ScroogeVM: Boosting Cloud Resource Utilization with Dynamic Oversubscription

Pierre Jacquet, Thomas Ledoux, Romain Rouvoy

SCROOGEVM: Boosting Cloud Resource Utilization with Dynamic Oversubscription

Pierre Jacquet,* ‡ Thomas Ledoux,† * Romain Rouvoy‡ *
* Inria, France
† IMT Atlantique, LS2N, UMR CNRS 6004, France
‡ Univ. Lille, CRIStAL, UMR CNRS 9189, France

***Abstract*—Despite continuous improvements, cloud physical resources remain underused, hence severely impacting the efficiency of these infrastructures at large. To overcome this inefficiency, *Infrastructure-as-a-Service* (IaaS) providers usually compensate for oversized *Virtual Machine*s (VMs) by offering more virtual resources than are physically available on a host. However, this technique—known as *oversubscription*—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 article studies how a greedy increase of the oversubscription ratio per *Physical Machine* (PM) and resources type can preserve performance goals. Keeping performance unchanged allows our 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, we investigate the impact of resource oversubscription on performance degradation. Subsequently, we conduct a comparative analysis of SCROOGEVM with state-of-the-art oversubscription computations. The results demonstrate that our approach outperforms existing methods by leveraging the presence of long-lasting VMs, while avoiding live migration penalties and performance impacts for stakeholders.

***Index Terms*—Cloud, IaaS, oversubscription.**

I. INTRODUCTION

Initially, virtualization technologies paved the way for building more energy-efficient cloud infrastructures by increasing the physical resources usage [11]. Unfortunately, most cloud operators keep observing that the VMs they host in their data centers are underused [19]. These observation 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 to provision 32 GB of memory to host a hypothetical workload of 18 GB.

Various cloud providers report on relatively low usage of their infrastructure, even over the past decade [7], [19], [21]. We believe that optimizing this usage is one of the most promising leads to reducing the energy consumption and carbon footprint of data centers. While CPU resources are often effectively allocated, as reported by [23] indicating that 80% of Azure servers have less than 15% unprovisioned cores, there is still substantial underutilization of CPU on PMs [21]. This underutilization highlights the significance of provisioned, yet unused, resources. We chose to address it through the prism of resource *oversubscription*, also known as *overcommitment* or *overbooking*, which is defined as the amount of virtual resources made available for each unit of physical resource [26]. Resource *oversubscription* allows cloud providers to boost physical resource utilization and is widely adopted in production, as reported in the literature [9], [19].

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 [18].

Instead of statistically determining an optimal global value, we argue that clusters can better benefit from a *dynamic* oversubscription ratio per PM. In particular, we believe that higher gains at lower risk can be achieved by learning from the deployed workloads and resource utilization of individual PM, instead of reasoning at the scale of a whole cluster. In this article, we 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 our 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 usages. 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 VMs requirements and can be used to balance the IaaS allocations and performances across all PM in a cluster.

In the remainder of this article, we start by discussing the state of the art in the area of cloud resource usage and oversubscription (cf. Section II). Then, we introduce the principles of greedy oversubscription and our implementation, named SCROOGEVM (cf. Section III). We also perform an empirical analysis of the impact of different parameters of our approach in an environment that mimics the characteristics of Microsoft Azure (cf. Section IV). Additionally, we evaluate SCROOGEVM oversubscription computation and compare it to other dynamic approaches (cf. Section V). Finally, we conclude and share the perspectives for this work in Section VI.

II. RELATED WORK

A. Improve Cloud Resources Usage

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 [8]. Therefore, even if energy proportionality kept improving over the last decade [43], [53], resource under-utilization inevitably imposes higher power consumption and hardware costs [19]. In this context, the utilization ratio of a cloud platform can be modeled as a workload consolidation problem for PM. This problem can be addressed statically (VM placement) and dynamically (VM live migration) through different scheduling strategies [2]. Dynamic placement, or consolidation, has been widely studied in the literature [27] to deliver significant gains based on the behavior of VM, but VM migrations remain costly and should be limited in a cloud infrastructure. Our approach only leverages the initial placement decision, hence minimizing its impact on cloud infrastructures. Unlike traditional bin-packing problems, VM usage window is dynamic, even if they are bounded by their inner configuration limits. We, therefore, aim at better estimating overall PM usage and providing more insightful feedback to the cloud provider.

More recently, new types of VM have been specifically promoted by cloud providers to increase their cloud utilization ratio. For example, unallocated resources can be used to deploy a SPOTVM [5], [16], [42], whose deployment is preempted when the allocated resources are requested by a traditional VM. This VM type is usually proposed at a lower price. The HARVESTVM [3] extends this concept and allows the size of a SPOTVM to be dynamically adjusted according to the resources available on the PM. The harvesting of unallocated—or unused—resources is performed in addition to CPU oversubscription on at least some cloud offers [6], [41]. Service providers do not usually communicate their oversubscription ratio, potentially considering this information as sensitive given its potential impact on performance. Our solution, instead, pays particular attention to performance consistency.

B. Hypervisor Oversubscription

Most hypervisors enable oversubscription by allowing the sum of all allocated virtual resources to exceed the PM capabilities [10], [45], [48]. Oversubscription relies on the fact that a VM resources usage is usually lower than its allocation. It may be explained by a low resource-demanding workload, but also by resource optimization implemented by the host. For example, hypervisors may implement *Kernel Samepage Merging* (KSM), ballooning, memory compression, or swap mechanisms to reduce VM usage from the host perspective. Some hypervisors, such as VSPHERE [49], natively support these features and may, therefore, oversubscribe their memory to higher levels.

