Cloud environments. Unlike other approaches, the introduced method allows for the down-scaling of workloads by design. This enables the evaluation of platforms on real hardware rather than simulated environments, which is particularly beneficial for research focusing on reducing the environmental impact of Cloud usage. As an illustration, I employ this tool, entitled* CLOUDFACTORY*, to quantify the cost associated with the "instantaneous provisioning" of Cloud resources, which can result in up to a 46% increase in the number of required servers. Subsequently, the remainder of this thesis leverages this tool to assess oversubscription strategies.*

For more than a decade, Cloud computing has been an active field of academic and industrial research, encompassing multiple challenges and contributions that are continuously delivered to improve the state-of-the-art. Nevertheless, assessing effective contributions in the domain of Cloud computing is notoriously hard [120], in particular, because of the difficulty of reproducing production-scale IAAS deployments.

To overcome this limitation, Cloud simulation platforms are commonly considered in the literature to evaluate new contributions for Cloud infrastructures [121]. Alternative contributions to this field include datasets and benchmarks that can be released and published by both academic and industrial practitioners. Unfortunately, Cloud providers may be reluctant to share production-scale datasets, due to the possibility of user identity exposure through trace information [122]. Furthermore, both Cloud-based simulations and experimental deployments tend to suffer from the same limitations in terms of representativeness. In particular, the replay of benchmarks and raw datasets may be difficult to apply when lacking production-scale infrastructures, while raw traces may be truncated to

deal with the limitation of a testbed infrastructure, hence questioning the representativeness of the resulting experiments. Furthermore, the applicability of developed contributions in a production-scale environment may be challenged by this limitation.

The main objective of this contribution, referred to as CLOUDFACTORY, is to enhance the evaluation and validation of Cloud contributions by allowing to use of production-like Cloud workloads. More specifically, CLOUDFACTORY addresses the challenge of reproducing IAAS workloads by leveraging representative infrastructure-scale models extracted from datasets of production-scale execution traces. To achieve this, CLOUDFACTORY combines two key components that can be used independently by Cloud practitioners and researchers to share insightful statistics that can be exploited to generate IAAS workloads at any scale—from single worker nodes to clusters—and apply them in simulations or experimental deployments. In particular, thanks to the workload analyzer that I made available, Cloud providers can share production-scale metrics without disclosing their raw datasets. Then, computer scientists can import these statistics into the workload generator that I packaged as a separate tool to produce various IAAS workloads that they can easily use and further share to evaluate their contribution, thanks to the bindings I deliver for several state-of-the-art evaluation platforms. Throughout the contribution, I illustrate the value of CLOUDFACTORY by reporting on the analysis of the Microsoft Azure dataset [23] and the generation of IAAS workloads for the CLOUDSIMPLUS simulation environment [31] and the CBTOOL benchmarking tool [48].

I first present an overview of CLOUDFACTORY architecture (see 3.1) before diving into its design (see Section 3.2, Section 3.3, and Section 3.4). Then, I explore a case study enabled by this contribution and share CLOUDFACTORY generated statistics from OVHcloud context (see Section 3.5). Finally, I discuss limits (see Section 3.6) and conclude this work (see Section 3.7).

3.1 CLOUDFACTORY overview

Figure 3.1 depicts an overview of the CLOUDFACTORY key components. In particular, I focus on generating IAAS workloads that are representative of production-scale deployments but can be provisioned at any scale—from single compute nodes to cluster-wide platforms or simulators.

To do so, I propose to convert raw datasets collected by Cloud providers into a compact set of IAAS statistics that are sufficient to deploy a production-like workload. Interestingly, this approach does not impose the disclosure of sensitive information from the perspective of Cloud providers, who can simply publish such cluster-wide statistics. It is the goal of the workload analyzer component detailed in section 3.2.

The workload generator leverages these statistics to generate a usable experiment workload, with the appropriate VM set and usage models. Implementation is detailed in section 3.3.

Finally, different binding options are available to exploit the results on several state-of-the-art evaluation platforms. Generated experiments workload may be executed on a simulator or on a physical system. Exporting capabilities are detailed in section 3.4.

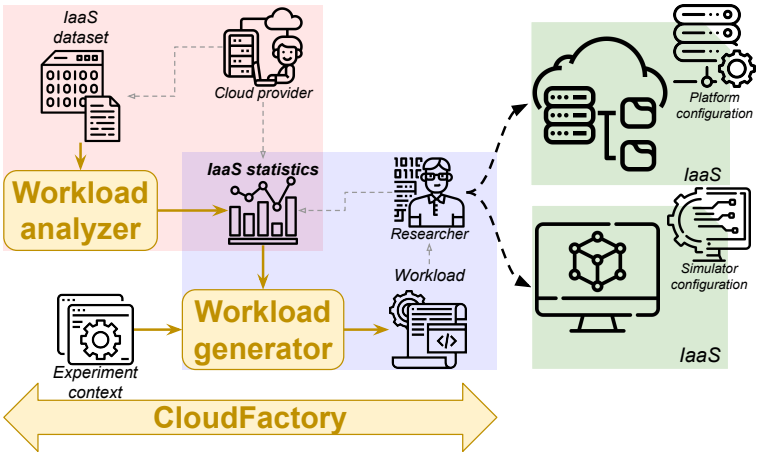

Fig. 3.1 Overview of CLOUDFACTORY

3.2 Compute high-level statistics

After deployment of a client-selected configuration, a VM enters its usage phase, during which resource utilization can vary substantially. This variability can be observed in different ways, ranging from complete idleness to periodic activity, bursty behavior, short lifespan, etc. To account for VM behaviors, I introduce a `workload analyzer`. This library builds a set of VM usage profiles from a Cloud dataset.

The realism of workloads is contingent upon their ability to replicate the behavior of actual systems. While replay-based IAAS workloads meet this requirement, they are often impractical to implement—due to privacy restrictions that limit sharing options, which in turn impacts the reproducibility of experiments. Furthermore, production-scale Cloud clusters cannot be easily reproduced for SUT experimental purposes, thus posing a material constraint.

3.2.1 Statistics identification

Metrics describing the SUT must be carefully selected to represent as faithfully as possible Cloud context specificities. The identification of relevant statistics was also motivated by metrics availability from currently published datasets. Furthermore, this set of statistics is voluntarily kept as small as possible to facilitate computation and sharing from undisclosed Cloud datasets.

CLOUDFACTORY statistics can be classified into two categories. At the micro level, VM resource usages are considered. At the macro level, the management of a dynamic set of VMs is studied.

VM resource usage is considered from the CPU perspective. In addition to CPU utilization bounds (see Section 3.2.2), I also take into account usage periodicity (see Section 3.2.3).

To account for VM management, CLOUDFACTORY considers VM configuration options distribution (see Section 3.2.4) and departure/arrival ratios (see Section 3.2.5).

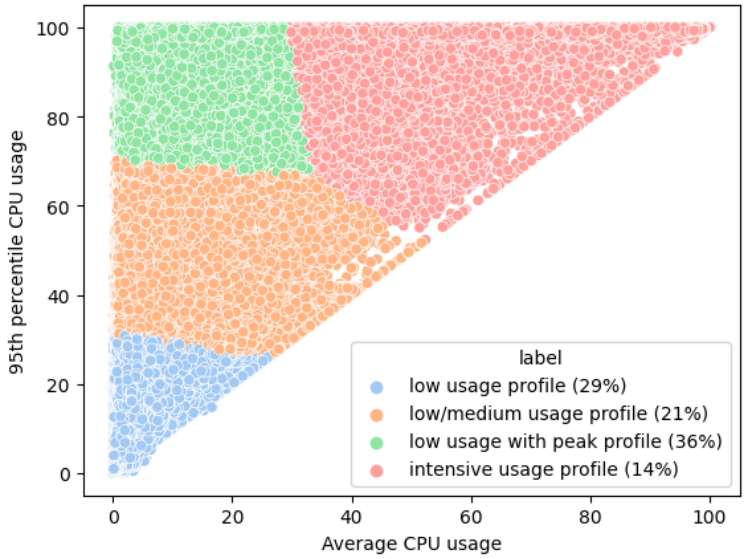

Fig. 3.2 4 VM usage profiles captured from Azure 2017 using k-means clustering ($k = 4$)

Except for VM configuration options distribution, statistics are computed according to profiles. The first step applies clustering to VMs, based on their usage similarities, to describe more accurately observed behaviors, as per-profile statistics. Options distribution is treated separately as offered configuration options may significantly change from one Cloud provider to another, which limits comparisons and export options.

3.2.2 Computing usage

The workload analyzer first classifies VMs based on their average and percentile CPU usage metrics to identify resource utilization profiles. I have selected the k-means clustering algorithm, due to its capability to capture a given number of clusters. This capability allows Cloud providers to fine-tune the granularity of information exposed to researchers by customizing the number of profiles through the parameter k. Figure 3.2 illustrates this classification on the Azure dataset VMs on four profiles ($k = 4$). Different VM clusters are deduced, from those with a low average and percentile usage to those with high average and percentile usage. Each cluster is subsequently translated into a profile, consisting of specific bounds on the observed average and percentile values. For instance, the blue cluster in Figure 3.2 is composed of VMs having an average CPU usage below 26% (X-axis), and peaks below 32% (captured by their 95th percentile, on Y-axis). VMs that match this profile are typically considered oversized. It is noteworthy that approximately 29% of the VMs observed in the Azure dataset belong to this particular category.

3.2.3 Periodicity ratio

VM periodicity is an important factor for scheduling strategies. A recurrent usage pattern can be leveraged by Cloud providers to identify interactive workloads [23]. In each cluster, I compute the ratio of VMs reporting a recurrent pattern considering a specified scope (set by default to one day). As in [23], I used a *Fast Fourier Transform* (FFT) algorithm to evaluate each VM behavior. Differences may remain, however, as the authors did not communicate the implementation details. The implementation is described in Algorithm 1. Specifically, I used a periodogram to estimate *Power Spectral Density* (PSD) frequencies. A VM is classified as *periodic* over a given window if the associated window frequency PSD exceeds a user-specified percentile in distribution. In the remainder of this chapter, the 99th percentile has been considered. On each profile, the periodicity is evaluated on a subset of VMs matching a lifespan duration condition, before being extrapolated to the overall quantities.

Algorithm 1 Perodicity detection algorithm

Input time series, period, threshold
Output Perodicity state

1: $\mathscr{M}_{periodogram} \leftarrow$ Generate periodogram from time series
2: $value \leftarrow \mathscr{M}_{periodogram}(period)$
3: $max \leftarrow percentile(\mathscr{M}_{periodogram}, threshold)$
4: **if** $value > max$ **then**
5: **return** `PERIODIC`
6: **end if**
7: **return** `NOT PERIODIC`

3.2.4 Computing VM distribution

VM configuration options distribution is an important statistic due to its impact on server consolidation. CHAMELEON and Azure datasets underline the disparity between the observed VM distributions. Therefore, I calculate the ratio of VM configuration, in terms of vCPU and vRAM, observed in Cloud datasets.

Avoiding bias Counting the VM configuration occurrences at the dataset scale biases the results, due to the short lifespan of some deployments. Instead of considering the number of configurations based on deployments, I compute the average VM distribution presence in nodes (see Equation 3.1).

$$\text{configuration}_{\text{freq}}(c) = \frac{\sum_{s \in S} \text{Frequency}(c,s)}{|S|} \tag{3.1}$$

This later facilitates experiments' representativeness in down-scaled clusters, where server configuration and arrival rates may be significantly lower than the ones observed in production, hence impacting the hosted distribution accuracy. The provided dataset is therefore considered through time windows of a chosen duration. Distribution presence is computed as the average value observed from

all considered time slices (S). With a sufficiently short time slice duration, this approach allows us to compute the average distribution of each VM configuration option at any given time, mitigating the impact of heterogeneous lifespan. The rest of this contribution considered a one-hour time slice duration.

Azure example Interestingly, applying this method to the Azure dataset reveals significant differences with their values, as they adopted the "deployment approach". For instance, the 14 GB configuration option, which is claimed to account for 11% of the VMs, is found to be half as present, accounting for only 6% of VMs over a typical hour. The configuration has indeed the lowest 90th percentile lifespan in the dataset. VM-creation test workloads are reported to be common on VMs belonging to Azure [23] may be based on this configuration, leading to a reduced average lifespan.

3.2.5 Departure & arrival ratios

The rate at which new VMs are deployed is a crucial metric for IAAS environments, as it bears a significant influence on the provisioning capabilities of the platform. Conversely, the lifespan of VM deployments may be viewed as a complementary micro-level indicator. In light of my concentration on IAAS workloads, I opt for the macro-level equivalent, namely the departure ratio. Consequently, both arrival and departure ratios are computed for each identified VM profile.

To analyze these metrics, I compute the number of newly arrived VMs relative to those who left within a default one-day window through all the analyzed datasets. With Equations 3.2 and 3.3, I transform these quantities into the average ratio observed in the specific context. If the arrival ratio is greater than the departure ratio, the profile is considered as *increasing*, while the opposite reflects a *decreasing* profile. In the Azure dataset, I observe that low utilization workloads have a lower degree of dynamism, with reduced departure and arrival ratios compared to more intensive workloads.

$$\dot{\text{arrival}}_{\text{IAAS}} = \frac{\sum_{s \in S} \frac{|departure_{\text{VM}}(s)|}{|\text{VM}(s)|}}{|\text{S}|} \tag{3.2}$$

$$\dot{\text{departure}}_{\text{IAAS}} = \frac{\sum_{s \in S} \frac{|arrival\text{VM}(s)|}{|\text{VM}(s)|}}{|\text{S}|} \tag{3.3}$$

3.2.6 Profiles examples on Azure dataset

The profiles built by the workload analyzer include the following statistics:

- *Profile weight in Cloud dataset*,
- *Average CPU usage bounds*,
- *95th percentile CPU usage bounds*,
- *Arrival & departure ratio*,

- *Periodicity ratio*.

The IAAS statistics computed for the 4 profiles illustrated in Figure 3.2 are summarized in Table 3.1.

Table 3.1 Detailed metrics of the 4 usage profiles computed by the workload analyzer on Azure dataset

Usage Profile	Percentage	Avg. CPU bounds	95th percentile CPU bounds	Arrival rate	Departure rate	Periodicity rate
Low	29%	[0.0;25.9]	[0.0;31.2]	15%	15%	0.4%
Low/Medium	21%	[0.1;51.9]	[26.0;70.2]	32%	32%	0.8%
Low with peak	36%	[0.2;33.3]	[66.6;100.0]	60%	60%	1.1%
Intensive	14%	[29.3;100.0]	[52.1;100.0]	57%	57%	0.8%

3.2.7 Generated statistics.

The workload analyzer data related to VM distribution and usage are converted to YAML files. The YAML format was chosen due to its user-friendly nature that facilitates comprehension. These files are not expected to disclose any sensitive information and to enable sharing with the wider community. Furthermore, they can also be modified by developers seeking to explore specific conditions. Finally, they can be packaged with any Cloud experiment to offer a reproducible Cloud environment for researchers who would like to compare their contributions with previous works published with this toolkit.

3.3 Generate production-scale workloads

I now detail how to generate workload matching high-level specifications. Beyond importing statistics, generating a production-scale workload requires describing the infrastructure considered for the SUT. This also involves specifying experiment duration and temporality patterns.

Long-run experiments are simplified by CLOUDFACTORY through the temporality concepts of slices and scope. A slice represents the minimum duration considered in the experiment, while scope refers to a multiple of slices. This can be used in two ways. The first is to accelerate real experiments by changing the virtual day's duration (i.e., scope) and a virtual hour (i.e., slice). The second is to study the effects of predictability on various windows (e.g., diurnal pattern, and hourly patterns).

The workload generator combines the IAAS statistics shared by Cloud providers (or computed from the workload analyzer) with user input on desired workload size and on temporality to build a consistent set of VMs. The set is composed of VM behavior attached to a VM configuration. A VM configuration is defined by characteristics related to provisioning:

- *number of vCPUs*,
- *size of allocated vRAM*,
- *time of provisioning* (as ticks),

- *lifespan* (as ticks/slices/slots).

While, the VM behavior is characterized by:

- hosted *application binary*,
- VM *workload*.

3.3.1 On VM configuration generation

CLOUDFACTORY first generates a set of VM configurations according to requested ratios for the SUT. The user can request a specific number of VMs to be generated. A more practical option—especially when studying the effects of load—is to request workloads in terms of the amount of vCPU and memory. In the latter case, CLOUDFACTORY applies an operational search approach where vCPU allocation is maximized while respecting constraints on the specified amount but also on the VM configurations distribution considered.

Each VM is randomly assigned a usage profile that conforms to the proportions reported by the workload analyzer.

If the arrival rate for a given profile is not zero, additional VMs is generated after the initial set is deployed, following a heavy-tail Gaussian to match the real pattern observed [23].

During this process, a lifespan is attributed to each VM so that the number of VM departures on each scope meets the targeted departure ratio of each profile.

3.3.2 On VM behavior generation

From the profile assigned to each VM, an activity workload must be generated. Computed profiles include bounds on average and percentiles values. A random selection is made within the specified limits for both the targeted average value and percentile value of the CPU. Therefore, a Gaussian distribution that aligns with these two specific characteristics is generated. Gaussian distributions were chosen for their similarities with VM CPU usage [34].

For a given VM, the obtained distribution is considered as a list of potential CPU usage. Based on experiment temporality, values from the distribution are selected and assigned to each slice as its targeted CPU usage. For VMs that exhibit periodic behavior, the values attributed to the first scope are repeated on the subsequent scopes until the VM reaches the end of its lifecycle.

3.3.3 A few words on reproducibility

CLOUDFACTORY provides the ability to dump and load an experiment in JSON format, allowing researchers to reproduce the same workload on different systems or temporality settings.

This feature is particularly useful in peer review processes, as the JSON dump can be shared along with the generated scenario scripts to facilitate communication on the evaluation process.

3.4 Exporters

CLOUDFACTORY can currently generate 3 export types, leading to 3 possible bindings.

3.4.1 CLOUDSIMPLUS

Overview

In a simulation-based experiment, a CLOUDSIMPLUS scenario can be generated as a Java program that incorporates the VM workloads. While this generated Java template file is readily usable, researchers may wish to customize it to evaluate specific clusters (e.g., number of DCs, number of nodes, size of nodes) or algorithms (e.g., orchestrators).

Implementation details

CLOUDSIMPLUS uses various object representations to simulate the execution of a Cloud environment. VM objects are instantiated according to the generated configuration. Internal applications are modeled using Cloudlet objects. I created a per-VM Cloudlet implementing a custom utilization model loaded from a Java properties file. To account for VM lifespan, a maximum Cloudlet duration is specified, and a broker destroys VM without any Cloudlet being assigned to it. The submission delay feature is used to manage arrival rates.

3.4.2 Bash

Overview

In the case of a physical testbed, the tool can generate VM workloads in the form of libvirt-based bash scripts.

CLOUDFACTORY outputs VM deployment script jointly with a VM workload script to deliver a reproducible behavior.

To mimic the heterogeneity of the Cloud workload, different images with different applications can be considered. The generated scripts transform targeted CPU levels into benchmarking tool parameters using editable configuration files. Additionally, constraints on VM distribution can be expressed at the generation level to prevent, for instance, memory-intensive benchmarks from being deployed on lightweight VM configurations.

Although this bash exporting option requires an initial setup involving the use of pre-generated qcow2 images with SSH keys already injected, it remains a simple way to deploy a large number of VMs on a worker node.

The resulting workload scripts can be used both locally and remotely, with *Network Address Translation* (NAT) management offered with the latter option.

Implementation details

Various applications are used to represent the heterogeneity of the workload in the Cloud.

Idle VMs are characterized as workloads with no specific CPU activity, except for the guest operating system. Batch-oriented VMs simulate CPU-intensive workloads using the StressNG load test with variable CPU usage. Database workloads are represented with VMs hosting a POSTGRESQL instance serving a TPC-C workload where request rates are adapted. Static websites are represented with a WordPress instance serving various request rates using wrk2 benchmarking tool. Microservices architectures are also considered using DEATHSTARBENCH [41], a social network serving various request rates using wrk2 benchmarking tool.

The reports obtained from the various benchmarking tools are stored as text files. To facilitate the exploration of application performance, I provide bash scripts that can generate a CSV format from these reports.

3.4.3 CBTOOL

Overview

To facilitate more complex deployments, the IBM CBTOOL framework can be used. From the generated CBTOOL script, VM workloads can be deployed on various Cloud environments, including GCP, AWS, and OpenStack clusters, with proper framework configuration.

Implementation details

The notion of a CBTOOL virtual application is used to model VMs and their associated load levels. Prior to deploying any new VM, a baseline virtual application is customized to match the desired load level, load duration, and VM size. CLOUDFACTORY slices are represented using "`waitfor`" instructions, with VM deployments and destructions at each occurrence to simulate the departure and arrival rate obtained from the modeling phase. The generated script is configured for the CBTOOL simulation mode, making it ready to use. It can be adapted by developers to their deployment context.

