# Virtual machine experience design : a predictive resource allocation approach for cloud infrastructures

Loïc Pérennou

le cnam

Ecole Doctorale Informatique, Télécommunications et électronique

Centre d'Études et de Recherche en Informatique et Communications

# THÈSE DE DOCTORAT

*présentée par* : **Loïc PÉRENNOU**

*soutenue le* : **23 octobre 2019**

*pour obtenir le grade de* : **Docteur du Conservatoire National des Arts et Métiers**

*Discipline* : **Informatique**

# Virtual Machine Experience Design : A Predictive Resource Allocation Approach for Cloud Infrastructures

**THÈSE dirigée par**

M. Gressier-Soudan Éric — *Professeur, CNAM*

**et co-encadrée par**

M. Lefebvre Sylvain — *Enseignant-Chercheur, ISEP*

**RAPPORTEURS**

M. Hagimont Daniel — *Professeur, IRIT/ENSEEIHT*

M. Lebre Adrien — *Professeur, IMT Atlantique*

**PRÉSIDENT DU JURY**

M. Hagimont Daniel — *Professeur, IRIT/ENSEEIHT*

**EXAMINATEURS**

Mme. Bouchenak Sara — *Professeure, INSA Lyon*

M. Gaaloul Walid — *Professeur, TSP*

**MEMBRES INVITÉS**

Mme. Chiky Raja — *Professeure, ISEP*

Mme. Callau-Zori Mar — *Docteure, Outscale*

# Acknowledgments

Throughout this PhD, I received invaluable support and advice from academic supervisors, colleagues at Outscale, friends and family. I will forever be thankful to them.

First and foremost, I would like to thank Laurent Seror, Outscale's founder and president, for his trust and for the impetus given to this project.

Eric Gressier-Soudan (CNAM), Raja Chicky (ISEP), Sylvain Lefebvre (ISEP) and Mar Callau-Zori (OUTSCALE), spent countless time advising me and providing warm support in difficult times. I am incredibly grateful for patiently transmitting me your wisdom.

As a member of Outscale's engineering team, I had the pleasure to work with Christophe, Fabien, Evgeny, Nizar, Aurélien, Rahma, Salim and Quentin. The team meetings and informal discussions with you were an amazing opportunity to benefit from your expertise on operating systems, networks, programming languages, technology in general and even economy and politics. I would also like to thank the development, quality assurance and operation teams who helped me acquire the data used in this project. I owe you so much!

Big up to Sathyia, Manuel, Amadou, Rayane, and Denis. I have been lucky to share time at the lab with you, and to become your friend.

I would never have completed this thesis without my family. Mom, Dad, Camille and Hugo, your love always pushed me to do better.

I also acknowledge the french National Association of Research and Technology (ANRT) for funding this thesis through grant CIFRE N. 2015/0934.

# Résumé

L'un des principaux défis des infrastructures d'informatique en nuage (cloud computing) est d'offrir aux utilisateurs une performance acceptable, tout en minimisant les besoins en matériel et énergie. Cette thèse CIFRE, menée en collaboration avec Outscale, un fournisseur de services cloud, vise à améliorer l'allocation des ressources de la plateforme grâce à de nouvelles sources d'information pertinentes.

Les caractéristiques de la charge soumise à l'orchestrateur déterminent dans quelle mesure il est possible d'allier performance et économie de ressources. La plateforme d'Outscale et sa charge de travail possèdent des caractéristiques particulières : avec $\sim 150k$ machines virtuelles créées par mois, la plateforme est un ordre de grandeur plus grande que des clouds privés, et un ordre de grandeur plus petite que des clouds dits "hyperscale", comme celui de Microsoft Azure. C'est pourquoi nos trois contributions visent à optimiser l'allocation des ressources en tenant compte des spécificités de la charge de travail.

En premier lieu, nous caractérisons la charge d'Outscale, d'après des traces d'exécution collectées sur la plateforme et publiées après anonymisation. L'originalité de notre travail réside dans la caractérisation du déploiement des principales ressources virtuelles (machines virtuelles, volumes, instantanés de volumes, groupes de sécurité, images systèmes) afin de comprendre leur association et le stress résultant sur l'orchestrateur. Par ailleurs, nous caractérisons l'utilisation de ressources matérielles ainsi que les interférences liées à la sur-allocation du CPU et au surcoût de la virtualisation.

En seconde contribution, nous proposons un modèle de prédiction de la durée de vie des VMs à partir de caractéristiques prédictives qui sont, ou pourraient être, disponibles à leur démarrage. En plus de caractéristiques déjà connues, comme la quantité de ressources demandée, et l'historique de consommation de l'utilisateur, nous proposons d'utiliser l'information contenue dans la configuration

réseau de la VM, les étiquettes textuelles attachées à la VM pour permettre son inventaire, ou encore des contraintes de placement.

Notre troisième contribution est l'évaluation de la sensibilité d'un algorithme de placement des VMs qui requiert une prédiction de leur durée de vie. Cet algorithme issu de la littérature, appelé RTABF, avait préalablement été évalué en considérant des prédictions de durée de vie parfaites. Nous avons donc évalué, via simulation, la sensibilité de RTABF à différents niveaux d'erreur de prédiction. Notre travail invalide l'utilisation des prédictions de durée de vie avec l'algorithme RTABF. Ainsi, la question de trouver une utilité aux prédictions se pose de nouveau.

En résumé, cette thèse contribue à l'amélioration des techniques d'allocation de ressources sur les plateformes cloud, à travers la caractérisation de la charge de travail, la prédiction de la durée de vie des VMs, et l'évaluation de la sensibilité d'un algorithme de placement de VMs aux erreurs de prédiction. L'algorithme étudié ne permet pas de faire bon usage des prédictions de durée de vie des VMs. Nous pensons néanmoins que de telles prédictions pourraient être utiles, notamment pour réduire le nombre de migrations nécessaires lors des maintenances des serveurs.

Mots-clés : Infrastructure à la demande, placement de machine virtuelle, plateforme cloud, orchestrateur

# Abstract

One of the main challenges for cloud computing providers remains to offer trustable performance for all users of a multi-tenant platform, while maintaining an efficient use of hardware and energy resources. In the context of this CIFRE thesis conducted with Outscale, a French public cloud provider, we performed an in-depth study aimed at making new sources of information available to improve resource management algorithms.

Resource allocation at Outscale must take into account the particularities of the platform and workload. With 150k VMs/month, Outscale's workload is an order of magnitude smaller than the workload of "hyperscale" providers such as Azure, and an order of magnitude larger than the workload of on-premise platforms. This thesis contains three contributions that aim at adapting Outscale's resource allocation policies to the specificity of its workload.

The first contribution is the characterization of Outscale's workload, based on traces that we collected. We characterize the correlated utilization of virtual resources to understand their association in user deployments and the resulting stress for the orchestrator. The originality of this work stands in the analysis of the whole set of resources consumed by users (volumes, images, snapshots, security groups), in addition to the virtual machines. Besides, we characterize the utilization of hardware resources, and we discuss the implications of CPU overcommit and interferences.

The second contribution is the prediction of the runtime of VMs based on features which are available when VMs start. We use well-known features from the literature, such as the amount of resources requested or the user account history; and we propose new features such as text tags used to describe VMs, or the network configuration.

The third contribution is the sensitivity analysis of Release-Time Aware Best-Fit (RTABF), a VM placement algorithm that requires the prediction of VM runtime to minimize the energy usage of

servers. We analyze the performance sensitivity of RTABF with respect to the prediction error. This work disproves the efficiency of the RTABF algorithm, and raises interesting questions on the utility of prediction for scheduling purposes.

In summary, we contributed to the management of resources in IaaS with the characterization of the workload, the prediction of VM runtime, and the sensitivity analysis of a VM placement algorithm with respect to prediction error. We were unable to prove that VM runtime predictions can be used by a placement algorithm to save energy. However, prediction models could serve other purposes, such as the differentiation of overcommit policies, or the admission of short-running VMs on servers scheduled for maintenance.

Keywords : Infrastructure as a Service, virtual machine placement, cloud platform, orchestrator

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

**Contenu**

## 1.1 Context

This industrial thesis was conducted in collaboration with Outscale, a public cloud provider. Cloud computing was invented to enable easy and affordable access to computing resources. It is used, for instance, to develop autonomous vehicles [3] or medical treatments [4], which have in common the need to find meaningful information in large datasets. Since the amount of data that must be collected, stored, and processed worldwide nearly doubles every two years [5], it is mandatory to optimize the affordability of computing resources.

Cloud computing services provide on-demand access to a pool of processing and storage resources accessible over the network, and shared by multiple users [6]. Resources are in self-service, thus users can adapt their utilization to their needs. Resource utilization is measured and users pay exactly for their usage. Hence, users are financially incentivized to release the resources they no longer need for the benefit of other users. In 2011, the National Institute of Standards and Technology determined three cloud computing service models, differentiated by the position of the boundary between the provider and user management domains [6] (Figure 7.1):

— In *Software as a Service* (SaaS), users share an application that is maintained by the provider. Users can parametrize restricted application settings.

— In *Platform as a Service* (PaaS), users deploy their application code in an execution environment that includes the necessary software libraries.

— In *Infrastructure as a Service* (IaaS), users deploy their application code, libraries and operating systems on the provider's hardware. The hardware is made of fundamental computing resources such as servers that embed RAM and CPU. Servers are connected, either physically or via a network, to block storage devices.

Complementary service models have emerged, such as Storage, Function, Acceleration or Container as a Service [7]. In this thesis, we focus on IaaS, the fastest-growing segment of the market [8], and the specialty of Outscale, our industrial partner.

Outscale has two key assets: 1) the hardware infrastructure built in partnership with CISCO, Intel, Netapp and Nvidia; and 2) a proprietary *orchestrator*, TINA OS, to manage the infrastructure and allocate resources to users. As shown on Figure 7.2, the *virtualization* of the infrastructure allows users to safely share hardware. For instance, the CPU and RAM of servers are allocated under the

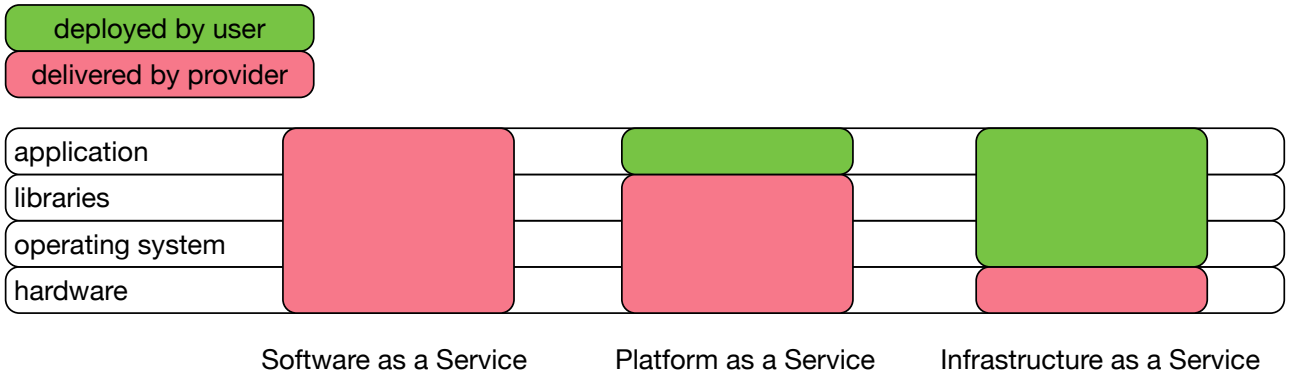



Figure 1.1: Perimeter of resources management responsibility in the three cloud service models

form of virtual machines (VMs), which provide an isolated execution environment for an OS and its applications [9]. Similarly, storage hardware is virtualized into volumes, images and snapshots. Volumes store user data, images are system templates, and snapshots store the state of a volume at some point in time. Network appliances are virtualized too, and they are configured with security groups. In this virtualized context, resource allocation is therefore defined as the mapping between the provider's hardware resources and the users' *virtual resources* [10].

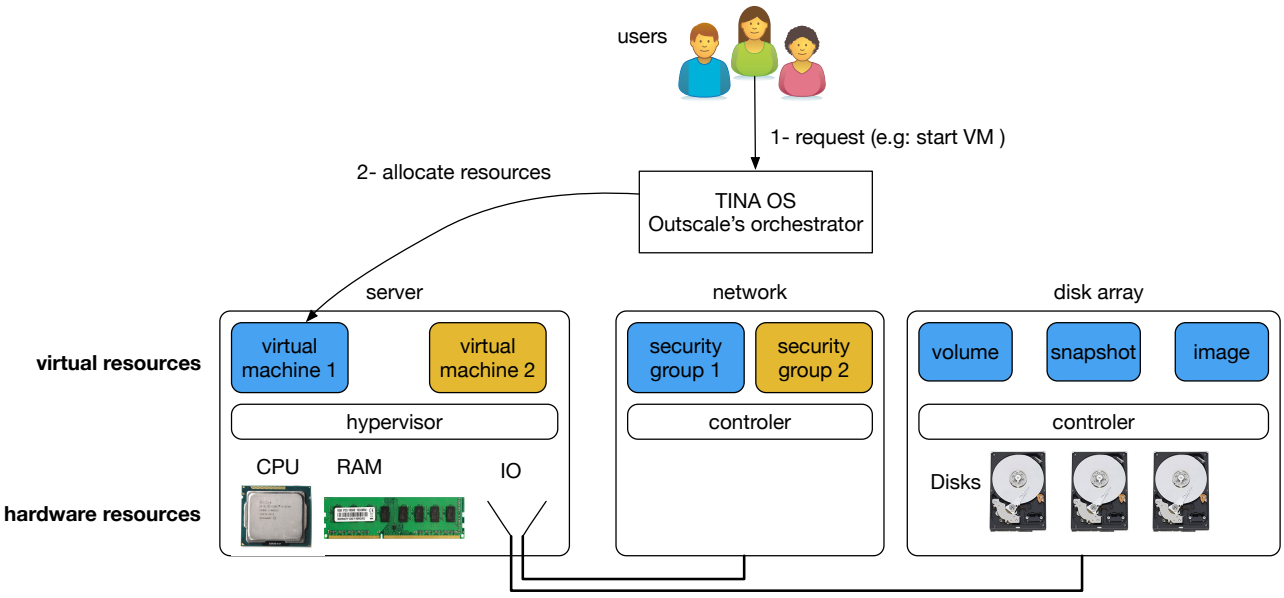



Figure 1.2: Outscale's IaaS platform

Resource allocation has three objectives:

— Service availability. In order to maximize user satisfaction, the provider seeks to maximize

its ability to respond positively to resource provisioning requests. Consequently, the provider seeks to minimize *fragmentation*, which is the existence of available resources distributed over the infrastructure but in amounts too small to be allocable [11]. Since servers bundle CPU, RAM, and possibly other resources such as GPUs, the fragmentation of a resource depends on the utilization of others.

— Service cost. The provider seeks to minimize service costs, which encompass hardware and energy expenditures. To maximize the utility of hardware, the provider can *overcommit* resources. Overcommitment is defined as the allocation of an indivisible hardware resource, such as a CPU core, to several users [12]. In addition, the provider seeks to minimize energy costs by shutting down unused servers.

— Performance. The provider seeks to give stable performance to the different service classes. Hence, the allocation of resources must take into account the heterogeneous performance of hardware, which has been acquired over the years. Moreover, the provider seeks to minimize performance *interferences* that arise when two users contend for a shared resource [13].

In summary, the orchestrator must take advantage of the complementarity between individual user workloads to maximize the profit generated from the utilization of the infrastructure, while controlling the Quality of Service (QoS) experienced by users. In the next section, we derive open problems pertaining to resource allocation in the specific context of Outscale's platform architecture and service constraints, and we introduce our contributions.

## 1.2 Research Problems and Contributions

The goal of this thesis is to help Outscale optimize the management of its infrastructure. Outscale's platform must be able to host as many virtual resources as possible and enforce a controlled level of QoS. The resource allocation algorithm is a heuristic, i.e., the translation of the expertise of administrators into a set of rules [14]. The quality of decisions depends on how well the heuristic suits the workload. We address the problem of improving resource allocation in two steps: 1) characterize the workload to identify opportunities regarding the management of resources, and 2) propose solutions suited to the workload.

### 1.2.1 Identify opportunities regarding the management of resources

Previous works in the literature have characterized the deployment of VMs [2, 15, 16], but neglected the utilization of volumes, snapshots and security groups. Since these virtual resources are used in association with VMs, there is a lot to discover from the characterization of co-deployments. For instance, the characterization of the correlated utilization of volumes and snapshots is required to optimize the management of storage.

Besides, workload characterizations in the literature focus on public hyperscale platforms hosting millions of VMs per month such as Microsoft Azure [2], and on-premise platforms hosting in the thousand of VMs [15, 16]. Neither hyperscale nor on-premise platforms have the same workload than Outscale: Outscale operates a public non-hyperscale platform with $\sim 100k$ VMs/month, which is an order of magnitude smaller than hyperscale platforms, and an order of magnitude bigger than on-premise ones.

Focusing on VMs, previous works from the literature described their utilization of hardware resources (CPU, RAM, IO) [15]. To our knowledge, there is no study of the QoS on an entire platform. To characterize the utilization of hardware resources and the QoS, it is mandatory to comply with Outscale's privacy policy, which forbids the insertion of code in system images. Thus, all monitoring must be done from the hypervisor.

The previous observations lead us to word four research questions: How do users deploy and associate virtual resources? In terms of VMs, is the workload received by Outscale's orchestrator different from the workload of other providers? How can such differences be used to improve the management of resources? Which QoS metrics are observable from the hypervisor, and what factors influence them?

To identify opportunities regarding the management of resources, we characterize Outscale's workload, based on traces that we collected. We characterize the correlated utilization of virtual resources to identify the sources of stress for the orchestrator. We characterize the utilization of hardware resources by VMs and the CPU contention and discuss the implications on QoS management. We compare the VM workload at Outscale with the workloads at Azure and Bitbrains, who are respectively hyperscale and small-scale providers. We find that bursty VM and snapshot creations stress the orchestrator. Long-lived volumes define the long-term storage space requirements. A majority of

VMs has an idle CPU utilization, but some VMs (especially small ones) experience CPU interferences. Finally, VMs request and use different amounts of CPU, RAM and disk at Outscale than at Azure or Bitbrains. This first contribution led to two publications:

— Loïc Perennou, Mar Callau-Zori, Sylvain Lefebvre, Raja Chiky. *Workload Characterization for a Non-Hyperscale Public Cloud Platform*, short paper, Proceedings of the IEEE International Conference on Cloud Computing (CLOUD '19).

— Loïc Perennou, Mar Callau-Zori and Sylvain Lefebvre. *Understanding Scheduler Workload on Non-Hyperscale Cloud Platform*, poster, Proceedings of the 19th ACM/IFIP Middleware Conference (Middleware '18).

### 1.2.2 Improve the allocation of resources to VMs by predicting the behavior of VMs

For the placement of VMs on servers, the orchestrator has to make online decisions, i.e. with a partial knowledge of the problem inputs [17, 18]. New problem inputs may come from user requests such as starting or stopping VMs, or administrator requests such as preparing a server for maintenance. To cope with an always changing problem definition, existing VM placement algorithms rely heavily on migrations [19]. A VM migration is the transfer of a VM state from a server to another via the network. Works in the state of the art proposed to use migrations to consolidate VMs on a minimum number of servers [20], and turn off unused servers to save energy [21]. Migrations are also used to perform server maintenance without requiring users to stop and restart their VMs. Unfortunately, migrations take time, consume resources [22], degrade the QoS of VMs [23], and sometimes fail across servers with heterogeneous characteristics. That is why administrators would rather "place the VMs where they can stay" [2], which requires to predict the behavior of a VM at startup. Given that Outscale's IaaS service is accessible via an API, we make the hypothesis that users have automated their deployments, and use VMs in a repeated and predictable fashion.

Three new questions arise: Can the behavior of a VM be predicted from information available when it is started? How can predictions be used to serve the purpose of VM placement? What is the impact of prediction errors on the placement of VMs? We answer these questions in two contributions:

1. We propose a model to predict the runtime of VMs based on metadata available at startup. Being interested in the prediction of a range rather than an exact value, we formulate this as

a classification problem. We show that the utilization of *tags*, which are freely-typed pieces of text used to describe VMs, improves significantly classification results compared to approaches which do not use tags. This contribution led to one publication:

— Loïc Perennou, Raja Chiky, *Applying Supervised machine learning to predict virtual machine runtime for a non-hyperscale cloud provider*, Proceedings of the 11th International Conference on Computational Collective Intelligence (ICCCI '19).

2. We analyze the sensitivity of a VM placement algorithm from the literature which requires predictions of VM runtimes to save energy. This VM placement algorithm, Release-Time Aware Best Fit (RTABF [24]), was previously evaluated under the assumption of perfect predictions. To complement the evaluation of RTABF and determine the required prediction accuracy, we analyze the sensitivity of RTABF with respect to the prediction error. We simulate the execution of RTABF for various levels of prediction error, and we compare the energy consumption of servers against any-fit and best-fit, two well-known VM placement algorithms unaware of the runtime. In order to compare our results with the original evaluation of RTATF, we use the same workload trace provided by Google for our simulations. We find that RTABF does not outperform best-fit even with perfect predictions, contrarily to what was reported in the original paper. This contribution led to one publication:

— Loïc Perennou and Sylvain Lefebvre. *Runtime Prediction Error Levels for Virtual Machine Placement in IaaS Cloud*, Proceedings of the 9th International Conference on Ambient Systems, Networks and Technologies (ANT '18).

The remainder of this manuscript is organized as follows. In Chapter 2, we present the state of the art on resource allocation algorithms in Infrastructure as a Service, the content and utilization of existing workload traces, and the utilization of machine learning in the context of resource allocation. In Chapter 3, we characterize Outscale's workload, and compare it with the Azure's and Bitbrains' workloads, two providers who offer similar services. In Chapter 4, we present models that aim at predicting the runtime of VMs when they start. Finally, in Chapter 5, we analyze the sensitivity of Release-Time Aware Best-Fit (RTABF) with respect to runtime prediction errors. Conclusions and future works are given in Chapter 6.

# Chapter 2

# State of the Art

**Contenu**

## 2.1 Introduction

Clouds are large-scale platforms that offer compute, network and storage resources to multiple users. In Section 2.2, we introduce the hardware and software architectures of clouds, and discuss the implications regarding the allocation of resources. Then, in Section 2.3, we focus on the allocation of compute resources of servers to VMs, which is also called *VM placement*. We survey state-of-the-art VM placement methods that consider various objectives and resource utilization models. In Section 2.4, we present methods that aim at predicting the resource utilization of VMs, or establish performance profiles. Finally, in Section 2.5, we survey the content and utilization of cloud workload traces for research on resource allocation.

## 2.2 Bottom Up View of the Datacenter Architecture

The section introduces the hardware and software architecture of a cloud datacenter from the bottom up, starting with the most elementary resource (the server) up to the most abstract one, an aggregate pool of compute, networking and storage resources managed by the orchestrator. Along the description, we present the fundamental implications of architecture patterns on resource management and QoS.