An overload situation occurs when a VM requires more resources than available from the PM. This may happen due to an undersized configuration, or due to an excessive oversubscription ratio. With an oversized CPU oversubscription ratio, VMs will not be able to use their allocated time slices, which may lead to *Service-Level Agreement* (SLA) violation. This highly impacts smaller VMs as usual hypervisors, such as KVM and VSPHERE, allocate an equal share of time-slices per *virtual CPU* (vCPU). When memory overload occurs, VMs will swap memory pages leading to highly degraded performance [36]. To avoid such critical situations, some cluster managers, such as OPENSTACK [39] and BORG [4], limit the oversubscription ratio to a common static value for each PM.

[17] aims to estimate the optimal cluster-scale CPU oversubscription for a given workload trace, according to an accepted risk, by modeling the problem as a bin packing with chance constraints. Unfortunately, they did not support memory. Furthermore, we argue that the oversubscription ratio is highly dependent on the workload and the PM configuration, which is highly heterogeneous and unforeseeable in the context of a IaaS platform. Typically, we believe that more gains could potentially be identified by computing a PM-scale resource oversubscription ratios, covering memory and CPU.

[19] computes a PM-scale CPU oversubscription ratio by considering the sum of VMs in a given percentile. However, this technique limits the potential gain, as it is unlikely that all peak VMs usages will occur at the same time [9]. Furthermore, they did not support memory oversubscription.

CLOUDVAMP [52] targets dynamic memory oversubscription by leveraging the amount of memory retrieved from ballooning. In our approach, we compute an oversubscription ratio without *a priori* knowledge of the platform optimization mechanisms, which allows us to propose a more generic approach that can work with ballooning, but also other types of memory optimization proposed by the literature. The *Dynamic Oversubscription ratio Adjustment* (DOA) is introduced in [51] to mitigate cascading failures. Due to its purpose, it is highly reactive and only takes into account the last resource usage of a given PM. We aim to provide a more stable and conservative ratio leveraging resource usage histories.

Finally, N-SIGMA [9] derives the CPU oversubscription from a prediction of the CPU usage peak, computed as $\overline{cpu}+N\times\sigma$, where $\overline{cpu}$ and σ capture the average and standard

deviation of CPU usage, respectively. The study, which draws upon Google Borg traces, recommends a $N = 5$ configuration. While this value effectively covers the worst-case scenarios observed in Borg, it also results in underutilized resources in the majority of cases. In light of this limitation, we propose to adjust any resource oversubscription by considering the quiescent state of the PM, especially in long-running IaaS workloads.

C. Memory Optimisation

In some cloud infrastructures, memory is reported as the system resource that limits the deployment of new VMs [25], due to the memory capacity wall [30]. However, it is more challenging to oversubscribe memory due to its inherent harvesting limits, compared to the CPU. For a given VM, its host memory usage is defined by *Resident Set Size* (RSS), the amount of physical memory allocated, in opposition to, for example, swap space. However, actively used pages are usually lower and, therefore, RSS can potentially be reduced close to the *Working Set Size* (WSS), the minimal amount of memory required [38]. WSS estimation may rely on different techniques, such as [50], 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 [14]. Different techniques also attempt to estimate the optimal WSS when the VM is under-provisioned, using a dedicated cache [29], [35]. WSS estimation may also benefit from new hardware features to reduce its performance overhead [12]. 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 [32]. While dynamic memory management reduces (ballooning, hotplug) or increases (hotplug) the RSS of VM, other mechanisms can also impact memory consumption. For example, KSM allows VMs to share common pages [31] 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 [34] 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 while keeping good I/O performance.

We argue that the proposed resources oversubscription ratios should be generic, and therefore independent from platform-specific optimizations. What limits the deployment of new VMs is the PM resources usage and thus forms the basis of our work. This usage can be reduced due to optimization or limitation on system resources, which gives us an indirect, but sufficient, view of the platform optimizations.

III. GREEDY OVERSUBSCRIPTION WITH SCROOGEVM

We introduce *greedy oversubscription* as a novel approach to implement a dynamic resource oversubscription strategy at the scale of individual PMs. More concretely, our 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, we 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.

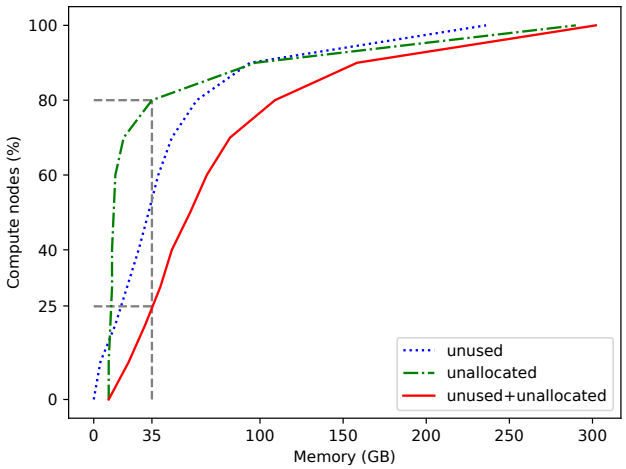

Fig. 1. CDF of memory availability in a IaaS infrastructure.