3.4.4 Others exporters

CLOUDFACTORY has been designed with compatibility considerations in mind. Other formats can be implemented through custom exporters, typically written in about 200 lines of code.

To ensure maximum compatibility with other formats, exporters should utilize the CPU usage list generated by CLOUDFACTORY instead of custom solution models. This approach helps to maintain homogeneity across different scenarios.

3.5 Case study

While CLOUDFACTORY can be used for multiple purposes, I demonstrate its added value to study cluster sizing issues, given multiple workload characteristics. Cloud physical resources remain underused, hence severely impacting the cost and efficiency of these infrastructures at large [23, 17]. To overcome this inefficiency, IAAS providers usually compensate for oversized VMs by offering more virtual resources than are physically available on a host. However, studying oversubscription strategies implies considering realistic VM usage. In this section, I highlight the distribution impact on the sizing of an IAAS cluster, based on the study of Chameleon and Microsoft Azure datasets.

3.5.1 Generate distributions

Chameleon-like and Azure-like distributions were computed using the workload analyzer from each dataset. The Azure 56 GB memory configuration, which represents 2% of VMs, has no equivalent in the Chameleon dataset. It was disregarded to facilitate comparisons. The computed distribution is detailed in Tables 3.2 and 3.3.

Table 3.2 CPU Distribution used for experiment

vCPU	Azure	Chameleon
1	**51%**	26%
2	25%	20%
4	17%	12%
8	7%	**42%**

Table 3.3 Memory Distribution used for experiment

vRAM (GB)	Azure
0.75	4%
1.75	**47%**
3.50	25%
7.00	16%
14.00	8%

vRAM (GB)	Chameleon
0.5	2%
2.0	24%
4.0	20%
8.0	12%
16.0	**42%**

3.5.2 Generate usage profiles

As the Chameleon dataset does not contain usage metrics, Azure ones—computed from four usage profiles inferred from the workload analyzer—were applied to both contexts. Furthermore, two deployment strategies were tested: a *static* and a *dynamic* deployment. In the static deployment case, all profiles were assigned arrival and departure ratios equal to 0. The initial set of VMs is therefore deployed immediately, and run on all experiment duration, with no new VMs arriving. In the case of

dynamic deployment, the computed departure and arrival ratios are left unchanged for each profile. Once the initial set of VMs is deployed, some of them are leaving and others arrive continuously, while maintaining a similar overall quantity. In this scenario, the provisioning capabilities were evaluated over a one-week window.

3.5.3 Experiment

Considering a hosting target of VMs, and given a VM distribution and deployment type, I compute the minimal amount of servers required to host the targeted number of VMs. To do so, the workload generator is used to obtain a CLOUDSIMPLUS workload for each combination, using a server baseline of 64 CPUs and 256 GB. The configuration baseline was chosen due to the prevalence of 64-core servers in at least one Cloud dataset [27] and an assumption that an allocation of 4 GB memory per core would be representative. Given a distribution and deployment type combination, CLOUDSIMPLUS simulations are run sequentially, with an increasing number of hosts, until the minimum amount of servers required is found. This is done by verifying that the default VM scheduler was able to schedule all the provisioned VMs. CPU oversubscription was supported in this experiment through CLOUDSIMPLUS MIPS notion, by implementing a per-VM MIPS baseline weighted by their maximum usage, allowing the VM orchestrator to allocate resources based on VM usage instead of VM provisioned resources. This mechanism can be described as an oracle-based oversubscription mechanism, as maximum usage is known before deployment. However, it remains pessimistic, as maximum usage of VMs is unlikely to happen at the same time, leading to potential gains, non-exploited here. A more realistic approach requires the deployment of a custom VM scheduler, which is out of the scope of this contribution. I did not consider live migration in this experiment.

3.5.4 Results

Results are shown in Figure 3.3. Compared to the static deployment, the flexibility required by a fluctuating number of provisioned VMs implies a significant overhead for both datasets, on average 1.31 more servers for Azure and 1.46 for Chameleon. While being relatively stable on an increasing hosting objective, this factor underlines that heavy-tail arrival rates, reflecting grouped VM deployments, impose an infrastructure cost. Conversely, in a system where the arrival rate can be managed based on, for example, a queuing mechanism, the cost associated with the same VM hosting objective is lower. Cloud providers can leverage this by offering their customers an incentive to choose a queued deployment instead of the state-of-the-art "as fast as possible" strategy.

The Chameleon distribution requires VMs with larger amounts of resources, on average. Therefore, the same VM hosting objective results in provisioning more resources, compared to the Azure distribution. Concretely, an average deployment requires about 2.25 vCPU and 4 GB of memory in the Azure context versus 4.5 vCPU and 9 GB for Chameleon. Chameleon needs on average 1.89 additional hosts for static deployments and 2.1 additional hosts for dynamic deployments, due to the amplified impact of VMs arrival bursts on newly provisioned resources.

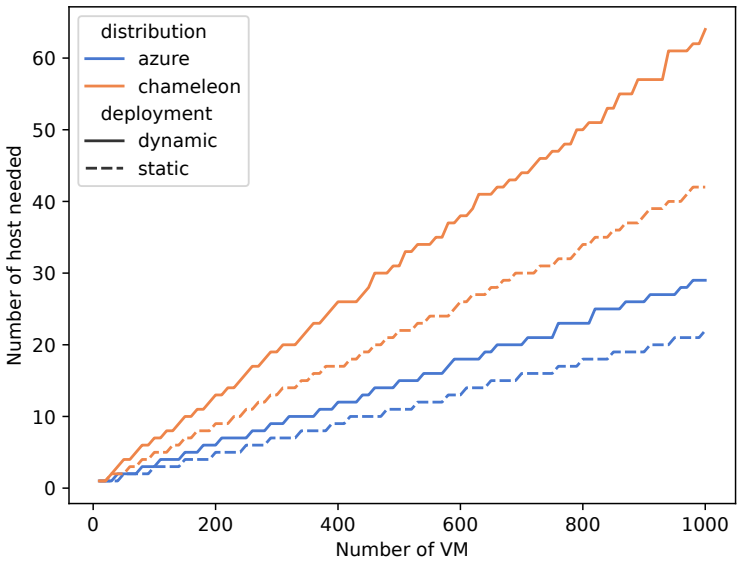

Fig. 3.3 IaaS hosting capacities required by VM distribution and deployment conditions

The achieved CPU oversubscription ratio, defined as the ratio between the vCPUs offered and those physically available, can also be estimated. Provisioned CPUs are extrapolated from the average deployment size applied to the number of VMs. Physical resources are computed from the number of hosts and their respective configurations. In Figure 3.4, one can observe that, while being higher than 1.0, oversubscription ratios remain significantly lower for dynamic deployments, compared to static ones. While dynamic deployments have the same VM hosting objective, the number of hosted VMs may be significantly higher at a given point, due to fluctuation in departure and arrival rate, reducing oversubscription to almost no gain on average (1.11 ratio for Chameleon, 1.18 for Azure). Due to the same usage being applied during CloudFactory workload generation, oversubscription ratios are similar on static deployments (1.63 and 1.57 for Chameleon and Azure respectively).

The key results of the experiment can be seen in Table 3.4.

Table 3.4 Experiment results summary

Infrastructure Configuration	**Servers to host 1,000 VMs**	**CPU oversubscription ratio**
Static Azure	22	1.57
Dynamic Azure	29	1.18
Static Chameleon	42	1.63
Dynamic Chameleon	64	1.11

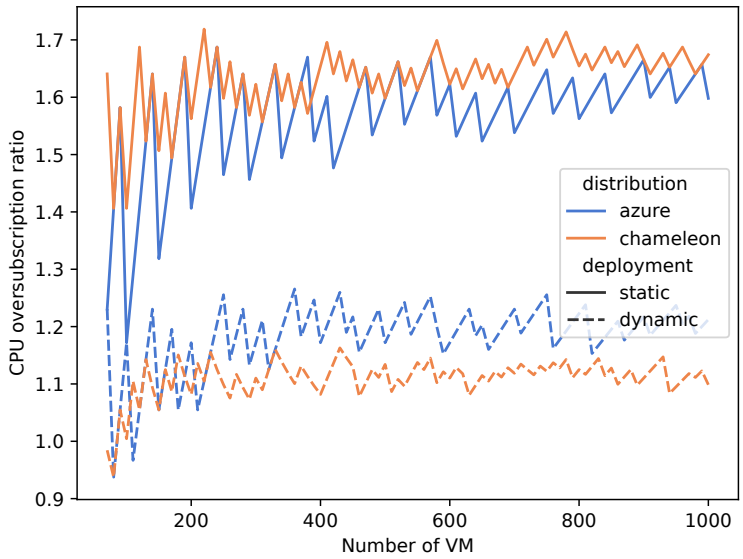

Fig. 3.4 CPU oversubscription ratio based on VM distribution and deployment condition

3.5.5 Adoption by the Cloud industry

Besides generating realistic IAAS workloads for Cloud experiments, CLOUDFACTORY aims to foster the sharing of high-level IAAS statistics to and from the Cloud ecosystem. In particular, OVHcloud[1] is a public Cloud provider recognized as the largest Cloud operator in Europe [123].

Thanks to the availability of the workload analyzer of CLOUDFACTORY, OVHcloud agreed to share the statistics of their *public Cloud* offer. Interestingly, I can compare these statistics with the ones of alternative providers, like Azure and Chameleon. Figures 3.5 and 3.6 are, thus, reporting on the distributions of vCPU and vRAM, respectively. Due to the diversity of configurations, I grouped some close vRAM configurations.

The findings from the VM distribution analysis suggest statistically significant differences among the three Cloud service providers.

Specifically, Azure exhibits a light CPU context in comparison to both Chameleon and OVHcloud. The configuration options that account for less than 1% have been excluded from the analysis. It is worth noting that the OVHcloud distribution includes the largest configuration option, with 16 vCPU (3% of VMs). Conversely, the Chameleon Cloud service provider records the highest average CPU option, reflecting its research-oriented objective. This outcome is attributed to the 8 vCPU option, which is the most prevalent among the Chameleon cluster VMs.

On the memory aspect, Azure is also predominantly characterized by small configurations. While Chameleon is a memory-intensive Cloud workload compared to Azure, it is exceeded by OVHcloud, primarily due to its percentage of options equal to or greater than 30 GB (8% of VMs). In contrast,

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

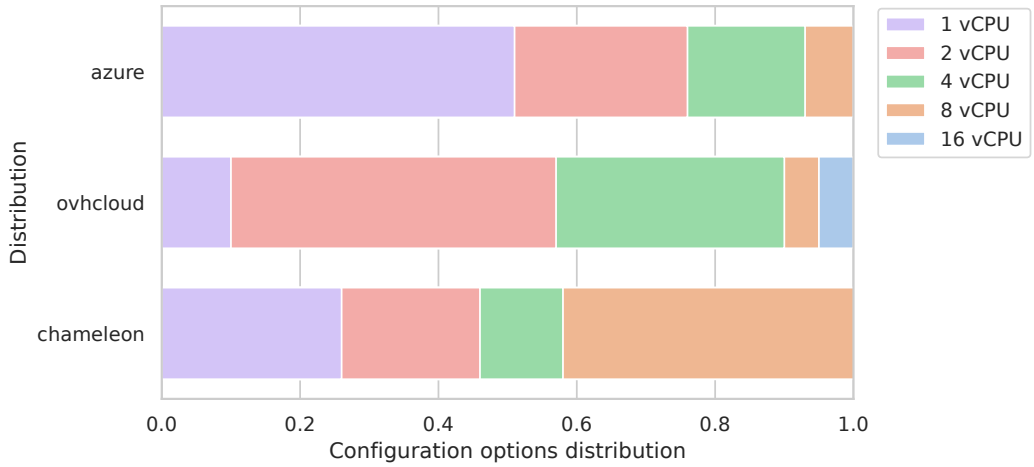

Fig. 3.5 Comparison of vCPU distributions across Cloud providers

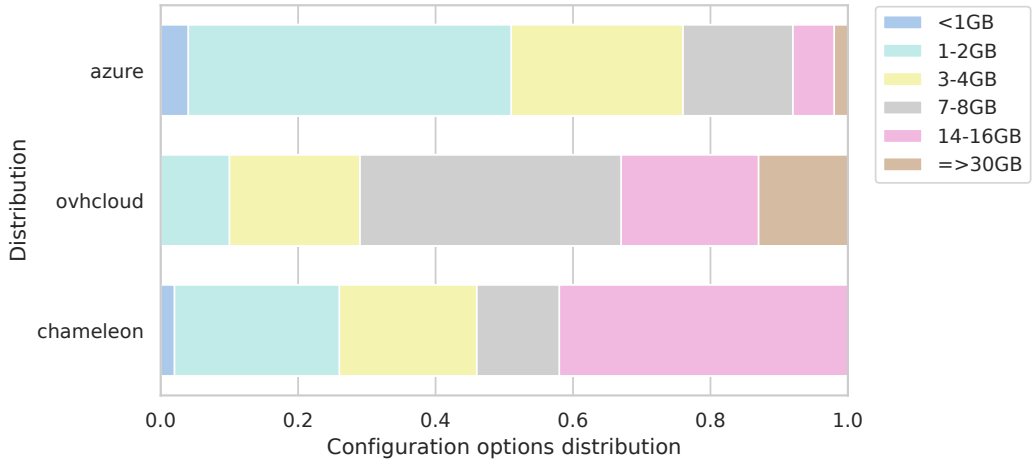

Fig. 3.6 Comparison of vRAM distributions across Cloud providers

Azure's most significant memory option is the 56 GB configuration, which accounts for only 2% of the total VMs. Notably, OVHcloud does not offer any memory options less than 2 GB.

The three distributions are available in the repository and can directly be used by the workload generator.

3.6 Limitations

CLOUDFACTORY aims to capture and reproduce global behaviors observed in IAAS environments. However, the use of statistical techniques introduces an inherent information loss, potentially leading to the omission of specific behaviors. In particular, the modeling of CPU usage patterns is based on Gaussian distributions, which may not accurately represent the behavior of all VMs under real-world conditions.

While grouped deployments are taken into account by utilizing a heavy-tail distribution to capture arrival rates, the tool subsequently treats individual VMs as independent entities. However, in reality, sets of VMs can be provisioned to host distributed applications, resulting in network traffic between them.

Furthermore, CLOUDFACTORY's usage profiling focuses on CPU usage and neglects other types of resources. This decision was made for two main reasons: the widespread availability of CPU usage in datasets and the presumed correlation between CPU usage and the utilization of other resources, based on the specific application being used. However, I acknowledge that certain benchmarks could benefit from incorporating information about the usage of other computing resources.

The tool offers the capability to scale an IAAS workload down to a single node. However, this functionality relies on the assumption that the workload is evenly distributed among nodes, which may not always hold in practice. For example, Cloud providers may manage dedicated clusters for different premium levels or service tiers.

3.7 Conclusion

In this contribution, I introduced CLOUDFACTORY, a realistic IAAS workload generator. I first provide a Cloud dataset analyzer, able to compute context-based metrics on VM configuration options distribution and usage. These statistics can be shared with the wider community, enabling researchers to consider multiple Cloud-based contexts. The statistics can also be used by the generator to compute a workload scenario in a simulated context, leveraging CLOUDSIMPLUS capabilities, or using a physical testbed, using bash or CBTOOL deployments. A case study example was presented with the cluster sizing example, underlying the importance of considering representative workload to produce coherent results.

By enabling the generation of realistic Cloud clients' behavior and, consequently, the replication of their incomplete usage of requested resources, this tool allows us to test oversubscription strategies in the remainder of this thesis. In these strategies, the incomplete utilization of resources is mitigated by sharing host cores among clients, while trying to preserve performances, an aspect that necessitates investigation on real platforms.

Chapter 4

Computing oversubscription ratios under stability consideration

Abstract: *In this contribution, I propose to compute oversubscription ratios at the server scope by assessing their quiescent state. This approach allows for adjusting the optimistic oversubscription degree on a per-PM basis, rather than defining it globally as seen in other dynamic approaches. I evaluate the stability of the workload on a per-server basis and use this information to tune the pessimistic/optimistic degree of oversubscription when considering the host for new deployments. The prototype, called* SCROOGEVM*, demonstrated superior performance compared to other dynamic approaches.*

Initially, virtualization technologies paved the way for building more energy-efficient Cloud infrastructures by increasing physical resource usage [124].

Unfortunately, most Cloud operators keep observing that the VMs they host in their data centers are underused [14]. These observations can be explained by several reasons: (i) their customers order cheap servers, without necessarily using them over time (as a consequence of a rebound effect); (ii) their customers over-provision their VMs to anticipate potential workload peaks; (iii) the fixed-size VM configuration might be inappropriate—*e.g.*, imposing a provision of 32 GB of memory to host a hypothetical workload of 18 GB.

Various Cloud providers report relatively low usage of their infrastructure, even over the past decade [14, 24, 125]. Optimizing this usage is one of the most promising leads to reducing the energy consumption and carbon footprint of DCs. While CPU resources are often effectively allocated, as reported by [15] indicating that 80% of Azure servers have less than 15% unprovisioned cores, there is still substantial underutilization of CPU on PMs [24]. This underutilization highlights the significance of provisioned, yet unused, resources. I chose to address it through the prism of resource *oversubscription*, also known as *overcommitment*, which can be defined as the amount of virtual resources made available for each unit of physical resource. Resource *oversubscription* allows Cloud

providers to boost physical resource utilization and is widely adopted in production, as reported in the literature [14, 17].

However, setting an appropriate oversubscription ratio for a given Cloud infrastructure remains an open challenge. While an under-loaded platform is inefficient, an over-loaded platform may impact its performance and stability. Therefore, state-of-the-art hypervisors adopt a configurable threshold of oversubscription ratio per resource type, and at the scale of a cluster. For example, the OPENSTACK platform sets by default the oversubscription ratios to 16:1 and 1.5:1 the oversubscription ratios for vCPU and vRAM, respectively. This implies that all the computing cores and memory of PMs are multiplied by 16 and 1.5 when exposed as virtual resources. Nonetheless, the effective value for each of these ratios has to be carefully estimated by Cloud providers by taking into account resource and workload characteristics, as well as an acceptable risk [106].

Even current dynamic approaches, such as [23, 17], do not provide inherent guidelines on how to adjust the optimistic/pessimistic degree of their respective approach. In this contribution, parameters are dynamically deduced from the PM state.

Instead of statistically determining optimal global parameters, I argue that clusters can better benefit from a *dynamic* oversubscription ratio per PM. In particular, I believe that higher gains at lower risk can be achieved by learning from the deployed workloads and resource utilization of individual PM. In this contribution, I propose SCROOGEVM, a new approach to dynamically increase the oversubscription ratio per PM, while maintaining performance goals. SCROOGEVM monitors effective PM resource usage by combining resource utilization metrics and overload signals. This approach, decoupled from the resource optimization of a given IAAS platform, allows the solution to be generic and leveraged by most Cloud schedulers in any IAAS context. SCROOGEVM leverages state-of-the-art machine learning techniques to capture periods of stability for a PM—i.e., periods of foreseeable resource usage. During these periods, SCROOGEVM can analyze the utilization statistics to better understand resource usage and increase the oversubscription ratio of each resource accordingly. SCROOGEVM, therefore, adopts a *greedy* approach to increase little-by-little oversubscription ratios, to never trigger VM consolidation phases, which could penalize the Cloud infrastructure and its customers. Step by step, and as long as a PM is labeled as stable, SCROOGEVM adjusts the amount of available resources by reasoning on the long term. This more natural solution to resource consolidation learns from VM requirements and can be used to balance the IAAS allocations and performances across all PM in a cluster.

I first introduce the principles of greedy oversubscription and an implementation, named SCROOGEVM (cf. Section 4.1). I also perform an empirical analysis of the impact of different parameters of this approach in an environment that mimics the characteristics of Microsoft Azure (cf. Section 4.2). Additionally, I evaluate SCROOGEVM oversubscription computation and compare it to other dynamic approaches (cf. Section 4.3). Finally, I conclude this work in Section 4.4.