### 2.2.1 Hardware Architecture of a Server

In 1971, Intel invented the microprocessor, an integrated circuit embedding all the transistors needed to process information in a computer. Since then, the improvement of chip manufacturing processes led to a miniaturization of transistor size. As transistors become smaller, they dissipate less heat, allowing an increase in microprocessor complexity and operating frequency. Until now, processor performance has doubled every two years [25]. Multi-task operating systems were created to take advantage of faster processors. A multi-task OS switches the process running on the processor. Multi-tasking introduces a fundamental tradeoff in CPU allocation: A high number of processes sharing the processor is cost-efficient, but processes risk to wait for the CPU.

Computer performance has also benefited from architectures with multiple processors working in parallel. In the Symmetric Multi-Processing System (SMP) architecture [1], shown on Figure 2.1, all processors have a uniform access cost to the memory and IO systems. Processors have a dedicated

cache, and may also share a second one, called the layer 2 cache (L2, not shown in the figure). In this architecture, the tradeoff between high utilization and execution latency applies to the memory bus and cache as well as processors. The memory can be accessed by a single processor at a time. A processor can evict data put in the L2 cache by another processor. Thus, the QoS given to a process depends on the behavior of all other processes.

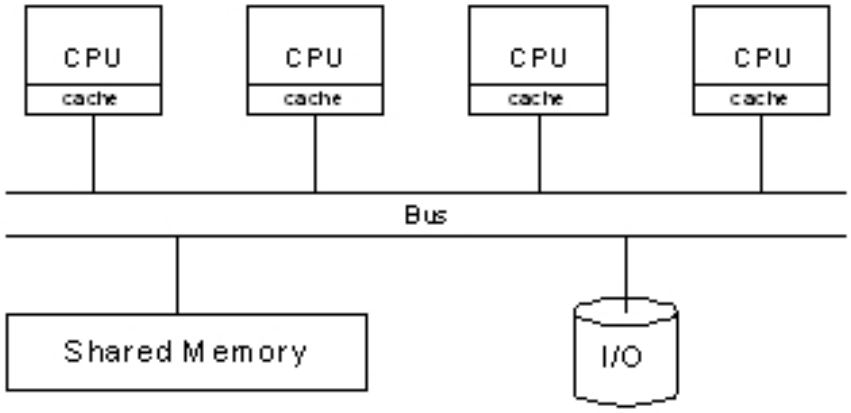



Figure 2.1: The Symmetric Multi-Processing Architecture, from [1]

The Non-Uniform Memory Access (NUMA) architecture was designed to overcome the limitation of the SMP architecture. As shown on Figure 2.2, the NUMA architecture divides the pool of processors and memory in nodes. It is 50% faster for a processor in a given node to access the local memory in the same node than the memory of remote nodes [26]. Provided that two processes do not share data, NUMA zones help to isolate them.

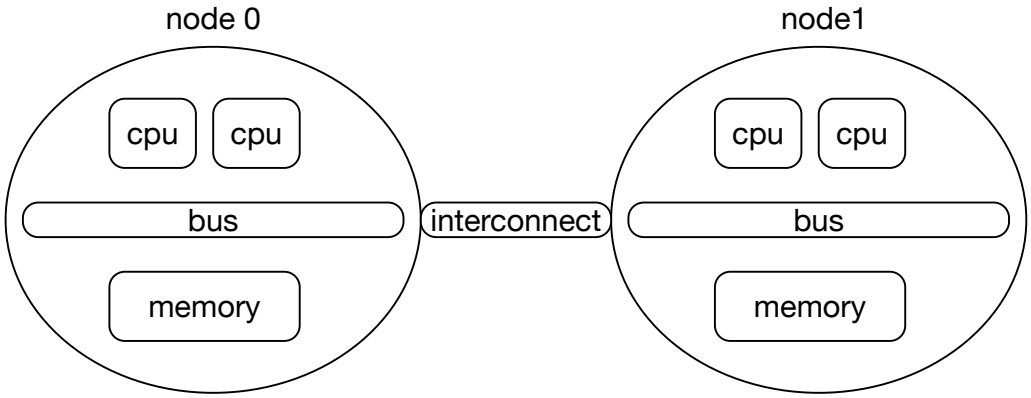



Figure 2.2: The Non-Uniform Memory Access Architecture

In the next section, we explain how virtualization allows to securely execute the application processes of several users on the same server.

### 2.2.2 Virtualization

Virtualization allows the concurrent execution of several Operating Systems (OSs) and their application processes on the same server [27]. The advantage is that applications can be consolidated to increase hardware utilization while keeping their own OS. They are isolated from each other, so if one application or OS crashes, the others are not impacted. And since the *guest* (OS + applications) are run by a software that emulates the behavior of hardware, their state can be duplicated or migrated to another server like any data structure.

Virtualization consists in giving the illusion to competing guest OSs that they control the hardware, by introducing a software agent called the hypervisor, or the virtual machine monitor [28]. The hypervisor exposes an interface to each guest OSs. It receives control instructions via these interfaces, arbitrates between them and performs the resulting operations on the hardware. As a result, hardware is shared between VMs.

Kernel Virtual Machine (KVM) is one of the most popular hypervisor, thanks to its reliance on hardware virtualization capabilities of modern CPUs, and the resource management capabilities of the linux OS [29]. A CPU with hardware virtualization capabilities can trap privileged instructions sent by a guest OS, signal the hypervisor and wait for further instructions [27]. This solution is easier to implement than paravirtualization, used by the Xen hypervisor [28], which requires to modify the guest OS such that privileged instructions are replaced by calls to the hypervisor [30]. And the overhead is lower than with binary translations, which supposes that all the guest's instructions are analyzed by the hypervisor at runtime, so that privileged ones can be changed [31].

Let us illustrate the implementation and benefits of virtualization for two privileged operations: network and the disk usage. For networking, one popular solution is VxLAN [32]. VxLAN is an encapsulation format used to create and isolate several virtual networks within a single Ethernet domain, and to propagate a virtual network across several Ethernet domains. The later feature makes it popular in cloud infrastructures which commonly span multiple datacenters. In the VxLAN specification, the hypervisor encapsulates the outgoing Ethernet frames of its VMs in a UDP datagram. The encapsulation masks the inner source and destination addresses to the network equipment until

decapsulation, which is done by the receiving hypervisor (Figure 2.3). Thus, there is no conflict if two VMs in different virtual networks (e.g., B and B' on the figure) have the same address. Users can freely configure their virtual networks.

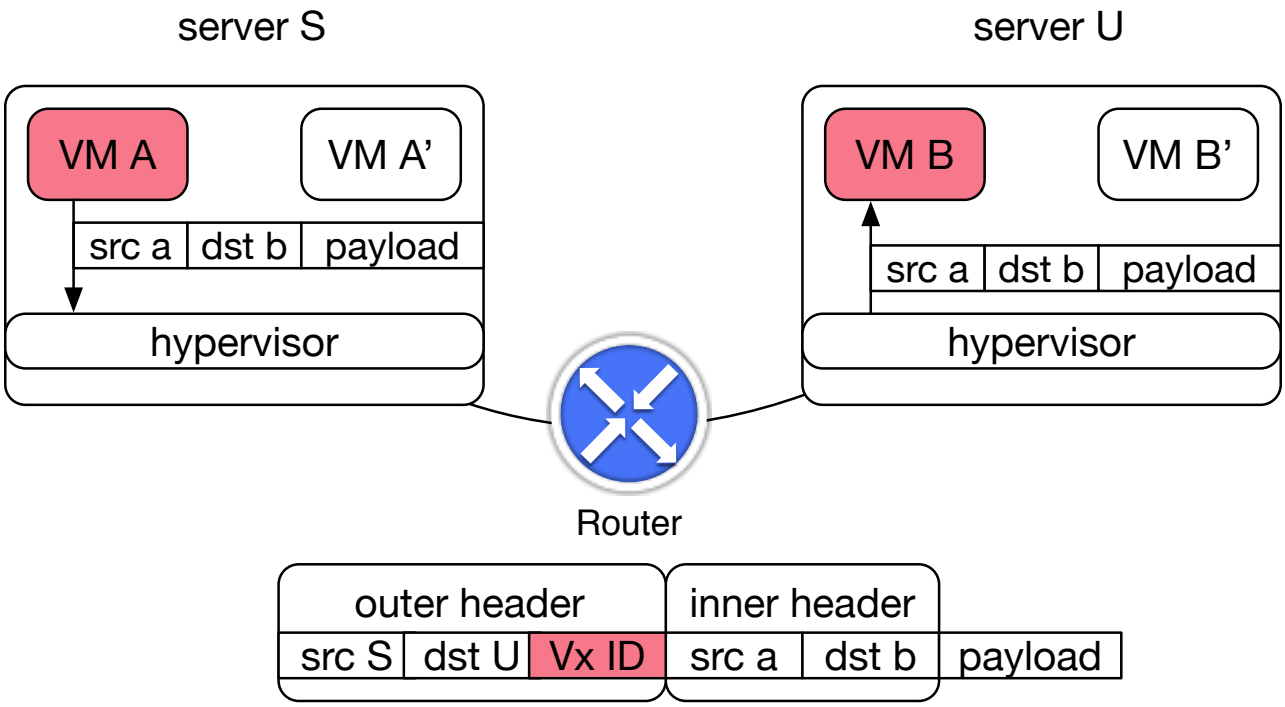



Figure 2.3: Traffic encapsulation

Disk virtualization consists in translating operations on block devices into operations on files. One popular file format for virtual disks, QCOW, offers two advantages: 1) base disk images are read-only files, so they may be shared among users to reduce storage space requirements; and 2) fresh data is written to an initially empty file, so storage can be overcommitted if users do not fill their virtual disks.

Both the virtualization of hardware resources and the consolidation of applications have implications on performance. Because the hypervisor is responsible for any operations made on hardware on behalf of the guest OS, the hypervisor and the guest need to synchronize. Lettieri et. al study the synchronization overhead of IO operations [33]. This is a first reason why the performance of a VM does not reach that of bare metal. The performance of a VM degrades further when it shares some resources with other VMs. This phenomenon is called *performance interference* [34]. Lee et al. study CPU, cache, network and storage interferences independently [35]. Podzimek et al. analyze the

impact of CPU pinning on the aggregate performance of co-located workloads [36].

### 2.2.3 Scaling Out The Architecture

There is always a limit on the amount of resources per server. To overcome this limit, cloud datacenters scale horizontally, and leverage the resources of thousands of servers connected via a network. The network is also used to connect servers to remote storage (Figure 7.2 from Chapter 1). Although remote storage introduces latency, is allows to execute a VM from any available server. The orchestrator manages all the equipment - servers, network and storage - in coordination. Upon reception of a request from a user or administrator, or from its own initiative, the orchestrator interacts with the specific equipment controllers to perform management operations.

In this distributed architecture, server resources are fragmented. A VM running on a server cannot leverage the CPU and memory resources from another server. Consequently, resources are wasted if unused by the local VMs. Besides, servers have heterogeneous characteristics because they are acquired over the years. In summary, in a distributed cloud architecture, the QoS given to a VM depends on the characteristics of its server, the background load created by co-located VMs, and even the load of VMs running on other servers in the case of networking and storage QoS.

## 2.3 Allocation of Resources to Virtual Machines

In this section, we survey the definitions and solutions of problems related to the placement of VMs. In the literature, VM placement is treated as a variant of the traditional bin packing problem, where the goal is to pack objects into bins [18]. Indeed, the orchestrator must pack VMs onto the minimal number of servers in order to save resources. In this case, the problem is defined as follows: Given a set of servers $S$ with resource capacity $C$, and a set of VMs $V$ with resource usage $u_1, \cdots, u_n \leq C$, find the minimum number of servers $B$ and a $B$-partition of $V$ such that $\sum_{vm_i \in S_k} u_i \leq C, \forall k \in [1, B]$. In some cases however, the cloud provider has other objectives than to minimize resource usage. We now survey the objectives, the modeling of problem variables, and the methods to search through the space of candidate solutions.

### 2.3.1 Objectives

Here, we list the objectives of resource allocation as well as the ways to combine them. Resource allocation must benefit to the cloud provider, who seeks to maximize the profitability of the infrastructure; and to the users, who want to have the performance that they pay for (no more, no less).

#### 2.3.1.1 Maximize Profitability

The cloud provider seeks to maximize the profitability of its infrastructure. This broad objective can be decomposed in two parts: minimize resource wastage, and allocate resources to the users who value them the most.

Wasted energy represents a significant fraction of data center operation costs [37]. Servers are not energy proportional: an idle server consumes half the energy of a fully loaded one [38]. And for every watt powering an idle server, an additional 0.6W is consumed by power supply and cooling systems [39]. Hence, the provider aims to minimize the amount of wasted energy. In [14], the orchestrator consolidates VMs on the fewest number of servers, which are chosen based on their energy efficiency. In [40], the orchestrator consolidates network flows between pairs of VMs onto physical links, such that the minimum number of switches and interfaces have to be powered on. In [41], the minimization of energy consumption by servers and switches is formulated as a joint optimization problem. In [42, 43], the VMs are placed on servers to minimize the utilization of network links. This objective allows to provide bandwidth to more VMs using the same network infrastructure, and hence increases profitability. It also benefits to users, because network latency decreases when traffic remains localized in the lowest network layers.

Another method to increase the profitability of the infrastructure is to lease spare resources at a discount price, under the condition that the provider can claim them back to service other users who are willing to pay more. The major public cloud providers have implemented this type of pre-emptible VMs. They are called spot instances in EC2, low-priority VMs in Azure, and pre-emptible VMs in Google Compute [44]. The problem then takes an economic aspect, because the objective is to allocate resources to users who value them the most. The auction system presented in [45] allows users to bid for VMs periodically. The provider then knows the demand and supply for each VM type. It computes the spot price above which VMs can run. It is possible that some VMs are forcefully stopped

when demand and spot price increase above their users' bids. One limitation is that the number of available slots for each type is fixed. So, it is possible that demand does not fit supply for all types, which would incur a loss of revenue. In [46], the scheduler adjusts the supply of VM types to their respective demand. The spot market is useful to avoid wasting spare resources. But it does not allow to fundamentally reduce resource consumption for a given workload.

#### 2.3.1.2 Respect Service-Level Agreement on Performance

In the interest of users, the objective of resource allocation is to enforce an appropriate level of performance for applications running onto VMs, or equivalently, to control the virtualization overhead and interferences between VMs. To do so, Delimitrou et al. model the affinity of applications with respect to various hardware configurations (e.g., CPU frequency, cache size ...), and also their co-location affinities [47]. In [48], the goal is to support different classes of service. The authors group VMs by performance class. An agent continuously monitors the performance of applications running onto VMs. If gold-class VMs are performing well, they give one of their dedicated CPU cores to silver VMs. The CPU can be claimed back if needed. In [49], the goal is to provide a homogeneous performance to identical tasks that are distributed across multiple servers. A statistical distribution of their performance is collected. Tasks whose performance is much lower than their siblings are detected, and their low-priority contenders are throttled. Further details on the modeling of application performance will be given in Section 2.3.2, and we will discuss the feasibility in the context of IaaS.

#### 2.3.1.3 Multi-objective resource allocation

The minimization of resource wastage and respect of performance SLAs are conflicting objectives. There are several ways to combine them. The first possibility is to treat one objective as a constraint, and evaluate solutions based on the second objective. For instance, in [50, 51], the algorithm minimizes the number of servers under the constraint that the probability of server overload is below a threshold. The second possibility is to combine the evaluation of multiple objectives with the same function. In [52], the objective function combines the energy efficiency and performance objectives. In that case, the difficulty is to determine the optimal weighting. The third possibility is to use the Pareto-optimal policy [53, 54]. A solution candidate is Pareto-optimal if it is impossible to make one criterion better off without making other criteria worsen off.

Solving these optimization problems requires a proper modeling of the workload. In the next section, we present the resource utilization and performance models.

### 2.3.2 Workload Models

VM placement is an optimization problem where the orchestrator aims at minimizing resource wastage and respect the performance SLA. The candidate solutions to this problem are compared with an objective function which takes in input the utilization of resources and the performance of VMs. This section compares how these two variables are modeled.

#### 2.3.2.1 Static Resource Utilization Model

In this model, the footprint of a VM is modeled by a static utilization of resources. In [55], the utilization is multi-dimensional and represents a fraction of a server's resources (CPU, RAM, disk, and bandwidth of the network access link). As observed by Rai et al. in [56], the problem with this model is that it does not allow to model resources that are shared by VMs on different servers, such as core network links. The authors propose to represent all datacenter resources as a large vector. As a result, resource utilization needs more memory to be encoded. In these models, performance is considered to be acceptable as long as the total utilization of a resource is less than a threshold [14]. The model is valid until the resource utilization changes, or existing VMs stop, or new VMs start. When changes occur, the provider must re-optimize resource allocation via costly migrations.

#### 2.3.2.2 Dynamic Resource Utilization Model

The dynamic utilization model aims long-term optimizations, as it captures changes in the resource utilization of VMs. The footprint of a VM is modeled by a time series [57, 58, 59]. The performance of a VM is estimated from the correlation of the resource utilization by the VM, and the aggregate utilization of neighbor VMs [58, 60]. This model supposes that VMs run indefinitely, and that resource utilization is periodic.

#### 2.3.2.3 Clairvoyant Resource Utilization Model

A clairvoyant model assumes the provider knows both the resource usage and the timing of future VM management requests. Zhao et al. consider a fully clairvoyant model where the provider knows the

timing of VM start and stop requests [61]. Srinivasan et al. consider that VMs start simultaneously at $t_0$, and their runtime is known [62]. Dabbagh et al. consider the case when the runtime of VMs is known, but not the start time [24].

Resource utilization models have three inherent limitations. Firstly, the utilization of low-level resources, like the L2 cache and the memory bus, cannot be measured. The lack of accountability can lead the scheduler to make the wrong choices. Secondly, the performance degradation experienced by an application running on a VM not only depends on the background load, but also on the nature of the application itself. Thirdly, the acceptable level of interference also depends on the application. Mars et al. differentiate user-facing, *latency-sensitive* applications such as web or database servers, and *batch* applications such as offline data analytics [63]. They have different QoS requirements: interferences are tolerated for batch applications, but latency-sensitive applications have a stringent deadline. Next, we introduce models that capture heterogeneous performance profiles.

#### 2.3.2.4 Performance Model

In this type of model, applications are profiled in a controlled environment to estimate their performance sensitivity to competing applications, and the amount of interferences they cause themselves. One of the main challenge is the profiling burden: it is too costly to benchmark all possible combinations of co-location. In [63], latency-sensitive applications are scheduled with a first benchmark that behaves like a batch application, creating pressure on the memory bandwidth. The sensitivity of the latency-sensitive application is measured. Then, batch applications are scheduled with a second benchmark to measure their exerted pressure. The utilization of two benchmarks as proxy for the two types of applications reduces the profiling costs to the total number of applications. However, this approximation is not accurate. In [47], each latency-sensitive application is profiled against two batch ones (in separate runs). Machine learning is used to infer the sensitivity of the latency-sensitive application with other batch applications. The model assumes that interferences are additive, i.e., that it is possible to estimate the interferences from a group of applications by summing their individual contributions. In [64], machine learning is used to infer the degree of interference resulting from multiple batch applications.

Performance models have a major limitation in the context of IaaS: due to the confidentiality policy, the provider does not know if a given VM hosts a latency-sensitive or a batch application.

This section completes the definition of the VM placement optimization problem. The next section surveys solving methods.

### 2.3.3 Solving of the resource allocation problem

According to Mann [18], the resource allocation is tackled under two types of initial conditions. The initial conditions apply to :

— the number of VMs - The resource allocation problem may apply to a single VM, a group of VMs related to a distributed application, or all the VMs in the datacenter.

— the timing - Resource allocation may apply to new VMs that are starting, or VMs that are already running on servers.

The most complex variant is the re-optimization of resources for all running VMs. The problem is NP-hard, i.e., the number of potential solutions is exponential with the number of VMs and servers [65]. In addition, migration cost must be accounted for. Architectural choices can help tackle the complexity. For instance, the set of servers can be partitioned to allow solving independent problem instances in parallel. Secondly, some methods speed up the exploration of the space of candidate solutions. In this section, we present the different methods for partitioning the problem and guiding the evaluation of candidate solutions.

#### 2.3.3.1 Problem Partitioning

Under a centralized architecture [14, 66, 20, 55], the scheduler optimizes resource allocation on all the infrastructure. The centralized problem definition allows to model complex constraints applied to arbitrary sets of VMs or servers. For instance, in [55], a user can ask the scheduler to spread a set of VMs on at least 3 servers, in order to guarantee that the application will be highly available. In theory, the centralized scheduler is omniscient, so it can find the global optimum to the resource allocation problem. But in practice, because of the NP-hardness, finding the solution in acceptable time (i.e., before the workload changes) for a large-scale datacenter is challenging [65]. Another limitation of the centralized architecture is its lack of fault tolerance. If a group of servers gets disconnected from the scheduler, then their resources cannot be managed.

Kesavan et al. describe a hierarchical architecture in [67]. In this architecture, servers are logically partitioned in independent clusters. A cluster scheduler allocates the resources within each cluster.

A global manager balances the load between clusters. It removes server capacity from underloaded clusters, and makes it available to overloaded ones. When a server is removed from a cluster, all its VMs are migrated to its peers. In [68], clusters are formed periodically and at random using a gossip protocol. Every time new clusters are formed, the different schedulers optimize resource allocation. Randomness increases the likelihood of convergence to an optimum. In [69], servers monitor themselves and, upon detection of an event (e.g., overload), initiate a ring cluster with their neighbors. If the problem cannot be solved with the help of a neighbor, then another one is added to the cluster, and so on until success. A deadlock may arise when two clusters cannot grow because there are no free servers left. In this case, they are merged.

In the fully distributed consolidation approach, couples of servers (a source and a destination) exchange one VM at a time. In [70, 71, 72], the source server chooses the destination. The scalability is limited because the source must monitor all potential destinations. In [73, 74], the source proposes its VM to all destinations, and they reply how much they would benefit from receiving it. Again, the communication overhead is large. In addition, it is unclear how this architecture can handle placement constraints applying to different VMs. Imagine, for instance, that VMs A,B,C,D should run on three distinct servers to ensure high availability. Initially, server 1 runs VMs A and B; server 2 runs VM C; and server 3 runs VM D. Imagine that server 1 is underloaded and offers to give VM A. Server 3 accepts the invitation. The exchange of VM A breaks the placement constraint because neither the source nor the destination know that other VMs (B and C) are both on a third server.

As shown in this section, the scheduler can optimize resource allocation for the entire datacenter, or just a partition. In the next section, we will present how the search for the best configuration can be performed from an algorithmic point of view.

#### 2.3.3.2 Exploring the Space of Candidate Solutions

When the scheduler manages an entire data center or cluster, the number of feasible solutions is large. Here, we present three families of algorithms that aim to optimize the exploration of the space of candidate solutions.