A. Principles of Greedy Oversubscription

a) Preliminary analysis of cloud resource availability: In 2017, Microsoft Azure released traces from a 3-month IaaS workload [19]. Interestingly, this dataset—reflecting a production-scale cloud infrastructure—reveals several insights that we leverage as part of this article:

- **Not all VMs resource requirements are the same**: VM configuration distribution—coined *size* in the article—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 VMs lifespan are equal**: Long-running VMs account for more than 95 % of the total core hours;
- **Not all VMs provisioned resources are used**: While resource usage figures are only provided for the CPU, a collaboration with OVHCLOUD[1] reveals that a production-scale IaaS infrastructure with guaranteed resources—*i.e.*, no oversubscription—succeeds to allocate most of the memory of PMs, with 80 % of their PMs reporting less than 35 GB of unallocated memory (cf. Figure 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 2 reports that 95 % of the PMs of OVHCLOUD infrastructure report on a variation of at most 6 GB over a 24-hour window.

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

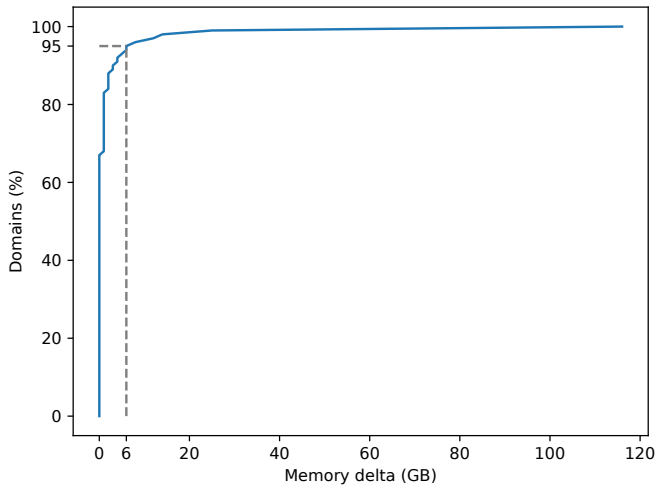

Fig. 2. CDF of memory allocation variations in a IaaS infrastructure.

b) 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:

$$\dot{\text{CPU}}_{IaaS} = \frac{targeted(\text{vCPUs})}{\sum_{p \in PM} \text{cores}(p)} \quad (1)$$

$$\dot{\text{RAM}}_{IaaS} = \frac{targeted(\text{vRAM})}{\sum_{p \in PM} \text{RAM}(p)} \quad (2)$$

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 a IaaS, these oversubscription ratios rely on a high-level analysis of workload patterns and an accepted risk in terms of performance degradation. However, a 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, we believe that the oversubscription ratios should rather be estimated in Rreal-time and at the scale of individual PM to minimize resource over-utilization while maximizing VMs performance.

c) 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, we studied potential oversubscription limitations.

We applied *Operations Research* (OR) techniques to maximize the number of VMs for a given PM, while respecting VMs size distribution reported by Azure. We 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 [47]. Moreover, the sum of VMs percentiles is known to overestimate actual requirements [9].

d) Capturing the effective resource utilization of hosted VMs: VMs hosted by a 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) [33]. 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, we 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. We 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.

e) Dynamic resource utilization profiling of hosted VMs: 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 [19], [20]. 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. We 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.

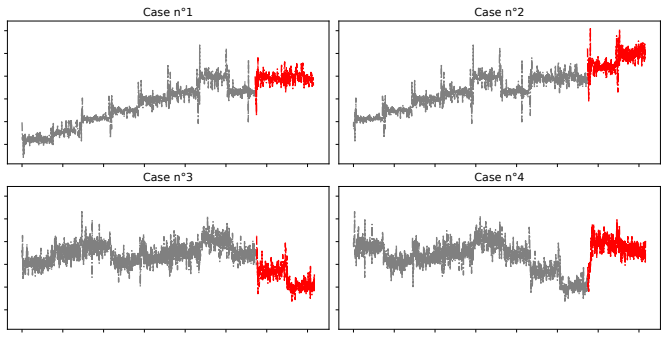

Fig. 3. Resource usage traces. Reference data is in grey, data seen as new in red

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? Our solution proposes to mitigate this issue with a *quiescent indicator*. This indicator answers our two previous questions by ensuring confidence in metrics computed on a quiescent PM and by properly reacting to a PM with an evolving workload.

f) 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, we consider that the PM resource becomes unstable and leaves 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 [15], where similarity measures are utilized to identify specific sequences in a series [1], [22]. However, there are some major distinctions in our notion of quiescence. Particularly, the observation of a pattern not previously encountered is common, as VMs-based workloads often exhibit a chaotic behavior [46] 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, we 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, we evaluate the PM quiescent state by comparing the current window samples to previous ones. To better understand the evaluation, we illustrated four resource usage traces from our workloads in Figure 3. We 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.

We now describe the stability detection heuristics we considered for this component of SCROOGEVM. As PM usage is usually compromised in a Gaussian distribution [28], 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 [19] and PM workloads [21] 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. We use a null-hypothesis significance test to compute the probability of obtaining the test result and reject it if it is below 5%.

Finally, our introduced quiescent classifier leverages a machine learning approach, using a *Long Short-Term Memory* (LSTM) model, known for its performance in predicting computing resource usage [20], [37]. As described in Algorithm 1, this classifier trains a model $\mathcal{M}_{LSTM}$ 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 $\mathcal{M}_{LSTM}$ 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, we 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 of 2.56 cores between both projection average errors. In our experiments, the 1% threshold was sufficient on large PMs.