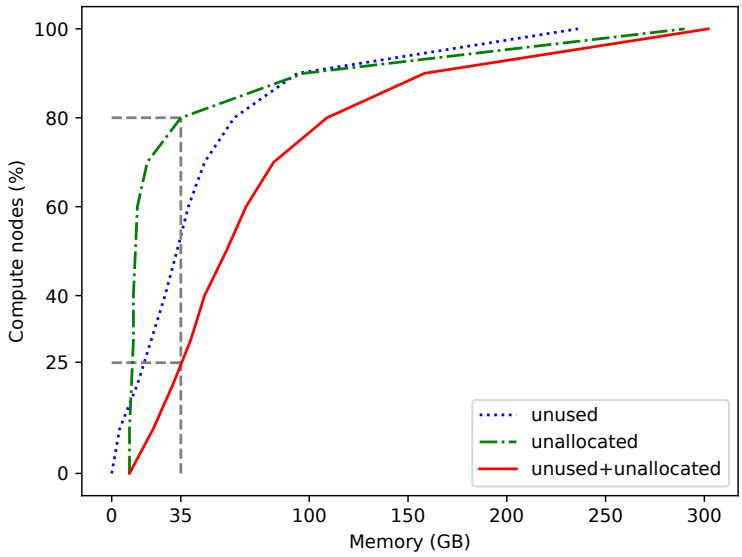

Fig. 4.1 *Cumulative Distribution Function* (CDF) of memory availability in an IAAS infrastructure.

4.1 Greedy oversubscription with SCROOGEVM

I introduce *greedy oversubscription* as a novel approach to implement a dynamic resource oversubscription strategy at the scale of individual PMs. More concretely, this approach gradually increases the oversubscription ratios intending to never trigger live migration, as long as a PM is assumed to be stable, thus reasoning over the long term. While migration consolidation techniques may penalize the stakeholders by imposing temporary service unavailability and complex migration strategies, I believe that greedy oversubscription can offer an alternative approach that incrementally increases resource utilization by hosting more and more VMs, while minimizing costly VMs consolidations.

4.1.1 Principles of greedy oversubscription

Preliminary analysis of Cloud resource availability In 2017, Microsoft Azure released traces from a 3-month IAAS workload [14]. Interestingly, this dataset—reflecting a production-scale Cloud infrastructure—reveals several insights that I leverage as part of this contribution:

- **Not all VM resource requirements are the same**: VM configuration distribution—coined *size* in the contribution—reports that typical VMs are quite small, with around 80 % of VMs having less than 2 vCPUs and 70 % of them less than 4 GB of memory;
- **Not all VM lifespans are equal**: Long-running VMs account for more than 95 % of the total core hours;
- **Not all VM provisioned resources are used**: While resource usage figures are only provided for the CPU, a collaboration with OVHcloud reveals that a production-scale IAAS infrastructure with guaranteed resources—*i.e.*, no oversubscription—succeeds in allocating most of the

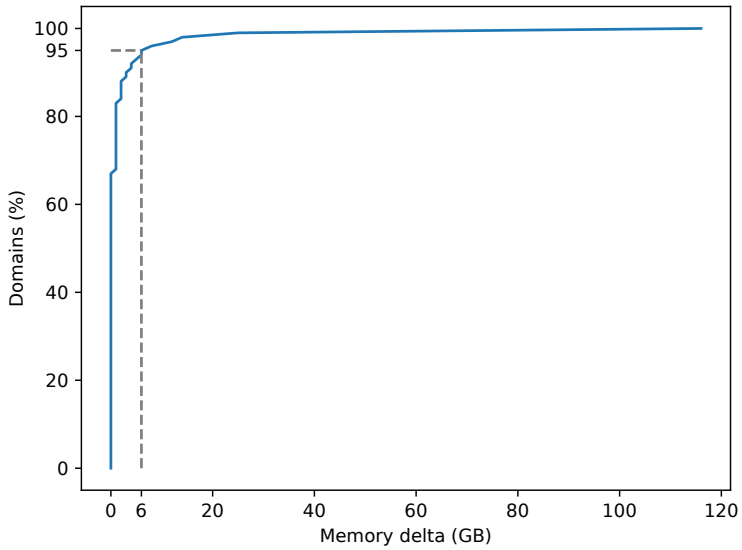

Fig. 4.2 CDF of memory allocation variations in an IAAS infrastructure

memory of PMs, with 80 % of their PMs reporting less than 35 GB of unallocated memory (cf. Figure 4.1). However, when including the amount of allocated memory that remains unused by their customers, one can observe that 75 % of their PMs have a potential of at least 35 GB of available memory.

Furthermore, when considering the evolution of memory allocation over time, one can observe that the memory effectively used by VMs remains stable. In particular, Figure 4.2 reports that 95 % of the PMs of OVHcloud infrastructure report on a variation of at most 6 GB over a 24-hour window.

Definition of the resource oversubscription ratio An *oversubscription ratio* is used to "map" a quantity of physical resources into virtual ones that can be exposed to third parties. For example, $\dot{\text{CPU}}$ and $\dot{\text{RAM}}$ are the oversubscription ratios commonly used by IAAS platforms to boost CPU and memory utilization, defined respectively as:

$$\dot{\text{CPU}}_{\text{IAAS}} = \frac{targeted(\text{vCPUs})}{\sum_{p \in PM} \text{cores}(p)} \tag{4.1}$$

$$\dot{\text{RAM}}_{\text{IAAS}} = \frac{targeted(\text{vRAM})}{\sum_{p \in PM} \text{RAM}(p)} \tag{4.2}$$

where $targeted(\text{vCPU})$ and $targeted(\text{vRAM})$ are the number of virtual resources that a Cloud infrastructure aims to offer for each PM, with regards to the number of cores and the quantity of memory exposed by the associated hardware configuration.

When statically defined at the scale of an IAAS, these oversubscription ratios rely on a high-level analysis of workload patterns and an accepted risk in terms of performance degradation. However, an IAAS is a general-purpose computing platform that can host a wide diversity of VMs, hence

inducing a different profile on resource utilization from one PM to another. Therefore, I believe that the oversubscription ratios should rather be estimated in real-time and at the scale of individual PMs to minimize resource over-utilization while maximizing VM performance.

Insights from Azure-like IAAS platforms IAAS bottlenecks may differ, depending on VM flavors, distribution and PM configuration. Based on Azure VM size distributions and IAAS configurations, I studied potential oversubscription limitations.

I applied *Operations Research* (OR) techniques to maximize the number of VMs for a given PM, while respecting VM size distribution reported by Azure. I considered a share of 4 GB memory per thread as a PM configuration baseline. Considering this PM configuration, and no resource oversubscription, the CPU is the bottleneck. For example, a 256-cores PM with 1 TB of memory can host a maximum of 101 VMs for a total of 251 vCPUs and 481 GB of VRAM. When increasing the CPU oversubscription to 2:1, more memory is consumed, with a total of 949 GB of VRAM provisioned across 204 VMs. A higher CPU oversubscription would, however, induce memory overload.

In the current Azure configuration, dynamic CPU oversubscription based on the sum of VM percentile is unlikely to improve resource usage compared to a static 2:1 ratio, as they do not oversubscribe memory [126]. Moreover, the sum of VM percentiles is known to overestimate actual requirements [17].

Capturing the effective resource utilization of hosted VMs VMs hosted by an IAAS are considered black boxes provisioned with a given amount of resources ordered by the customers. Nevertheless, from the perspective of a PM, system-level metrics can be monitored to better understand the resource utilization of hosted VMs. In particular, actual CPU and memory utilization can be observed using CPU usage and RSS, respectively.

In addition to these raw usage metrics, each PM can also monitor resource *overload signals*. A good indicator is the *scheduling latency* extracted from Linux scheduler statistics exposed by the *Process File System* (ProcFS) [127]. When overloaded, CPU time slices granted to each hosted process decrease, because the number of requested time slices is higher than the number physically available on the system. As a consequence, the scheduling latency—defined as the sum of task wait times—increases. Regarding memory, I used *page faults* count as memory signals. Major page faults are used by the Linux Kernel to increase the virtual address space of a given process. When overloaded, a new memory page requires the writing of an existing page, which can be accessed shortly after, resulting in a new page fault, thus increasing the overall amount.

The combination of *resource utilization metrics* and *resource overload signals* are the key features to better understand the effective exploitation of PMs by VMs. To achieve optimal usage of computing resources, every PM is expected to maximize its resource utilization metrics, while minimizing resource overload signals. To do this, PMs require a better understanding of their resource utilization profile by aggregating metrics, commonly adopted by the state of practice. Nevertheless, these

resource profiles are particularly relevant when the PM reaches a *quiescent state—i.e.*, their resource utilization is stable enough to be anticipated. I believe that the quiescent state of a PM can better reflect the actual resource availability and be exploited to increase the number of hosted VMs by gradually increasing the oversubscription ratio without triggering resource overload signals.

Dynamic resource utilization profiling of hosted VMs *Service-Level Agreement* (SLA) violation prevention constrains Cloud providers from sizing their infrastructure according to usage peaks. Quantiles and percentiles are commonly used to compute them at the cluster scale [128, 14]. A cluster resource usage distribution may be considered stable, making percentiles computation practical, but inappropriate at the scale of a PM due to potential instability. I consider it as an opportunity and leverage PM specificities, caused by their inner unique workloads evolution, to compute available resources using metrics closer to the real PM usage.

Given that PM reactivity is also a challenge, metrics computed at a PM scale must address two questions: how to ensure confidence in usage metrics computed from a smaller sample composed of a single PM? How to account for workload evolution that can be observed at the PM level? I propose to mitigate this issue with a *quiescent indicator*. This indicator answers the two previous questions by ensuring confidence in metrics computed on a quiescent PM and by properly reacting to a PM with an evolving workload.

Identifying quiescent states The quiescent state must be independently estimated for each resource of interest: a PM may have a stable memory usage, but an unstable CPU one. A resource is considered as *quiescent* whenever its current usage can be estimated from past observations. Thus, whenever the workload changes significantly, I consider that the PM resource left its quiescent state.

To the best of our knowledge, the detection of a quiescent state in IAAS PMs has never been extensively explored. It could be aligned with more general approaches in time series data mining [129], where similarity measures are utilized to identify specific sequences in a series [130, 131]. However, there are some major distinctions in our notion of quiescence. Particularly, the observation of a pattern not previously encountered is common, as VM-based workloads often exhibit a chaotic behavior [132] due to the variety of conditions affecting the load of each VM. These new patterns may still be associated with a quiescent state depending on their amplitude, as small changes may not significantly impact the overall trend.

Therefore, I evaluated different detection heuristics under multiple generated IAAS workloads. Under each workload, samples of resource usage are continuously monitored and attached to a time window of a fixed duration. Whenever a time window ends, I evaluate the PM quiescent state by comparing the current window samples to previous ones. To better understand the evaluation, I illustrated four resource usage traces in Figure 4.3. I want to compare if the latest received window (shown in red) may be considered as quiescent with regards to the historical data (in grey). The illustrated traces cover behaviors observed in production, where cases n°1 and n°4 should be labeled as quiescent, while cases n°2 and n°3 highlight a difference in scale in the new window.

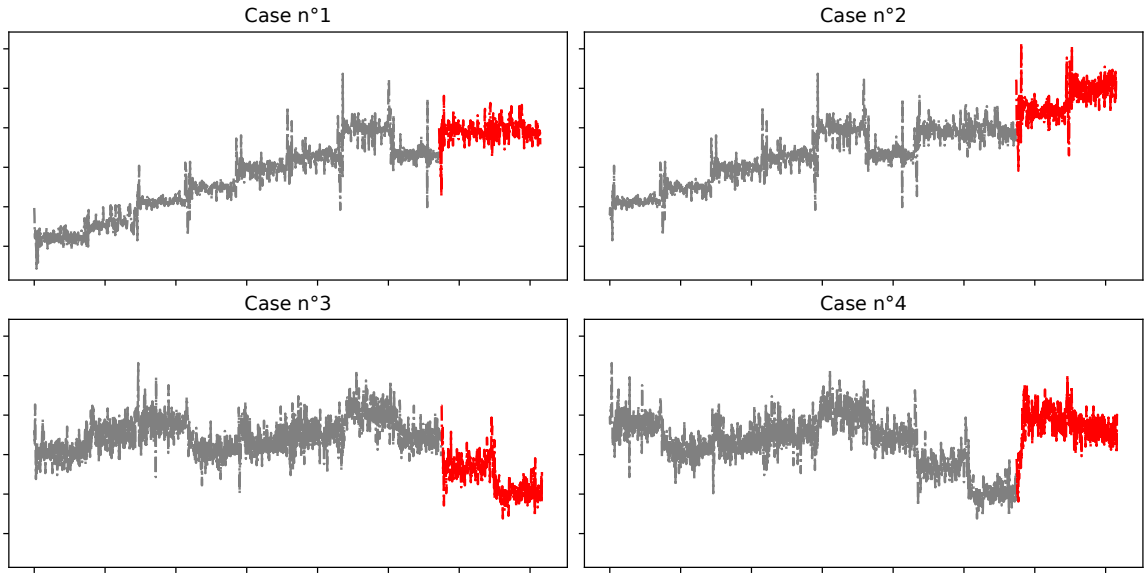

Fig. 4.3 Resource usage traces. Reference data is in grey, data seen as new in red

I now describe the stability detection heuristics I considered for this component of SCROOGEVM. As PM usage is usually compromised in a Gaussian distribution [133], the average classifier computes the new average value and compares it to bounds defined by average and standard deviation on historical data. It checks that the new average is included in $[\overline{old}-\sigma;\overline{old}+\sigma]$.

Percentiles are also commonly used to evaluate both VMs [14] and PM workloads [24] to account for exceptional situations—i.e., the ones likely to provoke SLA violations. The percentile classifier computes a given percentile on the new data and checks the following bounds: $[percentile(old)\times 0.8; percentile(old)\times 1.2]$.

The p-value classifier uses a common statistical test. I use a null-hypothesis significance test to compute the probability of obtaining the test result and reject it if it is below 5%.

Finally, the introduced quiescent classifier leverages a machine learning approach, using an *Long Short-Term Memory* (LSTM) model, known for its performance in predicting computing resource usage [134, 128]. As described in Algorithm 2, this classifier trains a model $\mathscr{M}_{LSTM}$ on $[i-n;i-1]$ historical windows (with $n\in\mathbb{N}, n\geq 1$) and then forecast the behavior observed during the last completed window i. To do so, the trained model $\mathscr{M}_{LSTM}$ is used to predict values on two series. The first one is on metrics set from the historical data, to evaluate the baseline model accuracy. The second one is on metrics extracted from window i. Then, the average error of both predictions is computed using *Root-Mean-Square Error* (RMSE). If the difference between the two predictions is significant, the PM is considered as `UNSTABLE`. As RMSE is expressed on the same scale as the unit being predicted, I consider the difference to be significant if it exceeds a percentage of the PM configuration. For example, a 256-cores PM with 1% threshold would tolerate a maximum difference

Table 4.1 Comparison of quiescent labels returned by classifiers

Case	**1**	**2**	**3**	**4**
Ground truth	**QUIESCENT**	**UNSTABLE**	**UNSTABLE**	**QUIESCENT**
Average	UNSTABLE	UNSTABLE	UNSTABLE	QUIESCENT
Percentile	QUIESCENT	QUIESCENT	QUIESCENT	UNSTABLE
***p*-value**	UNSTABLE	UNSTABLE	UNSTABLE	UNSTABLE
LSTM	QUIESCENT	UNSTABLE	UNSTABLE	QUIESCENT

of 2.56 cores between both projection average errors. In my experiments, the 1% threshold was sufficient on large PMs.

Algorithm 2 Quiescent state detection algorithm

Input Historical dataset, last window
Output Quiescent state

1: $\mathscr{M}_{LSTM} \leftarrow$ Generate model from historical dataset
2: $forecasted \leftarrow predict(\mathscr{M}_{LSTM}, historical\_set)$
3: $predicted \leftarrow predict(\mathscr{M}_{LSTM}, last\_window)$
4: $\delta \leftarrow |RMSE(forecasted) - RMSE(predicted)|$
5: **if** $\delta > threshold$ **then**
6: **return** UNSTABLE
7: **end if**
8: **return** QUIESCENT

LSTM training phase can be tuned by so-called hyper-parameters. I intentionally considered "low" values to reduce the ability of the resulting model to anticipate previously unseen behaviors. It also reduces the training phase to a few seconds, making it practical for live usage. One should be reminded that I do not use LSTM to predict future behaviors but to detect the occurrence of unforeseen behaviors. More specifically, I reduce the ability of LSTM to predict new behaviors by setting a few hidden layers (less than 10) and a low number of considered time steps.

I assess the LSTM classifier by comparing it to the other classification techniques: average, percentile, and *p*-value based. Table 4.1 summarizes the labels returned by each classifier when evaluating the red window of Figure 4.3. One can observe that the average classifier often proves inadequate in labeling a state of quiescence when the history exhibits an ascending or descending trend, as evidenced in the first case, due to its previous average value being biased. Similarly, the percentile classifier tends to exhibit bias based on the number of values that significantly deviate from the history within the latest window. The conventional bounds commonly employed for *p*-value hypothesis testing appear unsuitable in our specific context. However, employing an LSTM based approach appears to succeed in accurately detecting both unstable states.

To further assess these quiescent state classifiers, I consider the state-of-the-art metrics adopted for the evaluation of classifiers [135]. I labeled a sequence of windows covering the CPU traces of an IAAS platform, from which are issued the previous four examples and compared this ground truth to the labels returned by the four classifiers mentioned earlier. By counting the number of *true positives*

Table 4.2 Quantitative evaluation of quiescent state classifiers

Classifier	Accuracy	Precision	Recall	F-score
Average	0.68	0.8	0.52	0.63
Percentile	0.57	0.55	**0.87**	0.67
P-value	0.52	**1.0**	0.06	0.12
LSTM	**0.88**	0.93	0.84	**0.88**

(TP), *false positives* (FP), *true negatives* (TN), and *false negatives* (FN), I assessed the following indicators:

- Precision: $TP/(TP+FP)$
- Recall: $TP/(TP+FN)$
- Accuracy: $(TP+TN)/(TP+FP+FN+TN)$

In addition, the F-score, defined as the harmonic mean of the precision and the recall, can be used to get a performance score. As reported in Table 4.2, LSTM outperforms other classifiers with an F-score of 0.88. In the following sections, I therefore adopt it as the quiescent state detector of SCROOGEVM.

Estimating the available vCPU & vRAM resources Two VMs requiring the same amount of resources may have different workload patterns, due to the diversity of services hosted by Cloud infrastructures. When adopting a cluster-wide static oversubscription, a pessimistic approach tends to be adopted, leading to lower gains by potentially lowering the resource utilization of PMs. Therefore, instead of assuming the number of vCPUs and the available vRAMs are statically defined, based on the number of physical resources and the associated oversubscription ratio, this approach aims to continuously adjust the number of virtual resources to be offered by estimating their actual value depending on the PM quiescent state.

N-SIGMA method introduced in [17] may be seen as partially dynamic. In N-SIGMA, a fixed value of N is employed in the computation of peak resource usage using the formula $\overline{cpu}+N\times\sigma$, where $\overline{cpu}$ is the average value and σ the standard deviation. However, it should be noted that the standard deviation, which is utilized in this computation, is influenced to some extent by the quiescent state of the PM. Nevertheless, relying solely on the standard deviation as a proxy for resource stability is inadequate, as high amplitude values should be regarded as stable if the observed pattern is consistently reproduced over time. On a quiescent PM, the amplitude previously seen is likely to be reproduced, making a more optimistic peak prediction possible. We, therefore, introduce a quiescent-aware computation of N. At its maximum, it should handle worst-case scenarios observed in the Borg context, where there is a relatively high VM churn ($N=5$). Conversely, the minimum value of N is chosen to detect quiescent states while minimizing mispredictions. Through empirical deduction, a value of $N=2$ is determined. At each time slice, the PM's quiescent state is assessed, and the *streak* of the PM is adjusted accordingly, being either increased or decreased, based on predetermined values.

This dynamic adjustment of N allows SCROOGEVM for adaptive peak prediction based on the current state of the PM.

On a quiescent PM, the ratio is deliberately decreased using a low value, taking advantage of long-running VMs to improve resource utilization. The rest of this contribution used a 0.1 step.

On an unstable PM, the ratio is increased asymmetrically. However, it is important to consider that the associated standard deviation is likely to have increased in such cases. Additionally, the set of VMs on a given PM is unlikely to have been entirely refreshed since the previous observations. In practical terms, a decreasing value of 0.2 was found sufficient to mitigate the occurrence of most mispredictions.

Resources availability estimated by SCROOGEVM is, then, mapped to PM-scale oversubscription ratios, $\dot{\text{CPU}}_{PM}$ and $\dot{\text{RAM}}_{PM}$, with the following formula:

$$\dot{\text{CPU}}_{PM} = \frac{\mathit{provisioned}(\text{vCPUs}) + \lfloor \mathit{available}(\text{cores}) \rfloor}{\sum \text{cores(PM)}} \tag{4.3}$$

$$\dot{\text{RAM}}_{PM} = \frac{\mathit{provisioned}(\text{vRAM}) + \lfloor \mathit{available}(\text{RAM}) \rfloor}{\sum \text{RAM(PM)}} \tag{4.4}$$

where $\mathit{provisioned}(\text{vRES})$ captures the amount of currently provisioned virtual resources and $\lfloor \mathit{available}(\text{RES}) \rfloor$ the identified unused resources.[1]

Dynamic oversubscription can leverage existing Cloud platforms and monitoring infrastructure to be easily implemented. I assume that proposing a new VM orchestrator is out of the scope of this contribution, and I rather focus on the evaluation of PM-scale oversubscription ratios, which can eventually be leveraged by any legacy or new scheduler. Beyond state-of-the-art scheduling policies, I believe that the greedy oversubscription approach paves the way for the design of new policies that privileges the PM with the highest oversubscription ratio and/or the ones with the longest quiescent state, hence offering a more natural consolidation of Cloud resources.