Greedy algorithms decompose the problem into stages and choose the local optimum for each one. Both the method used to decompose the problem and the cost function used to compare partial solutions are chosen according to past experience on similar problems. Hence, greedy algorithms are

heuristic methods that trade solving optimality for speed. For instance, the placement of a set of VMs on a set of servers such that the number of servers is minimized can be solved with the First Fit Decreasing (FFD) heuristic. The rationale behind FFD is that big VMs should be placed first, and then small ones should fill the remaining "holes". So, FFD first orders VMs by size, and then from the biggest to the smallest one, puts them in the first available server with sufficient remaining capacity. Modified versions of the algorithms have been proposed to benefit from the knowledge of multiple resource dimensions [35] and runtime [62, 24]. Heuristics have two limitations. First, they are designed by humans for whom reasoning about conflicting objectives at the same time is difficult. Secondly, their performance is not guaranteed, and it depends on the assumptions made on the problem inputs [35].

A metaheuristic makes less assumptions about the problem than a heuristic, and thus may perform better on different problem variants. It randomly explores space of candidate solutions. A configuration candidate is a mapping between the list of VMs and the list of servers. The search is guided by probabilities such that the algorithm is more likely to explore the neighborhood of a good candidate than a poor one. The Ant Colony Optimization metaheuristic, which is inspired by how ants find their food, is used in [68, 53]. Each ant builds a candidate solution in parallel. Candidates are compared, and a pheromone is deposited on the best VM/server pairs. Being guided by pheromones, ants are likely to explore similar candidates in the next rounds.

In the constraint programming approach [20, 55], the solution search is exhaustive. It is nonetheless driven by heuristics, such as the first-fail approach. This approach consists in assigning the variables that have the tightest range first, so that the algorithm detects non-viable solutions as early as possible. When a non-viable solution is encountered, the algorithm backtracks, i.e., it changes the last variable assignment. A constraint solver that solves large-scale cloud resource allocation problems on a GPU is presented in [56].

### 2.3.4 Discussion

This section introduced the different objectives, models and solving methods used for the optimization of resource allocation. The main objectives are to minimize the amount of resources needed to support the workload, and to maximize the performance of applications running onto VMs. From our point of view, the main challenge is to find a common model accommodating both these objectives.

Modeling the performance of applications is challenging in IaaS because the provider does not know what type of application runs on a VM. Currently, providers assume that performance is met if the server load is under a given threshold. It is thus necessary to obtain a precise performance model for IaaS. Besides, another challenge, inherent to combinatorial optimization problems, is to find the best tradeoff between the optimality of resource allocation, and the speed of convergence. For instance, a centralized scheduler architecture coupled with a constraint programming algorithm or metaheuristics can find good configurations with few migrations, but they may need a long search time. In the next section, we will study the utilization of machine learning to optimize resource allocation.

## 2.4 Machine Learning Based Approaches

Cloud datacenters offer a large pool of resources accessible on-demand. Consequently, VM placement is online and large-scale. The risk, for the orchestrator, is to have the problem definition change before an acceptable solution is found and enforced. Machine learning, which explores the design of algorithms that learn to perform a task from experience instead of being explicitly programmed [75], can be used to speed up the search of the optimal VM placement, or to anticipate changes in resource utilization and take pro-active decisions. For instance, machine learning is used to predict future resource utilization based on past observations, and to migrate VMs out of servers that will become overloaded. Machine learning can also be used to approximate the performance profile of VMs, which allows to compute a VM placement configuration faster than with an accurate performance model. This section surveys the utilization of machine learning for the prediction of resource utilization and the approximation of VM performance. Then, we discuss the applicability of approaches based on machine learning to the context of IaaS.

### 2.4.1 Prediction of resource utilization

The resource utilization of VMs is dynamic. Machine learning is used to anticipate changes in utilization, and optimize resource allocation pro-actively. We list three types of models. In the *single VM* model, the resource utilization of each VM is modeled independently. In the *multiple VM* model, a model predicts the resource usage of a group of VMs. While both models aim to make predictions during the execution of VMs, a *startup* model gives a prediction when the VM is started.

#### 2.4.1.1 Single VM Prediction Model

Many solutions have been proposed to predict the resource usage of a VM based on its past utilization. A survey of time series forecasting techniques is given in [76]. The paper also presents a decision tree allowing to choose the best technique based on the problem context, and feedback over prediction quality.

Both *Press* [77] and *Agile* [78] make short term CPU usage predictions using signal processing. *Press* applies a Fast Fourier Transform to time series in order to isolate their strongest periodic components. If strong periodicity is found, the next sample of the series is extrapolated. Otherwise, the series is modeled with a Markov chain. *Agile* uses wavelets instead of Fourier transform, because wavelets are better at analyzing acyclic patterns. In [50], the time series of CPU usage is decomposed into a periodic components and a residual. The residual is modeled as an Auto Regressive process: its predicted value at $t$ is a linear combination of the previous values.

In [79], the CPU utilization of a VM is modeled with a bag of neural network. The period of the time series, measured with the autocorrelation, is used as a feature. Bagging, which is the process of training several learners with random splits of data and combining their predictions, is used to reduce overfitting.

Server utilization is modeled with a Bayesian classifier in [80]. The authors propose 10 high-level features derived from the CPU load, compare them and show that keeping 3 particular ones gives optimal results. The prediction window is divided exponentially such that a short-term interval is small and benefits from a high resolution prediction.

#### 2.4.1.2 Multiple VM Prediction Model

In [81], a single Hidden Markov Model is trained for a group of VMs. This choice is motivated by the observation that individual VMs have a noisy resource usage. This noise can be filtered out by taking advantage of load correlations that exist between VMs, which arise because VMs are collaborating to support complex services. The same argument is used in [82], and a deep neural network is used to predict the CPU load of all VMs. In [83], a method called "tracking the best expert" is used to predict traffic demand. Traffic demand between all pairs of VMs is represented as a matrix. The future matrix is estimated with a linear combination of past measurements. The weights are adjusted online every

hour.

#### 2.4.1.3 Predictions served at startup

Predictive systems presented so far can only help to optimize resource allocation after the start of the VM, because they require the observation of its past resource usage. Other systems are designed to serve predictions when the VM starts, in order to optimize the initial resource allocation.

In [2], the CPU usage, running time, deployment size and workload class of VMs is predicted with an Extreme Gradient Boosting Tree. A deployment is a cluster of collaborating VMs. The two workload classes are delay-insensitive and latency-critical. The proposed scheduler uses the prediction of maximum CPU usage to overcommit this resource.

On high performance computing and grid platforms, one of the scheduler's objective is to minimize the total execution time of the job waiting queue. Historically, users provided a maximum allowed job run time to help scheduling. But, since their estimation is often too cautious, this is not optimal. Yadwadkar et al. use support vector machines to predict the likelihood of a task of a data mining job to be a straggler [84]. Straggler tasks are the ones that finish late due to contention for shared resources, and they slow down the job completion. A differentiated resource allocation that depends on job runtime is proposed in [85]. Short batch jobs are consolidated aggressively whereas long-running and interactive ones have dedicated resources. A prediction of job runtime is made at startup, and the model uses support vector machines too. The features correspond to the job's resource request. Linear regression is used in [86] to predict the exact job run time. Interestingly, many features are related to the previous jobs of the same user, e.g. their running time, submission time, initial resource request and user submission rate. In [87], decision trees are found to perform well on predictions of runtime from only 6 days of training data.

### 2.4.2 Approximation of interference profiles

In Section 2.3, we presented placement techniques that respect performance SLAs. Finding the interference level between all possible combinations of co-located VMs is cumbersome and hinders scalability. In this section, we survey how machine learning simplifies the approximation of interference profiles.

#### 2.4.2.1 Identification of groups of VMs with similar resource usage variations

Some works [58] use the correlations between resource usage time series as a proxy for interferences. Instead of reasoning about the interferences between single VMs (which supposes computations between all VM pairs), Verma et al. proposed to identify groups of VMs with similar resource usage patterns thanks to clustering techniques [57]. They use k-means to find groups of VMs whose peak utilization is correlated. In [60], a 2-phase method is proposed: first, VMs are clustered based on the values of peak resource usage. Then, sub-clusters are found based on correlations. In [88], spectral clustering is applied on the histograms of resource utilization. Multiple metrics are taken into account (CPU, system call rate, cache miss rate and IO rate). In [89], hierarchical clustering is performed to find groups of disks with uncorrelated access and consolidate them on the same array. The cost of computing pairwise correlation remains, but reasoning about clusters instead of single disks still reduces the complexity of resource allocation.

#### 2.4.2.2 Speeding up Interference Measurement

In some works, the orchestrator minimizes the performance degradation of co-located applications [63]. Given the sheer arrival rate of new applications in the data center, it is impossible to profile all combinations of co-locations. Delimitrou et al. argue for the use of collaborative filtering to speed up interference profiling [47]. A new application is profiled against only two existing ones, and the affinity with other applications is inferred. Linear regression is used in [64] to model the affinity of a new application with respect to a group of existing ones. However, this technique cannot be used by IaaS providers because the provider does not know which application runs on a VM.

### 2.4.3 Discussion

Machine Learning is used to predict resource utilization, or approximate interference profiles. Predictions allow to make pro-active decisions and approximations allow to speed up the comparison of VM placement configurations. Most approaches require VMs to run for some time before any decision can be made. Given that cloud computing is best suited for users with temporary needs, it is not certain that the orchestrator has sufficient time to model the behavior of VMs before they stop. A promising approach is to make predictions at startup, based on the information contained in the VM request, and the resource usage of other VMs that were executed previously. Only one paper applied

this approach to a IaaS workload [2]. The ongoing research on the utilization of machine learning for VM placement requires datasets for model design, training and evaluation. In the next section, we survey the content and characterization efforts of cloud workload traces.

## 2.5 Characterization of Cloud Workload Traces

The ability of the orchestrator to achieve the optimal resource allocation depends on the fitness between the algorithm and the characteristics of the received workload. This is why it it necessary to take into account the characteristics of the workload for the design and evaluation of orchestrators [90]. Calzarossa et al. published a recent survey (2016) on workload characterization that covers cloud platforms, among others [91]. The survey reports findings on workload characteristics, but not the content of the different sources of data. In addition, since cloud computing is an emerging topic, several new analyses have been published since then. In this section, we compare the content of cloud workload traces and survey the insights obtained from their characterization. Then, we identify remaining opportunities regarding workload characterization in the context of IaaS.

### 2.5.1 Comparison of Workload Traces

Here, we compare cloud traces related to the allocation of hardware resources to virtual ones. The comparison is based on the workload type (VMs or jobs), the platform scale, the tenancy (public and private), and the public availability of data. A summary is provided in Table 7.1.

#### 2.5.1.1 VM Based Workload

Regarding public clouds, Azure published in 2017 a 30 days-long trace including 2M VMs [2]. This hyperscale platform is used by numerous users for a wide range of applications. The trace contains the number of cores and the amount of memory requested by VMs, their start and stop times, their utilization of CPU (min, average and maximum utilization in 5-minutes windows) and deployment ID. A deployment is a set of related VMs that belong to the same user and are located in the same cluster. Bitbrains, a company that specializes in the application hosting business for the finance and insurance sector, published two traces [15] from a non-hyperscale public platform. The first trace reports the execution of 1250 VMs connected to a Storage Area Network (SAN), while the second reports the execution of 500 VMs connected to SAN or to a slower Network Attached Storage (NAS). The traces

include the resource request of VMs (number and speed of cores, memory) as well as the resource utilization (CPU, memory, disk and network) sampled every 5 minutes. Since VMs run throughout the duration of their respective trace (1 and 3 months), the trace does not contain VM state change events.

Regarding private clouds, Eucalyptus Systems published traces from six customer platforms that rely on the open-source Eucalyptus scheduler [92]. The traces contain a description of the hardware (the largest platform has 31 servers with 32 cores each), the state of VMs (start and stop times), their CPU request and placement, but no measurements of effective resource utilization. A trace including 11k VMs was collected over a year on the Czech scientific cloud CERIT-SC [16]. The platform is managed by the OpenNebula scheduler [93]. Two use cases are given: cloud VMs are started and managed by users to run the application of their choice, whereas grid VMs are used to run scientific jobs. The trace contains the resource requested by VMs and their role but not their effective resource utilization.

In this description, we include two traces that were not published, but were nonetheless characterized (the characterization findings for all traces are given in the next section). Nutanix collected traces from 2000 private enterprise clusters mainly deployed in remote and branch offices [94]. The traces include the resource request and applicative role of VMs, hardware configurations and failure events. Disk utilization is reported at the cluster level, but not for single Ms. Finally, IBM collected traces from six cloud data centers. Traces report the correlated utilization of VMs and images, as well as statistics on image content similarity at the block level [95].

#### 2.5.1.2 Job Based Workload

In 2011, Google published a trace from one of its private and hyperscale cloud backend. The platform ran two classes of workloads: user-facing jobs, such as a web service with stringent latency requirements; and batch jobs such as offline data analysis. The 7M jobs were scheduled on 12.5k servers during 29 days [96]. A job comprises one or more tasks, each of which is executed on a container. The trace reports the relationship between users, jobs and tasks; the state of tasks (pending in a queue, running, etc...), their attributes (resources requested, constraints guiding the choice of server, and scheduling priority); the state of servers and their attributes (normalized resource capacity and opaque hardware characteristics), and the resource usage of running tasks. Regarding resource

utilization, CPU, IO and memory metrics are reported, as well as processor performance counters such as the number of cycles per instruction (CPI) and the number of memory accesses per instruction (MAI). Resource utilization was measured every second, which allows to report average and maximum utilization every five minutes.

A similar job-based trace collected on 4k servers during 8 days was published by Alibaba [97]. Resource utilization is reported at the task and server levels. In addition to CPU, memory, disk IOs and network bandwidth utilization, the CPI and the number of cache misses per kilo instruction (MPKI) are reported to characterize performance.

In the Google and Alibaba traces, server resources are virtualized with containers. This is different from the IaaS model where servers execute VMs. Because VMs run a guest OS, they provide more isolation at the cost of having more overhead than containers [98]. Based on these differences, the deployment patterns and resource utilization described here for containers is unlikely to be representative of VMs.

The comparison of workload traces is summarized in Table 7.1. Google's and Alibaba's traces report QoS metrics, but the job-based workloads has a different nature than IaaS workloads. Among VM-based workloads, we can identify two types of contexts from the platform scale. The Azure trace reports the utilization of a general-purpose, hyperscale platform (2M VMs/month). On the other hand, the other traces describe specific use cases like finance or scientific computing on smaller platforms (1k VMs/month for Bitbrains and SCERIT-SC). Being a general-purpose, non-hyperscale cloud platform with still  100k VMs/month, Outscale's context is not represented in the available workload traces. Moreover, only the IBM trace, which is not published, reports the utilization of resources other than VMs (images). The utilization of volumes, snapshots and security groups has not received attention prior the collection of Outscale's trace.

### 2.5.2 Characterization of Workload Traces

Here, we survey the characterization of public and private traces of cloud workloads. The works surveyed shed light on the management of server and storage systems, failures at the hardware and software levels, economics of resource pricing and patterns of application deployment.

| trace | entity | hyper scale | state events | virtual machine utilization | | | | | other virtual resources |
|---|---|---|---|---|---|---|---|---|---|
| | | | | CPU | RAM | Disk | Net. | interf. | |
| Google [96] | job | yes | yes | yes | yes | yes | – | yes | – |
| Alibaba [97] | job | yes | yes | yes | yes | yes | yes | yes | – |
| Azure [2] | VM | yes | yes | yes | – | – | – | – | – |
| Eucal. Sys. [92] | VM | – | yes | – | – | – | – | – | – |
| SCERIT-SC [16] | VM | – | yes | – | – | – | – | – | – |
| Bitbrains [15] | VM | – | – | yes | yes | yes | yes | – | – |
| IBM [95] | VM | – | yes | – | – | – | – | – | images |
| Nutanix [94] | VM | – | – | – | – | – | – | – | – |
| Outscale | VM | – | yes | yes | yes | yes | – | yes | yes |

Table 2.1: Comparison of the content of cloud workload traces

#### 2.5.2.1 Server Management

Reiss et al. characterize the heterogeneity of the Google workload [99]. Heterogeneity in hardware configurations, job resource request and runtimes reduce the effectiveness of slot-based scheduling. Comparisons of the Google workload with grid and HPC workloads are given in [100, 101]. Lu et al. characterize the spatial and temporal server load imbalance resulting from the job churn at Alibaba [97]. Statistical modeling of jobs [102] and VMs [92] allow the generation of synthetic traces and the evaluation of VM placement algorithms. Cano et al. analyze the resource requests of VMs as a function of their applicative role [94] in private entreprise clusters sold by Nutanix. Regarding the characterization of effective resource usage, Mishra et al. identify groups of jobs with similar resource usage [103], and Dumont et al. identify atypical VMs [104]. Cortez et al. characterize the CPU utilization of VMs from the Azure trace [2]. They propose a system that predicts the CPU utilization of starting VMs. Predictions are used to find the best server. Based on unnamed traces, Birke et al. characterize the correlations of CPU, memory and disk utilization on the same server [105], while Verma et al. explore the correlations between servers that host a distributed application [57].

#### 2.5.2.2 Storage Management

Storage administrators must make several choices to optimize storage cost and performance. Cano et al. quantify the size needed for storage caches, and the space savings obtained by using compression

and deduplication [94] in Nutanix clusters. They also show that customers tend to fill storage space progressively, hence storage requirements are often predictable. Peng et al. analyze the correlated utilization of VMs and images in IBM datacenters [95]. They show that images are very good candidates for deduplication for two reasons. Firstly, some images are used repeatedly. Secondly, the content of two images can be similar, particularly if they contain successive OS releases.

#### 2.5.2.3 Resource Pricing

The cloud provider fixes the price of resources such that revenue is maximized. Kilcioglu et al. explain the current prevalence of static pricing models based on the low volatility of resource requests [106]. They use a trace from an unnamed hyperscale platform. However, the stronger volatility of effective resource usage implies that, in the future, customers will take advantage of elasticity to minimize costs. Then, demand fluctuations will need to be balanced with dynamic prices, as in other industries such as tourism.

#### 2.5.2.4 Failure Analysis

Traces can be used to troubleshoot failures that hinder the proper execution of jobs and VMs. In [107], Chen et al. investigate the presence of patterns in the characteristics of failed jobs. They find that a significant fraction of resources are wasted on jobs that fail or get killed. They show that the termination status of jobs is correlated with pre-launch attributes such as priority or user ID, and post-launch attributes such as resource usage. To save resources, they propose to make failure predictions based on features that can be measured early in the job lifecycle. Cano et al. characterize the annual return rate for several categories of hardware components, and compare the mean repair time for hardware and software failures [94] in Nutanix clusters. Schroeder et al. study the occurrence of failures specific to DRAM memory chips in Google's platform [108]. The authors describe the incidence of several factors such as temperature, chip age, chip density and server load. Similarly, Pinheiro et al. characterize the occurrence of disk failures [109].

#### 2.5.2.5 Application Deployment

Traces are also used to characterize the deployment of applications. In 2013, He et al. showed that few customers used value-added services managed by the provider, like load balancers or a domain

naming systems [110]. Moreover, the authors found that the majority of web service deployments were made in a single availability zone. In a study performed in 2017, Cortez et al. argued that users spread deployments across several zones to increase fault tolerance [2]. They showed that 80% of cloud deployments were made of at most 5 VMs. As shown by these two consecutive studies, the utilization of the cloud is evolving with time. Hence, providers need to characterize user deployment regularly to decide of the development roadmap.

### 2.5.3 Discussion

The characterization of cloud workloads is critical to optimize the management of servers and storage , minimize the occurrence of failures, and guide the service development roadmap. The survey of workload characterizations led us to identify requirements for a trace to be used in resource management. In the context of IaaS, the trace must report a VM-based workload, with the state of VMs changing as they are started and stopped by users. The trace must include the amount of resources requested by VMs, and periodic measurements of the effective utilization. The first trace matching these requirements was published by Azure in 2017, after the start of our project. Information regarding interferences on shared resources (e.g., CPU or memory) would also be valuable to characterize the quality of service provided to VMs. Some works measured the performance variability of VMs by running benchmarks within them [111]. However, benchmarks reflect the point of view of a single cloud user, not the point of view of the provider regarding the entire set of VMs. The latter case requires to perform measurements from the host, and on the entire platform. In addition, we observe that existing traces focus on virtual machines, but generally do not report the utilization of other virtual infrastructure resources, such as images, volumes, snapshots and security groups. We found a single exception in [95], where the correlated utilization of VMs and images is characterized to optimize the management of storage. As Outscale's storage system hosts ten times more snapshots and volumes than images, it is crucial to characterize the lifecycle of all types of virtual resources.

## 2.6 Conclusion

The combination of the large platform size and dynamicity of VM workloads challenges the optimization of server resource allocation in clouds. Current production systems like Openstack make the allocation based on the request, but not on the effective utilization. Ideally, resource allocation should

be pro-active, lightweight considering the cost of migrations, and quick to enforce because load keeps changing. As seen in Section 2.3, many works explore the tradeoffs between the solution quality and search time. Works presented in Section 2.4 use machine learning to predict resource utilization or approximate an interference model. Machine learning is essential to speed up the search and enforcement of resource allocation. Yet, most approaches based on machine learning require to observe the execution of VMs. In few works, predictions are based on the utilization of VMs that were executed previously. This approach could be well adapted to enhance resource allocation for VMs that are just starting and for which real utilization has not been observed yet. And the problem of finding the features with the most predictive power is still opened.

# Chapter 3

# Characterization of Outscale's Workload

**Contenu**

## 3.1 Introduction

As discussed in the previous chapter, workload characterization is essential to the optimization of automated resource management by the orchestrator. So far, previous works have focused on public, general-purpose, hyperscale platforms with 2M+ of VMs, or small on-premise platforms with 10k- of VMs. Besides, previous works focused on the lifecycle of VMs, their utilization of hardware resources and their relationship with images. To our knowledge, previous works from the literature did not characterize the utilization of virtual resources other than VMs and images, or the interferences among VM and hypervisor threads that share the CPU.

To complement previous studies, we perform a comprehensive workload characterization of Outscale's public, general-purpose and non-hyperscale IaaS cloud platform. To this end, we collected a three-month-long trace from Outscale's European region. The contributions presented in this chapter are:

— We characterize user requests related to the management of virtual resources. We study the lifecycle and correlated utilization of resources. We discuss the implications of the lifecycle of volumes and snapshots on the management of storage hardware.

— Focusing on VMs, we characterize their individual utilization of hardware resources such as CPU, memory and disk. We show that CPU and disk utilization are skewed, e.g., 70% of VMs never use more than 20% of their requested CPU. This makes CPU overcommit for idle VMs attractive, but we identify VM churn as an obstacle to its achievement.

— From a provider's point of view, we characterize interferences resulting from the CPU overcommitment and the virtualization overhead. We discuss the implications for the automated optimization of the placement of VMs and hypervisor threads on CPU cores.