Algorithm 1 Quiescent state detection algorithm

Input Historical dataset, last window
Output Quiescent state

$\mathcal{M}_{LSTM} \leftarrow$ Generate model from historical dataset
$forecasted \leftarrow predict(\mathcal{M}_{LSTM}, historical\_set)$
$predicted \leftarrow predict(\mathcal{M}_{LSTM}, last\_window)$
$\delta \leftarrow |RMSE(forecasted) - RMSE(predicted)|$
if $\delta > threshold$ **then**
    return `UNSTABLE`
end if
return `QUIESCENT`

LSTM training phase can be tuned by so-called hyperparameters. We 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 we do not use LSTM to predict future behaviors but to detect the occurrence of unforeseen behaviors. More specifically, we 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.

TABLE I
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

TABLE II
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**

We assess the LSTM classifier by comparing it to the other classification techniques: average, percentile and p-value based. Table I summarizes the labels returned by each classifier when evaluating the red window of Figure 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 a LSTM based approach appears to succeed in accurately detecting both unstable states.

To further assess these quiescent state classifiers, we consider the state-of-the-art metrics adopted for the evaluation of classifiers [40]. We labeled a sequence of windows covering the CPU traces of a 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* (TP), *false positives* (FP), *true negatives* (TN), and *false negatives* (FN), we 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 II, LSTM outperforms other classifiers with a F-score of 0.88. In the following sections, we therefore adopt it as the quiescent state detector of SCROOGEVM.

g) 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, our 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 [9] 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 $\overline{cpu}$ and σ. 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 article 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 in our tests 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:

$$\dot{\text{CPU}}_{PM} = \frac{provisioned(\text{vCPUs}) + \lfloor available(\text{cores}) \rfloor}{\sum \text{cores(PM)}} \quad (3)$$

$$\dot{\text{RAM}}_{PM} = \frac{provisioned(\text{vRAM}) + \lfloor available(\text{RAM}) \rfloor}{\sum \text{RAM(PM)}} \quad (4)$$

where $provisioned(\text{vRES})$ captures the amount of currently provisioned virtual resources and $\lfloor available(\text{RES}) \rfloor$ the identified unused resources.[2]

Dynamic oversubscription can leverage existing cloud platforms and monitoring infrastructure to be easily implemented. We assume that proposing a new VM orchestrator is out of the scope of this article, and we rather focus our contribution 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, we believe that

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

our 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.

B. Implementation of SCROOGEVM

Our 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 our 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 recognize 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 [44] and updated periodically. Aggregation, storage, and analysis steps can be performed on a dedicated node to 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 illustrates the integration of SCROOGEVM in a IaaS platform, like OPENSTACK, to guide the deployment of new VMs towards 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 resources 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 1.

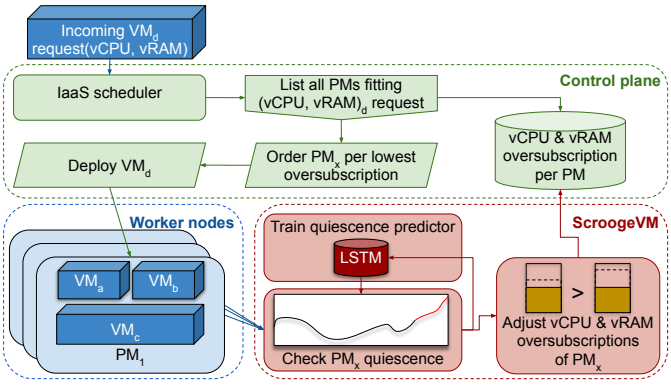

Fig. 4. Overview of the integration of SCROOGEVM in a IaaS platform to guide the deployment of new VMs

IV. EMPIRICAL ANALYSIS

Cloud simulators, such as [9] and [13], lack of 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. Our protocol is built upon the IaaS workload characteristics of Microsoft Azure [19] to consider a realistic cloud infrastructure.

A. Experimental Settings & Evaluation Protocol

We evaluated various ratios of oversubscription with different workload intensities by gradually increasing the number of VMs, while monitoring different metrics. In our experiments, CPU resources did not enable any specific optimization mechanisms. Memory resources implemented a ballooning mechanism that periodically reduces VM configurations to their observed max peak memory usage for a few seconds before restoring it. In our experiments, this principle was sufficient to reduce VM RSS without any noticeable impact on their performance.

a) Input workload: We developed a realistic workload generator to inject a VM workload according to the IaaS statistics reported in the Microsoft Azure dataset [19]. The scripts obtained from this generator 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. Our generator adapts VMs size distribution to a specific PM configuration by providing a list of required VMs and workload scripts.

The workload intensity of each VM is generated from the CPU utilization characteristics of the Microsoft Azure dataset. To avoid unrealistic steady load caused by usual benchmarks, we periodically reload them with different parameters. Benchmark parameters used are computed based on a targeted CPU intensity. This CPU intensity changed through time by randomly selecting values from a Gaussian distribution around the targeted CPU intensity. While we cannot ensure that a selected benchmark parameter value will exactly reach the targeted CPU intensity, we roughly ensure that the order of magnitude is reached based on empirical analysis.