4.1.2 Implementation of SCROOGEVM

This contribution, SCROOGEVM, involves a lightweight resource probe deployed on each PM. Metrics of interest are retrieved from three main sources: the virtualization platform (using `libvirt` in my case), ProcFS, and performance counters. Files of interest include `/proc/schedstat` and `/proc/[pid]/stat`, which expose *scheduler latency* and *page fault* metrics, respectively. Page faults are not extracted from the associated performance counter, as the combination of both major and minor faults does not permit to identify an overload situation. Monitoring of ProcFS metrics requires retrieving the VM PIDs, which is done via *cgroupfs*. For a given VM, reported values are the sum of its PIDs values.

Probe data is then exposed and processed by a PROMETHEUS monitoring solution [136] and updated periodically. Aggregation, storage, and analysis steps can be performed on a dedicated node to

[1] RES and vRES refers indifferently to CPU/RAM and vCPU/vRAM, respectively.

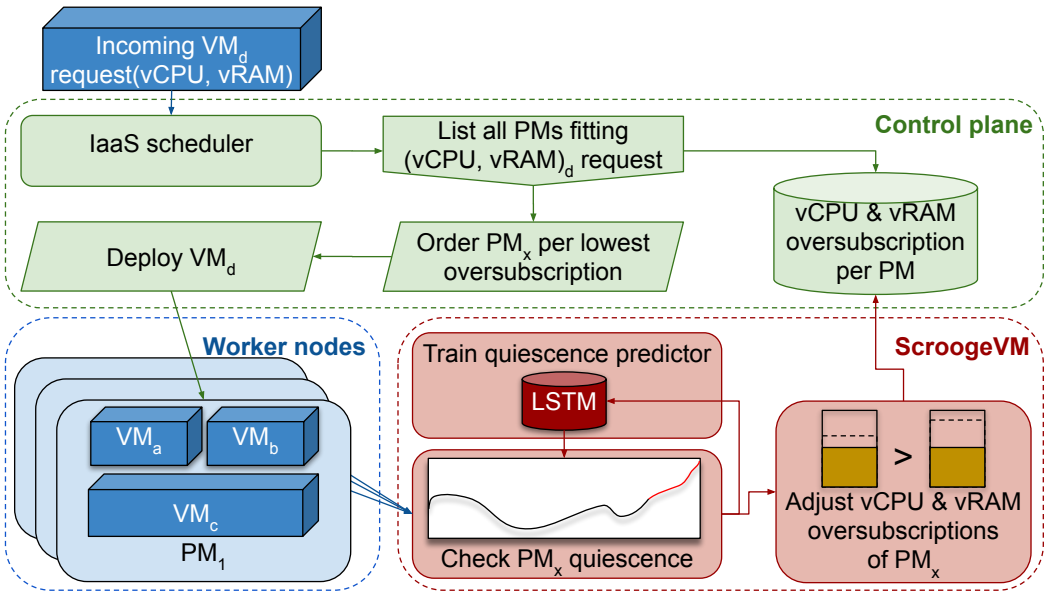

Fig. 4.4 Overview of the integration of ScroogeVM in an IaaS platform to guide the deployment of new VMs

address scalability issues in production environments. The current implementation of ScroogeVM reports on the estimated PM resources as a file, periodically updated, which can be parsed by any third-party orchestration solution.

Figure 4.4 illustrates the integration of ScroogeVM in an IaaS platform, like OpenStack, to guide the deployment of new VMs toward the least oversubscribed PMs and preserve the performance of hosted services. This integration leverages ScroogeVM to maintain an up-to-date oversubscription ratio per PM, hence offering a more dynamic indicator to select the most suitable PM that can satisfy a given VM deployment request. In this configuration, the resource oversubscription ratios (incl. vCPU and vRAM) are independently adjusted if a PM is in a quiescent state, which leads to a more careful and realistic estimation of resource availability that can be reported to the control plane. The quiescent state of each PM is periodically assessed at the end of time windows (of a configurable duration). That is, ScroogeVM uses the trained LSTM model to predict the vCPU and vRAM usages and compare them against recent history as described in Algorithm 2.

4.2 Empirical analysis

Cloud simulators, such as [17] and [30], lack valuable metrics for implementing greedy oversubscription. We, therefore, consider a more empirical protocol to study the impact of a set of parameters on dynamic oversubscription computation. This protocol is built upon the IaaS workload characteristics of Microsoft Azure [14] to consider a realistic Cloud infrastructure.

4.2.1 Experimental settings & evaluation protocol

I evaluated various ratios of oversubscription with different workload intensities by gradually increasing the number of VMs while monitoring different metrics. During the experiments, memory allocation was optimized using a ballooning mechanism that periodically reduces VM configurations to their observed max peak memory usage for a few seconds before restoring it. This principle was sufficient to reduce VM RSS without any noticeable impact on their performance.

Input workload Using the generator introduced in Chapter 3, we generated workloads matching statistics from the Microsoft Azure dataset [14]. The scripts obtained configure a representative workload of a PM hosting a set of production-scale VMs. With this method, VMs are deployed on a dedicated PM, allowing us to monitor the metrics of interest for SCROOGEVM. For each VM, a benchmark is selected to emulate its activity. The workload intensity of each VM is generated from the CPU utilization characteristics of the dataset. While I cannot ensure that a selected benchmark parameter value will exactly reach the targeted CPU intensity, I roughly ensure that the order of magnitude is reached based on empirical analysis.

Applications under study The platform workload relies on heterogeneous applications to emulate the diversity of situations covered in a Cloud infrastructure. Concretely, I deploy applications from the DeathStarBench as a representative microservices architecture [137], TPC-C for database workloads, a WordPress application, batch using CPU-intensive workloads, and idle VMs. When applicable, workload generation uses related benchmarking tools, which can be executed remotely or locally.

Metrics of interest In addition to the system metrics, I also monitored the performances obtained from applications hosted by the different VMs for validation purposes. Figure 4.5 depicts the evolution of hosted application performance through their 90^{th} percentile response times. A first set of VMs hosted a PostgreSQL database stressed under a TPC-C benchmark. A second set considers a DeathStarBench application *SocialNetwork* deployed with the "*read home timeline*" benchmark. Response times of all the VMs hosted on are aggregated in the graph. From the PM point of view, scheduling metrics were retrieved and a ratio was computed based on the scheduler latency and processes runtime. Due to the benchmark heterogeneity, performance degradations do not follow the same pattern, but one may observe that the lower the scheduler latency, the better the application performance. I also considered metrics, such as tail latency, requests throughput, and errors with similar observations. We, therefore, consider the scheduler latency as a relevant metric to assess VM applications from the host perspective in a black-box environment, like an IAAS platform. The impact on batch performance was not particularly investigated as I expected it to be similarly affected by an overall platform performance degradation.

Hardware settings In the experiments, I used the PM described in Table 4.3.

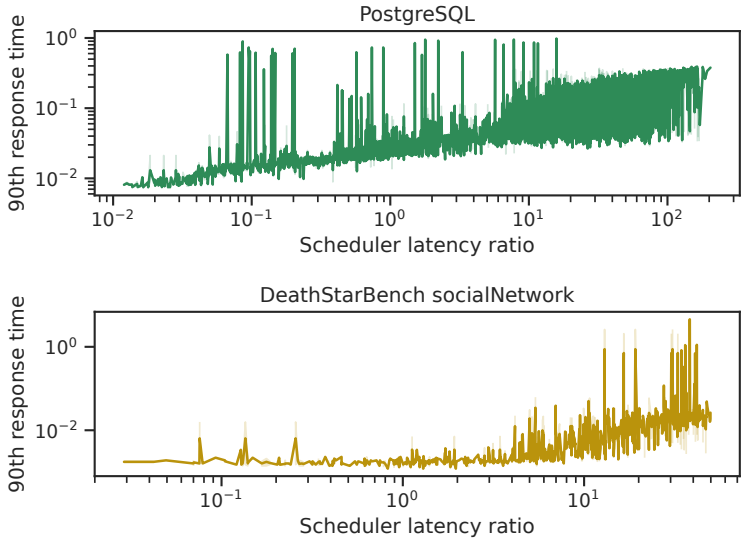

Fig. 4.5 Scheduler latency impact on VM performance (log-scale axes)

Table 4.3 Hardware configuration of IAAS PM

Processor	AMD EPYC 7662 64-cores ×2
Total threads	2 × 64 cores × 2 hyperthreads = 256
Memory	1 TB
Operating System	Linux Redhat 8.6
Virtualization Platform	QEMU & KVM 7.1

Figure 4.6 summarizes the profile of an experiment executed from the workload generator. An Azure-like workload was incrementally injected while maintaining the VM distribution and activity percentages. As a consequence, the number of VMs increases, while resource usage (such as CPU and memory) reaches their PM configuration limits. The ballooning mechanism previously described allowed SCROOGEVM to oversubscribe memory up to 3:1 and CPU up to 7:1. The workload was intentionally generated with unrealistic oversubscription ratios to investigate precursor indicators of an overloaded system.

4.2.2 Impact of the sampling period

The sampling period impacts the metrics volume and its associated processing capacity. As memory is exposed as a quantity, it is not subject to the smoothing effect. The main risk may be to miss potential peaks by only considering the last reported amount of memory, which is unlikely due to the RSS being generally linear. A more sudden change may be caused by a ballooning reduction window, with different effects depending on its conservative strategy level: if the reduction is exaggerated, a given VM on a PM may report an underestimated RSS. In the experiments, the sampling period did not have any significant impact on its oversubscription ratio.

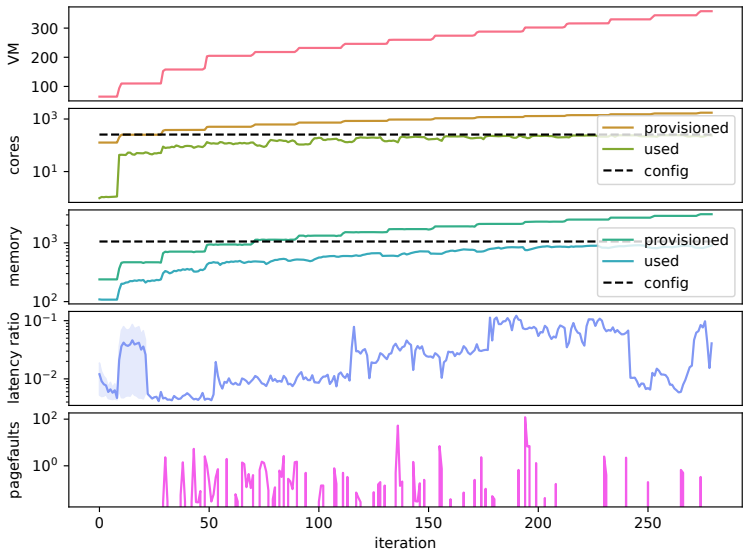

Fig. 4.6 Overview of our collected metrics

CPU utilization is typically reported as a percentage of the PM CPU time to the corresponding elapsed time. Therefore, a higher window length tends to smooth out extreme values and avoid overreaction. A longer window duration, therefore, leads to an increase in the estimated available quantities, as well as the associated oversubscription ratio. In the rest of this contribution, I used a 5s aggregation window.

4.2.3 Impact on the VM performances

In the context of an IAAS, performances from the perspective of the Cloud provider can only be evaluated as a black box, as no access to VM workload performance indicators is granted. To do so, I used the resource overload signals introduced in Section 4.1.1. Technically, system-wide scheduler latency is exposed by the Linux kernel. When the operating system is under-loaded, tasks assigned to the cores do not significantly affect the latency. I also consider runtime statistics to compute the average scheduling latency overhead. This is a more comprehensive metric to quantify an overload. One may potentially assess an overload severity considering latency evolution. Figure 4.7 compares the latency on two different workload intensities. Both workloads start from a $\dot{\text{CPU}} = 1{:}1$ oversubscription baseline composed of VMs distributed in terms of size and workload intensity, according to the Microsoft Azure dataset. The heavy one was progressively increased every two virtual days using 8-cores VMs with CPU-intensive tasks simulating aperiodic batch activities using up to all provisioned resources. The light workload was increased at the same frequency, with the same VM size. However, two-thirds were idle, and one-third were implemented with various workloads (including databases, micro-services, static websites, and batches) consuming less than the provisioned resources. Despite having the same VMs distribution, the latency distribution from the $\dot{\text{CPU}} = 2{:}1$ oversubscription is degraded on the heavy workload, compared to the light one. This can

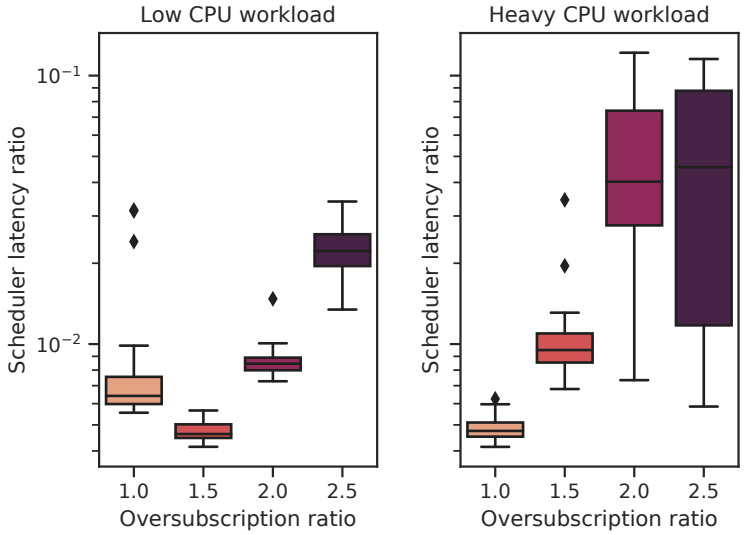

Fig. 4.7 CPU performance comparison for 2 workloads (log-scale Y axis, lower is better)

be deducted from a higher average value (in nanoseconds), but also from a larger dispersion. The light workload had no performance degradation, due to CPU-slicing competition at this oversubscription ratio. It did not exceed a 10% scheduler latency degradation until reaching $\dot{\text{CPU}} = 3.5{:}1$ (not visible in the graph). During the experiments, this threshold was unnoticeable from the VM's perspectives.

When it comes to memory, page faults evolve differently in an oversubscribed scenario. VMs RSS was periodically reduced, based on their usage profiling. Retrieved pages were not allocated to specific workloads, allowing any other VM process to use them, or allowing the probe to increase the amount of unused resources. As previously, I considered light and heavy memory workloads in Figure 4.8. Starting from $\dot{\text{RAM}} = 1{:}1$ (according to Microsoft Azure distribution), I incremented the heavy workload with 2 VMs of 16 GB every two virtual days, which simulated aperiodic memory-intensive activities consuming all their allocated memory. Moreover, no ballooning was enabled, leading to a never-decreasing linear behavior in RSS. The lighter workload used an Azure-like VM distribution that did not overload CPU resources and ballooning was enabled. Major page faults are retrieved from all active VMs using their PIDs file systems. The sum of major page faults of all VMs is reported for different oversubscription ratios. For the heavy workload, this ratio could not be increased more than $\dot{\text{RAM}} = 1.6{:}1$, due to an *Out Of Memory* (OOM) situation. On average, less than one major page fault occurs during the aggregation window (5 s) in an under-loaded situation. This average increases slightly in an overloaded PM, as outliers increase by an order of magnitude, with some reaching thousands of major page faults. Since there was no significant resource overload signal on the last oversubscription ratio, the extreme values must be considered when studying memory oversubscription. $\dot{\text{RAM}} = 1.4{:}1$ was the last healthy memory oversubscription ratio for the heavy workload, while the most representative workload was able to reach a ratio of $\dot{\text{RAM}} = 2.4{:}1$.

The performance feedback at the scale of a PM can be integrated by the cluster when the oversubscription ratio is periodically incremented. For example, the IAAS infrastructure can decide

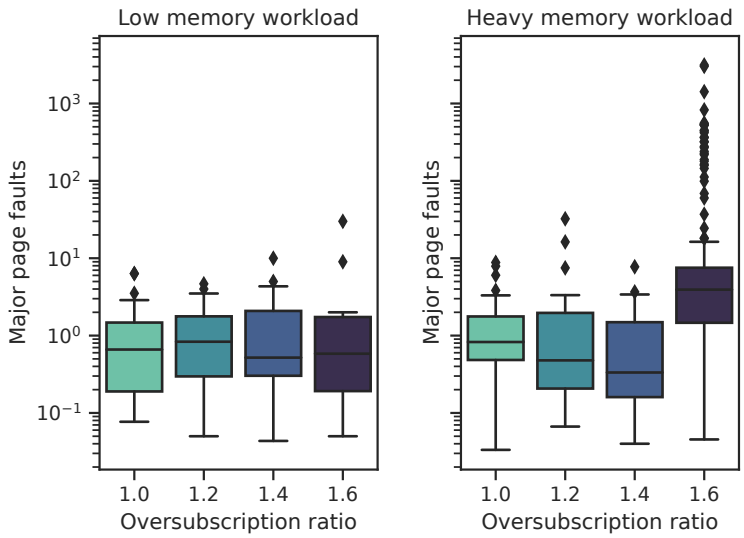

Fig. 4.8 Memory performance comparison for 2 workloads (log-scale Y axis, lower is better)

to balance performances across PM and, therefore, uses this feedback to prioritize the increase of oversubscription ratios for PM that are the least impacted by performance variations. Alternatively, when combined with application-level performance metrics, the performance feedback can also be used to tune *Service-Level Objective* (SLO) to make sure that SLA are not violated by the increase of oversubscription ratios.

4.3 Validation

I evaluated the greedy oversubscription strategy implemented by SCROOGEVM with other dynamic oversubscription estimation mechanisms.

The first method called doa, implements the *Dynamic Oversubscription ratio Adjustment* (DOA) mechanism described in [102]. DOA increases available resources by a fixed ratio of 5% of PM configuration until a 95% max resource usage is reached. Once this threshold is reached, the available resources are reduced to 50% of the PM configuration. The second method nsigma refers to N-SIGMA, configured with $N = 5$ as recommended in [17]. The third method is Borg-default alike (borg), which mimics a static oversubscription mechanism where deployed VMs are estimated to leave 10% of their resources unused. This percentage, corresponding to a 1.1:1 oversubscription, is based on the empirical study from [17]. The fourth method, RC-like (rclike), calculates the used resources by summing up the percentiles of hosted VMs. Specifically, the 99^{th} percentile is employed based on the analysis presented in [17].

Given that the aforementioned oversubscription strategies primarily focus on the CPU resource, I conduct a comparative evaluation of SCROOGEVM against these strategies by considering CPU traces derived from different IAAS workload traces.

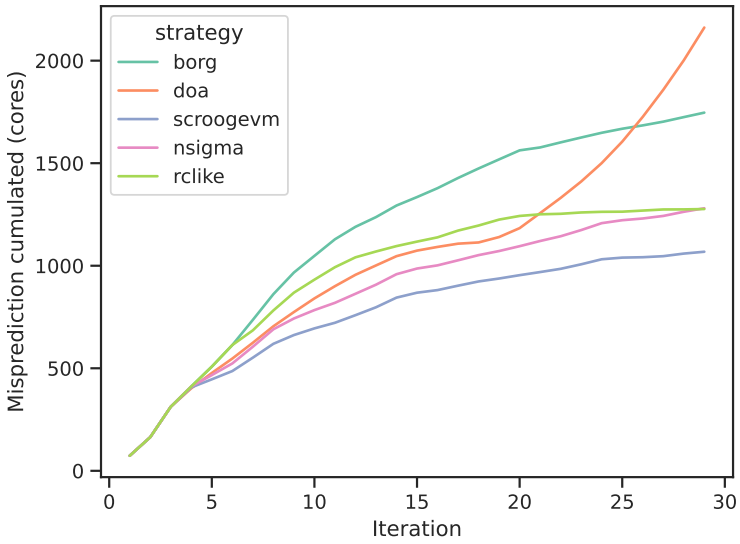

Fig. 4.9 Cumulated mispredictions (cores) under decreasing CPU workload (lower is better)

Table 4.4 Comparison of oversubscription strategies (decreasing CPU)

Strategy	mispredictions	violations	reductions
borg	6.8	0.0	0.2
doa	18.4	4.1	0.0
nsigma	5.0	0.0	0.3
rclike	5.0	0.1	0.2
scroogevm	4.2	0.0	0.3

For any given CPU usage trace, strategies are compared based on their predictions. At the end of each iteration, the discrepancy (denoted as δ) between the predicted resource usage at the beginning of the iteration and the actual usage observed during the period is computed. A positive δ reflects a conservative approach lowering the available resources. I label such a situation as a *misprediction* and I aim at minimizing mispredictions to improve the accuracy of the strategy. On the other hand, a negative δ reflects the overestimation of available resources, thus imposing performance penalties on the hosted VMs. This situation is referred to as a *violation*. In addition to the two prediction accuracy metrics, I also consider a third metric that focuses on the evolution of predictions. Specifically, I focus on the "reductions" in resources estimated as free over time. The objective is to keep these reductions to a minimum to ensure stability when deploying new VMs.