The rest of the chapter is organized as follows. In Sec. 3.2, we explain the methodology for the collection of the workload traces from Outscale's platform. Then, we characterize the workload in terms of virtual resources management operations (Sec. 3.3), VMs' utilization of hardware resources (Sec. 3.4) and CPU interferences (Sec. 3.5). In Sec. 3.6 we compare the behavior of internal users (the staff or the orchestrator itself) with clients; and in Sec. 3.7, we perform the comparison of Outscale's workload with Azure's.

## 3.2 Data Collection

In order to provide an overview of the utilization of virtual resources, we have collected and published [1] two workload traces from Outscale's European region, from August to October 2017 (3 months). The first trace reports the virtual resource management operations performed by users, and the second trace reports the hardware resource usage of VMs.

### 3.2.1 Traces of Resources Management Operations

We focus on the five most common virtual resources: VMs, images, volumes, snapshots, and security groups. Table 3.1 describes the virtual resources along with the recorded fields and the allowed management operations. To obtain the trace, we performed an offline parsing of data collected in the scheduler's centralized log repository (*syslog*). Logs are semi-structured messages with two different formats (JSON and XML). The extraction of exploitable records from raw logs involved three steps. First, we removed duplicate messages resulting from repeated requests for an idempotent operation. Then, we correlated the user's request log with the orchestrator response to filter out invalid operations (e.g., clients requesting resources above their quota). Finally, we matched the logs that referenced the same virtual resource in order to get the whole resource lifecycle and fields. For example, a volume created from a snapshot inherits its size, so we matched the two records of resource creation. The dependency to anterior actions is the main cause of missing information in our trace. Additionally, we discarded less than 1% of logs because they were truncated.

### 3.2.2 Traces of The Resource Usage of VMs

Regarding the collection of hardware resource utilization metrics, we had the restriction to not deploy any software inside the guests because of privacy policy. Hence, we deployed an ad hoc probe on hosts. To perform periodic and synchronized measurements across servers, we scheduled the recurrent execution of the probe with *cron* [2] instead of running a continuous service. The stages of execution of the probe are shown on Figure 3.1. The probe first measures the resource usage of VMs during 5 minutes. Then, to avoid sending all measurements simultaneously to the centralized database, it

1. an anonymized sample of the collected workload traces will be downloadable at http://www.github.com/outscale/OutscalePublicWorkloadTrace
2. *cron* allows users to schedule the execution of programs in Linux.

| Virtual resource | Description | Record fields | Management operation |
|---|---|---|---|
| VM | emulated physical machine | ID<br>source image ID<br>number of vCPUs<br>amount of RAM<br>family<br>type<br>OS<br>tag[a] | run<br>start<br>stop<br>terminate<br>put in sec. group |
| Image | VM template with at least a boot volume | ID<br>source VM ID<br>tag | create from VM<br>delete |
| Volume | emulated block storage device | ID<br>source snapshot ID<br>type<br>size<br>tag | create<br>delete<br>attach to VM<br>detach from VM |
| Snapshot | point-in-time backup of a volume | ID<br>source volume ID<br>size<br>tag | create from volume<br>delete |
| Security group | virtual network filtering appliance[b] | ID<br>tag | create<br>delete |

[a] A tag is a freely-typed text describing a virtual resource
[b] We did not collect network rules because they include confidential IPs

Table 3.1: Description of virtual resource management traces

sleeps during a random amount of time before transmission. We used *flock*[3] to ensure that there was no more than one probe running in each phase (collect and sleep). In addition, we used *timeout*[4] to prevent the existence of zombie probes hanging or hogging resources.

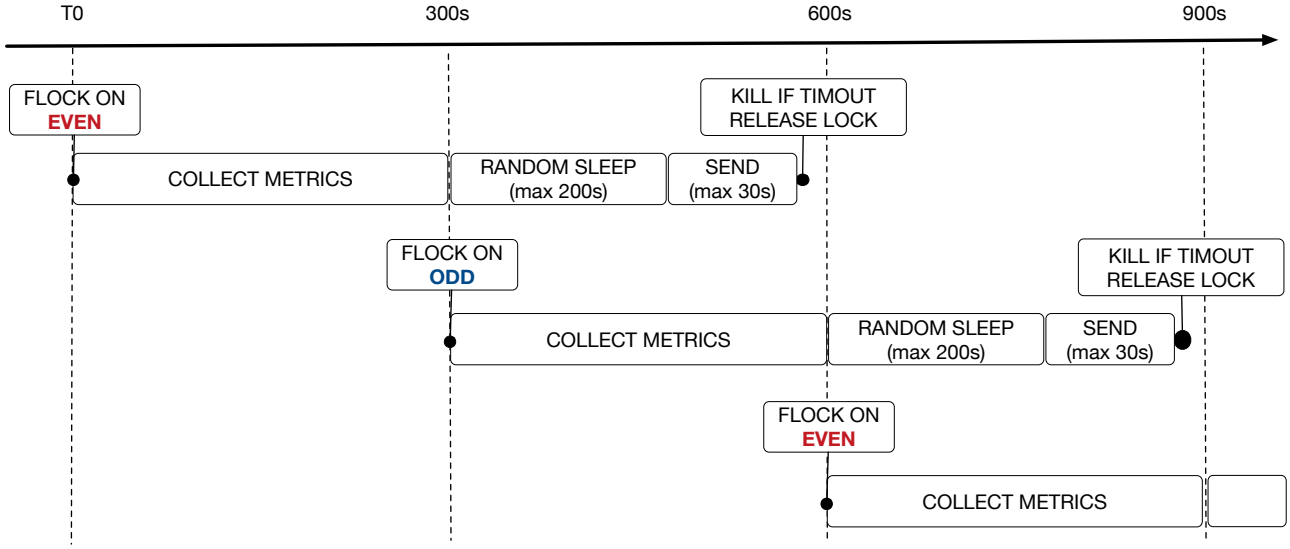



Figure 3.1: Stages of execution of the resource probe script

As shown on Figure 3.2, Outscale's hosts run the open source KVM/QEMU hypervisor [112]. QEMU monitors resource utilization of VMs and exposes metrics through a management socket. The socket was already used to receive commands from the orchestrator, therefore, it was not accessible to the probe. Instead, the probe leveraged Python's *psutil* library to read the utilization metrics reported by the OS in the */proc* filesystem. The probe retrieved the hardware resource utilization of the QEMU processes, which encompasses both the utilization of guests, the management overhead, and a fraction of the virtualization overhead. QEMU launches threads for the virtualization of CPU and disks, but optionally relies on a separate process (vhost) to handle network virtualization. When we designed the probe, we did not know the existence of the vhost process, so the network virtualization overhead is not counted in the resource usage.

Table 3.2 summarizes fields collected about VM resource usage. The first metric is `cpu` utilization. It is the sum of the utilizations of management, vCPU, and disk threads in user and system execution

3. *flock* places a lock on a file descriptor before calling a program, therefore it ensures that a single program instance can exist at any time.
4. *timeout* kills a program after the timeout given in argument.

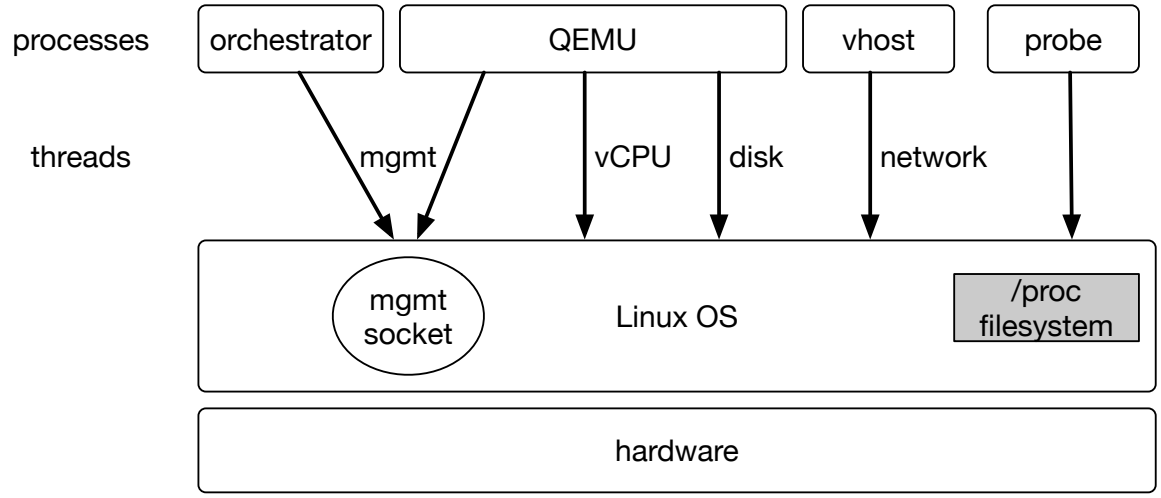



Figure 3.2: Integration of the probe in the virtualization stack

modes. To monitor memory utilization, we track the resident set size metric (`rss`), corresponding to the amount of memory stored in RAM (i.e., not swapped). Since servers are configured not to swap on disks, we are not losing memory traceability using `rss`. On the other hand, `rss` may overestimate RAM utilization by considering memory shared by multiple QEMU processes. However, knowing that two processes only share hypervisor libraries, the memory overestimation is reduced. In the case of disk usage, we have measured the number of bytes per second for `read` and `write` operations. Since disk operations go across the network, we have focused on the throughput instead of the IO rate. Two final metrics were collected to study CPU interactions between VM processes. `cpu_affinity` tracks the VMs that are using the same CPU cores. Involuntary context switches (`ics`) measure the number of times the threads of a VM got their CPU pre-empted by the OS. This occurs when the time slice of a thread expires and other threads need the CPU, or when kernel threads must handle a hardware interrupt. Our measure of `ics` does not include the contribution of vCPU threads, because their values could not be accessed until the termination of the VM. Hence, `ics` measures the CPU contention between management and disk threads of VMs on the first hand, and the threads of other VMs on the second hand.

Despite making probes connect to the database at random times, we have lost some records due to high load on the database. The reason is that we had initially considered a one-month collection time, but when we decided to extend measurements for two more months, we didn't scale the database sufficiently. We also discarded unreliable measurements taken at the beginning or end of VM executions,

| Metric | Description |
|---|---|
| `cpu` | CPU utilization, in % of CPU requested |
| `rss` | Resident Set Size, in % of RAM requested |
| `read` | number of bytes read per second |
| `write` | number of bytes written per second |
| `cpu_affinity` | list indicating on what CPU cores the VM is running |
| `ics` | number of involuntary CPU context switches per sec. |

Table 3.2: Description of VM resources usage traces

or when some counters overflowed. However, the 270M measurements guarantee the robustness of the analysis.

### 3.2.3 Implementation of Data Collection in Compliance With Security Standards

Outscale's work and organization processes comply with state of the art security standards. Outscale complies with the ISO 27001-2013 norm [5], and SecNumCloud [6] compliance is being evaluated by the French National Agency for Security of Information Systems (ANSSI).

We collected the workload trace in compliance with these security standards. The design and implementation of the solution took four months: I developed the probe, the operation team developed deployment scripts, and I developed a script to extract the logs of the scheduler every day because the log repository only keeps a 15-day history. The operation team deployed and tested the solution on an integration platform during two months, and then a pre-production platform during two months. The team validated the accuracy of the measured resource utilization metrics, checked that the probe did not interfere with the execution of VMs and that the access to the database was secure. We then proceeded to the deployment on the production platform. Unfortunately, after one month of data collection, we realized that we were missing one third of scheduler logs, because a new scheduler front-end had been added without updating the log extraction scripts. As this prevented us from correlating the logs to obtain the entire lifecycle of virtual resources, we had to restart the collection of data. Therefore, because of the need to implement an ad-hoc solution, because of the obligation to comply with the highest security standards, and because of an unexpected platform update, it took a year to collect a three-month-long workload trace.

5. The ISO 27001-2013 norm is available at https://www.iso.org/fr/isoiec-27001-information-security.html
6. The SecNumCloud norm is available at https://www.ssi.gouv.fr/uploads/2014/12/secnumcloud_referentiel_v3.1_-anssi.pdf

## 3.3 Understanding Management Operations

In this section, we characterize the virtual resource requests. We analyze traces about the five virtual resources described in Table 3.1. We study time patterns about creation time and lifetime, the amount of VMs resources requested, and the correlation between different virtual resources.

### 3.3.1 Time Patterns in Virtual Resource Management

In presence of workload peaks, creation events can stress the orchestrator, since it requires to allocate hardware resources suddenly. Figure 3.3 explores weekly patterns in the number of virtual resources created. As VMs can be started several times, we consider each start event instead of creation ones.

VM start events (black line) have a daily periodicity and a peak load at midnight. With an average of 240 VM starts per hour and a maximum peak value of 890, the workload in terms of VMs varies by a factor of 4. In the case of images (blue line), we also observe a daily periodicity, but this time with two peaks per day, one in the afternoon and one at midnight. Given that there are fewer images created than VMs, images are not usually used to back up VMs. Regarding volumes (purple line), we see a drop in activity around 8 PM. In fact, lower demand around 8 PM is also observable for VMs and images. Snapshots (yellow line) are the most often created virtual resources, and with the strongest periodicity. There are six snapshot bursts per day, 5 minor ones (800/h), and a major one (1400/h) at midnight. This reveals that users make automatic backups with scripted snapshots every 4 hours. Besides, major snapshot bursts occur when daily VM bursts start to fade off. This suggests that some users backup data once their VMs have finished their tasks. Given their popularity and periodicity, snapshot creations could be an important source of stress for the orchestrator. Major snapshot bursts represent four times the average workload, and two and a half for minor bursts. Finally, the number of security groups (red line) created is stable during the trace (40/h).

The platform must have enough capacity to allocate resources during the runtime. Figure 3.4 presents the distribution of resource lifetime through a boxplot based on the 1st quantile (Q1), the median (Q2), the 3rd quantile (Q3) and an interquartile range set to 1.5. We have only considered resources created and terminated within the 3-month collected interval, that is, resources that completed their lifecycle. All the types of resources have items that reach the maximum observable

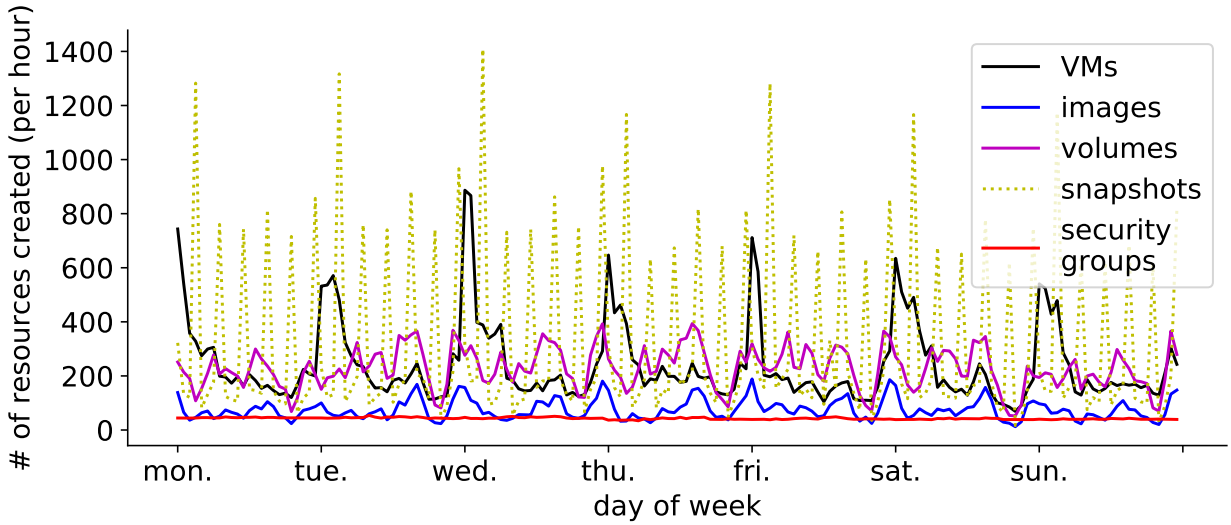



Figure 3.3: Resources creation time

lifetime, however, all their Q3 values are lower than 10 days. In the case of VMs, resources are usually not used more than 2 hours (Q2 = 18min and Q3 = 2h 14min). Even images have larger lifetimes than VMs (median = 1 day 4h and Q3 = 4 days 10h). That means, images are not used to create VMs during a long period. In the case of volumes, the distribution is wider than for VMs, meaning that volumes persist even when VMs to which they were attached have been terminated. Snapshots, which we characterized as automatized backup mechanisms, survive a short period (Q3 $\leq$ 5 days). Finally, security groups have a lifetime lower than VMs (Q3 = 5min), which seems to indicate that they are created for short-time tasks.

### 3.3.2 Resources requested by VMs

Figure 3.5 presents the resources requested by VMs. The first two boxplots represent the requested CPU and RAM. Q3 corresponds to 4 vCPU and 16GB. VMs requesting more than 8 vCPU (resp. 39 GB) are considered outliers. The third boxplot shows that VMs running more than 39min represent less than 25% of the set. If we compare Q3 values of runtime and lifetime (resp. 39min and 2h 14min), we observe that runtime is notably shorter than lifetime. The reason is that VMs can be run several times in their lifetime. The next boxplot shows the storage capacity attached to VMs; knowing that Outscale's default boot volume is 8GB, and the maximum volume capacity is 14TB. The median attached capacity (9GB) indicates that nearly half the VMs have no other volume that the

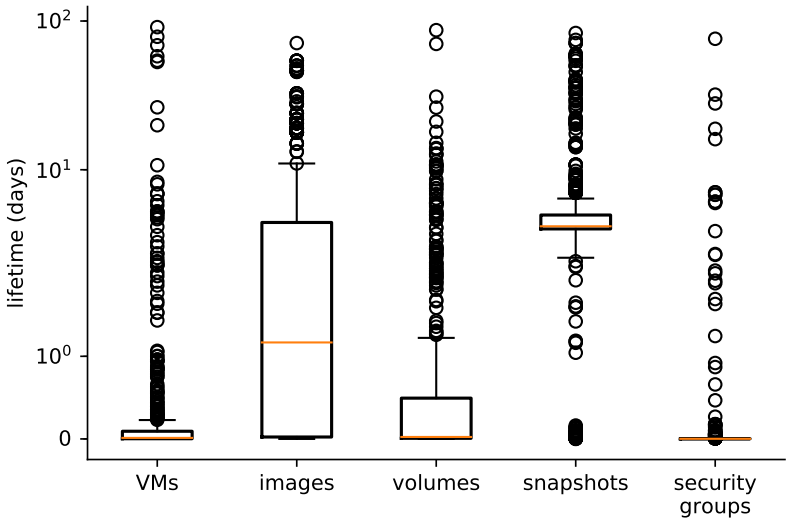



Figure 3.4: Resources lifetime

default one. The maximum (42TB) corresponds to 3 of the largest volumes. The last boxplot shows the number of security groups assigned to the same VM. Having a security group for each application is a good security practice to avoid mixed up rules. The figure shows that customers enforce security with this technique (median = 4 and maximum = 11).

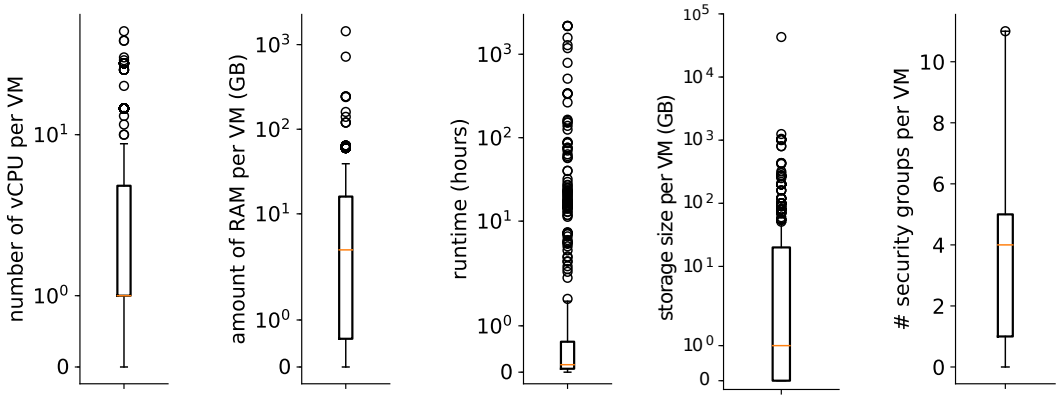



Figure 3.5: Requested VMs resources

### 3.3.3 Correlations Between Virtual Resource Requests

In this section we study dependencies between resources, Figure 3.6 presents different correlations.

Regarding the first boxplot, 75% of security groups are used by two VMs or less. This shows that users generally do not reuse security groups for different deployments. For example, clients deploying the same environment several times can simplify operations by creating a specific security group for each environment (even if they could reuse the same). However, there is a security group used more than $10^5$ times. Another feature of virtualization is to reuse the same images to deploy several VMs. The next boxplot shows the number of VMs deployed from the same image. Most of the images are used once (median = Q3 = 1); outliers basically correspond to default images provided by Outscale. The last two boxplots show snapshots/volumes correlations. The first of the two counts the number of snapshots (i.e., backups) taken for the same volume. 75% of volumes are snapshotted less than twice, but one has been snapshotted a thousand times. Finally, the last boxplot shows the number of volumes deployed from the same snapshot. We observe similar outliers in the number of VMs deployed from the same image. This effect is due to the fact, when a user creates an image from a VM, the orchestrator snapshots the VM's volumes on behalf of the user.