According to Azure, 30% of long-lasting VMs can be estimated as having a diurnal usage pattern, we also take it into account using the same proportion by reproducing usage patterns through time. The effective duration of an "hour", called in our context *slice*, and a "day", called a *scope*, can be modified to simplify the experiments. Specified arrival rate and VMs lifetime characteristics are reproduced in our scripts based on these settings.

b) Applications under study: Our benchmark workload relies on heterogeneous applications to emulate the diversity of situations covered in a cloud infrastructure. Concretely, we deploy applications from the DeathStarBench as a representative microservices architectures [24], TPC-C for

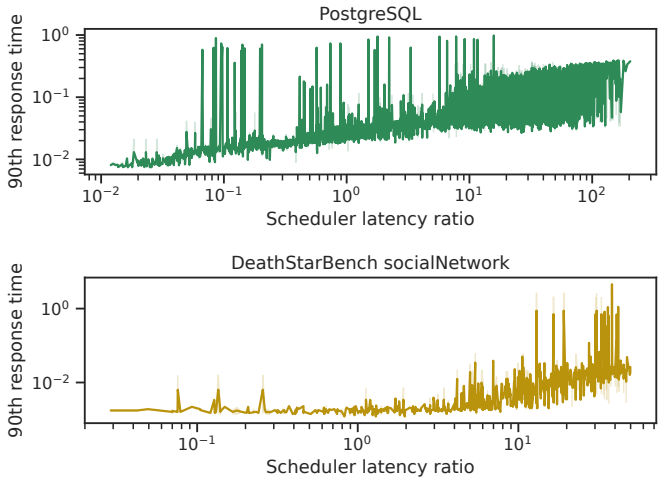

Fig. 5. Scheduler latency impact on VM performance (log-scale axes)

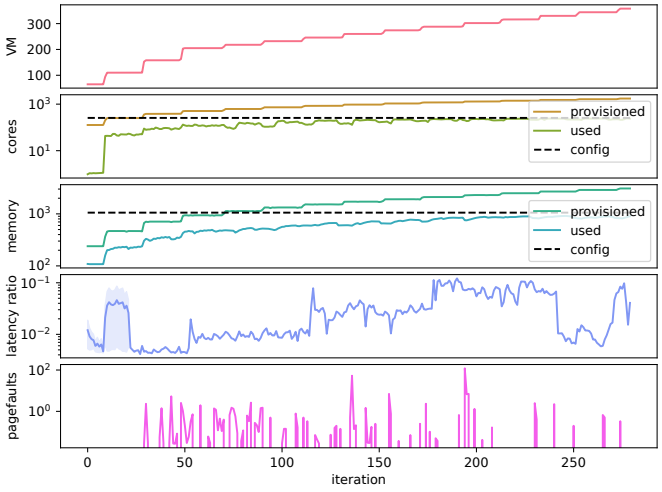

Fig. 6. Overview of a profile of our collected metrics

databases workloads, a WordPress application, batch using CPU-intensive workloads, and idle VM. When applicable, workload generation uses related benchmarking tools, which can be executed remotely or locally.

c) Metrics of interest: In addition to the system metrics, we also monitored the performances obtained from applications hosted by the different VMs for validation purposes. Figure 5 depicts the evolution of hosted application performance through their 90^{th} percentile response times. A first set of VM 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 VM hosted on a PM are aggregated in the graph. From the PM point of view, scheduler metrics were retrieved and a ratio was computed based on the scheduler latency and processes runtime. Due to heterogeneous benchmarks, performance degradations do not follow the same pattern, but one may observe that the lower the scheduler latency, the better application performance. We 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 VMs application performance from the host perspective in a black-box environment, like a IaaS platform. Impact on batch performance was not particularly investigated as we expected it to be similarly affected by an overall platform performance degradation.

d) Hardware settings.: In our experiments, we used the PM described in Table III.

TABLE III
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 6 summarizes the profile of an experiment executed from our workload generator. An Azure-like workload was incrementally injected while maintaining the VMs distribution and activity percentage. 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.

B. 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 our experiments, the sampling period did not have any significant impact on its oversubscription ratio.

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 article, we used a 5s aggregation window.

C. Impact on the VM Performances

In the context of a 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, we used the resource overload signals introduced in Section III-A. Technically, system-wide scheduler latency

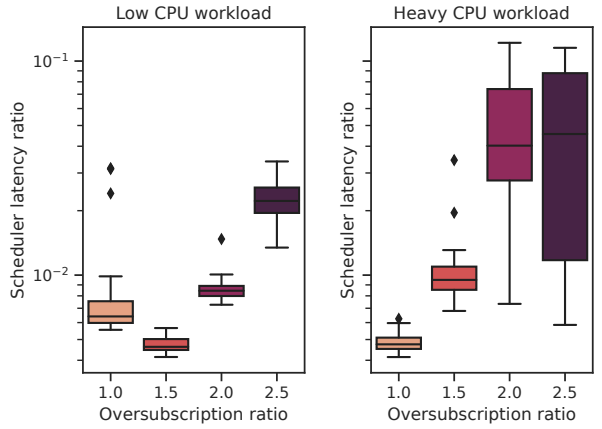

Fig. 7. CPU performance comparison for 2 workloads (log-scale Y axis)

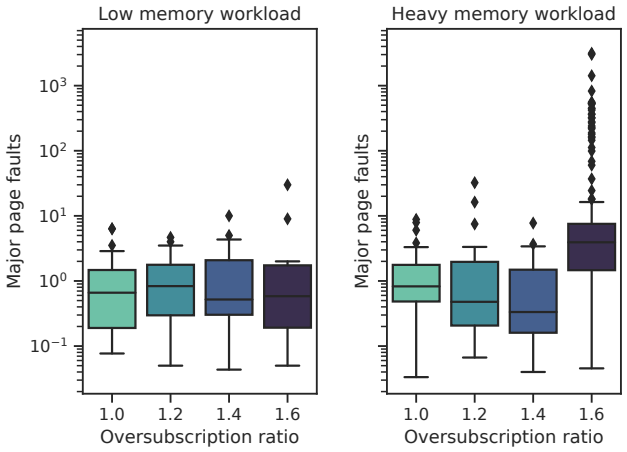

Fig. 8. Memory performance comparison for 2 workloads (log-scale Y axis)

is exposed by the Linux kernel. When the operating system is under-loaded, tasks assigned to the cores do not significantly affect the latency. We 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 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 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). In our experiments, this threshold was unnoticeable from the VMs perspective.