Figure 4.9 depicts the performances of each strategy on an IaaS workload reflecting a decreasing trend in the number of allocated resources (by reducing the number of provisioned VMs). The accumulated *misprediction* is used to emphasize underused CPU resources (expressed as cores) over a specific duration. Although oversubscription strategies may exhibit similar estimations at a specific iteration, the cumulative impact of even small variations can have significant consequences over time. Unsurprisingly, borg static oversubscription of 1.1 is one of the most pessimistic mechanisms. On

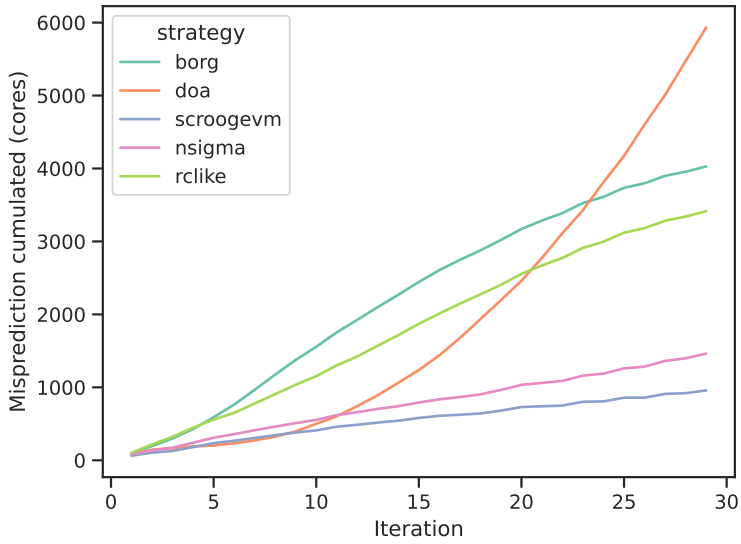

Fig. 4.10 Cumulated mispredictions (cores) under increasing CPU workload (lower is better)

Table 4.5 Comparison of oversubscription strategies (increasing CPU)

Strategy	mispredictions	violations	reductions
borg	15.7	0.0	0.0
doa	23.2	22.4	0.3
nsigma	5.7	0.0	0.2
rclike	13.3	0.0	0.0
scroogevm	3.7	0.0	0.2

the other hand, nsigma and rclike methods demonstrate similar predictions. It is noteworthy that the SCROOGEVM approach consistently exhibits the lowest misprediction.

Table 4.4 reports on the detailed cumulated metrics for each strategy. Mispredictions, violations, and reductions are expressed as ratios of PM configuration (cf. Table 4.3).

The high violation rate of DOA results from the heuristics that increase available resources as long as the usage threshold is not met. Resulting numbers are therefore unrealistic if the scheduler did not fully allocate the resources previously seen as available, which is highly dependent on VM arrival rate. Therefore, the resources seen as available may be underestimated, leading to high violations. It can be observed that SCROOGEVM achieves the lowest misprediction ratio without any violations, while also maintaining similar reduction phases compared to the other strategies. The oversubscription ratios calculated from the SCROOGEVM approach result in a misprediction gain of 0.8 compared to the nsigma strategy. This translates to a gain of 204 cores, concerning the PM settings, while maintaining realistic ratio values.

The behavior of oversubscription mechanisms can vary when applied to an IAAS workload that exhibits an increasing trend. This is primarily due to the lack of historical data available for newly deployed VMs. Consequently, strategies based on VM metrics tend to be more conservative, as

evidenced by borg and rclike in Figure 4.10. In contrast, strategies based on PM metrics—nsigma and scroogevm—demonstrate lower mispredictions.

Under an increasing trend, the performance of different strategies is compared in Table 4.5. The presence of long-running VMs contributes to higher oversubscription ratios. This is primarily due to the asymmetry between the calculation of used resources and available resources. Used resources are typically estimated from actual usage, while available resources are proposed based on the requested amount (e.g., at a 1:1 ratio) for newly provisioned resources. Consequently, the higher number of long-running deployed resources leads to increased oversubscription ratios because their impact is relatively lower compared to non-deployed resources. When comparing the mispredictions ratios, results highlight the effectiveness of the SCROOGEVM approach, which achieves a gain of 512 cores (twice the PM settings) compared to the nsigma strategy, without any violations.

Overall, one can observe that the solution implemented by SCROOGEVM outperforms the state-of-the-art of oversubscription strategies by delivering a greedy mechanism that accurately estimates the amount of available CPU resources that can be recycled and further provisioned, by a Cloud infrastructure. By leveraging quiescent state detection, SCROOGEVM only increases PM oversubscription ratio when available resources are expected to not be required in the future for its set of VMs.

I believe that the long-term gains can be significant. When considering N-SIGMA as the baseline maximizing the number of allocated VMs and minimizing the number of unused cores, one can observe that SCROOGEVM succeeds in deploying an average of 3 additional VMs per server, and in reducing the number of unused cores by 16%. Under favorable conditions (increasing trend), SCROOGEVM even improves the state-of-the-art by up to 7 additional VMs per server, with a reduction of 35% of unused cores, without SLA violation. In contrast to N-Sigma, the overhead primarily arises from the LSTM model training, taking approximately 1 second on a server without a discrete GPU unit. This level of overhead suggests practical viability for production contexts.

4.4 Conclusion

In this contribution, I propose a novel approach, named SCROOGEVM,[2] to implement a greedy resource oversubscription strategy at the scale of individual PMs. Instead of configuring static and cluster-wide parameters, I propose to dynamically estimate the optimal oversubscription ratio per PM, while maintaining performance goals. The degree of optimism in the oversubscription computation is adjusted per PM, instead of defining it globally as seen in other dynamic approaches.

This method, decoupled from the resource optimization of a given IAAS platform, allows SCROOGEVM to be generic and deployed with most Cloud schedulers operating an IAAS. I evaluated SCROOGEVM with an IAAS workload from Microsoft Azure and reported on potential gains exceeding state-of-the-art strategies.

[2]https://github.com/jacquetpi/scroogevm

While this work aimed to enhance the computation of current oversubscription ratios, it does not fundamentally challenge the current oversubscription paradigm, wherein a server is associated with a single ratio. Exploring the adaptation of VM hosting to better align with an oversubscribed context holds promise for improving resource packing efficiency. In the subsequent chapters of this thesis, we delve into the exploration of multi-oversubscribed PMs, based on either vCPUs or VM offerings.

Chapter 5

Introducing per-vCPU oversubscription

Abstract: *In this contribution, I first evaluate the degree of parallelism in the workloads of OVHcloud's client VMs by monitoring their per-vCPU usage. After identifying that vCPUs are not equally utilized, I propose a new oversubscription paradigm where some vCPUs offered to VMs are more powerful than others based on their underlying applied oversubscription ratio. The prototype, entitled SweetSpotVM, demonstrates that this paradigm can mitigate the performance degradation of oversubscription.*

Addressing the challenge of underutilized resources in Cloud DCs remains a significant concern [138, 139, 29], aiming to reduce both costs and ecological footprints of these virtual platforms. The consolidation of workloads onto a smaller set of PMs improves efficiency, considering the non-linear relationship between PM power consumption and workload [138, 140]. 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 [81, 15] to more passive approaches based on sharing [141]. Oversubscription 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 vCPUs exposed to 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. SMT, initially introduced in 1995 [69], 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, I propose exposing vCPUs with various performance levels to VMs.

This contribution introduces per-vCPU performance variations using a novel oversubscription paradigm. Instead of managing oversubscription at the granularity of a VM, I 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 other resources (oversubscribing to an n:1 ratio with $n > 1$) across various oversubscription levels.

In the remainder of this contribution, I first motivate the need for an oversubscription paradigm closer to the actual usage (cf. Section 5.1) based on an empirical analysis of OVHcloud production environment, one of the largest European Cloud operators [123]. We, then, detail how oversubscription can be implemented at the vCPU granularity (cf. Section 5.2). The prototype is evaluated in Section 5.3 using realistic IAAS workloads and reports on its ability to selectively guarantee computing resources. Finally, I conclude this work in Section 5.4.

5.1 Motivation

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

5.1.1 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 several 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 [14, 27, 28]. 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

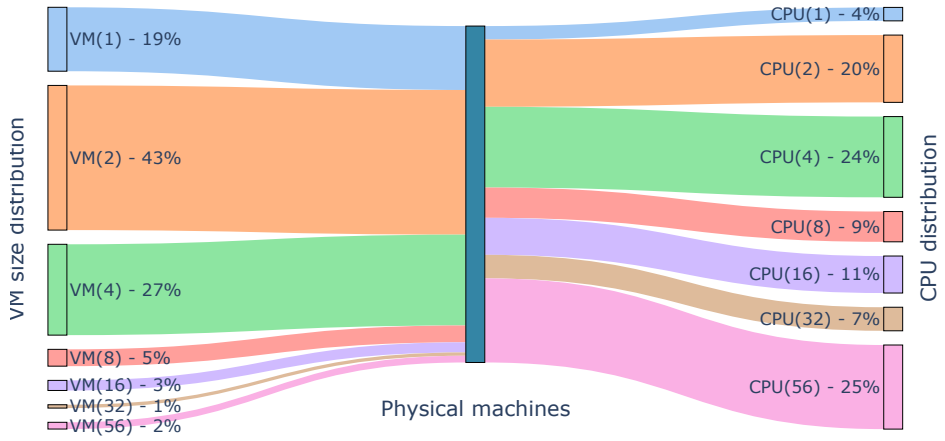

Fig. 5.1 Mapping the distributions of VM sizes to the physical CPUs provisioned by the OVHcloud infrastructure

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 a production-scale IAAS environment operated by OVHcloud, a major Cloud provider. 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. I consider a premium offering that delivers dedicated resources—i.e., no oversubscription is implemented (1:1)—thereby precluding the examination of contention situations.

Figure 5.1 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 is 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 [14], the share of provisioned CPU highlights that the largest VM configurations are the ones that consume most of the computing resources exposed by the PMs, with 76 % of the CPUs being provisioned by VMs requesting at least 4 vCPUs.

Beyond this first observation, I further dive into the effective usage of individual vCPUs by the different VM configurations. To do so, I derive a utilization metric, and I order the vCPUs from the most utilized (designated as vCPU0) to the least utilized (designated as $\mathsf{vCPU}(n-1)$, 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 CFS queue calibration mechanism [142]. The substantial differences observed in vCPU time after a 5-minute interval underscore the significance of these variations,

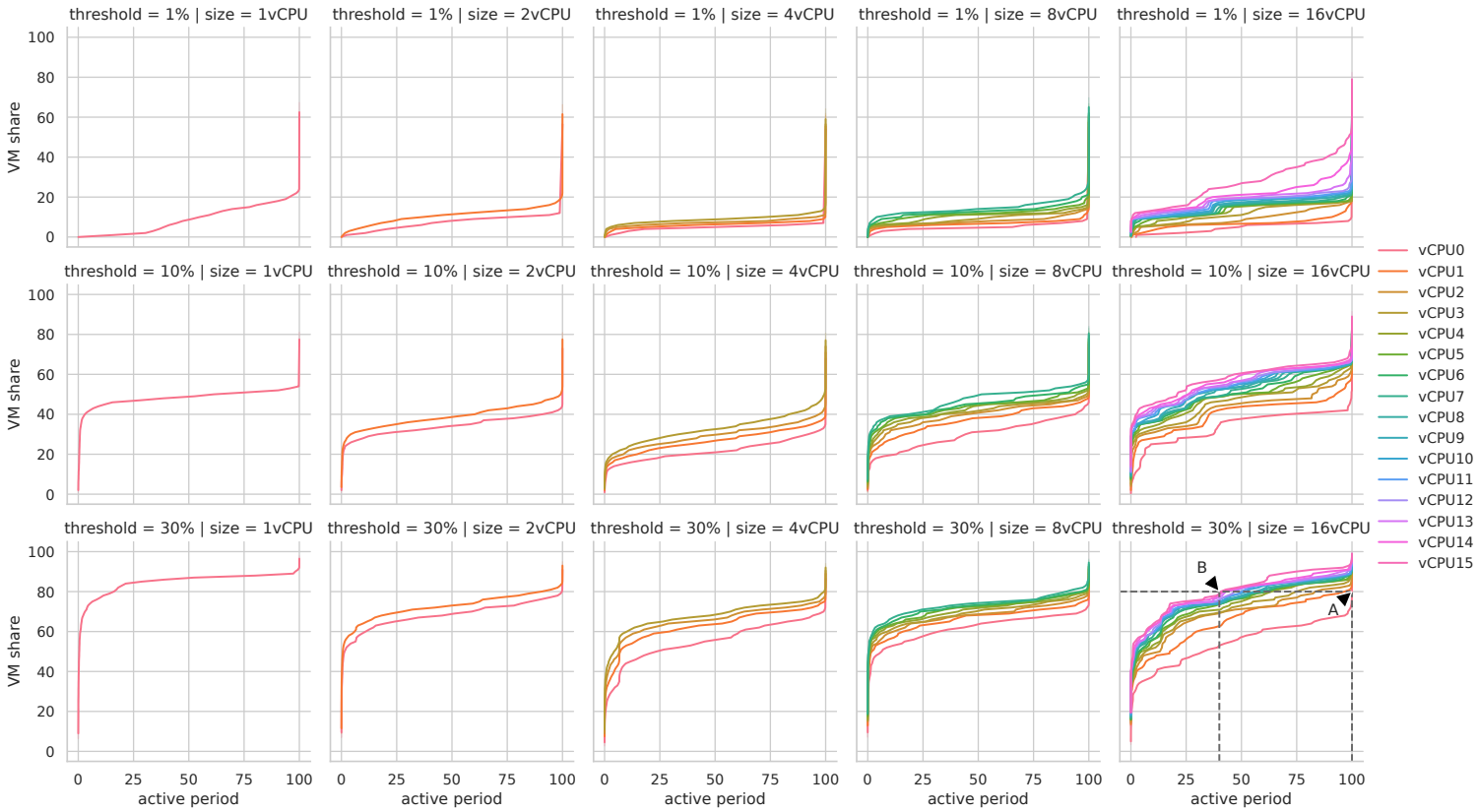

Fig. 5.2 CDF of individual vCPU utilization ratios of various VMs profiles hosted by OVHcloud

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, I examined 3 values to label a vCPU as active: 1%, 10%, and 30%.

The results are plotted as CDFs in Figure 5.2. 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*).

I first observe that the vCPU0 line can be distinguished 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 for 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.

I 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, many reasons 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.

5.1.2 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 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 in edge situations, leads to configure pessimistic ratios, hence missing resource optimization opportunities.

The mapping of the 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 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 [107] and workload heterogeneity [14, 17]. I believe that this shift in scope 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

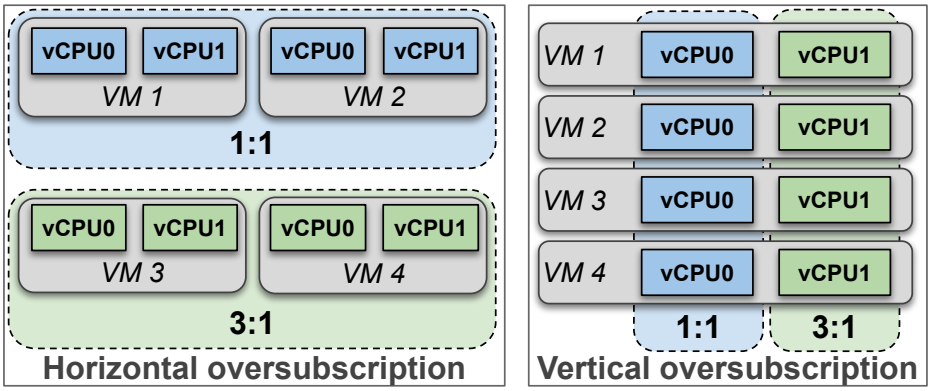

Fig. 5.3 Transitioning from horizontal CPU oversubscription to vertical oversubscription as implemented by SWEETSPOTVM

equally in a *horizontal* manner, as depicted in Figure 5.3, I 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.

In a similar way 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).

5.2 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, the 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, I explore the coexistence of various oversubscription ratios on a single PM.

5.2.1 Local scheduler

A Cloud scheduling architecture can be summarized as two main software components [143, 57].

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 ratio and its 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 5.2.2.

The *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.

5.2.2 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.

I identified and addressed two main challenges. Firstly, the performance and isolation of PM core selection is intricately linked with the PM topology, as established in existing literature [69]. However, the applicability of proposed heuristics across diverse architectures is imperative, given the heterogeneous configurations prevalent in Cloud data centers [107] Secondly, the size of the vCPU pools needs to be dynamically adjusted to cope with unforeseen IAAS workloads (covering VMs of various sizes).

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. For example, manufacturers' drivers can impact the scheduling by 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 this strategy. In practical implementation, instead of selecting individual cores, I opt for the selection of a range of cores, even in the case of premium vCPUs.

The selection of cores is undertaken to closely align 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 universally shared are common, such as in multi-socket architectures or with AMD EPYC processors.

Algorithm 3 Distance (Δ) computation between 2 cores

Require: $core_0$, $core_1$
Ensure: Δ
1: $\Delta \leftarrow 0$
2: **for** `<cachelevels>` **do**
3: **if** LEVEL$(core_0) ==$ LEVEL$(core_1)$ **then**
4: **return** Δ
5: **end if**
6: $\Delta \leftarrow \Delta + 10$
7: **end for**
8: **return** $\Delta +$ NUMA-DISTANCE$(core_0, core_1)$

To account for it, I introduce a core distance metric extending the NUMA distance [144]. 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. I retrieve this data and compute distances between each core, as described in Algorithm 3. While the incremental value is arbitrary, I 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, I rather considered them as a group and pin them to a range of physical CPUs. If the number of vCPUs matches the number of physical CPUs associated, the non-oversubscribed status is sustained, and the approach lets 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.

Pool resizing

A VM with n vCPUs may have a maximum of n different oversubscription 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 deployment. This flexibility accommodates the uncertainty associated with the number of VM creation requests, enabling this 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.

5.2.3 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 misleads 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 a 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 the vertical context, oversubscription may occur earlier, as I 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, I 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, my strategy relies on the pooling of oversubscribed vCPUs when feasible, effectively leveraging all resources that remain unallocated by the current hosted VMs. Upon deployments, I also prevent VMs from being oversubscribed against themselves by verifying that the pool size is greater than the requested virtual resources.

5.2.4 Oversubscription templates

In a vertical oversubscription scenario, the vCPUs of a given VM may be oversubscribed to different ratios. This is achieved using what I 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, I believe that using progressive oversubscription ratios is a good practice. The VM should be aware of the performance of its cores to leverage the most of the 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.

5.3 Empirical evaluation

In this section, I discuss how the vertical oversubscription paradigm was evaluated.

5.3.1 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, the 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 the development of a specific Linux scheduler and/or a driver.

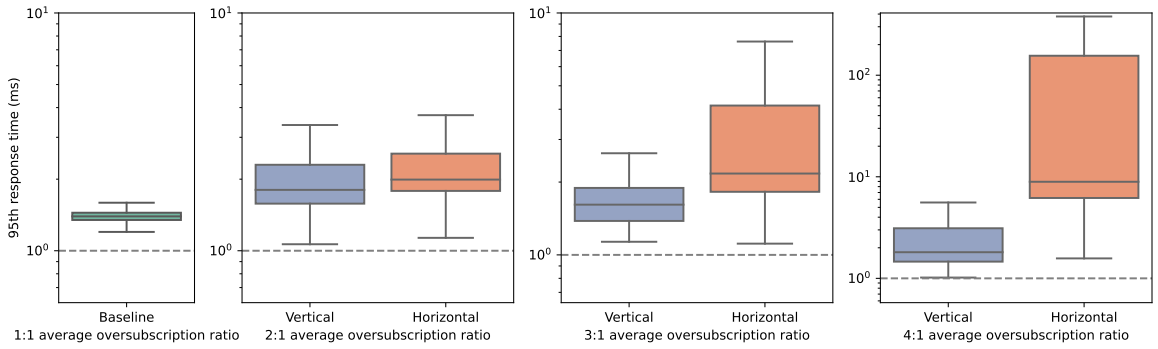

Fig. 5.4 DEATHSTARBENCH *social network* response time on different oversubscription scenarios

For the evaluation, I 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 VM cores. Deactivation is executed in decreasing core index order, deactivating the farthest index cores as they are the least powerful ones in the 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 it to handle an increasing intensity in the workload.