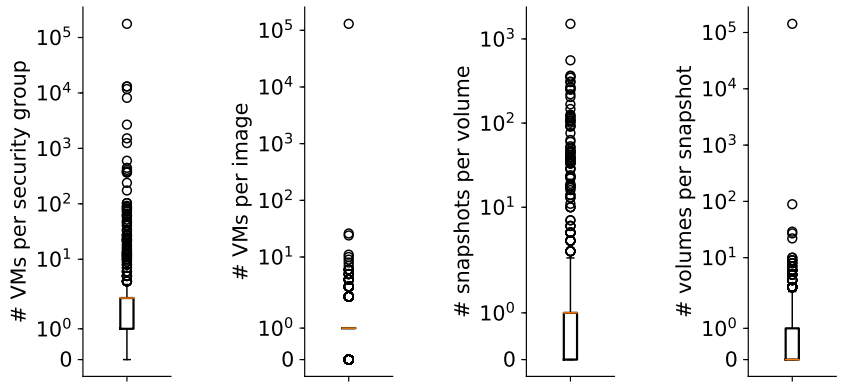



Figure 3.6: Correlations between virtual resource requests

**Consequence 1** *The utilization patterns of volumes and snapshots affect the management of storage hardware. 90% of volumes live less than 5 days. They are seldom filled with data in this interval, and consequently the average volume size is half the requested size. So, storage space can be overcommitted by a factor of two. In addition, short-lived volumes do not have to be taken into account for long-term capacity planning. The placement of the remaining volumes (8% of which are snapshotted more than five times) is more complex. They fill up progressively, and recurrent snapshots require additional space.* ***In order to maximize the overcommitment of storage space and balance the effective***

***utilization of equipment, the minority of long-lived volumes should be spread on separate shards.*** *This policy supposes that volume lifetime can be determined at creation time.*

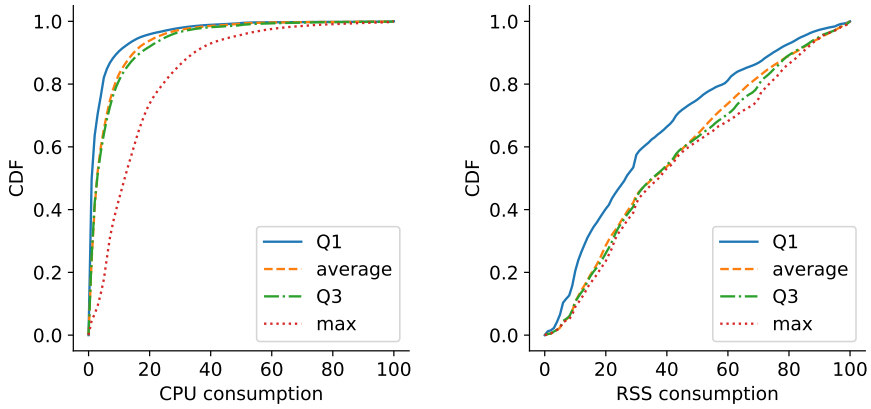
## 3.4 Understanding The Resource Usage of VMs

This section analyzes the effective vCPU, RAM and disk utilization of VMs, according to the first four metrics of Table 3.2. For CPU and RSS, which are allocated by the scheduler based on a request, we study the relative utilization (in % of the request). This enables us to compare utilization of VMs whose requests vary by an order of magnitude. For each metric, we have studied the consumption per VM through four indicators: 1st quantile (Q1), average, 3rd quantile (Q3) and maximum. We report the cumulative distribution function (CDF) for each couple of metric and indicator.

Figure 3.7a presents the CDF of the CPU utilization (%). In general, 99% of VMs have low CPU utilization with a maximum utilization lower than 50%. However, ∼2,600 VMs (0.9% of the trace) consume on average 42% of their requested CPU. Some VMs even consume 100%. The generally lower CPU consumption justifies the overcommitment of CPU resources; however, the provider must pay attention to minimize the impact of interferences on the perceived QoS. We will study CPU interferences in Sec. 3.5.

Figure 3.7b shows that the CDF of the RSS utilization (%) is close to linear. VMs have a larger consumption of RSS than CPU. This can be explained by the fact that the guest OS never releases memory to the host, since it prefers recycling it for cache, and no memory overcommit mechanism is implemented. In general, the difference between VMs' maximum and average RSS consumption is 10%, which indicates that RSS utilization is more stable than for CPU.

Figure 3.8 presents the disk utilization in terms of read load (kB/sec), write load (kB/sec), and the ratio of write load over read load. 75% of VMs read at an average rate of 377 kB/s and write at an average rate of 1900 kB/s. However, Q3 peaks at 1280 kB/s for reads and 4920 kB/s writes, multiplying by four the average load. We observe a larger write load than read: on average, VMs write 7 times more data than what they read. That could be due to two reasons. First, the guest OS caches data for read operations while write ones cannot be cached. Also, the guest OS could be configured such that any time it accesses a file, it updates the *atime* attribute (time of last access) and eventually triggers a disk write.




(a) CPU utilization (b) RSS utilization

Figure 3.7: VMs resources utilization

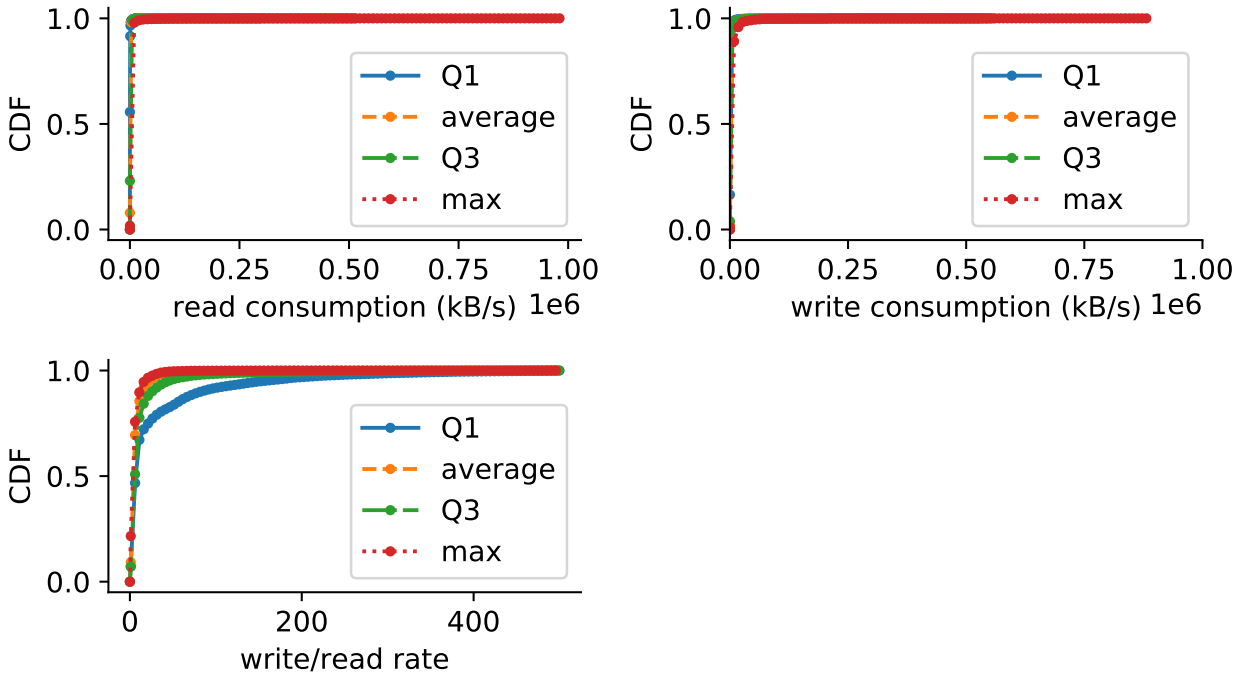



Figure 3.8: Disk utilization

Finally, to characterize the dispersion of resource utilization, we compute the relative standard deviation (RSD) as the ratio between the standard deviation and the average (for non-zero average) in Figure 3.9. We observe that the largest dispersion is achieved on reads, and the lowest on RSS.

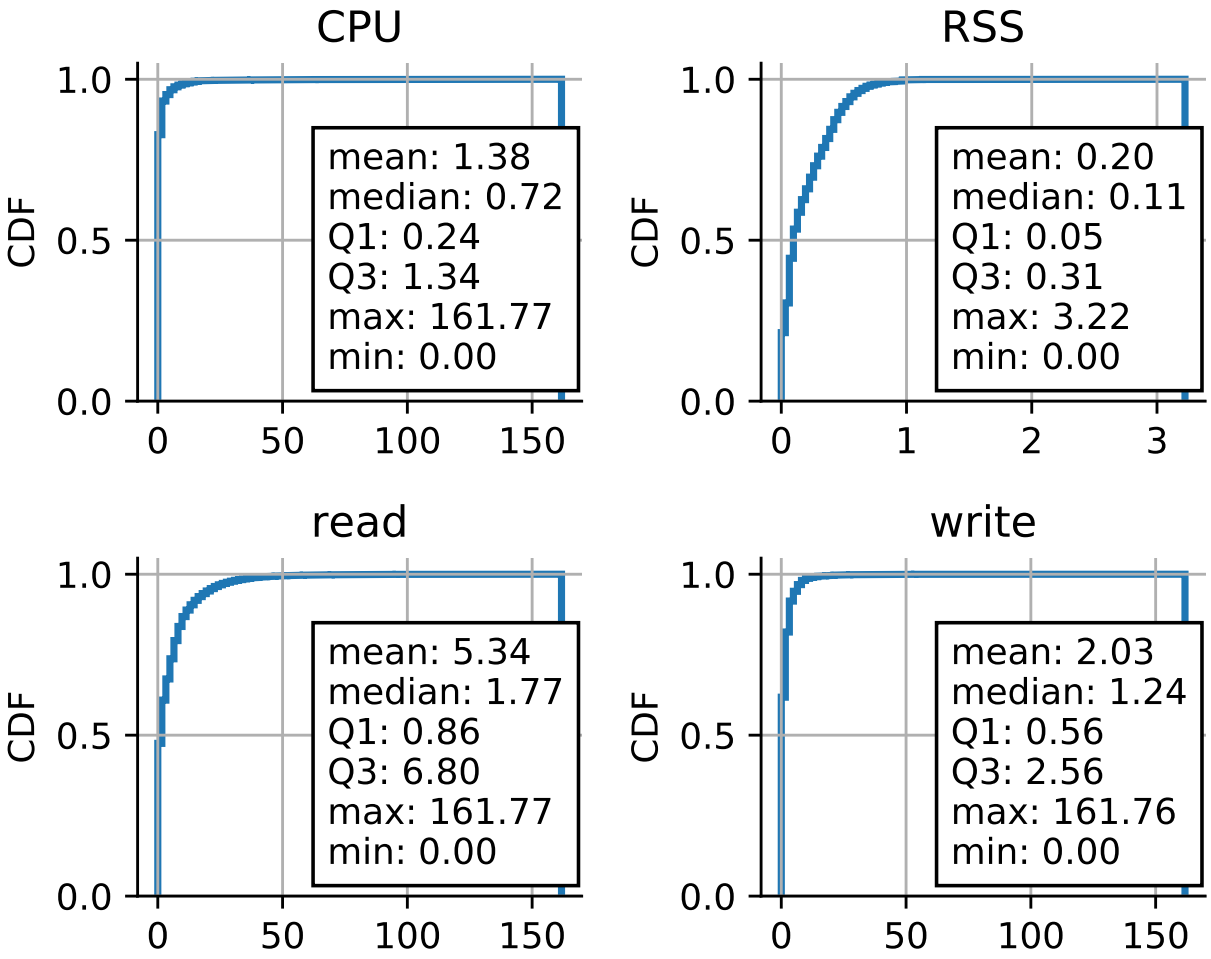



Figure 3.9: Relative STD

**Consequence 2** *70% of VMs never use more than 20% of the CPU they requested over 5-minute windows. Up to three VMs using less than 20% of CPU could share this resource to target the 40-60% utilization level recommended for mixed workloads [9], and to save 46% in CPU costs. A popular approach is to monitor the resource utilization of VMs and then consolidate the ones with compatible workloads through migrations [59].* ***At Outscale, the benefit of VM migration to manage server resources could be reduced: 90% of VMs run less than an hour, and it is not***

***worth consuming network resources to migrate the memory state of a VM that is about to be deleted.***

## 3.5 Understanding Interferences Between VMs

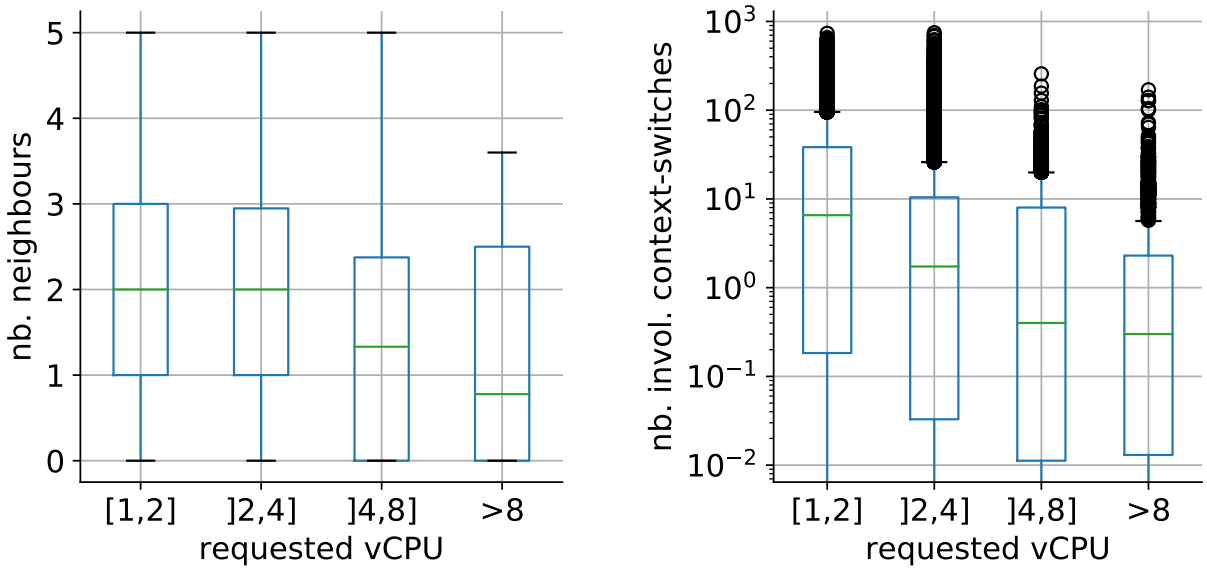
We have seen that a large number of VMs request more resources than they use. While resource overcommitment has been presented as a common approach to avoid waste caused by over-sized demand, providers must pay attention to the impact on QoS. Characterizing VMs interactions helps to understand the tradeoff between high resource usage and performance.

This section analyses VMs interferences focusing on CPU. Given a VM, we study two metrics: 1) the average number of VMs neighbors by core, or in other words, the number of VMs sharing CPU with the current VM; and 2) the number of involuntary context switches per second, which is the number of times the management and disk threads of a VM must release a CPU core for allowing another VM or the hypervisor to perform work. Both measures give information about the QoS of the VM. A large number of neighbors gives better hardware utilization and efficiency for the cloud provider, at the expense of lower QoS for the client if/when its neighbors are fully using the CPU. Therefore, we use involuntary context switches to evaluate the contention on the CPU resource.

This analysis aims to understand VM interactions according to the number of vCPU requested. To this end, we have divided the dataset in four classes according to the number of vCPU requested by each VM: [1,2], ]2,4], ]4,8], 8+ vCPU. Figure 3.10a shows the average number of neighbors by core with a boxplot. VMs requesting up to 4 cores have a median of 2 neighbors, while other classes have a lower median. However, in general, VMs have a low number of concurrent neighbors ($Q3 \leq 3$).

Figure 3.10b presents the number of involuntary context switches per second. The largest number of involuntary context switches per second is achieved for small VMs because most of these VMs are sold as low performance resources. The largest VMs, sold as high performance ones, experience a lower number of involuntary context switches per second. Finally, some VMs raise up to 1 involuntary context switch every millisecond, we therefore argue that these are VMs performing CPU or disk-intensive tasks, they are more impacted by resource overcommitment and the virtualization overhead.

**Consequence 3** *Users are interested in having VMs with acceptable and constant performance. In-*




(a) VMs neighbors

(b) Involuntary context switch

Figure 3.10: VMs interferences according to requested vCPU

*voluntary context switches measure the performance degradation resulting from virtualization and the sharing of CPU between VMs.* ***To control the performance of VMs as measured by the number of involuntary context switches, a controller could map dynamically VM and hypervisor threads to CPUs.*** *Besides, a perfect performance model should make the distinction between interferences between VMs and the virtualization overhead. By comparing the* `ics` *metric of non-overcommitted VMs with the entire population, we estimate that the virtualization overhead represents, on average, 40% of the metric value. Additional work is required to obtain finer interference models with metrics that are observable by the provider.*

## 3.6 Comparative Study of The Consumption of Clients and Internal Users

In this section, we compare the resource usage of clients and internal users in Outscale's trace. Internal users are responsible to maintain the cloud platform in an operational state as well as to perform improvements. Some of the projects related with internal users are monitoring, logging, data-warehousing, pre-sales demonstrations for clients, quality services. The provider has more information

about applications and users' behavior for internal users than for clients. We compare both kind of users with respect to the utilization of virtual resources and VMs usage.

Table 3.3 reports the number of virtual resources created by clients and internal users. Clients create $2/3^{rd}$ of VMs, their footprint is larger if we consider the number of core-hours they requested (80%). Clients are responsible for the creation of images, volumes and snapshots more than 90% of the time. However, they create only 3% of security groups. More than half of the internal security groups corresponds to quality validation of the platform. While clients make 1.5 snapshots per volume on average, the rate falls down to 0.1 for internal users. In general, services deployed internally do not require to backup volumes with high frequency due to two main reasons: some volumes do not store important data, others are part of distributed and fault tolerant systems that can recover automatically in case of failure.

| | Number of resources | | |
|---|---|---|---|
| Resource | Clients | Internal | Total |
| VM | 318,486 (66%) | 161,905 (34%) | 480,391 |
| Image | 131,730 (99%) | 623 (1%) | 132,353 |
| Volume | 428,722 (92%) | 36,910 (8%) | 465,632 |
| Snapshot | 634,673 (99%) | 3,778 (1%) | 638,451 |
| Security group | 2085 (2%) | 87,278 (98%) | 89,363 |

Table 3.3: Number of resources: clients vs. internal

Table 3.4 reports the utilization of hardware resources by VMs created by clients and internal users. For each utilization metric, we compute two indicators: the average and the standard deviation of the distributions. Relatively to their request, client VMs use on average six times less CPU that internal ones. However, they use 15% more RSS. On average, client VMs read 8 times more and write 2.5 times more data than internal VMs. They experience 6 times more involuntary CPU context switches. Regarding the dispersion of the distributions, we observe that read and write rates are highly variable ($\sigma \geq 10\mu$) for both types of VMs.

We conclude that clients create the majority of virtual resources except for security groups where the internal quality assurance service reverses the trend. Finally, client VMs use less CPU than internal ones, but more memory and storage.

| | VMs consumption: average ± std | | |
|---|---|---|---|
| Metric | Clients | Internal | All |
| `cpu (% of request)` | 2 ± 7 | 12 ± 25 | 4 ± 12 |
| `rss (% of request)` | 52 ± 25 | 37 ± 26 | 50 ± 25 |
| `read (kB/s)` | 8 ± 721 | 1 ± 200 | 7 ± 670 |
| `write (kB/s)` | 53 ± 865 | 22 ± 1018 | 50 ± 890 |
| `ics (/s)` | 6 ± 9 | 1 ± 4 | 5 ± 8 |

Table 3.4: VMs consumption: clients vs. internal

## 3.7 Comparative Study With Other Traces

In this section we compare Outscale's workload with those of Azure and Bitrbrains. The Azure and Bitbrains traces were presented in Section 2.5.1. Similarly to Outscale, Azure and Bitbrains provide a public VM-based service. Whereas Outscale operates a mid-sized platform and provides only generic images, Azure has an hyperscale platform and some of its images are specialized. Specialized images allow the provider to know the applicative role of a VM. On another hand, Bitbrains is specialized in hosting business-critical applications in long-running VMs, contrary to cloud VMs that are mostly short-running.

### 3.7.1 Comparison With Azure

We use a public sample of the Azure trace characterized in [2] to compare the start time of VMs, the number of vCPU and amount of RAM they request, and the amount of CPU they effectively use.

In Figure 3.11, we plot the average number of VMs started per hour over a weekly basis. Since the start time of the Azure trace is unknown, we cannot correlate daily patterns between Outscale and Azure. However, we observe that both workloads have a daily periodicity. With 240 VMs/h on average, the scheduling workload of Outscale is 11 times lower than at Azure (2600 VMs/h). Outscale's peak load (890 VMs/h) represents 4 times the average load, whereas Azure's peak load (4400 VMs/h) is two times its average one.

Figure 3.12a shows the percentage of VMs according to the number of vCPU requested. Both providers have a majority of VMs ($\sim$ 75%) asking 1 or 2 vCPU, and VMs bigger than 8 vCPU are unusual ($\leq$ 5%). Outscale clients request twice as many VMs with 4 vCPU than Azure, whereas VMs with 8 vCPU are requested three times more often at Azure (15%) than Outscale (5%). Consequently,

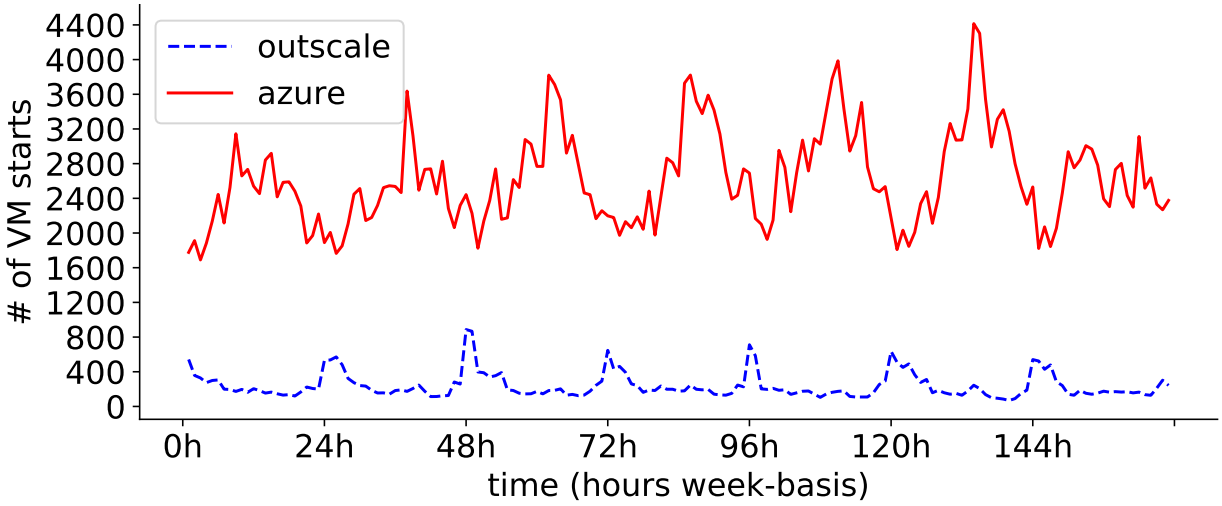
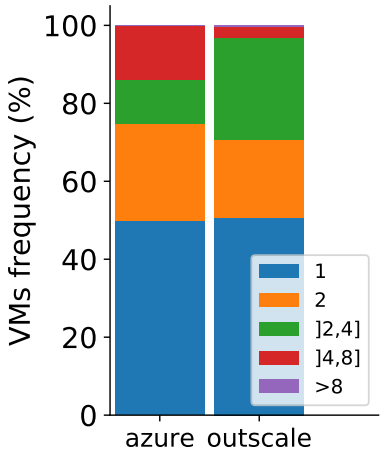


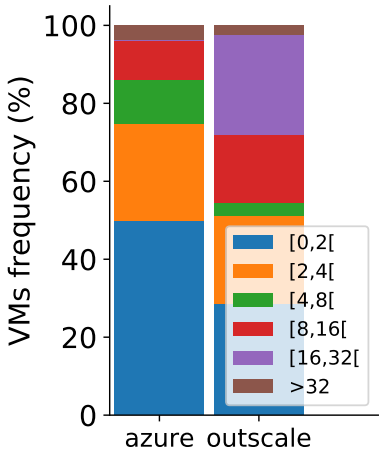

Figure 3.11: Start time

Outscale VMs request 13% less vCPU than at Azure.