When it comes to memory, page faults evolve differently in an oversubscribed scenario. In our context, 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, we considered light and heavy memory workloads in Figure 8. Starting from $\dot{\text{RAM}} = 1{:}1$ (according to Microsoft Azure distribution), we 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 VMs 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 our 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 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.

V. Validation

We 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 [51]. 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 [9]. The third method is Borg-default like (borg), which mimics a static oversubscription mechanism where deployed VMs are estimated to let 10% of their resources unused. This percentage, corresponding to a 1.1:1 oversubscription, is based on the empirical study from [9]. 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 [9].

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

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. We label such a situation as a *misprediction* and we 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 to hosted VM. This situation is referred to as a *violation*. Our objective is to avoid such violations. In addition to the two prediction accuracy metrics, we also consider a third metric that focuses on the evolution of predictions. Specifically, we 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 9 depicts the performances of each strategy on a 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 the other hand, nsigma and rclike methods demonstrate similar predictions. It is noteworthy that the SCROOGEVM approach consistently exhibits the lowest misprediction.

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

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 VMs 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

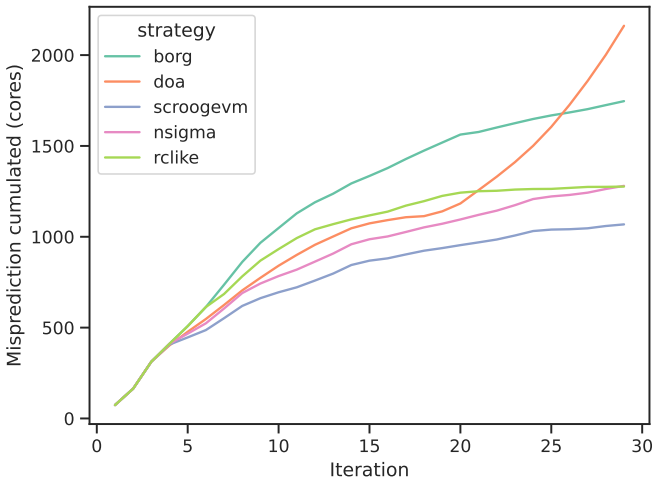

Fig. 9. Cumulated mispredictions (cores) under decreasing CPU workload

TABLE IV
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

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 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 IV. 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

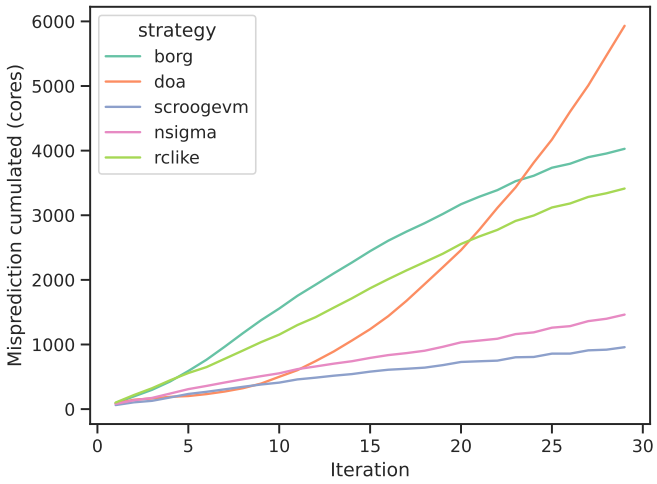

Fig. 10. Cumulated mispredictions (cores) under increasing CPU workload

TABLE V
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

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.

We 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, our overhead primarily arises from the LSTM model training, taking approximately 1 second in our tests on a server without a discrete GPU unit. This level of overhead suggests practical viability for production contexts.

VI. CONCLUSION

In this article, we propose a novel approach, named SCROOGEVM,[3] to implement a greedy resource oversubscription strategy at the scale of individual PMs. Instead of configuring a static and cluster-wide ratio, we propose to dynamically estimate the optimal oversubscription ratio per PM, while maintaining performance goals. This approach, decoupled from the resource optimization of a given IaaS platform, allows SCROOGEVM to be generic and deployed with most cloud schedulers operating a IaaS. We evaluated SCROOGEVM with an IaaS workload from Microsoft Azure and reported on potential gains exceeding state-of-the-art strategies.

Short-term perspectives on this work include the consideration of more diverse datasets to recommend to public and private cloud operators parameters according to their workloads. Exploring various performance objectives aligned with the diverse premium VM policies offered by Cloud providers is a potential future direction. We also aim to improve oversubscription assessment with complementary VMs profiling operation. The inclusion of SCROOGEVM in a cloud stack, like OPENSTACK, is also of particular interest to demonstrate the benefits of greedy oversubscription for a cloud orchestrator.

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

ACKNOWLEDGMENTS

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

REFERENCES

[1] Rakesh Agrawal, Christos Faloutsos, and Arun 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*, pages 69–84. Springer, 1993.