This approach implies a certain latency before activating all cores, however, I 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 I start in the worst case from having only vCPU0 activated).

5.3.2 On workload generation

The 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, CLOUDFACTORY (Chapter 3) was used for the workload generation. The workload was composed of an increasing number of VMs, each having 8 vCPU, overall matching the CPU usage observed in the Azure context [14].

The considered VMs hosted two distinct types of applications. Firstly, there was a micro-services architecture, known as *Social Network*, derived from the DEATHSTARBENCH [41]. 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.

5.3.3 Experimental IaaS platform

I used the PM described in Table 5.1 as a worker node. Of the 256 cores, 20 were kept for the monitoring and components parts, leading to 232 usable cores. Memory was not a limiting factor in all the experiments reported afterward.

Table 5.1 Hardware settings of the IAAS worker node

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

5.3.4 Experimental results

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

The evaluation and comparison of the considered oversubscription techniques was conducted in both horizontal (single ratio) and vertical (multiple ratios) approaches. Specifically, I 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 5.2. While these choices are arbitrary and can be customized by the Cloud provider, I 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 5.2 Oversubscription templates considered in the 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 5.4. 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 the vertical oversubscription template keeps maintaining VM performances closer to their optimal values, succeeding in hosting more VMs—i.e., 3× 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 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 use their most powerful cores (the less shared ones) in priority, performances can be close to the optimal.

The VM count is emphasized in Figure 5.5. The vertical 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, the 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

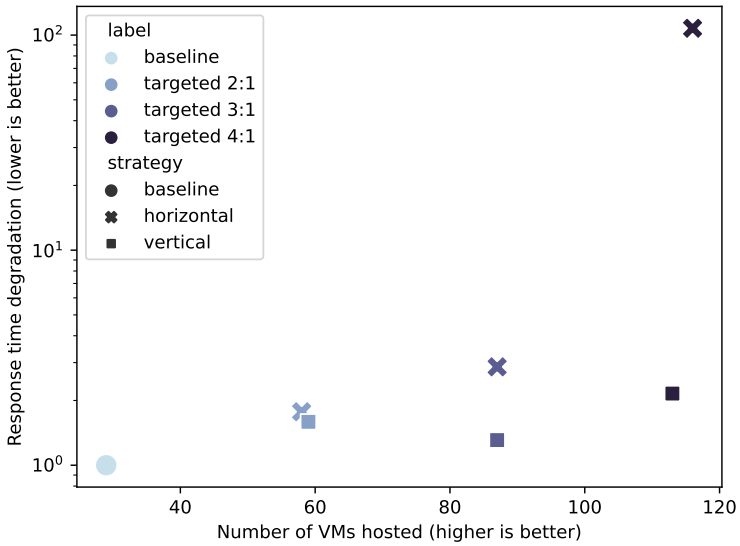

Fig. 5.5 Performance degradation in response time of the *social network app* of DEATHSTARBENCH (as multiple of the baseline, lower is better)

(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 mitigated by SWEETSPOTVM compared to state-of-the-art strategies enforcing horizontal oversubscription.

By dedicating CPUs to the VMs and oversubscribing others more aggressively, the 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, $r = 3.0$, the 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 5.6. The premium subset, dedicated to all vCPU0, provisions 59 physical cores under the considered 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.

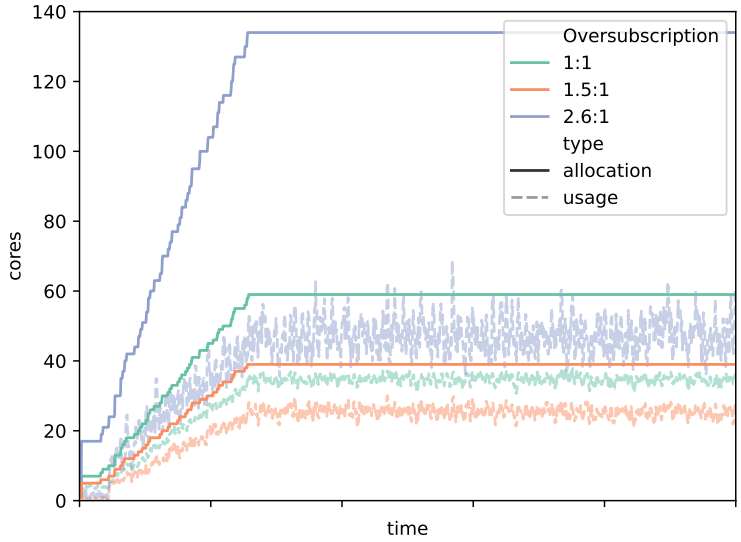

Fig. 5.6 Evolution of resources allocation and usage per oversubscription level for a SWEETSPOTVM template targeting 2:1

5.3.5 On the provisioning of small VMs

This contribution 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 5.1. Nonetheless, nothing prevents SWEETSPOTVM from hosting smaller VM configurations in conjunction with larger VMs as used to be the 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 drawn from Figure 5.2.

5.4 Conclusion

In conclusion, this contribution 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, I 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 $n > 1$).

This contribution is implemented as a functional software prototype, leveraging cache level distance between cores to efficiently segregate multiple oversubscription levels. The 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.

Different perspectives can be identified for this work, notably on the applied oversubscription templates. While these templates are defined statically in this contribution, I 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.

Furthermore, this contribution does not consider the existence of distinct premium tiers, as all VMs adhere to the same oversubscription template. In practice, certain VMs require all their resources to be guaranteed. Exploring the complementarity, in terms of hosting needs, between oversubscribed and non-oversubscribed VMs is addressed in the next chapter.

Software artefacts

For the sake of the reproducibility of the empirical results I shared in this contribution, the software prototype is made publicly available.[1] In particular, I documented an offline mode to reproduce reported experiments. Furthermore, the daemon used in the experiments to prioritize the cores with lower indexes inside the VMs is also available.[2]

[1]https://github.com/jacquetpi/sweetspotvm
[2]https://github.com/jacquetpi/cpu-staker/

Chapter 6

Balancing complementary oversubscription levels

Abstract: *In this contribution, I address the problem of scattered resources. After identifying that different oversubscription ratios may primarily fill different kinds of resources in their distinct clusters, I propose to colocate VMs of different ratios. I facilitate the integration into existing Cloud schedulers by exposing virtual clusters to the control plane and by enhancing their PM score-based selection mechanisms with a new metric. This introduced cluster architecture, referred to as SlackVM, reduced the number of servers by up to 9.6% compared to current packing strategies.*

Over the past decade, there have been notable improvements in the power efficiency of DCs. However, following the Jevons paradox [145], the increasing demand for data processing and storage has risen at an even greater rate, resulting in a continued upward trajectory in power consumption [11].

As some DCs are close to their maximum efficiency in terms of PUE [12], 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 PM [138, 139, 29]. In particular, consolidating 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. As such, unallocated PM resources may be leveraged by *spot* VM [75–77], while resources unused by tenants may be used by *harvest* VM [146]. *Disaggregated* VMs were also recently proposed to leverage resource fragmentation [83, 68]. 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 [147], as they lack guarantees on availability or performance. This contribution 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.

I 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, I aim to enhance PM packing with a focus on long-term services rather than small, ephemeral tasks (e.g., FaaS Computing [148]). It avoids reliance on a complex equilibrium notion between infrastructure gaps and client demands [149]. Furthermore, this approach effectively reduces the IAAS PM cluster size, contributing to an improvement in the carbon footprint of ICT, which is known to grow at a fast rate [11].

Oversubscription is commonly adopted to increase resource usage. This contribution 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, I demonstrate that different oversubscription levels may saturate different types of physical resources. By combining different oversubscription levels, I, therefore, illustrate the potential to leverage their complementary nature, thereby reducing the number of PMs required to handle an IAAS workload.

This 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. I assess this approach on both a physical platform and a simulator, evaluating performance and noticing significant gains at scale. Specifically, the 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 contribution, I explain why oversubscription co-hosting is important (Section 6.1), present an overview of SLACKVM architecture (Section 6.2), before diving into its design (Section 6.3 & Section 6.4). I evaluate this approach (Section 6.5) before concluding this work (Section 6.6).

6.1 Cloud resource balance

In this section, I discuss how oversubscription levels impact the packing of underlying PMs. I start by describing the VM resources allocation (Section 6.1.1) and then, I compare it to PM configurations to identify bottlenecks (Section 6.1.2).

6.1.1 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 [18]. While hypervisor constraints do not inherently preclude the proposition of intermediate sizes, this practice is

commonly adopted to facilitate VM packing [150]. 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 is often either CPU-bound, resulting in underutilized memory, or memory-bound, leading to underutilized CPU [151, 152]. Other types of resources, such as networks, are less likely to limit deployments [153] and are not considered in this paper.

To further dive into this issue, I analyzed the VM size distribution reported in [18] to compute the average VM size for both Azure and OVHcloud infrastructures (cf. Table 6.1). This first illustrates that allocations can significantly differ between Cloud providers.

Table 6.1 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

However, in an oversubscribed environment, resource allocation may deviate from the deployment request. It is now interesting to identify which server resource is depleted first under different oversubscription policies. To achieve this, I can compare the *Memory per Core* (M/C) ratio of hosted VMs to the PM configurations [154]. When these ratios do not align, one resource will typically be exhausted before the other, resulting in stranded resources. I begin by reporting on the M/C ratio of allocated resources.

Without oversubscription (1:1), I directly compute the M/C ratio from Table 6.1, 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 6.2 reports on the M/C ratio per Cloud provider, across three different levels of CPU oversubscription.

Table 6.2 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 my estimations of the M/C ratio for oversubscribed VMs, the average vCPU and vRAM deployments sizes were re-computed from the VM size distributions

where VMs having more than 8 GB were excluded. While this approximation may not be perfectly accurate, I 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 [155, 103], resulting in similar variations in the M/C ratio.[1].

6.1.2 Cloud resources collapse differently

While IAAS workload M/C ratio may evolve [154], 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. I 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.

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 an M/C ratio of 2 GB per core, all the workloads outlined in Table 6.2 experience memory saturation, as the minimal VMs ratio is higher (2.1 GB on Azure 1:1 level).

Nonetheless, I 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 ($5.8 > 4$).

[1]For instance, OPENSTACK's default oversubscription ratios are 16:1 for CPU and 1.5:1 for DRAM [101]

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, this contribution 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.

6.2 SLACKVM overview

A Cloud scheduler architecture consists of two key components [143, 57]. 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 (cf. Figure 6.1). 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'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 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.

I 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 6.1, 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*.

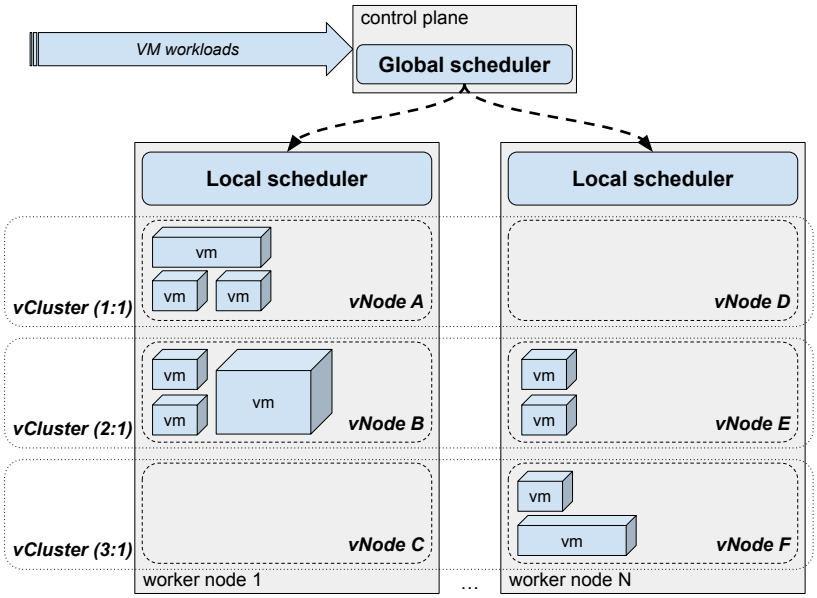

Fig. 6.1 Overview of SLACKVM components

In this 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. I delve into how the local scheduler manages vNodes in Section 6.3.

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. I explore the adaptation of Cloud scheduling for *vClusters* in Section 6.4.

6.3 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, the *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 this 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, I prefer to harness the dynamic capabilities inherent to the 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 6.3.1), 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.

I also discuss requirements in workload variability in Section 6.3.2

6.3.1 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 aims to allocate vNodes to achieve 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.

Favoring 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, I allocate each vNode to a separate physical socket, as this provides the best isolation on the same PM [156]. 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, I carefully examine the cache level being shared between cores. In a setting with n cache levels, I first attempt to guarantee isolation between cores at the n^{th} level. If not feasible, I 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 CFS, the Linux scheduler. This task is intricate 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. I only consider resources through collections of physical cores. The responsibility of selecting the most suitable core from the specified vNode is kept by 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 the core pinning strategy.

Exposing a virtual topology

To accommodate various CPU topologies, I leverage the core distance metric introduced in Chapter 5 (Algorithm 3). This metric considers the shared cache levels between two cores to provide a more complete evaluation of core proximity compared to the traditional NUMA distance NUMA distance [144].

When extending a vNode, I 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.

The computed distance between cores is what allows the local scheduler to be generic when pinning cores on physical machines. 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 in this context when a VM is being deployed or destroyed. Such events do not happen at a significant frequency in the realm of CPU operations.

6.3.2 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 this context, two remediation strategies can be identified.

Critical-size strategy

The principle behind oversubscription relies on the assumption that resources unused 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 as more VMs are deployed—as the probability of all VMs simultaneously reaching their peak usage diminishes, thus establishing a theoretical minimal deployment size threshold. A critical size may, therefore, be defined. In practice, a worker node starts being oversubscribed when the sum of its allocation exceeds its configuration. I argue that usage heterogeneity comes from VM usage and, therefore, should be taken into account using a VM count instead of a sum of allocation while defining the critical size.

Oversubscribed vNodes may adopt a strategy of waiting until they have reached a critical size before initiating any reduction in physical resources relative to virtual ones. This approach provides a robust guarantee of heterogeneity for each vNode. While this approach may seem intuitive, it has a significant side effect that can counteract the gains achieved in a baseline oversubscription scenario.

Since there is a critical size requirement for each oversubscription level (i.e., per vNode), it can result in an unrealistic number of VMs to host to satisfy the conditions of all oversubscription levels. This becomes especially challenging for PM with 32 cores or less, even with optimistic critical VM numbers defined (e.g., less than 5 VMs).

In practice, reaching this condition is difficult and, paradoxically, this strategy can increase the overall number of PMs required to host the VM workload, as oversubscription may not be applied in most vNodes.

In the rest of this contribution, this strategy was dismissed in favor of a second one, which capitalizes on pooling capabilities among vNodes based on their physical proximity.

Best-effort strategy

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 CFS 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, the best-effort 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. While the best-effort approach does not provide the same level of guarantees as the critical-size approach, I assert that I still adhere to the n:1 oversubscription condition and prevent VMs from being oversubscribed with themselves. Oversubscribed VMs are typically chosen by customers when performance is not their primary concern, and thus, I believe that this represents a reasonable compromise that balances resource efficiency and fairness in allocation.

6.4 Global scheduler incentive

Instead of proposing a new IAAS scheduler, I 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 [58, 157]. I introduce a new metric in existing scoring mechanisms to enhance VMs' complementarity. This new metric leverages the 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*. I 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 4 Progress towards *target ratio* computation

Require: $configPM$, $allocPM$, vm
Ensure: $progress$
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 emphasize 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. The methodology is rooted in this consideration.

Algorithm 4 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 prioritized. 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, my preference is to deploy the considered VM on a PM with a lighter workload, as this improves the 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 usage. Oversubscribed vNodes are considered through the PM allocation, and not, for example, the sum of exposed vCPUs. This approach enables the 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.

6.5 Empirical evaluation

The proposed solution was tested on both a physical platform, as detailed in Section 6.5.1, and a simulator, as described in Section 6.5.2.

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 this context, as I highlighted in Section 6.1, where the distribution of typical VM sizes from Cloud providers has a noticeable impact on the M/C ratio.

For both of the platforms, I employed CLOUDFACTORY (Chapter 3 as the workload generator. I extended the initial version of the generator to incorporate oversubscription considerations. These modifications enabled this 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 the evaluation.

6.5.1 Evaluation in the wild

I now turn my attention to presenting an example of the operational behavior of the *local scheduler* in the context of a physical platform. Prior research has extensively examined the performance implications of pinned resources [156, 158, 159]. My focus is on comparing the performance of this strategy with the baseline scenario, where a PM hosts VMs at a single oversubscription level without pinning considerations. Additionally, I aim to evaluate the *local scheduler* ability to isolate VMs from distinct vNodes.

Physical experimentation settings

In the experiments, I used the PM described in Table 6.3. The hardware settings of this worker include 256 threads and 1 TB of memory, resulting in an M/C ratio of 4 GB per thread.

Table 6.3 Hardware settings of the 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

I 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 VM, the CPU usage patterns obtained from CLOUDFACTORY were translated into application loads. Among the VMs, 10% was set to idle, 60% underwent a CPU benchmark using `stress-ng` [160], and the remaining VMs were composed of interactive applications. Specifically, I selected the *social network application*, a micro-service architecture from the DEATHSTARBENCH [41], 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.

I considered three distinct oversubscription levels: 1:1, 2:1, and 3:1. In the baseline scenario, the three oversubscription levels are hosted separately. The 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. The *local scheduler* does not impose a limit on the considered oversubscription levels and can host more vNodes with more oversubscribed

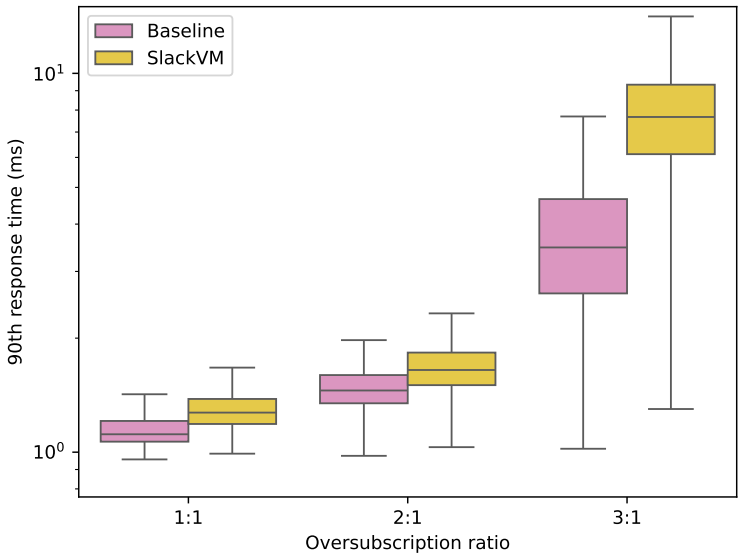

Fig. 6.2 Comparison of 90^{th} percentile response times for the DEATHSTARBENCH *Social network app* (log-scale Y axis)

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 these 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 slice contention).

The 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.

Performance results

Performances between SLACKVM and the dedicated clusters were compared through the 90^{th} response times that I measured during the empirical experiments. Observed median values are reported in Table 6.4, while the performance distribution per oversubscription ratio can be visualized in Figure 6.2.

Table 6.4 Performance comparison by the median of the 90^{th} response times measured

Oversubscription levels	Baseline (ms)	SLACKVM (ms)
1:1	1.16	1.27 (×1.09)
2:1	1.46	1.65 (×1.13)
3:1	3.47	7.67 (×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, I demonstrate with SLACKVM that performance can still be isolated within a single PM. In my experiment, all three oversubscription levels were hosted concurrently on a single PM.