For RAM (Figure 3.12b), we observe more notable differences between providers than with the distributions of vCPU. For instance, VMs with 4-8 GB are not very popular at Outscale ($\leq 5\%$); but represent 15% of Azure's workload. On the other hand, VMs with 16-32 GB are more frequent at Outscale (30%) than Azure ($\leq 5\%$). On average, VMs at Outscale request 16% more RAM that at Azure.




(a) vCPU requested




(b) RAM requested

Next, we study how the VMs in both clouds make use of the CPU they requested. Figure 3.13 shows the distribution of mean CPU utilization (%) of VMs at Outscale and Azure. The CPU utilization of VMs at Outscale is skewed, and 90% of VMs have a mean utilization lower than 10%. However, this is the case for 40% of VMs at Azure. In summary, Azure's VMs consume more than Outscale's (10% of Azure's VMs use more than 50% of their CPU).

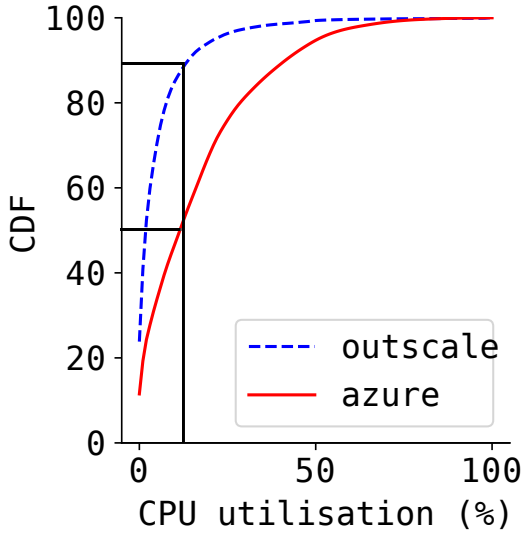



Figure 3.13: CPU usage

### 3.7.2 Comparison With Bitbrains

The comparison of Outscale's and Bitbrains' workloads is based on a prior characterization of the Bitbrains trace [15]. In Table 3.5, we compare the two workloads in terms of CPU and RAM requested per VM, as well as the effective utilization of disk. The main difference lies in disk utilization: whereas the average read throughput is three times larger than the write throughput at Bitbrains, it is 15 times smaller at Outscale.

**Consequence 4** *In this section, we showed that Outscale must handle a workload that is substantially different from Azure and Bitbrains.* ***Since the performance of a resource allocation algorithm depends on the characteristics of the workload, we need to perform complementary evaluations of state of the art algorithms, to evaluate if their performance remains acceptable with Outscale's workload.***

| Metric | Traces | Mean | Min | Q1 | Median | Q3 | Max |
|---|---|---|---|---|---|---|---|
| Cores req. | Bitbrains | 3.3 | 1 | 1 | 2 | 4 | 32 |
| | Outscale | 2.3 | 1 | 1 | 1 | 4 | 64 |
| RAM req. (GB) | Bitbrains | 10.7 | 0.0 | 1.27 | 3.98 | 15.59 | 511 |
| | Outscale | 7.5 | 0.5 | 0.6 | 3.75 | 16 | 1440 |
| read_disk (MB/s) | Bitbrains | 0.3 | 0.00 | 0.00 | 0.00 | 0.00 | 1411 |
| | Outscale | 0.39 | 0.00 | 0.01 | 0.07 | 0.38 | 980 |
| write_disk (MB/s) | Bitbrains | 0.1 | 0.00 | 0.00 | 0.00 | 0.01 | 188 |
| | Outscale | **1.58** | 0.00 | 0.03 | 0.19 | **1.90** | 547 |

Table 3.5: Statistics of resource consumption for Bitbrains and Outscale

## 3.8 Conclusions

Characterizing cloud workloads is required to optimize resource management processes such as capacity planning, resource allocation and performance control. Based on a three-month-long workload trace collected on Outscale's non-hyperscale public IaaS cloud, we characterized the utilization of virtual resources (VMs, images, volumes, snapshots and security groups), the utilization of hardware resources (CPU, memory and disk), and the CPU interferences. We discussed the implications of skewed utilization patterns on resource management. Only 10% of volumes live more than 5 days, but they have a larger footprint on storage backends because they fill up progressively and are snapshotted recurrently. Periodic snapshot creations bursts require to design a scalable orchestrator that can perform this operation transparently for users. 70% of VMs never use more than 20% of their requested CPU. However, even a reasonable overcommit ratio (often less than 3) generates involuntary context switches, especially for the smallest VMs. We doubt of the benefits of migrations to consolidate VMs with compatible workloads and separate incompatible VMs, because 90% of them run less than 1h. Moreover, we showed that the characterized workload was different from the ones observed on Azure, and Bitbrains, two providers with comparable VM-based services. This calls for the design of workload-specific orchestrators. A major feature of Outscale's workload is the high virtual resource churn, which indicates users automatize their deployments. We argue that automated deployments are repetitive, and therefore predictable. In the next chapters, we will seek to find if a machine learning system can predict the runtime of a VM with sufficient accuracy to be used by a VM placement algorithm.

# Chapter 4

# Prediction of VM runtime

**Contenu**

## 4.1 Introduction

As seen in the previous chapter, one of the main characteristics of Outscale's workload is the heterogeneity of VM lifetime. Previous works from the literature argued for the need to choose the placement of a VM based on its runtime [61, 62, 24]. Yet, VM runtime is unknown when the orchestrator makes this choice. In this chapter, we use supervised machine learning to mitigate this uncertainty. Supervised machine learning aims at making predictions about a phenomenon, based on past observations. In this use case, we predict the runtime of VMs based on information that is, or could be made available, when the VMs start, and based on the knowledge of past VM runtimes. We present a feature engineering and modeling pipeline to predict the runtime of VMs. The proposed features capture a variety of concepts. The novelty of our work is to leverage new parameters from the VM creation request, and text tags, which are freely-typed text strings used to describe VMs for their inventory. The combination of the proposed features with third-party features from the literature allow our model to outperform previous works.

This chapter is structured as follows. Section 4.2 gives an overview of supervised machine learning. Our experimental setup is presented in Section 4.3. Section 4.4 presents the implementation, and results are presented in Section 4.5.

## 4.2 Supervised Machine Learning

Machine learning explores the design of algorithms that learn how to perform a task from experience instead of being explicitly programmed [75]. The task is embodied by an unknown function. The goal is to find the best possible approximation in a large set of *hypothesis functions*. The quality of the approximation is computed from the experience given as numeric data, and from the *cost function* of choice. In summary, machine learning is a combination of three components [113]:

— A representation of the space of possible hypothesis functions.

— A cost function required to distinguish good hypothesis functions from bad ones, given some experience data.

— An optimization method to search the space of hypothesis functions.

In this Chapter, the task to be solved is the prediction of the runtime of VMs based on features available when they start. The problem is an instance of *supervised* machine learning, because both the

input and the expected output of the hypothesis function are provided as experience. The experience is therefore encoded with paired variables $(\mathbf{x}, \mathbf{y})$; where the runtime $\mathbf{y}$ is the expected output of the hypothesis function evaluated on features $\mathbf{x}$. Since the experience data includes runtime measurements, it is generated from VMs whose execution is finished. Hence, the underlying assumption is that historical patterns of cloud utilization can be used to estimate the runtime of new VMs.

Supervised machine learning problems are categorized according to the nature of the predicted variable $\mathbf{y}$. Binary classification stands for when $\mathbf{y}$ can only take two values, multiclass classification stands for when $\mathbf{x}$ can take one value of a set, multilabel classification stands from when $\mathbf{x}$ takes several values simultaneously, and regression stands for when $\mathbf{x}$ takes real values. Most algorithms handle regression and classification problems with minor variations. In the rest of this chapter, we will focus on classification.

In the next section, we present decisions trees, one of the most widely used class of supervised machine learning algorithms.

### 4.2.1 Decision Trees

In decision trees, the hypothesis function is a series of tests performed on the input. The successive tests and their outcome are represented by a downward tree, as shown in Figure 4.1. The nodes of the tree are therefore partitions of the data space. The leaves are the most elementary partitions. The observations that fall in a leaf are used to compute the output. The output is either a class prediction, or class probability estimates. The first advantage of decision trees is that they are non-parametric, i.e., they do not make any assumption about the data. They can also handle heterogeneous data composed of ordered and categorical variables.

Building the tree is a greedy optimization. At every node, the algorithm chooses the test that best discriminates observations with respect to the predicted variable. Hence, the features used in the first tests (closest to the root) are the most informative. The conferred interpretability is another advantage of decision trees.

For the sake of simplicity, we will consider the most common architecture for decision trees, where a test applies to a single feature and has a binary outcome. In alternative architectures, tests apply to a linear combination of features [114] and have non-binary outcomes [115, 116]. The interested reader

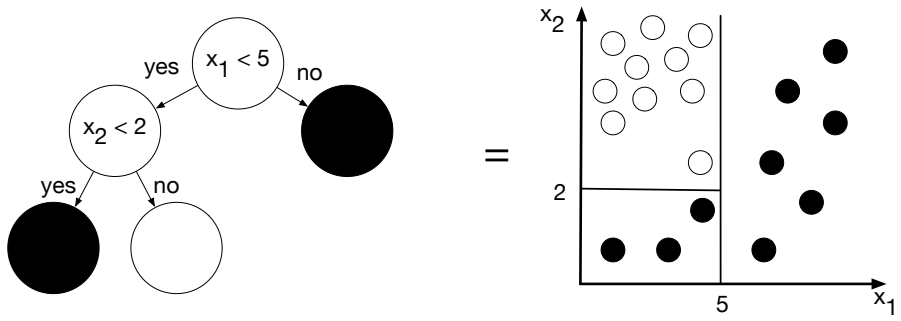



Figure 4.1: A decision tree (left) and the resulting partitioning of the data space (right) on a classification problem.

can find a survey of decision trees in [117].

Formally, let $\theta = (f, r)$ be a criteria that allows to split the set of observations in node $S$ into two children, $S^r$ and $S^l$, by testing if the value of the feature $f$ is in range $r$. The algorithm searches the most discriminant criteria. The idea is to choose the criteria that leads to the largest decrease of the *impurity* in the child nodes compared to the parent node (Equations 4.1 and 4.2). The impurity of a child node is minimal (and the impurity decrease is maximal) when the child node contains homogeneous observations. The formula of cross-entropy, the most popular impurity function for classification, is given in Equation 4.3 for $K$ classes.

$$\theta^\star = \arg\max_\theta \Delta I_\theta(S) \tag{4.1}$$

$$\Delta I_\theta(S) = I(S) - \frac{1}{|S|}(|S_r|I(S_r) + |S_l|I(S_l)) \tag{4.2}$$

$$I(S) = -\sum_{k=1}^{K} P[y = k|S]log(P[y = k|S]) \tag{4.3}$$

Decision trees are interpretable because they allow to rank the contribution of individual features. Node importance $\Delta I'$ is defined as the decrease in node impurity weighted by the fraction of samples reaching that node (Equation 4.4). The importance $F_f$ of a feature $f$ is the sum of the importance of nodes that split on that feature, normalized by the total node importance (Equation 4.5).

$$\Delta I'(S) = \frac{|S|}{N}\Delta I(S) = \frac{|S|}{N}I(S) - \frac{1}{N}(|S_r|I(S_r) + |S_l|I(S_l)) \tag{4.4}$$

$$F_f = \frac{\sum_{S \text{ splits on } f} \Delta I'(S)}{\sum_S \Delta I'(S)} \tag{4.5}$$

Depth is an important parameter of decision trees. There is a risk of underfitting with small trees. They may fail to model even the training data. On the other hand, there is a risk of overfitting with large trees: they may manage to model the training data very well, but fail to generalize on new data because they have isolated small subspaces and are sensitive to noise. One solution to find the best tradeoff is tree pruning. Pruning involves the utilization of a cross-validation dataset to test the model on new data. There are two kinds of pruning methods. Pre-prunning consists in building the tree until the cross-validation error stops decreasing as expected after a node split. With the post-pruning method, the tree is fully grown, then leaves are merged until the cross-validation error stops decreasing.

Another approach to optimize the tradeoff between under and overfitting is to use ensemble methods.

### 4.2.2 Ensemble Methods

The goal of ensemble learning is to exploit the strengths of diverse models and mitigate individual weaknesses [118].

There are two approaches to build ensemble models.
*Bagging* trains each model on a random subset of the data. Therefore, it provides implicit diversity. Random forest train decision trees on random sets of samples drawn with replacement, and consider node splitting on random subsets of features [119]. Extremely randomized trees go even further, as they choose node splits based on random thresholds [120]. On the other hand, *Boosting* algorithms explicitly control diversity, and ensure individual models are different with active measurements. The most popular, Adaboost, trains models sequentially on a subset of samples drawn such that samples that were previously misclassified have a higher probability to be included [121].
Bagging methods work best with complex models (e.g., fully grown decision trees), in contrast with boosting methods which usually work best with weak models (e.g., shallow decision trees). Moreover, bagging methods have a computational advantage over boosting methods, because they allow to train models in parallel on random subsets of the data that fit in memory.

There are two methods for combining model outputs. A linear combination of outputs is done

when class probability estimates are expected, otherwise a majority vote for the most likely class is used.

### 4.2.3 Evaluation Metrics for Classification

In this section, we present the most common metrics used to evaluate classification models.

#### 4.2.3.1 Accuracy

Accuracy is the fraction of correct predictions (Equation 4.6). The shortcoming of accuracy is when class cardinality is unbalanced. For, instance, if 90% of samples are in a class, a naive model that always predict that class will have a 90% accuracy.

$$accuracy = \frac{1}{n_{samples}} \sum_{i=1}^{n_{samples}} 1(\hat{y}_i = y_i) \tag{4.6}$$

#### 4.2.3.2 F1 score

The F1 score (Equation 4.7) is a popular classification metric when the problem involves unbalanced classes. The F1 score is the harmonic mean of precision $P$ and recall $R$. Precision is the fraction of samples in a class X that are actually classified as belonging to X (Equation 4.8). Recall is the fraction of samples classified as belonging to X that are actually in X (Equation 4.9). Precision and recall are calculated from the confusion matrix which holds the fraction of true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN), shown in Table 4.1. The combination of precision and recall makes the F1 score suited for problems with unbalanced classes, because a naive model that always predicts the majority class will either obtain good precision and bad recall, or the opposite, but cannot do well on both. And since the harmonic mean used in the computation is more conservative than the arithmetic or geometric means, the F1 score will be closer to the lowest value, and will show that the model is in fact not able to discriminate the classes. When the problem involves $C > 2$ classes, precision and recall are first computed for each class, and then averaged. The *macro-averaged* precision is given in Equation 4.10. Class imbalance is not taken into account, so that classes with relatively few samples contribute as much as others to the average score.

$$F1 = 2\frac{P * R}{P + R} \tag{4.7}$$

$$P = \frac{TP}{TP + FP} \tag{4.8}$$

$$R = \frac{TP}{TP + FN} \tag{4.9}$$

$$\text{if C} > 2, P = \frac{1}{C}\sum_{c=1}^{C} P_c \text{ (same for R)} \tag{4.10}$$

| | | Observation | |
|---|---|---|---|
| | | positive | negative |
| Prediction | positive | true positive (TP) | false positive (FP) |
| | negative | false negative (FN) | true negative (TN) |

Table 4.1: Confusion matrix

### 4.2.4 Feature Encoding and Scaling

Learning machines need numeric data to work with. In this section, we describe methods to encode different sources of data, from sensors to human-generated text, into features acceptable by a machine.

Physical quantities measured by sensors, are naturally expressed with numbers. When the algorithm is based on distance computations, it is recommended to scale features so that each of them contributes equally to computations. The main scaling method, *standardization*, gives each feature component $x$ a mean of 0 and a standard deviation of 1 (Equation 4.11).

$$x' = \frac{x - \mu}{\sigma} \tag{4.11}$$

Categorical features can take one value from a set. Unlike numeric features, the set cannot be ordered. When the set cardinality is greater than two, the most popular encoding is *dummy encoding*, i.e., the decomposition into a set of binary components. Table 4.2 shows the dummy encoding of a categorical feature such as a color.

Text is unstructured, so text processing is more complex than for structured data. In order to use text data, it is necessary to encode variable-length documents into fixed size feature vectors. This process is called *text vectorization*. Text vectorization consists in the following steps [122]:

1. pre-processing: letters are written in lower-case, words are reduced to stems.

| Gender | component 1 | component 2 | component 3 |
|---|---|---|---|
| green | 1 | 0 | 0 |
| yellow | 0 | 1 | 0 |
| red | 0 | 0 | 1 |

Table 4.2: Dummy encoding of a categorical feature (color) into binary components

2. tokenizing: each stem in the corpus is given a token ID, using spaces and punctuation as token separators.
3. counting: the occurrence of tokens is counted in each document.
4. normalizing: counts are normalized by the total document word count. Optionally, the result is weighted to diminish the importance of words that occur frequently in the set of documents. That is because very frequent words, like "the", do not bring any specific information.

## 4.3 Experimental Setup For Predicting VM Runtime

In this section, we present our approach to predict the runtime of VMs. Given the wide range of runtimes ($[0s, \infty]$), predicting an approximate value with a classification algorithm is sufficient for the purpose of VM placement. Besides, the impact of prediction error depends on the actual VM runtime: an over-prediction of one day for a VM that runs one hour is relatively larger than for a VM that runs one month. To reflect the decreasing impact of absolute prediction error as runtime increases, we define classes with increasingly large runtime ranges, as shown in Table 4.3. We set the class boundaries such that they are easy for humans to work with, and they allow to compare our results with an existing work [2].

| class name | S | M | L | XL |
|---|---|---|---|---|
| runtime range | $[0, 15min]$ | $]15min, 1h]$ | $[1h, 24h]$ | +24h |
| class cardinality (% of total) | 52 | 21 | 17 | 10 |

Table 4.3: Runtime class definition

In a previous work [2], Cortez et al. predicted the runtime of VMs with an ensemble of decision trees. The authors reported the prediction results, but did not explain precisely which features they used nor how they trained the model. Yet, the search of the best features and algorithms is critical for the development of VM placement solutions based on machine learning. For the feature search, our contribution lies in the utilization of novel features extracted from the API call for the VM creation,

and features extracted from text tags, which are used to describe VMs. For the learning algorithm, we focus on Extremely Randomized Trees [120], a bagging ensemble method based on decision trees and presented in Section 4.2.2 We anticipate that unbalance class cardinality (Table 4.3) will be problematic for Extremely Randomized Trees: since a tree is fitted on a random subset of samples, the majority class is given more importance. To alleviate class unbalance, we will use adapted training methods from the state of the art and we will propose a new approach. In the following paragraphs, we detail the proposed approaches for the search of features and training methods robust to class unbalance.

### 4.3.1 Feature Search

To predict the runtime of VMs, we are looking for features that are available when the VMs start, or could be made so with minimal impact on the user experience. Our work builds upon previous works in grid and cloud platforms [2, 123]. We propose three feature sets with an increasing number of features. In the first set, we include features extracted from the API call of the VM request. In the second feature set, we add features presented in previous works, that synthesize the user account state and history. In the third set, we include novel features extracted from text tags. The next three sections detail the content of feature sets, and a summary is given in Table 4.4.

#### 4.3.1.1 Features from the API call of the VM request

The most straightforward features are extracted from the API call. We class the features in six groups, the first two groups were used in previous works from the literature, the next four are proposed by us. The *resource request* group describes the amount of resources requested (vCPU, RAM), the VM type and the system OS [2]. The *time* group describes the timing of the API call to the orchestrator [123]. We extend these well-known features with the following four groups. *Placement affinity* describes user inputs regarding the choice of server. For instance, to guarantee high availability of an application, the user may ask the orchestrator to put its new VM on a different server than an existing VM. *Network* features describe networking setup for the VM, such as the number of security groups the VM belongs to. *Ephemeral storage* describes the amount of storage that will not persist if the VM stops. We did not use features related to persistent storage, because it is provisioned after the VM is started. Hence, this information cannot be used to make predictions at startup. Finally, we include *miscellaneous* user inputs, such as the number of VMs requested simultaneously through

a single API call.

#### 4.3.1.2 Features from the account state and history

Previous works from the literature show that current and historic user account state can be used for job runtime prediction in grids. Tsafrir et al. estimate the runtime of a job by simply averaging the last two jobs of the user [123]. Gaussier et al. propose an extended set of historic features including the average resource request and average runtime of all previous jobs [86]. In addition, they also synthesize the current account state with features like the sum of the runtime of currently running jobs. The only feature that cannot be transposed from grids to clouds is the maximum allowed time slot before killing the job. This comes from the fact that grids schedulers aim a fair resource allocation (allocating equal resources to users) whereas clouds allow users to run VMs for as long as long as they pay.

#### 4.3.1.3 Features extracted from text tags

We propose to generate features from text tags. Text tags are strings of text that users attach to VMs to simplify their inventory. We have inspected tags and observed that some of them provide rich information on the context of VM deployments. For instance, consider the following set of tags attached to three VMs:

```
   DEPLOYMENT=MY_APPLICATION VM=WEB_SERVER
DEPLOYMENT=MY_APPLICATION VM=DATABASE
DEPLOYMENT=MY_APPLICATION VM=LOAD_BALANCER
```

The three VMs belong to the same web application deployment. Within a deployment, the `VM` tag refers to the service name (web server, database or load balancer).

Currently, Outscale's API does not allow users to pass tags as parameters to the VM creation call. Users must first request a VM, then tag it with a subsequent call. We will measure the contribution of tags to the prediction accuracy, and assess if it is worth modifying the API to allow their utilization for runtime prediction at startup. To extract features from text tags, we performed the text vectorization technique presented in Section 4.2.4.