[2] Raja Wasim Ahmad, Abdullah Gani, Siti Hafizah Ab. Hamid, Muhammad Shiraz, Abdullah Yousafzai, and Feng Xia. A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *Journal of Network and Computer Applications*, 52:11–25, 2015. doi:10.1016/j.jnca.2015.02.002.

[3] Pradeep Ambati, Inigo Goiri, Felipe Frujeri, Alper Gun, Ke Wang, Brian Dolan, Brian Corell, Sekhar Pasupuleti, Thomas Moscibroda, Sameh Elnikety, Marcus Fontoura, and Ricardo Bianchini. Providing SLOs for Resource-Harvesting VMs in cloud platforms. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 735–751. USENIX Association, November 2020.

[4] George Amvrosiadis, Jun Woo Park, Gregory R. Ganger, Garth A. Gibson, Elisabeth Baseman, and Nathan DeBardeleben. On the diversity of cluster workloads and its impact on research results. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 533–546. USENIX Association, July 2018.

[5] Microsoft Azure. Azure spot virtual machines, 2020. URL: https://azure.microsoft.com/en-us/pricing/spot.

[6] Microsoft Azure. Virtual machine memory allocation and placement on azure stack, 2022.

[7] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007. doi:10.1109/MC.2007.443.

[8] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *IEEE Computer*, 40, 2007.

[9] Noman Bashir, Nan Deng, Krzysztof Rzadca, David Irwin, Sree Kodak, and Rohit 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, page 556–573. Association for Computing Machinery, 2021. doi:10.1145/3447786.3456259.

[10] CPU Virtualization Basics. overcommiting pcpus on individual xenserver vms, 2018. URL: https://support.citrix.com/article/CTX236977/overcommiting-pcpus-on-individual-xenserver-vms.

[11] Anton Beloglazov, Rajkumar Buyya, Young Choon Lee, and Albert Zomaya. *Chapter 3 - A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems*, volume 82 of *Advances in Computers*. Elsevier, 2011. doi:10.1016/B978-0-12-385512-1.00003-7.

[12] Stella Bitchebe, Djob Mvondo, Laurent Réveillère, Noël de Palma, and Alain Tchana. Extending intel pml for hardware-assisted working set size estimation of vms. In *Proceedings of the 17th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE 2021, page 111–124. Association for Computing Machinery, 2021. doi:10.1145/3453933.3454018.

[13] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exper.*, 41(1):23–50, jan 2011. doi:10.1002/spe.995.

[14] Jui-Hao Chiang, Han-Lin Li, and Tzi cker Chiueh. Working set-based physical memory ballooning. In *10th International Conference on Autonomic Computing (ICAC 13)*, pages 95–99. USENIX Association, June 2013.

[15] Tak chung Fu. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181, 2011. URL: https://www.sciencedirect.com/science/article/pii/S0952197610001727, doi:https://doi.org/10.1016/j.engappai.2010.09.007.

[16] Amazon Elastic Compute Cloud. Amazon ec2 spot instances, 2022. URL: https://aws.amazon.com/ec2/spot/.

[17] Maxime C. Cohen, Philipp W. Keller, Vahab Mirrokni, and Morteza Zadimoghaddam. Overcommitment in cloud services – bin packing with chance constraints. arXiv, 2017. doi:10.48550/ARXIV.1705.09335.

[18] Maxime C. Cohen, Philipp W. Keller, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Overcommitment in cloud services - bin packing with chance constraints. *CoRR*, abs/1705.09335, 2017. URL: http://arxiv.org/abs/1705.09335, arXiv:1705.09335.

[19] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 153–167. Association for Computing Machinery, 2017. doi:10.1145/3132747.3132772.

[20] Jean-Emile Dartois, Anas Knefati, Jalil Boukhobza, and Olivier 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*, pages 89–98. IEEE, December 2018. doi:10.1109/CloudCom2018.2018.00030.

[21] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-efficient and qos-aware cluster management. *SIGPLAN Not.*, 49(4):127–144, feb 2014. doi:10.1145/2644865.2541941.

[22] Craig L Fancoua and Jose 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)*, volume 4, pages 1906–1911. IEEE, 1996.

[23] Alexander Fuerst, Stanko Novaković, Íñigo Goiri, Gohar Irfan Chaudhry, Prateek Sharma, Kapil Arya, Kevin Broas, Eugene Bak, Mehmet Iyigun, and Ricardo Bianchini. Memory-harvesting vms in cloud platforms. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 583–594, 2022.

[24] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Yuan He, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Rick Lin, Zhongling Liu, Jake Padilla, and Christina Delimitrou. An open-source benchmark suite for cloud and iot microservices, 2019. doi:10.48550/ARXIV.1905.11055.

[25] Jing Guo, Zihao Chang, Sa Wang, Haiyang Ding, Yihui Feng, Liang Mao, and Yungang Bao. Who limits the resource efficiency of my datacenter: An analysis of alibaba datacenter traces. In *Proceedings of the International Symposium on Quality of Service*, pages 1–10, 2019.

[26] Rachel Householder, Scott Arnold, and Robert Green. On cloud-based oversubscription. *International Journal of Engineering Trends and Technology*, 8(8):425–431, feb 2014. doi:10.14445/22315381/ijett-v8p273.

[27] Muhammad Imran, Muhammad Ibrahim, Muhammad Salah Ud Din, Muhammad Atif Ur Rehman, and Byung Seo Kim. Live virtual machine migration: A survey, research challenges, and future directions. *Computers and Electrical Engineering*, 103:108297, 2022. doi:10.1016/j.compeleceng.2022.108297.