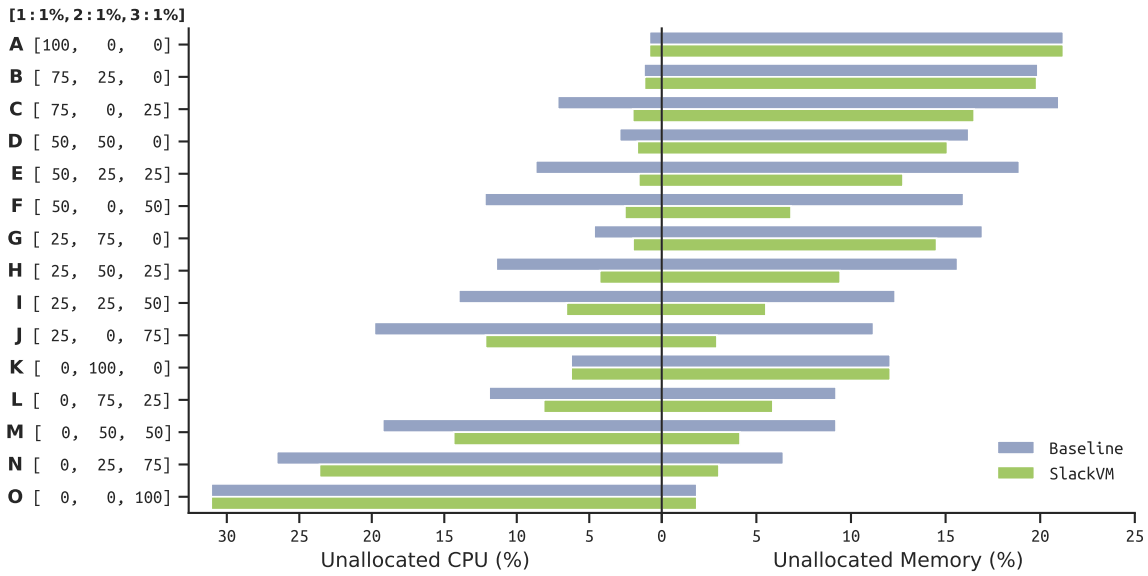

Fig. 6.3 Comparison of unallocated resource ratios between *dedicated clusters* (baseline) and SLACKVM when considering the OVHcloud setups

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 [161], the vNodes performance overhead is primarily due to the heterogeneity between cores. In a classic setting, CPU scheduler mechanisms do not exploit 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. Interestingly, this performance penalty remains limited in non-oversubscribed environments.

On the one hand, one can observe that the heuristics used by SLACKVM affect the oversubscribed VMs in priority, which are—by design—less prone to enforcing performance guarantees with strict 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, it is now interesting to investigate 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).

6.5.2 Evaluation at scale

Evaluating IAAS schedulers is known to be challenging, due to the lack of detailed information on current solutions used in production [162]. Cloud schedulers compute a fitting score for suitable PMs

based on hundreds of undisclosed rules [58, 14, 60, 59]. I choose to focus on improving packing efficiency. I evaluate the newly introduced metric, which indicates progress toward the perfect M/C ratio and its impact on reducing cluster size. First-fist scheduling serves as the baseline to evaluate it. This scheduling strategy is commonly employed to assess packing efficiency [162, 163], as it fills existing servers before considering new ones for deployments [55, 163]. In practice, Cloud providers may guide workload packing by adjusting the weight of this metric in their scoring mechanism, alongside their other criteria.

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

Simulated experimentation settings

I also implemented SLACKVM in the CLOUDSIMPLUS simulator [31], which is a derivative of CLOUDSIM [30]. Specifically, I created a particular worker type, utilizing the *local scheduler* heuristics to accommodate VMs from various oversubscription levels. Additionally, I 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, I adopted the provided specifications from OVHcloud and Azure Cloud providers. As Cloud providers do not disclose the share of VMs oversubscribed at each level, I 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. I simulated DC workloads over a week, adhering to the arrival and departure rates of VMs. For each workload, a CLOUDSIMPLUS simulation was initiated, starting from an empty cluster and progressively increasing until the minimal number of PMs was determined. Each PM within the cluster offered 32 cores and 128 GB of memory, resulting in an M/C ratio of 4 GB per core. To account for the typical largeness of Cloud workloads, I express gains in percentage values, as the approach scales with the cluster size.

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 better packing).

On the reduction of stranded resources Figure 6.3 compares the share of unallocated resources for various distributions of oversubscription levels. These distributions are ordered from the least oversubscribed (distribution A, including 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 6.3 highlights the resource allocation biases in the DC using the baseline *First-Fit* scheduling. These biases are a combination of the individual limiting factors 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 the 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%. The 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 the vNodes is let as a future work.

On the reduction of the cluster size Beyond resource allocation, I also study the PM gains achieved from the distributions involving the above 3 oversubscription levels, as depicted in Figure 6.4. 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 an 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 6.3, as well as in the values reported along the diagonal in Figure 6.4.

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

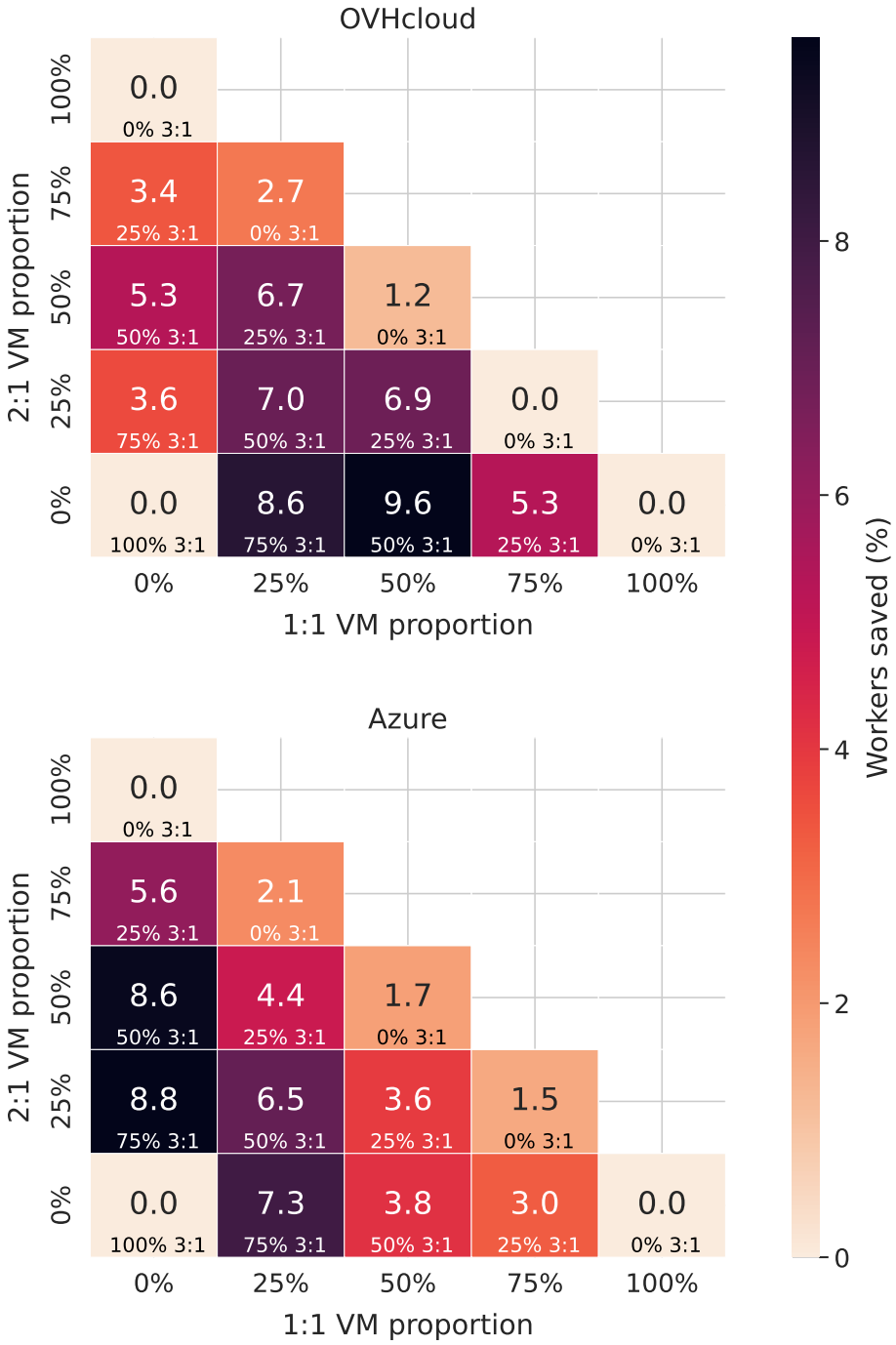

Fig. 6.4 SLACKVM gains in terms of PM (%) for various oversubscription distributions (the 3:1 VM distribution corresponds to the 100 complement of the other two distributions)

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 the 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 the approach required only 75 PMs overall. This highlights the significant PM utilization optimization achieved by SLACKVM through the use of the 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 configuration 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% of 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 the 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 this 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 the analysis but can be considered as an additional benefit of this approach.

Production-ready schedulers may therefore benefit from incorporating the 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.

6.6 Conclusion

In this contribution, I have demonstrated that different oversubscription levels can saturate different physical resources. Building upon this insight, I explored the complementarity of oversubscription levels. I introduced SLACKVM, an 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. The 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.

Software artefacts

The *local* and *global schedulers* implementations are publicly available.[2] To encourage the reproduction of the results, the extended version of CLOUDSIMPLUS[3] and CLOUDFACTORY[4] are also publicly available online. The latest repository contains instructions on how to re-generate simulations. An offline mode is notably documented to reproduce reported experiments.

[2]https://github.com/jacquetpi/slackvm
[3]https://github.com/jacquetpi/cloudsimplus-slackvm
[4]https://github.com/jacquetpi/cloudfactory-slackvm

Chapter 7

Conclusion

While Cloud Computing has made significant strides in improving energy efficiency within the ICT *sector, its potential remains constrained by the underutilization of provisioned servers. Considering the environmental footprint of server manufacturing and the superior energy efficiency achieved at higher usage levels, a crucial objective is to maximize the utilization of existing hardware before investing in new infrastructure. In this context, oversubscription is a potential solution, recognizing that not all clients require all their resources all the time. However, Cloud providers have been hesitant to adopt oversubscription in production widely, preferring instead to utilize alternative workloads to fill resource gaps. Through this thesis, I have demonstrated that oversubscription can indeed play a complementary role. The potential gains are substantial, and exploring new methods of utilizing oversubscription can lead to improvements in performance, server packing efficiency, and overall resource utilization.*

7.1 Contributions

In my contributions, oversubscription was proven to be an efficient way to counterbalance unused resources. While its performance degradation can be mitigated by carefully studying the oversubscription ratio (SCROOGEVM) or by adapting the paradigm (SWEETSPOTVM), we also discovered that oversubscription is an effective method to reduce unallocated resources (SLACKVM). By combining the gains from both unused and unallocated resources, the utilization of PMs can be significantly increased, leading to more efficient overall resource usage.

7.1.1 Improving IaaS experiments using realistic users' behavior

To facilitate the study of IAAS platforms, I developed a workload generator capable of producing experiments that mirror typical client behaviors. As behaviors may vary between different Cloud providers, influenced by factors such as the VM catalog offered by the provider, I initially introduce

a library aimed at promoting the sharing of high-level statistics. This library parses typical Cloud datasets and enables a given Cloud provider to share its characteristics without necessitating the publication of the entire dataset. I illustrate the potential differences between providers by introducing the context of OVHcloud and comparing it to two other providers. Subsequently, I present a generator capable of parsing these statistics and generating a workload in the form of a script. The generator supports scripts for both simulator-based and platform-based experiments.

7.1.2 Computing oversubscription ratios under stability consideration

The process of oversubscription relies on calculating a ratio that determines how much more virtual resources are offered compared to physical resources. While this ratio has traditionally been determined manually, recent approaches have demonstrated that dynamic computation at the server level allows for a more optimistic oversubscription, leading to greater gains. This is achieved by considering the specific workload characteristics of individual servers. I propose customizing the computation of the per-server ratio by taking into account their quiescent state. I define a method for assessing this quiescent state using an LSTM model and propose an algorithm to adapt the optimistic degree of an oversubscription computation. Through the evaluation, I demonstrate that this model offers a means to enhance current oversubscription computation without necessitating any live migrations.

7.1.3 Introducing per-vCPU oversubscription

Through the analysis of traces from OVHcloud IAAS platform, I initially highlight that while large VMs constitute a minority in the total VM count, they are responsible for the majority of vCPUs provisioned in the cluster. Upon further examination of the individual usage of vCPUs on large VMs, I establish that they exhibit heterogeneity, implying that their internal workloads cannot be uniformly distributed. Building upon this insight, I propose a novel oversubscription paradigm. This approach involves exposing vCPUs of varying performance levels, with some being more powerful than others, through underlying oversubscription segregation on the host. I propose and evaluate a prototype, proving that this approach enables the deployment of the same number of VMs associated with highly oversubscribed environments while mitigating performance degradation.

7.1.4 Balancing complementary oversubscription levels

While oversubscription traditionally aims to utilize resources left unused by clients, I demonstrate its potential to also reduce the number of unallocated resources. A VM deployment involves requesting various types of resources, with CPU and memory being key components. When either of these resources is fully allocated, the server cannot accommodate new VM deployments. Through an analysis of the ratio between CPU and memory in both VM creation requests and server hardware, I observe that premium VMs tend to primarily utilize the CPU of their host, whereas oversubscribed VMs tend to utilize memory. Consequently, I propose co-hosting premium and non-premium VMs on

the same PMs to leverage their complementary resource usage patterns. Utilizing a newly introduced progress indicator aimed at achieving the optimal memory per core ratio, I demonstrate potential savings of up to 9.6% of servers.

7.2 Perspectives

Finally, I propose perspectives on this work by outlining short-term extensions, related to the oversubscription principle, and describing long-term objectives, related to the broader scope of reducing the impact of Cloud computing.

7.2.1 Short-term perspectives

Implementation of multiple and dynamic ratios per server

While this thesis delved into the potential of utilizing oversubscription with finer granularities, it explored two distinct visions. Firstly, I enhanced oversubscription at the server level by considering its quiescent state, as discussed in Chapter 4. Secondly, I introduced multiple oversubscription levels within a single server by segregating its resources, as detailed in Chapters 5 and 6. However, these visions are not mutually exclusive, and there may be value in exploring the capabilities enabled by multiple concurrent and dynamic oversubscription levels. Specifically, dynamic oversubscription levels provide the flexibility to choose a more appropriate proxy for defining resource allocation, such as preserving performance to a desired level rather than adhering strictly to a predefined virtual-to-physical resource ratio. I believe that dynamic oversubscription is essential for the broader application of the principles outlined in Chapter 5, where defining a per-vCPU oversubscription level may present practical challenges.

On others resources dynamic oversubscription

This thesis predominantly focused on exploring CPU oversubscription within an IAAS context. However, it's noteworthy that other common processing units, such as GPUs, are gaining prominence, especially with the widespread adoption of *Large Language Model*s (LLMs). While GPUs serve different use cases compared to CPUs, they tend to consume more power and are also more expensive per unit. Consequently, there exists both a financial and environmental incentive to use the minimal quantity. Recent research conducted on an Alibaba GPU cluster sheds light on the fragmentation of workloads in DCs [164]. This fragmentation underscores what I perceive as potential opportunities for oversubscription in GPU usage.

As GPU may be shared between clients transparently [165], diving into oversubscription capabilities is interesting.

Linux schedulers cooperation with oversubscription

Recent kernel changes have simplified the evaluation of alternative Linux schedulers. Leveraging eBPF capabilities, process scheduling mechanisms can now be dynamically adjusted at runtime, eliminating the need for kernel recompilation or restart. This opens up possibilities for exploring diverse schedulers tailored to specific needs, such as those of IAAS workloads. Investigating synergies between oversubscription mechanisms and scheduling policies could enhance resource utilization or preserve performance. Additionally, addressing side-channel attack concerns by considering scheduling strategies, such as round-robin placement to mitigate cache-level conflicts, presents a promising avenue for further research.

7.2.2 Long-term perspectives

Sharing is caring

By putting multiple clients on computing units, oversubscription is a way to share the physical resources. While oversubscription leads to time slice share based on the number of processes running, other kinds of proxy can be used to manage it. As an example, [166] proposed to share resources based on a requested frequency while [139] proposed to share resources based on SLOs. Exploring the impact of the chosen proxy, especially in a black-box context, can be a way to improve packing.

Reducing is caring

The specific optimization required for Cloud applications remains unclear. Particularly, while reducing platform usage is essential to meet the Paris Agreement targets, there is a lack of studies on the feedback mechanisms provided by Cloud providers to their clients. It is important to investigate whether the resource usage feedback provided to clients is accurate, including whether it accounts for any "margin" and, if so, the magnitude of this margin. Additionally, understanding whether clients typically follow this feedback and identifying what types of feedback would be most effective for encouraging sustainable practices is crucial. This research could help develop more precise and actionable feedback systems, ultimately promoting more efficient resource use and aiding in the alignment with global sustainability goals.

Renouncing is caring

The work presented in this thesis is only relevant in the context of stable or decreasing demand for Cloud services. Evaluating the environmental impact of a service relative to its social or environmental utility is therefore crucial. Conducting a *Life Cycle Assessment* (LCA) of a virtualized service remains an open question, especially given the dynamic nature of resource allocation, whether through scaling (vertical/horizontal) or oversubscription. The usage of resources is dynamic, and power consumption is influenced by the workloads that are co-located on the same servers. Developing a model that

accurately captures both the utility of a service, likely using a social sciences-based approach, and its environmental impact should be an integral part of a comprehensive decision-making framework.

References

[1] T. W. Crowther, H. B. Glick, K. R. Covey, C. Bettigole, D. S. Maynard, S. M. Thomas, J. R. Smith, G. Hintler, M. C. Duguid, G. Amatulli, *et al.*, "Mapping tree density at a global scale," *Nature*, vol. 525, no. 7568, pp. 201–205, 2015.

[2] WWF, "Living planet report 2022," 2022. Available at https://www.worldwildlife.org/pages/living-planet-report-2022.

[3] S. Seibold, M. M. Gossner, N. K. Simons, N. Blüthgen, J. Müller, D. Ambarlı, C. Ammer, J. Bauhus, M. Fischer, J. C. Habel, *et al.*, "Arthropod decline in grasslands and forests is associated with landscape-level drivers," *Nature*, vol. 574, no. 7780, pp. 671–674, 2019.

[4] K. Lee and M. Greenstone, "Air quality life index annual update," *Energy Policy Institute, University of Chicago*, 2021.

[5] I. T. Cousins, J. H. Johansson, M. E. Salter, B. Sha, and M. Scheringer, "Outside the safe operating space of a new planetary boundary for per-and polyfluoroalkyl substances (pfas)," *Environmental Science & Technology*, vol. 56, no. 16, pp. 11172–11179, 2022.

[6] C. Mora, B. Dousset, I. R. Caldwell, F. E. Powell, R. C. Geronimo, C. R. Bielecki, C. W. Counsell, B. S. Dietrich, E. T. Johnston, L. V. Louis, *et al.*, "Global risk of deadly heat," *Nature climate change*, vol. 7, no. 7, pp. 501–506, 2017.

[7] The Shift Project, "Energy-climate scenarios: Evaluation and guidance," 2019. Available at https://theshiftproject.org/en/article/energy-climate-scenarios-evaluation-guidance-report/.

[8] L. Grimal, I. Di Loreto, N. Burger, and N. Troussier, "Design of an interdisciplinary evaluation method for multi-scaled sustainability of computer-based projects. A workbased on the Sustainable Computing Evaluation Framework (SCEF)," *LIMITS Workshop on Computing within Limits*, 2021.

[9] A. Barrau, *L'hypothèse K.: la science face à la catastrophe écologique*. Edition Grasset, 2023.

[10] C. Freitag, M. Berners-Lee, K. Widdicks, B. Knowles, G. S. Blair, and A. Friday, "The real climate and transformative impact of ict: A critique of estimates, trends, and regulations," *Patterns*, vol. 2, no. 9, p. 100340, 2021.

[11] 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.

[12] 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-centre-facilities-2023-09-05_en.

[13] J. Guo, Z. Chang, S. Wang, H. Ding, Y. Feng, L. Mao, and Y. Bao, "Who limits the resource efficiency of my datacenter: An analysis of alibaba datacenter traces," in *Proceedings of the International Symposium on Quality of Service*, pp. 1–10, 2019.

[14] 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, Association for Computing Machinery, 2017.

[15] 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.

[16] S. A. Baset, L. Wang, and C. Tang, "Towards an understanding of oversubscription in cloud," in *2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE 12)*, 2012.

[17] N. Bashir, N. Deng, K. Rzadca, D. Irwin, S. Kodak, and R. Jnagal, "Take it to the limit: Peak prediction-driven resource over-committment in datacenters," in *Proceedings of the Sixteenth European Conference on Computer Systems*, EuroSys '21, p. 556–573, Association for Computing Machinery, 2021.

[18] 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)*, (Boston, United States), pp. 81–91, IEEE, 2023.

[19] P. Jacquet, T. Ledoux, and R. Rouvoy, "ScroogeVM: Boosting Cloud Resource Utilization with Dynamic Oversubscription," *IEEE Transactions on Sustainable Computing (TSUSC)*, 2024.