Table 4.4 summarizes the composition of feature sets. In addition to the 25 features used in

previous works, we will use 14 features from the API call and more than 100 from text tags.

| group name | # | summary of features | origin |
|---|---|---|---|
| API call: resource request | 4 | amount of CPU & RAM requested<br>VM type<br>OS | [2] |
| API call: time | 9 | abolute components: day, hour, minute<br>sinusoidal components | [123] |
| API call: server affinity | 4 | tenancy (shared or dedicated server)<br>attract/repulse | ours |
| API call: network | 5 | number of security groups or bypass option<br>membership to a virtual private cloud (VPC)<br>attachment of elastic public IP(s) | ours |
| API call: ephemeral storage | 1 | total size of ephemeral storage | ours |
| API call: miscellaneous | 4 | min and max number of VMs requested in the call<br>VM shutdown policy | ours |
| account state and history | 12 | individual and averaged runtimes of the last 3 VMs<br>average runtime and CPU request of all finished VMs<br>maximum and sum of runtimes among running VMs<br># of VMs running and sum of their CPU requests<br>time elapsed since the last VM stop | [123] |
| text tags | 100+ | freely-typed text describing VMs<br>and their associated resources | ours |

Table 4.4: Composition of the feature sets

### 4.3.2 Dealing With Unbalanced Classes

We implemented three approaches to deal with unbalanced classes. The first two approaches are based on re-sampling the training data. In the first approach, we under-sample the majority class to the cardinality of the second largest class. In the second approach, we over-sample the two

minority classes by duplicating observations. The third approach, adapted from [124], is to change the algorithm and split the multiclass classification task into several binary classification ones. As shown on Figure 4.2, we propose a cascade of binary classifiers, where the output of a classifier is either the final output, or a call to the next classifier in the cascade.

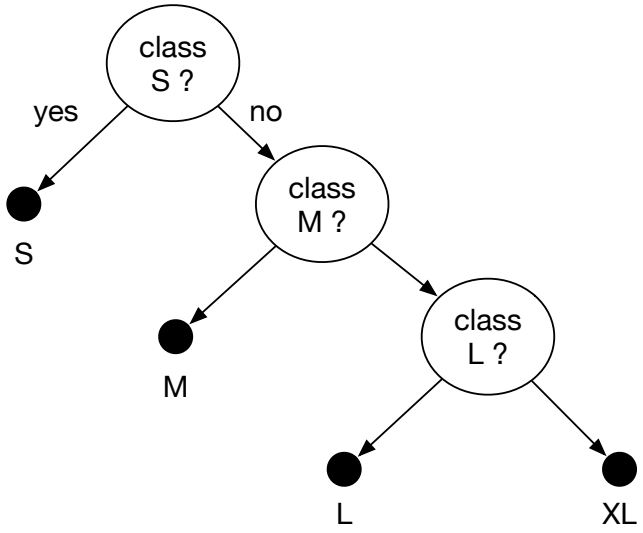



Figure 4.2: Proposed cascade classifier

## 4.4 Implementation

For the implementation, we used scikit-learn [125]. It is the standard python library for data mining and machine learning. Scikit-learn includes feature processing procedures, learning algorithms, and other methods to automatize model evaluation and selection. In this section, we present scikit-learn, identify some of its limitations, and introduce our proposed extension.

### 4.4.1 Presentation of Scikit-Learn

Scikit-learn offers three main classes:

— A *Transformer* takes some input data, estimates some parameters, and transforms the input based on the estimated parameters. For instance, a standardizer takes some input data $X$, estimates $\mu$ and $\sigma$, and returns a transformed, standardized version of $X$.

— A *Predictor* predicts the output data $y$ based on the input $X$ and some estimated parameters. For instance, a decision tree is implemented as a predictor.

— A *Pipeline* is a chain of transformers with a final predictor. Pipelines are convenient because they offer a single interface to assemble, train and test complex models.

### 4.4.2 Limitations of Scikit-Learn

Scikit-learn has the two following limitations:

— Transformers are applied on all columns of the input $X$. As we are working with heterogeneous data (numeric and categorical variables, or even text variables), we want to apply different transformers to subsets of features.

— Some transformers, e.g. the dummy encoder or text vectorizer, explode a feature column into multiple ones. Scikit-learn loses the mapping between the original features and the final ones. As we want to interpret the contribution of features to the learning task, we need the pipeline to carry the meaning of the final features.

Ibex [1] and sklearn-pandas [2] are two libraries that seek to tackle these limitations. However, Ibex does not allow to apply different transforms to subsets of features, and sklearn-pandas did not output feature names until 2017-05-13 (after we started the implementation). Therefore, we implemented a custom extension to scikit-learn.

### 4.4.3 Implementation of our Scikit-Learn Extension

In order to apply specific transformers according to the feature type, and to carry feature names through the entire pipeline, we implemented the following classes:

— *ColumnExtractor* selects the features to be passed to the next transformer in a pipeline. It allows to select the transform applied to a subset of features.

— *FeatureUnion* applies a list of transforms on the input and merges results column-wise. Each transform in the list operates on a subset of features.

— Standardizer, Text vectorizer, or Dummy encoder are wrappers over the native scikit-learn classes with the same names. We implemented wrappers to work with Pandas [3] data frames instead of NumPy [4] matrices, which enables to carry feature names down the pipeline.

1. https://github.com/atavory/ibex
2. https://github.com/scikit-learn-contrib/sklearn-pandas
3. Pandas is a package for data analysis in Python
4. NumPy is the fundamental package for scientific computing with Python

— *ColumnMemorizer* memorizes column names as seen by the predictor, once all transforms have been applied. This class is used to determine the most important features.

In addition, we implemented the *CascadeClassifier* class following the *Predictor* abstraction to mitigate class unbalance.

The resulting pipeline is shown on Figure 4.3. For the experiments, we separated the dataset into a training and a test set. The training set was composed of 80% of samples drawn at random, the test set was composed of the remaining 20%.

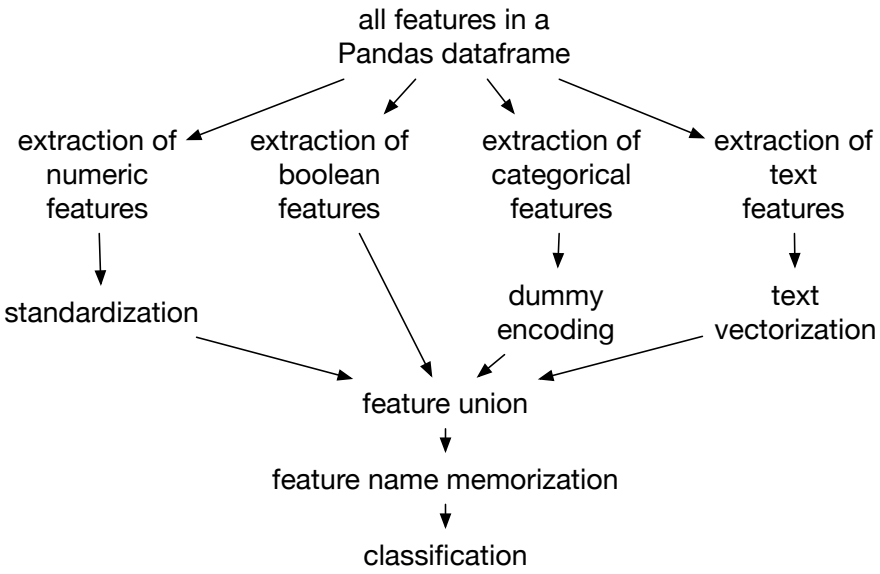



Figure 4.3: Implementation of the proposed feature processing and classification pipeline

## 4.5 Results

In this section, we present the results obtained on the classification of VM runtime. We evaluate the gain in prediction accuracy resulting from the utilization of features extracted from the API call, the account state and history, and text tags. We compare our results, obtained on Outscale's workload trace, with the works of Cortez et al. [2]. In this previous work, the authors predicted the runtime of VMs based on the Azure dataset. We choose this baseline because the runtime of VMs has no upper bound on both platforms, and the same problem formulation is made (four-class classification).

### 4.5.1 Importance of the feature set on model performance

First, we evaluate the contribution of the three feature sets on the model performance. Figure 7.3 presents the classification performance, as measured by the F1 score, with the incremental addition of features. The first model uses the features from the API call of the VM request and reaches an F1 score of 0.76, which is 0.01 better than the baseline pictured by the black horizontal line. In the second model, we added features that synthesize the current and historic user context. The performance improves by 0.11 (F1=0.87). In the third model, we added the tag features and the performance improves by 0.04 (F1 = 0.91). The results show that features extracted from the account state and tags are useful for VM runtime prediction.

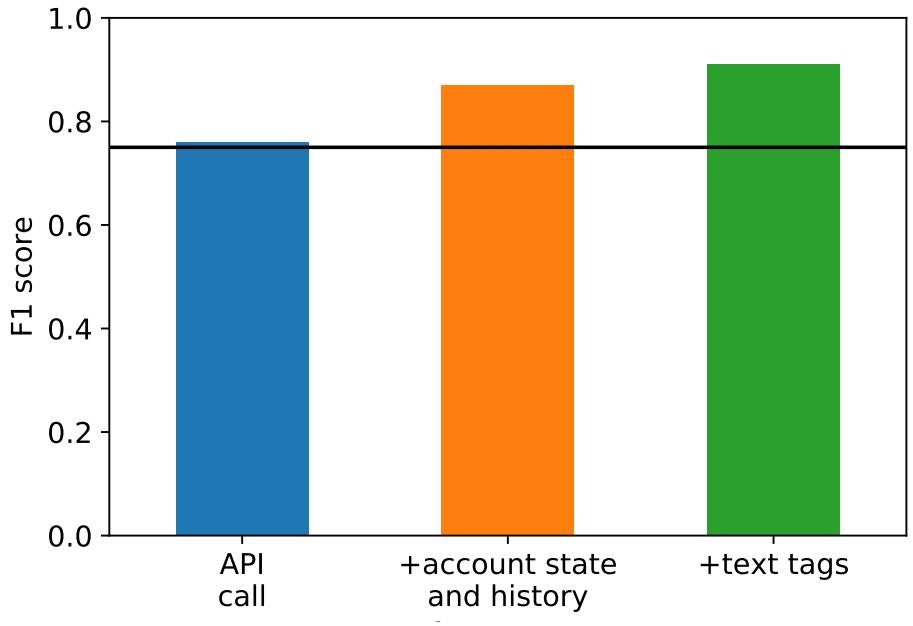



Figure 4.4: Model performance with the incremental addition of features. The horizontal line corresponds to the baseline on the Azure workload [2].

Next, we determine which individual features bring the most valuable information. Figure 7.4 shows the ranking of the top twenty features, by importance. To respect privacy, tag features are indexed by increasing importance instead of being named. There is a tie for the first place between the number of security groups associated with the VM and a tag word. This shows, again, that the proposed networking and tag feature sets are valuable. The next three features relate to the user

context, including the number of VMs running and the total amount of CPU and RAM requested. Then, 4 of the next 5 most important features describe the timing of the VM request. Among the top 20 features, 11 are related to the user account state and history, 6 describe the API call timing and security group membership, and 3 are text tags. We observe that the most straightforward features, like the CPU and RAM requested by a VM, do not belong to this set. This result demonstrates the need to use a mix of complex features to obtain accurate predictions.

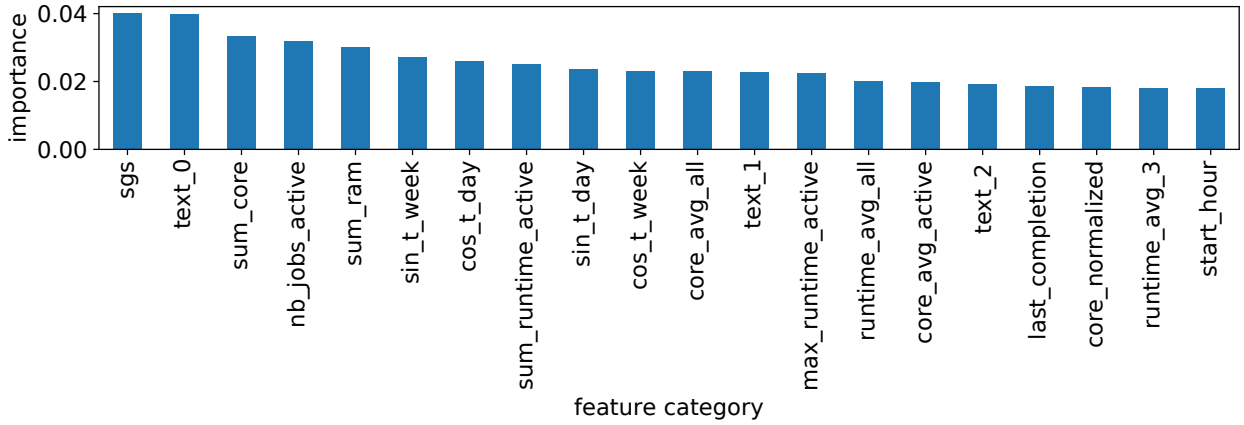



Figure 4.5: The twenty most important features

One always prefers models with relatively few features, because they are easier to train and understand. Figure 4.6 shows the evolution of prediction performance with respect to the number of text features included in the dataset. The features are ranked by occurrence of the corresponding word in the dataset, the most frequent words are included first. The performance quickly increases from 0 to 100 features, and then reaches a plateau. Hence, based on our data, it is sufficient to use the top 100 most frequent words as features. Using more words just slows down the training process.

### 4.5.2 Comparison of learning methods

To make accurate predictions, it is necessary to find the method that works best on the data. The performance of a model is determined by several parameters. For Extreme Randomized Trees, our chosen algorithm, performance is a function of the sampling of the training data, the number of trees in the ensemble, and tree depth. In this section, we present the results of the search for the optimal learning method.

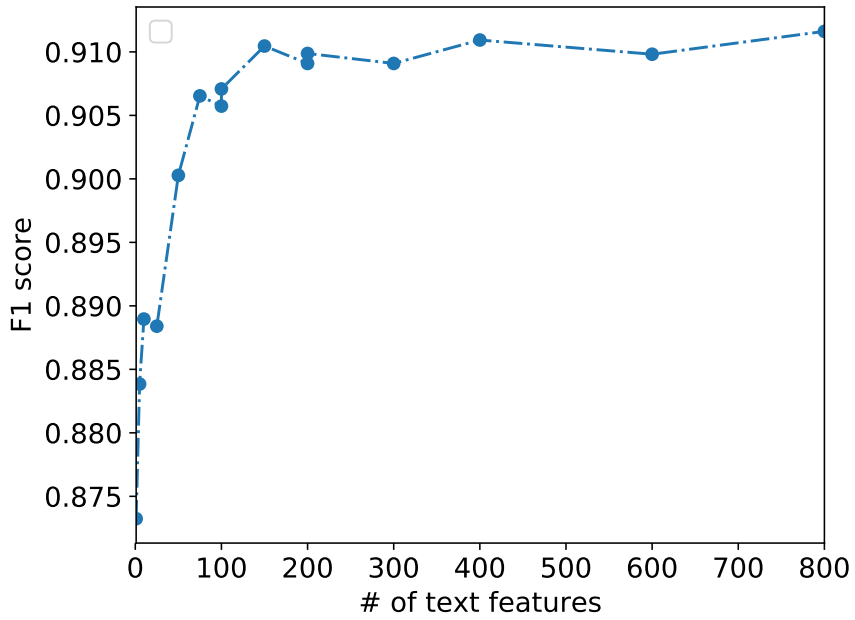



Figure 4.6: Evolution of prediction performance with the number of tag features included in the dataset

We anticipated class unbalance to be the main obstacle to learning. Figure 4.7 illustrates the relationship between prediction performance per class and class cardinality, for an ensemble of ten trees of depth ten. For the S, M and L classes, the per-class F1 score is correlated with class cardinality: the smaller the class, the smaller the F1 score. As a result, the averaged F1 score reaches 0.65, which is less than the 0.91 score of the best-performing model. Correlation does not imply causality. However, class unbalance is a well-known problem in machine learning [126], and it manifests with the observed correlation. Following the hypothesis that our model was suffering from class unbalance, we tested alternative learning methods.

To tackle the effect of class unbalance, we modify the learning method at the data level and algorithmic level. At the data level, we under-sample the majority class, or over-sample the minority one. At the algorithmic level, we propose a cascade of extreme randomized trees. Figure 4.8 compares model performance for the different learning methods. Model performance is comparable for the default, under-sampling, and cascade methods (F1=0.65). Over-sampling achieves the worst performance (F1=0.54). This could be explained by the fact that information of the minority class is duplicated, making the algorithm prone to overfitting the data. Overall, none of the tested method

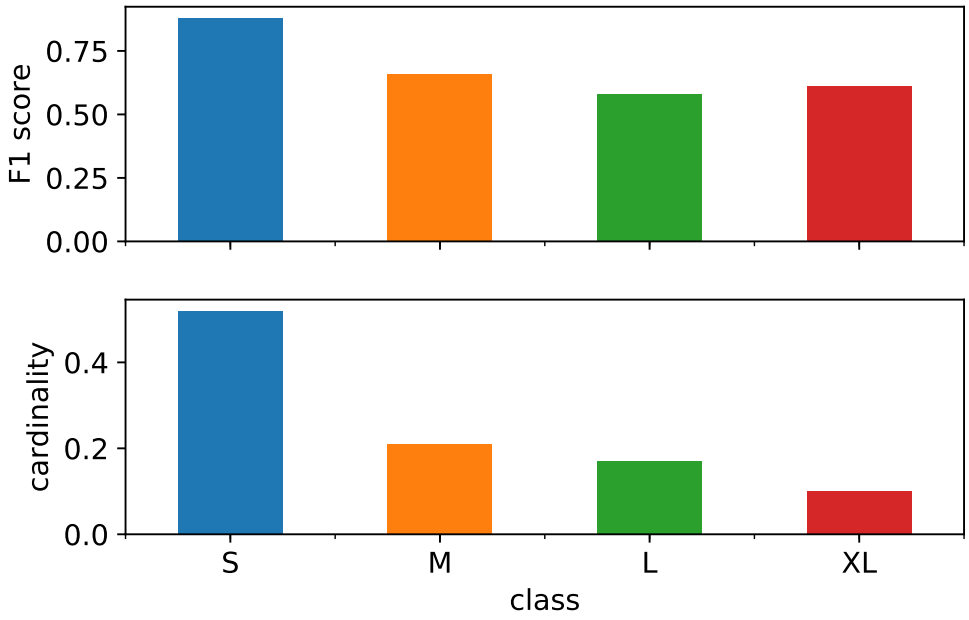



Figure 4.7: Relationship between model performance and class cardinality

brings a significant improvement over the default one. In light of these results, we re-considered our initial hypothesis. Could the poor results obtained on the minority classes be explained by the choice of hyperparameters, rather than skewed class cardinality?

The performance of a model depends on the choice of hyperparameters. For extremely randomized trees, the two hyperparameters are the number of trees in the ensemble, and the tree depth. Figure 4.9 shows the influence of the hyperparameters values on model performance. The most impactful parameter is tree depth. The F1 score increases with tree depth, until an asymptote is reached for a depth of 75 (F1=0.91). From our point of view, a depth of 75 is relatively large with respect to the number of variables, which is 190. This shows that in order to make good predictions, the algorithm has to either test a large number of binary variables, or test continuous variables repeatedly. The number of trees in the ensemble is less impactful than tree depth, but again, the F1 score increases with it (+0.03 from 5 trees to 75 trees in the ensemble). The use of multiple trees allows to have a complex model (deep trees), while preventing overfitting. The smaller sensitivity to the number of trees suggests that data is not noisy, and therefore the model is not subject to much overfitting.

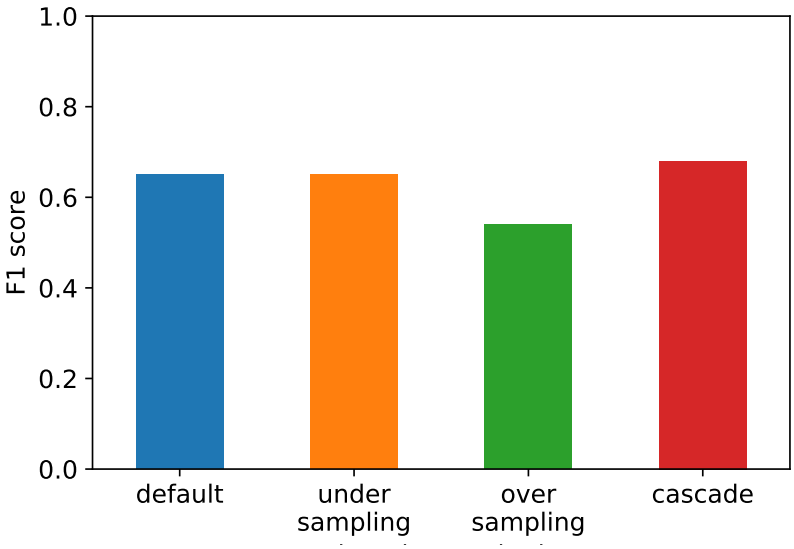



Figure 4.8: Relationship between model performance and learning method

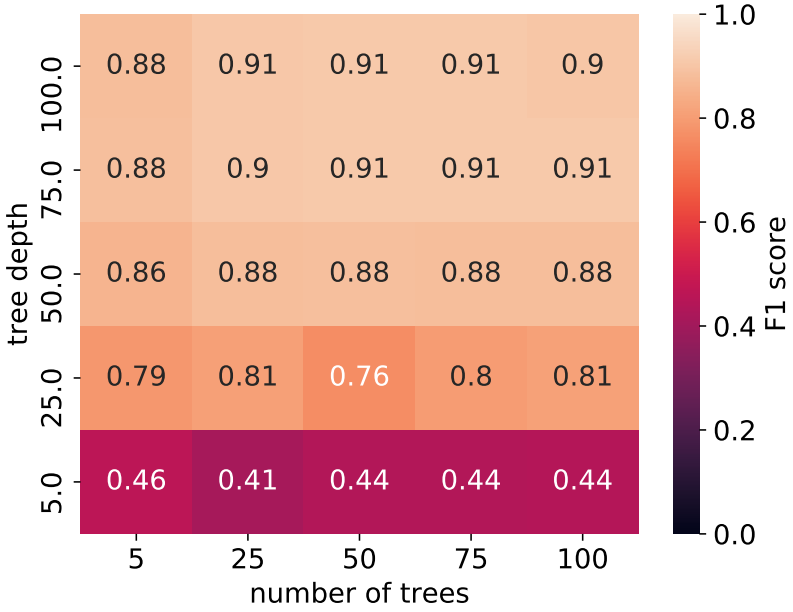



Figure 4.9: Hyperparameter sweep over tree depth and number of trees in the ensemble

## 4.6 Conclusion

The runtime of VMs is heterogeneous. Taking VM runtime into account for the placement of VMs could allow to optimize the management of resources. Yet, the runtime of a VM is unknown when it starts.

In this chapter, we proposed an approach to predict the runtime of VMs at startup, using supervised machine learning. Our contribution lies in the utilization of new features extracted from the parameters of the API call of VM creation, and from text tags. Tags are currently not available at VM start time because they are passed in a subsequent API call. Nonetheless, we evaluated their contribution to the prediction model, to assert the potential benefit of an API change allowing to capture them at VM start time. From the API call, we extracted novel features pertaining to the networking and storage configuration of the VM, the placement affinity and other miscellaneous inputs. In addition, we added features used in previous works that synthesize the user account state and history. With this extensive set of features, we obtained an F1 score of 0.91 (0.87 without tags) on a runtime classification problem with four classes. This is a significant improvement over the baseline model from the literature [2] reporting F1=0.75.