[28] Pawel Janus and Krzysztof Rzadca. Slo-aware colocation of data center tasks based on instantaneous processor requirements. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, page 256–268, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3127479.3132244.

[29] Stephen T Jones, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. Geiger: monitoring the buffer cache in a virtual machine environment. *ACM Sigplan Notices*, 41(11):14–24, 2006.

[30] Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K. Reinhardt, and Thomas F. Wenisch. Disaggregated memory for expansion and sharing in blade servers. *SIGARCH Comput. Archit. News*, 37(3):267–278, jun 2009. doi:10.1145/1555815.1555789.

[31] Linux. Kernel samepage merging, 2022. URL: https://www.kernel.org/doc/html/latest/admin-guide/mm/ksm.html.

[32] Linux. Memory hot(un)plug, 2022. URL: https://www.kernel.org/doc/html/latest/admin-guide/mm/memory-hotplug.html.

[33] Linux. Scheduler statistics, 2022. URL: https://docs.kernel.org/scheduler/sched-stats.html.

[34] Linux. zram, 2022. URL: https://docs.kernel.org/admin-guide/blockdev/zram.html.

[35] Pin Lu and Kai Shen. Virtual machine memory access tracing with hypervisor exclusive cache. In *Usenix Annual Technical Conference*, pages 29–43, 2007.

[36] Germán Moltó, Miguel Caballer, and Carlos de Alfonso. Automatic memory-based vertical elasticity and oversubscription on cloud platforms. *Future Generation Computer Systems*, 56:1–10, 2016. doi:https://doi.org/10.1016/j.future.2015.10.002.

[37] Langston Nashold and Rayan Krishnan. Using lstm and sarima models to forecast cluster cpu usage, 2020. doi:10.48550/ARXIV.2007.08092.

[38] Vlad Nitu, Aram Kocharyan, Hannas Yaya, Alain Tchana, Daniel Hagimont, and Hrachya Astsatryan. Working set size estimation techniques in virtualized environments: One size does not fit all. *ACM SIGMETRICS Performance Evaluation Review*, 46:62–63, 06 2018. doi:10.1145/3292040.3219642.

[39] OpenStack. overcommiting cpu and ram, 2022. URL: https://docs.openstack.org/arch-design/design-compute/design-compute-overcommit.html.

[40] John Paparrizos, Yuhao Kang, Paul Boniol, Ruey S. Tsay, Themis Palpanas, and Michael J. Franklin. TSB-UAD: an end-to-end benchmark suite for univariate time-series anomaly detection. *Proc. VLDB Endow.*, 15(8):1697–1711, 2022.

[41] Google Cloud Platform. over-committing cpus sole tenant vms, 2020. URL: https://cloud.google.com/compute/docs/nodes/overcommiting-cpus-sole-tenant-vms.

[42] Google Cloud Platform. Preemptible vm instances, 2020. URL: https://cloud.google.com/compute/docs/instances/preemptible.

[43] George Prekas, Mia Primorac, Adam Belay, Christos Kozyrakis, and Edouard Bugnion. Energy proportionality and workload consolidation for latency-critical applications. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, page 342–355. Association for Computing Machinery, 2015. doi:10.1145/2806777.2806848.

[44] Prometheus. Prometheus, 2022. URL: https://prometheus.io/.

[45] Proxmox. Proxmox ve administration guide, 2022. URL: https://pve.proxmox.com/pve-docs/pve-admin-guide.pdf.

[46] Kashifuddin Qazi, Yang Li, and Andrew Sohn. Workload prediction of virtual machines for harnessing data center resources. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 522–529, 2014. doi:10.1109/CLOUD.2014.76.

[47] Kirtana Venkatraman. Virtual machine memory allocation and placement on azure stack, 2019. URL: https://azure.microsoft.com/en-us/blog/virtual-machine-memory-allocation-and-placement-on-azure-stack/.

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

[49] vmware. Memory overcommitment, 2019. URL: https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.resmgmt.doc/GUID-895D25BA-3929-495A-825B-D2A468741682.html.

[50] Carl A. Waldspurger. Memory resource management in vmware esx server. *SIGOPS Oper. Syst. Rev.*, 36(SI):181–194, dec 2003. doi:10.1145/844128.844146.

[51] Haoyu Wang, Haiying Shen, and Zhuozhao Li. Approaches for resilience against cascading failures in cloud datacenters. In *2018 IEEE 38th*

International Conference on Distributed Computing Systems (ICDCS), pages 706–717, 2018. doi:10.1109/ICDCS.2018.00074.

[52] Dan Williams, Hani Jamjoom, Yew-Huey Liu, and Hakim 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, page 205–216. Association for Computing Machinery, 2011. doi:10.1145/1952682.1952709.

[53] Daniel Wong and Murali Annavaram. Knightshift: Scaling the energy proportionality wall through server-level heterogeneity. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 119–130, 2012. doi:10.1109/MICRO.2012.20.

Pierre Jacquet is a PhD candidate at Inria and the University of Lille. He received a Master's degree in computer science and engineering from Paris XII in 2021. He is a member of the Spirals and Stack teams and his research interests include resource scheduling and sustainable computing.

Thomas Ledoux is a Full Professor at IMT Atlantique and a member of the Stack team (a joint team with Inria and LS2N). He works in the field of software architectures for distributed systems and is particularly interested in sustainable computing.

Romain Rouvoy is a Full Professor at the University of Lille and a member of the Spirals team (a joint team with Inria and CRIStAL). His research lies at the intersection of software engineering and distributed systems, addressing issues, such as sustainable software development and user privacy concerns.