[20] P. Jacquet, T. Ledoux, and R. Rouvoy, "SweetspotVM: Oversubscribing CPU without Sacrificing VM Performance," in *24th IEEE/ACM international Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, (Philadelphia, United States), IEEE, 2024.

[21] P. Jacquet, T. Ledoux, and R. Rouvoy, "SlackVM: Packing Virtual Machines in Oversubscribed Cloud Infrastructures," in *26th IEEE International Conference on Cluster Computing (CLUSTER)*, (Kobe, Japan), IEEE, 2024.

[22] P. Jacquet, T. Ledoux, and R. Rouvoy, "La chasse au gaspillage dans le cloud et les data centers," *The Conversation, France*, 2023.

[23] 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 *SOSP*, pp. 153–167, ACM, 2017.

[24] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and qos-aware cluster management," *SIGPLAN Not.*, vol. 49, p. 127–144, feb 2014.

[25] Y. Zhang, G. Prekas, G. M. Fumarola, M. Fontoura, I. Goiri, and R. Bianchini, "History-Based harvesting of spare cycles and storage in Large-Scale datacenters," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, (Savannah, GA), pp. 755–770, USENIX Association, Nov. 2016.

[26] J. Mambretti, J. Chen, and F. Yeh, "Next generation clouds, the chameleon cloud testbed, and software defined networking (sdn)," in *2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*, pp. 73–79, 2015.

[27] Alibaba, "The alibaba clusterdata201708 trace data," 2018. Available at https://github.com/alibaba/clusterdata.

[28] Google, "Borg cluster traces," 2019. Available at https://github.com/google/cluster-data.

[29] 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.

[30] 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.

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

[32] S.-H. Lim, B. Sharma, G. Nam, E. K. Kim, and C. R. Das, "Mdcsim: A multi-tier data center simulation, platform," in *2009 IEEE International Conference on Cluster Computing and Workshops*, pp. 1–9, 2009.

[33] H. Casanova, A. Legrand, and M. Quinson, "Simgrid: A generic framework for large-scale distributed experiments," in *Tenth International Conference on Computer Modeling and Simulation (uksim 2008)*, pp. 126–131, IEEE, 2008.

[34] A. Lebre, J. Pastor, and M. Südholt, "Vmplaces: A generic tool to investigate and compare vm placement algorithms," in *Euro-Par 2015: Parallel Processing: 21st International Conference on Parallel and Distributed Computing, Vienna, Austria, August 24-28, 2015, Proceedings 21*, pp. 317–329, Springer, 2015.

[35] M. Silva, M. R. Hines, D. Gallo, Q. Liu, K. D. Ryu, and D. d. Silva, "Cloudbench: Experiment automation for cloud environments," in *Proceedings of the 2013 IEEE International Conference on Cloud Engineering*, IC2E '13, (USA), p. 302–311, IEEE Computer Society, 2013.

[36] B. Sun, B. Hall, H. Wang, D. W. Zhang, and K. Ding, "Benchmarking private cloud performance with user-centric metrics," in *2014 IEEE International Conference on Cloud Engineering*, pp. 311–318, 2014.

[37] Council, Transaction Processing Performance, "Tpc benchmark c standard specification," 1990.

[38] SPEC, "Spec cpu® 2017," 2017. Available at https://www.spec.org/cpu2017/.

[39] SPEC, "Specmail2009," 2009. Available at https://www.spec.org/mail2009/.

[40] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, A. Fox, and D. Patterson, "Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0," in *Proc. of CCA*, vol. 8, p. 228, Citeseer, 2008.

[41] 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.

[42] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, (New York, NY, USA), p. 143–154, Association for Computing Machinery, 2010.

[43] M. Rak and G. Aversano, "Benchmarks in the cloud: The mosaic benchmarking framework," in *2012 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pp. 415–422, IEEE, 2012.

[44] G. Aversano, M. Rak, and U. Villano, "The mosaic benchmarking framework: Development and execution of custom cloud benchmarks," *Scalable Computing: Practice and Experience*, vol. 14, no. 1, pp. 33–46, 2013.

[45] M. Cunha, N. Mendonca, and A. Sampaio, "A declarative environment for automatic performance evaluation in iaas clouds," in *2013 IEEE Sixth International Conference on Cloud Computing*, pp. 285–292, IEEE, 2013.

[46] N. Yigitbasi, A. Iosup, D. Epema, and S. Ostermann, "C-meter: A framework for performance analysis of computing clouds," in *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 472–477, 2009.

[47] B. Varghese, L. T. Subba, L. Thai, and A. Barker, "Container-based cloud virtual machine benchmarking," in *2016 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 192–201, 2016.

[48] M. B. Chhetri, S. Chichin, Q. B. Vo, and R. Kowalczyk, "Smart cloudbench – automated performance benchmarking of the cloud," in *2013 IEEE Sixth International Conference on Cloud Computing*, pp. 414–421, 2013.

[49] SPEC, "Specvirt datacenter 2021," 2021. Available at http://spec.org/cloud_iaas2018/.

[50] V. Makhija, B. Herndon, P. Smith, L. Roderick, E. Zamost, and J. Anderson, "Vmmark: A scalable benchmark for virtualized systems," tech. rep., Technical Report TR 2006-002, VMware, 2006.

[51] Y. Chen, A. S. Ganapathi, R. Griffith, and R. H. Katz, "Towards understanding cloud performance tradeoffs using statistical workload analysis and replay," *University of California at Berkeley, Technical Report No. UCB/EECS-2010-81*, 2010.

[52] E. F. Boza, C. San-Lucas, C. L. Abad, and J. A. Viteri, "Benchmarking key-value stores via trace replay," in *2017 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 183–189, 2017.

[53] J. Yin, X. Lu, X. Zhao, H. Chen, and X. Liu, "BURSE: A bursty and self-similar workload generator for cloud computing," *IEEE Trans. Parallel Distributed Syst.*, vol. 26, no. 3, pp. 668–680, 2015.

[54] SPEC, "Spec cloud iaas 2018," 2018. Available at http://spec.org/cloud_iaas2018/.

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

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

[57] 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.

[58] 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.

[59] 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.

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

[61] Z. Zhang, C. Li, Y. Tao, R. Yang, H. Tang, and J. Xu, "Fuxi: a fault-tolerant resource management and job scheduling system at internet scale," in *Proceedings of the VLDB Endowment*, vol. 7, pp. 1393–1404, VLDB Endowment Inc., 2014.

[62] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for {Fine-Grained} resource sharing in the data center," in *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*, 2011.

[63] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*, pp. 1–16, 2013.

[64] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in *SIGOPS European Conference on Computer Systems (EuroSys)*, (Prague, Czech Republic), pp. 351–364, 2013.

[65] X. Wang, H. He, Y. Li, C. Li, X. Hou, J. Wang, Q. Chen, J. Leng, M. Guo, and L. Wang, "Not all resources are visible: Exploiting fragmented shadow resources in shared-state scheduler architecture," in *Proceedings of the 2023 ACM Symposium on Cloud Computing*, SoCC '23, (New York, NY, USA), p. 109–124, Association for Computing Machinery, 2023.

[66] C. Cérin, T. Menouer, W. Saad, and W. B. Abdallah, "A new docker swarm scheduling strategy," in *2017 IEEE 7th International Symposium on Cloud and Service Computing (SC2)*, pp. 112–117, 2017.

[67] A. Ruprecht, D. Jones, D. Shiraev, G. Harmon, M. Spivak, M. Krebs, M. Baker-Harvey, and T. Sanderson, "Vm live migration at scale," *SIGPLAN Not.*, vol. 53, p. 45–56, mar 2018.

[68] 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.

[69] 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.

[70] I. Stoica and H. Abdel-Wahab, "Earliest eligible virtual deadline first: A flexible and accurate mechanism for proportional share resource allocation," *Old Dominion Univ., Norfolk, VA, Tech. Rep. TR-95-22*, 1995.

[71] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, p. 46–61, jan 1973.

[72] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *IEEE Computer*, vol. 40, 2007.

[73] D. Wong and M. Annavaram, "Knightshift: Scaling the energy proportionality wall through server-level heterogeneity," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 119–130, 2012.

[74] G. Prekas, M. Primorac, A. Belay, C. Kozyrakis, and E. Bugnion, "Energy proportionality and workload consolidation for latency-critical applications," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, p. 342–355, Association for Computing Machinery, 2015.

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

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

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

[78] P. Ambati, I. Goiri, F. Frujeri, A. Gun, K. Wang, B. Dolan, B. Corell, S. Pasupuleti, T. Moscibroda, S. Elnikety, M. Fontoura, and R. Bianchini, "Providing SLOs for Resource-Harvesting VMs in cloud platforms," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 735–751, USENIX Association, Nov. 2020.

[79] B. Reidys, J. Sun, A. Badam, S. Noghabi, and J. Huang, "BlockFlex: Enabling storage harvesting with Software-Defined flash in modern cloud platforms," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, (Carlsbad, CA), pp. 17–33, USENIX Association, July 2022.

[80] Y. Zhang, I. n. Goiri, G. I. Chaudhry, R. Fonseca, S. Elnikety, C. Delimitrou, and R. Bianchini, "Faster and cheaper serverless computing on harvested resources," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, (New York, NY, USA), p. 724–739, Association for Computing Machinery, 2021.

[81] 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.

[82] W. Zhang, B. Chen, Z. Han, Q. Chen, P. Cheng, F. Yang, R. Shu, Y. Yang, and M. Guo, "{PilotFish}: Harvesting free cycles of cloud gaming with deep learning training," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pp. 217–232, 2022.

[83] 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.

[84] S. Legtchenko, H. Williams, K. Razavi, A. Donnelly, R. Black, A. Douglas, N. Cheriere, D. Fryer, K. Mast, A. D. Brown, A. Klimovic, A. Slowey, and A. Rowstron, "Understanding Rack-Scale disaggregated storage," in *9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 17)*, (Santa Clara, CA), USENIX Association, July 2017.

[85] A. Klimovic, C. Kozyrakis, E. Thereska, B. John, and S. Kumar, "Flash storage disaggregation," in *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, (New York, NY, USA), Association for Computing Machinery, 2016.

[86] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient memory disaggregation with infiniswap," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pp. 649–667, 2017.

[87] Y. Qiao, C. Wang, Z. Ruan, A. Belay, Q. Lu, Y. Zhang, M. Kim, and G. H. Xu, "Hermit: Low-Latency, High-Throughput, and transparent remote memory via Feedback-Directed asynchrony," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, (Boston, MA), pp. 181–198, USENIX Association, Apr. 2023.

[88] Z. Ruan, M. Schwarzkopf, M. K. Aguilera, and A. Belay, "AIFM: High-Performance, Application-Integrated far memory," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 315–332, USENIX Association, Nov. 2020.

[89] C. Wang, H. Ma, S. Liu, Y. Li, Z. Ruan, K. Nguyen, M. D. Bond, R. Netravali, M. Kim, and G. H. Xu, "Semeru: A Memory-Disaggregated managed runtime," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 261–280, USENIX Association, Nov. 2020.

[90] Z. Ruan, S. Li, K. Fan, M. K. Aguilera, A. Belay, S. J. Park, and M. Schwarzkopf, "Unleashing true utility computing with quicksand," in *Proceedings of the 19th Workshop on Hot Topics in Operating Systems*, pp. 196–205, 2023.

[91] C. A. Waldspurger, "Memory resource management in vmware esx server," *SIGOPS Oper. Syst. Rev.*, vol. 36, p. 181–194, dec 2003.

[92] J.-H. Chiang, H.-L. Li, and T. cker Chiueh, "Working set-based physical memory ballooning," in *10th International Conference on Autonomic Computing (ICAC 13)*, pp. 95–99, USENIX Association, June 2013.

[93] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Geiger: monitoring the buffer cache in a virtual machine environment," *ACM Sigplan Notices*, vol. 41, no. 11, pp. 14–24, 2006.

[94] P. Lu and K. Shen, "Virtual machine memory access tracing with hypervisor exclusive cache.," in *Usenix Annual Technical Conference*, pp. 29–43, 2007.

[95] Linux Documentation, "Memory hot(un)plug," 2022. Available at https://www.kernel.org/doc/html/latest/admin-guide/mm/memory-hotplug.html.

[96] Linux Documentation, "Kernel samepage merging," 2022. Available at https://www.kernel.org/doc/html/latest/admin-guide/mm/ksm.html.

[97] Linux Documentation, "zram," 2022. Available at https://docs.kernel.org/admin-guide/blockdev/zram.html.

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

[99] 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.

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

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

[102] H. Wang, H. Shen, and Z. Li, "Approaches for resilience against cascading failures in cloud datacenters," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 706–717, 2018.

[103] 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.

[104] J. Sheng, L. Wang, F. Yang, B. Qiao, H. Dong, X. Wang, B. Jin, J. Wang, S. Qin, S. Rajmohan, Q. Lin, and D. Zhang, "Learning cooperative oversubscription for cloud by chance-constrained multi-agent reinforcement learning," in *Proceedings of the ACM Web Conference 2023*, WWW '23, (New York, NY, USA), p. 2927–2936, Association for Computing Machinery, 2023.

[105] V. Govindaraju, V. Raghavan, and C. R. Rao, *Big data analytics*. Elsevier, 2015.

[106] M. C. Cohen, P. W. Keller, V. Mirrokni, and M. Zadimoghaddam, "Overcommitment in cloud services: Bin packing with chance constraints," *Management Science*, vol. 65, no. 7, pp. 3255–3271, 2019.

[107] 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.

[108] D. Williams, H. Jamjoom, Y.-H. Liu, and H. Weatherspoon, "Overdriver: Handling memory overload in an oversubscribed cloud," in *Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '11, p. 205–216, Association for Computing Machinery, 2011.

[109] B. Riad, Z. Abdelhafid, and M. Ramdane, "An autonomous architecture for managing vertical elasticity in the iaas cloud using memory over-subscription," in *Applied Computational Intelligence and Mathematical Methods: Computational Methods in Systems and Software 2017, vol. 2*, pp. 50–61, Springer, 2018.

[110] IBM, "z/vm – a brief review of its 40 year history," 2012. Available at http://www.vm.ibm.com/vm40hist.pdf.

[111] R. A. Meyer and L. H. Seawright, "A virtual machine time-sharing system," *IBM Systems Journal*, vol. 9, no. 3, pp. 199–218, 1970.

[112] U.S. Bureau of Economic Analysis, "Private fixed investment, chained price index: Nonresidential: Equipment: Information processing equipment: Computers and peripheral equipment," 2024. Available at https://fred.stlouisfed.org/series/B935RG3Q086SBEA.

[113] G. Amvrosiadis, J. W. Park, G. R. Ganger, G. A. Gibson, E. Baseman, and N. DeBardeleben, "On the diversity of cluster workloads and its impact on research results," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pp. 533–546, USENIX Association, July 2018.

[114] M. Tirmazi, A. Barker, N. Deng, M. E. Haque, Z. G. Qin, S. Hand, M. Harchol-Balter, and J. Wilkes, "Borg: the next generation," in *Proceedings of the fifteenth European conference on computer systems*, pp. 1–14, 2020.

[115] AWS, "The ec2 approach to preventing side-channels," 2024. Available at https://docs.aws.amazon.com/whitepapers/latest/security-design-of-aws-nitro-system/the-ec2-approach-to-preventing-side-channels.html.

[116] OVHcloud, "Discovery," 2024. Available at https://us.ovhcloud.com/public-cloud/discovery/.

[117] Azure, "B-series burstable virtual machine sizes," 2024. Available at https://learn.microsoft.com/en-us/azure/virtual-machines/sizes-b-series-burstable.

[118] AWS, "Burstable performance instances," 2024. Available at https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-performance-instances.html.

[119] Redhat OpenShift, "Overcommitting," 2024. Available at https://docs.openshift.com/container-platform/3.11/admin_guide/overcommit.html.

[120] J. Byrne, S. Svorobej, K. M. Giannoutakis, D. Tzovaras, P. J. Byrne, P.-O. Östberg, A. Gourinovitch, T. Lynn, *et al.*, "A review of cloud computing simulation platforms and related environments.," *CLOSER*, pp. 651–663, 2017.

[121] N. Mansouri, R. Ghafari, and B. M. H. Zade, "Cloud computing simulators: A comprehensive review," *Simulation Modelling Practice and Theory*, vol. 104, p. 102144, 2020.

[122] S. M. Ali and G. Kecskemeti, "Sequal: an unsupervised feature selection method for cloud workload traces," *The Journal of Supercomputing*, pp. 1–19, 2023.

[123] OVHcloud, "Who are we?," 2023. Available at https://www.ovhcloud.com/en/about-us/.

[124] A. Beloglazov, R. Buyya, Y. C. Lee, and A. Zomaya, *Chapter 3 - A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems*, vol. 82 of *Advances in Computers*. Elsevier, 2011.

[125] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[126] K. Venkatraman, "Virtual machine memory allocation and placement on azure stack," 2019. Available at https://azure.microsoft.com/en-us/blog/virtual-machine-memory-allocation-and-placement-on-azure-stack/.

[127] Linux Documentation, "Scheduler statistics," 2022. Available at https://docs.kernel.org/scheduler/sched-stats.html.

[128] J.-E. Dartois, A. Knefati, J. Boukhobza, and O. Barais, "Using Quantile Regression for Reclaiming Unused Cloud Resources while achieving SLA," in *CloudCom 2018 - 10th IEEE International Conference on Cloud Computing Technology and Science*, pp. 89–98, IEEE, Dec. 2018.

[129] T. chung Fu, "A review on time series data mining," *Engineering Applications of Artificial Intelligence*, vol. 24, no. 1, pp. 164–181, 2011.

[130] R. Agrawal, C. Faloutsos, and A. Swami, "Efficient similarity search in sequence databases," in *Foundations of Data Organization and Algorithms: 4th International Conference, FODO'93 Chicago, Illinois, USA, October 13–15, 1993 Proceedings 4*, pp. 69–84, Springer, 1993.

[131] C. L. Fancoua and J. C. Principe, "A neighborhood map of competing one step predictors for piecewise segmentation and identification of time series," in *Proceedings of International Conference on Neural Networks (ICNN'96)*, vol. 4, pp. 1906–1911, IEEE, 1996.

[132] K. Qazi, Y. Li, and A. Sohn, "Workload prediction of virtual machines for harnessing data center resources," in *2014 IEEE 7th International Conference on Cloud Computing*, pp. 522–529, 2014.

[133] P. Janus and K. Rzadca, "Slo-aware colocation of data center tasks based on instantaneous processor requirements," in *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, (New York, NY, USA), p. 256–268, Association for Computing Machinery, 2017.

[134] L. Nashold and R. Krishnan, "Using lstm and sarima models to forecast cluster cpu usage," 2020.

[135] J. Paparrizos, Y. Kang, P. Boniol, R. S. Tsay, T. Palpanas, and M. J. Franklin, "TSB-UAD: an end-to-end benchmark suite for univariate time-series anomaly detection," *Proc. VLDB Endow.*, vol. 15, no. 8, pp. 1697–1711, 2022.

[136] Prometheus, "Prometheus," 2022. Available at https://prometheus.io/.

[137] 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 cloud and iot microservices," 2019.

[138] L. A. Barroso, U. Hölzle, and P. Ranganathan, *The datacenter as a computer: Designing warehouse-scale machines*. Springer Nature, 2019.

[139] 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.

[140] 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.

[141] 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.

[142] 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.

[143] 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.

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

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

[146] 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*, EuroSys '21, p. 1–16, Association for Computing Machinery, 2021.

[147] 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.

[148] 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.

[149] 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.

[150] 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.

[151] 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.

[152] 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, Association for Computing Machinery, 2022.

[153] 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.

[154] H. Li, D. S. Berger, L. Hsu, D. Ernst, P. Zardoshti, S. Novakovic, M. Shah, S. Rajadnya, S. Lee, I. Agarwal, *et al.*, "Pond: Cxl-based memory pooling systems for cloud platforms," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 574–587, 2023.

[155] 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.

[156] 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.

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

[158] 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.

[159] 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.

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

[161] 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.

[162] 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.

[163] 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.

[164] Q. Weng, L. Yang, Y. Yu, W. Wang, X. Tang, G. Yang, and L. Zhang, "Beware of fragmentation: Scheduling GPU-Sharing workloads with fragmentation gradient descent," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, (Boston, MA), pp. 995–1008, USENIX Association, 2023.

[165] B. Wu, Z. Zhang, Z. Bai, X. Liu, and X. Jin, "Transparent GPU sharing in container clouds for deep learning workloads," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, (Boston, MA), pp. 69–85, USENIX Association, Apr. 2023.

[166] E. Cadorel and R. Rouvoy, "Enabling dynamic virtual frequency scaling for virtual machines in the cloud," in *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 336–346, 2022.