To achieve a positive result, we compared several learning algorithms based on extremely randomized trees. Initially, we observed that prediction accuracy of the minority classes was lower than for classes with many items. This observation is symptomatic of the class unbalance problem, therefore, we tested alternative approaches at the data and algorithmic level. At the data level, we under-sampled the majority classes, or over-sampled the minority classes. At the algorithmic level, we decomposed the four-class classification problem into a cascade of four binary classification problems. None of these approaches increased model performance. We then resorted to tune two hyperparameters, namely, the number of trees in the model, and tree depth. We observed that tree depth needed to be set higher than expected and found the best F1 score (0.91) for a depth of 75. Therefore, the symptom observed initially (low accuracy for the minority class) was a consequence of model underfitting, rather than class unbalance. Our work testifies that finding the optimal learning method and hyperparameters is challenging, especially when the maximum achievable performance is unknown. We hope that this work will be used by Outscale to optimize the placement of VMs based on predictions of their runtime. We also hope that the hands-on experience gathered will be useful to researchers working on similar

problems.

Although we proved that tags are helpful to make predictions of VM runtimes, we still do not know if it is absolutely necessary to change the API to acquire them at start time, because we do not know what prediction accuracy is required. In the next chapter, we will determine the accuracy required for the prediction system to be used for the placement of VMs.

# Chapter 5

# Sensitivity Evaluation of RTABF

**Contenu**

## 5.1 Introduction

In the previous chapter, we designed and evaluated a promising approach to predict the runtime of VMs at startup. In this chapter, we aim to determine the level of accuracy required for a prediction system to be used for the optimization of VM placement. In 2014, Dabbagh et al. proposed Release-Time Aware Best Fit (RTABF), a VM placement heuristic that minimizes the energy consumption of servers [24]. RTABF assumes a perfect knowledge of VM runtime on startup. The amount of energy saved by RTABF is unknown in a real-world setup, where predictions of runtime may not be accurate. While this work originally aimed at evaluating the suitability of integrating our prediction system with the RTABF algorithm, we were unable to reproduce the results obtained by Dabbagh et al. in their paper.

The remainder of this chapter is structured as follows: In Section 5.2, we detail two baseline VM placement algorithms as well as RTABF. In Section 5.3 we present our comparison methodology and experimental setup. We present results and conclusions in Sections 5.4 and 5.4.1, respectively.

## 5.2 Comparison of online VM placement algorithms

In this section, we present Any Fit (AF) and Best Fit (BF), the two most popular online VM placement algorithms [74, 41, 127], as well as Release-Time Aware Best Fit (RTABF). The three algorithms are used for the initial placement of new VMs, not for the consolidation of running VMs.

### 5.2.1 Any Fit

Any Fit (AF) simply chooses a random server among the set of servers with sufficient available resources to host the new VM.

### 5.2.2 Best Fit

Best Fit (BF) is the most popular VM placement heuristic. It chooses the server with the minimum - but sufficient - available resources.

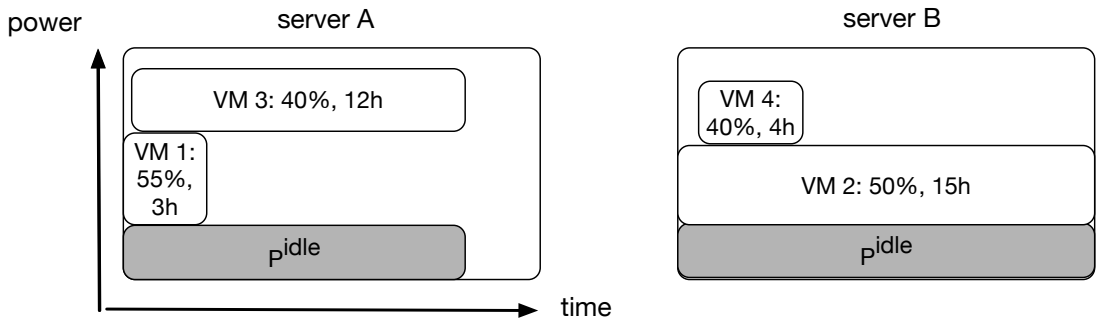
### 5.2.3 Release-Time Aware Best Fit

Release-Time Aware Best Fit (RTABF) was proposed by Dabbagh et al. [24] in 2014. RTABF extends Best Fit by taking into account the runtime of VMs. RTABF assumes the runtime of a VM is known when the start request is made. RTABF seeks to co-locate VMs that will stop simultaneously. The authors argue that this approach is more energy-efficient than BF, because servers can execute a given workload in less time than with BF. The time difference translates into saved energy because servers are powered off.

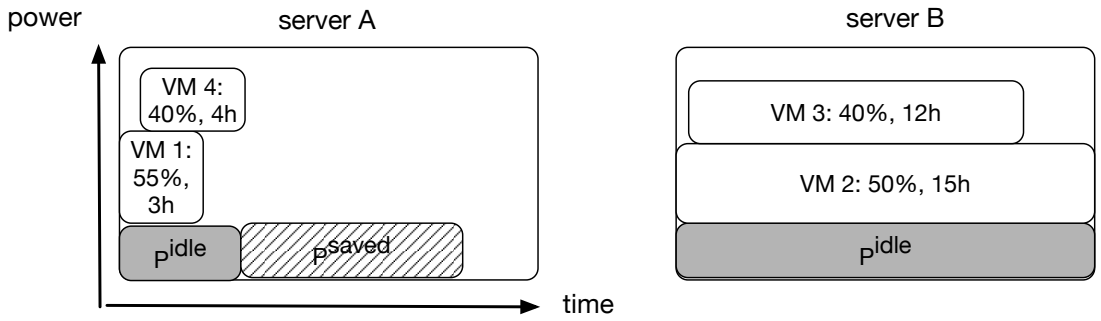
To illustrate the potential superiority of RTABF against BF, consider the case where two VMs are sequentially placed on two used servers. Server A already hosts VM 1 with 55% of CPU usage and 3h of remaining runtime. Server B already hosts VM2 with 50% of CPU usage and 15h of remaining runtime. VM 3 (40% CPU, 12h runtime) and VM 4 (40% CPU, 4h) are started in sequence. Best Fit (Figure 7.5a) places VM 3 on server A and then VM 4 on server B, based only on the CPU utilization. With BF, server A is used for 12h and server B is used for 15h. On the other hand, RTABF (Figure 7.5b) places VM 3 on server B because it avoids extending the utilization time on server A. Server A receives VM 4, and is turned off when VM 4 stops. The energy saved by RTABF with respect to BF is equal to the energy consumed by an idle server for 8 hours, because RTABF allows to turn off server A before BF.

RTABF chooses the best server for a VM based on two metrics. The temporal slack $\alpha$ measures the amount of resources available on the server. The difference with best fit is that instead of measuring resources in *cpu*, slack is measured in $cpu * seconds$. The rationale is that hosts with fewest available and sufficient slack should be chosen. The uptime extension $\beta$ is proportional to the additional time a host would have to stay on if it received a new VMs. The uptime extension should be minimized, to avoid keeping servers half used. The total cost $\gamma_{i,j}$ of placing VM $i$ on host $j$ is the sum of the temporal slack and uptime extension (Figure 5.2). It is computed as follows:

Let $s_i$ the start time of VM $i$, $r_i$ the predicted stop time (release time) of VM $i$, and $y_i$ the predicted runtime of VM $i$ (Equation 5.1). For two VMs $i$ and $k$ (with $k$ already running), let $y_k^{\prime i}$ be the estimate of the remaining runtime of VM $k$ bounded by the runtime of VM $i$ (Equation 5.2). $\delta_{i,j}$ is the estimated additional time host $j$ would have to stay ON if it was to receive VM $i$ (Equation 5.3). $\alpha_{i,j}$ is the temporal slack, with $C_j$ the cpu capacity of host $j$ and $c_k$ the cpu required by VM $k$ and




(a) Best Fit




(b) Release-Time Aware Best Fit

Figure 5.1: Comparison between Best Fit (BF) and Release Time-Aware Best Fit (RTABF). VMs 3 and 4 have to be placed on servers A and B, where VMs 1 and 2 are already running. RTABF saves energy over BF by taking into account the runtime of VMs. With RTABF, server A is turned off earlier than with BF.

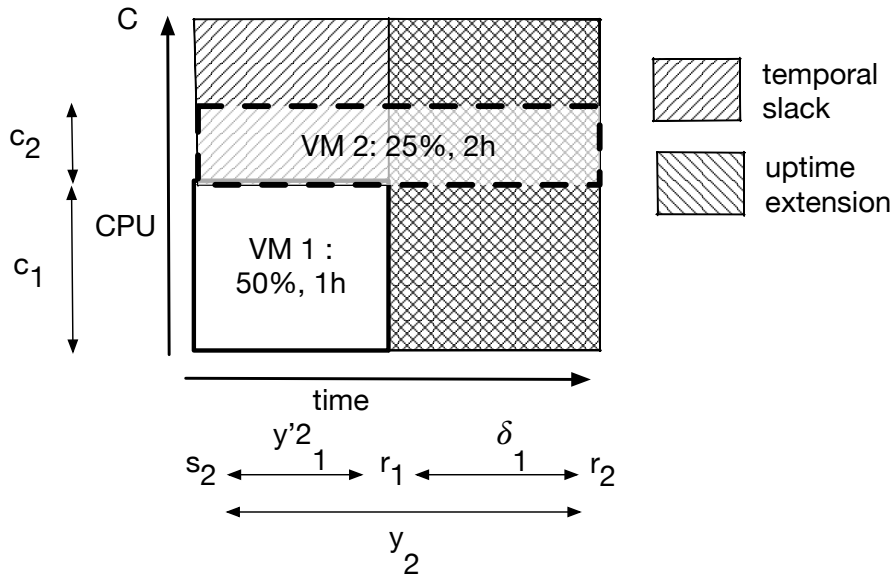



Figure 5.2: Computation of the placement cost with Release Time-Aware Best Fit (RTABF)

$\mathcal{N}_j$ the set of VMs on host $j$ (Equation 5.4). $\beta_{ij}$, the uptime extension (Equation 5.5). Finally, $\gamma_{ij}$ is the total cost for placing VM $i$ on server $j$ (Equation 5.6). RTABF chooses the server with minimum cost (Equation 5.7).

$$r_i = s_i + y_i \tag{5.1}$$

$$y'^i_k = \min\{r_k, r_i\} - s_i \tag{5.2}$$

$$\delta_{i,j} = \max\left(0, r_i - \max_{k \in \mathcal{N}_j}\{r_k\}\right) \tag{5.3}$$

$$\alpha_{i,j} = C_j \times y_i - \sum_{k \in \mathcal{N}_j} c_k y'^i_k \tag{5.4}$$

$$\beta_{i,j} = C_j \times \delta_{i,j} \tag{5.5}$$

$$\gamma_{i,j} = \alpha_{i,j} + \beta_{i,j} \tag{5.6}$$

$$j^\star = \arg\min_j \gamma_{i,j} \tag{5.7}$$

Although it is possible to feed RTABF with predictions of VM runtimes obtained from machine learning systems, predictions will necessarily contain some error. In the next section, we present our methodology to evaluate the sensitivity of RTABF with respect to the prediction error, as measured per the energy savings against AF and BF.

## 5.3 Experimental Setup

In this section we present our experimental setup for the evaluation of the sensitivity of RTABF with respect to the error in runtime prediction. In order to test RTABF with controllable error levels, we simulate a cloud infrastructure and workload. We use the Google workload trace [96] to compare our results with the original evaluation of RTABF [24], which was obtained assuming a perfect knowlege of VM runtimes. This section presents the workload trace, the infrastructure model and the energy consumption model.

### 5.3.1 Workload Trace

The Google workload trace, presented in details in Chapter 2, spans 29 days and reports the execution of millions of tasks on a platform composed of 11k servers [96]. In order to speed up the simulation but still obtain robust results, we make ten samples of the original trace. One sample is

generated with a three-steps pipeline. Firstly, we select the tasks submitted by 10% of users chosen at random. Then, we select the tasks started within a random five-day window. Finally, 20% of the tasks of each user are deterministically selected (one out of 5 tasks ordered in time). The number of VMs as a function of time for one generated trace is reported in figure 5.3. This Figure shows that the workload goes through major peaks and troughs. The workload dynamicity is sufficient to compare the performance of different VM placement heuristics.

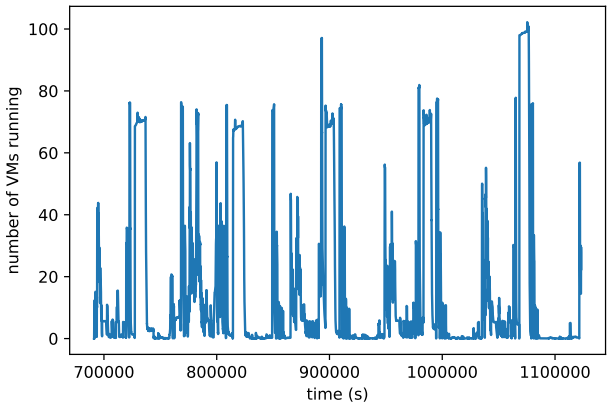



Figure 5.3: Number of virtual machines running over time in one trace sample

To enable the evaluation of RTABF, we include the runtime $y$ of a VM in the corresponding start request. We then add to the runtime the prediction error $e$ that would be inevitable if RTABF was used in the real world (Equation 5.8). The error is randomly generated from a Gaussian distribution with mean $\mu = 0s$ and standard deviation $\sigma$ (Equation 5.9). The parameter $\sigma$ therefore controls the level of error. We apply ten error levels to each trace, from $\sigma = 0s$ (no error) up to $\sigma = 3000s$. This is a wide range considering that 90% of VMs run less than $1000s$ [128].

$$\hat{y} = y + e \tag{5.8}$$

$$E \sim \mathcal{N}(\mu = 0, \sigma^2) \tag{5.9}$$

### 5.3.2 Infrastructure Model

Cloud infrastructures are heterogeneous. The Google workload trace reports three types of servers whose capacities are given relatively to the largest server capacity. We use the same model in our

simulation. The characteristics of the simulated infrastructure are reported in Table 5.1.

| server class | cpu capacity | Idle Power ($P^{idl}$, in Watts) | Max Power ($P^{max}$, in Watts) | count |
|---|---|---|---|---|
| S | 0.25 | 43 | 152 | 25 |
| M | 0.5 | 46 | 237 | 125 |
| L | 1 | 79 | 521 | 70 |

Table 5.1: Server characteristics

Since host configurations are heterogeneous, simply measuring the number of allocated hosts by each policy would not be a very good efficiency indicator. Therefore, we use energy consumption as a measure of efficiency of the studied heuristics. In the following section we detail our proposed energy consumption model.

### 5.3.3 Energy Consumption Model

Modeling the energy consumption of individual servers [129], or whole data centers [130], is an active research topic. It was shown in a large scale experiment that cpu load is the single metric that allows to best approximate server power consumption [131]. The reason is that CPU is the server component that has the most dynamic power range [38]. In numerous works [14, 73, 132, 127, 24], the power draw of a server is modeled with an affine function of the CPU utilization $u(t)$ (Equation 5.10). The two parameters of the equation are $P^{max}$, the maximum power draw, and $P^{idl}$, the power drawn when the server is powered on and idle.

$$P(t) = a * u(t) + b \tag{5.10}$$

$$a = P^{max} - P^{idl} \tag{5.11}$$

$$b = P^{idl} \tag{5.12}$$

$$u(t) \in [0, 1] \tag{5.13}$$

$P^{idl}$ captures the fact that servers are not energy-proportional, they consume energy even when they do not perform any work [38]. The implication for the placement of VMs is that, to minimize the energy consumption, servers should either be turned off or be highly utilized, as shown on Figure 5.4.

As in [132], we used different coefficient values according to server size. Bigger servers consume more power than smaller ones in the idle state, and they have a larger power range. We used the

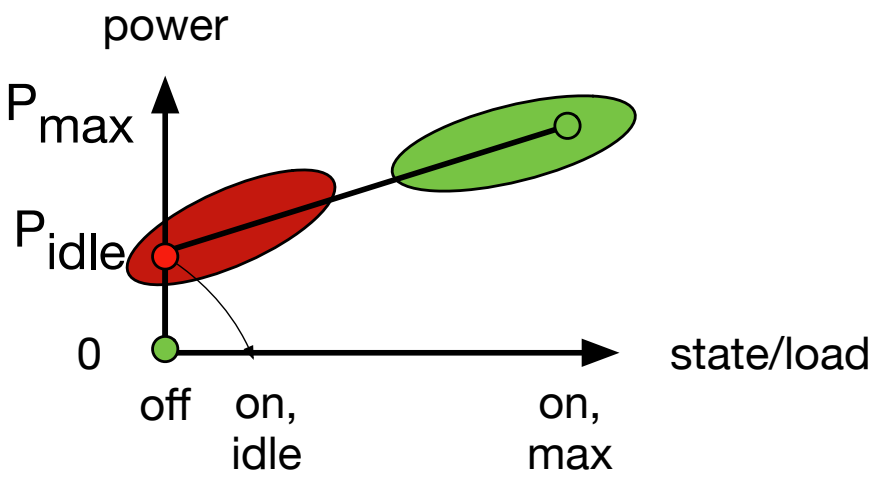



Figure 5.4: Implication of the non-energy-proportionality of servers. The most energy-efficient states (in green) are when the server is turned off or highly utilized.

values given by the Cisco UCS power calculator [133] for a B200 M4 blade server with 6, 12 and 24 cores, given in Table 5.1. Based on the work of Orgerie et al. [134], our model accounts the energy consumed during server state switches. We call $E_{on}$ and $E_{off}$ the energy needed to turn a server on and off, and give their values in Table 5.2. Finally, we note $N_{on}$ and $N_{off}$ the number of times servers are turned on and off during the execution of the simulation. These definitions enable to derive the total energy consumption for a simulation as in Equation 5.14.

$$E = \int \sum_{j} P_j(t)dt + N_{on}E_{on} + N_{off}E_{off} \tag{5.14}$$

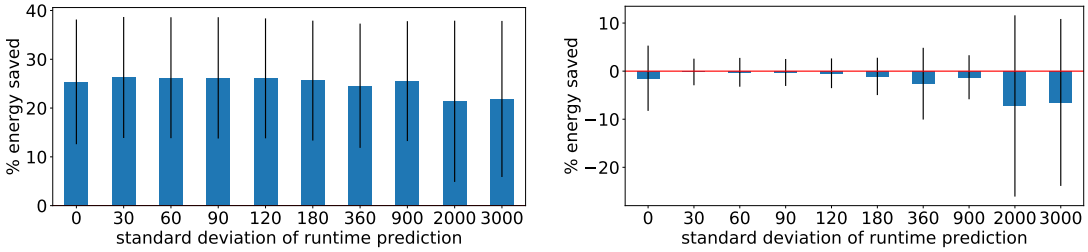
| symbol | value | description |
|---|---|---|
| $E_{on}$ | 24536 J | energy spent for switching a server on |
| $E_{off}$ | 1501 J | energy spent for switching a server off |

Table 5.2: Energy consumed during server state switches

## 5.4 Results

This section reports the energy savings of RTABF with respect to Any Fit and Best Fit, the two baselines that do not use the knowledge of VM runtime. Figure 7.6 presents the average and dispersion of energy savings over the ten sample traces. RTABF is evaluated for ten levels of Gaussian prediction

error, displayed on the horizontal axis. Figure 7.6a shows that RTABF consistently saves 25% energy over Any Fit. Figure 7.6b shows that RTABF does not outperform Best Fit, even when the prediction error is null. As expected, the performance of RTABF degrades once the prediction error reaches a threshold (900s).




(a) against Any Fit

(b) against Best Fit

Figure 5.5: Percentage of energy saved by RTABF against any-fit and best fit, averaged over 10 simulations

These results highlight that Best Fit works well without using any information of VM runtime. The use of VM runtime by RTABF does not bring significant improvement in energy consumption, according to our model. The relationship between VM/job runtime and the share of resource consumed might explain why the knowledge of runtime does not allow RTABF to outperform BF: "Most jobs are short, but short jobs contribute little to utilization. Even though less than 2% of jobs run for longer than one day, such jobs account for over 80% of the recorded usage by [...] requested CPU-or memory-days" [135]. Because of the weight of long-running jobs, there may be insufficient opportunities to shutdown servers and save energy.

### 5.4.1 Conclusion

In this chapter, we evaluated the sensitivity of RTABF, a VM placement algorithm that requires predictions of VM runtimes. Through simulations based on the Google workload trace, we measured the energy savings of RTABF with respect to Best Fit and Any Fit, for various levels of prediction error. Our results show that RTABF does not outperform the Best Fit baseline, possibly because jobs running more than one day consume 80% of resources and prevent the algorithm from shutting down servers to save energy. From these results, we conclude that online VM placement with RTABF is not a valid use case for a runtime prediction system. We believe that it is necessary to look for other

resource allocation algorithms to make use of predictions.

# Chapter 6

# Conclusion and Perspectives

**Contenu**

## 6.1 Conclusion

Cloud computing services provide a shared and highly-available pool of processing and storage resources accessible on-demand, to anyone on the planet. Cloud platforms have become very popular because they allow to amortize the cost of a computing infrastructure over several organizations. Because cloud infrastructures are large-scale, the allocation of resources to users is automated. Therefore, the two main assets of a cloud provider are the infrastructure, and the orchestrator, which is the software that manages the infrastructure and allocates resources. The orchestrator must take advantage of the complementarity between user workloads to maximize the profit generated from the utilization of the infrastructure, while guaranteeing an acceptable Quality of Service (QoS) to users. Since the quality of resource allocation depends on the fitness between the algorithm and the workload, we addressed the problem of improving resource allocation at Outscale in two steps: 1) characterize the workload to identify opportunities regarding the management of resources, and 2) propose solutions suited to Outscale's workload.

### 6.1.1 Identified Opportunities Regarding the Management of Resources

In our survey of cloud workloads (Chapter 2), we showed that works from the literature focused on the deployment of VMs, in hyperscale platforms ($\sim 1M$ VMs/month) or small-scale ones ($\sim 1k$ VMs/month). However, Outscale's platform is non-hyperscale ($\sim 100k$ VMs/month), and deployments are composed of multiple virtual resources (VMs, but also images, volumes, snapshots and security groups). Each time a virtual resource is used, the orchestrator must allocate hardware resources from compute, network, and storage equipment. Besides, focusing on VMs, previous works in the literature characterized their hardware resource requests and effective usage, but not their QoS, which is degraded when two VMs contend for a resource.

In order to characterize Outscale's workload and identify opportunities regarding resource allocation, we have collected two traces from Outscale's European region, from August to October 2017 (3 months). The first trace includes requests of virtual resource management operations sent by users, the second includes measurements of hardware resource usage by VMs. To record the second trace, we deployed an ad-hoc probe on servers in production. In addition to classic measurements of CPU, RAM and disk utilization, we collected the number of involuntary context switches ($ics$), an interfer-

ence metric that counts the number of times the threads of a VM got their CPUs pre-empted by the hypervisor to execute the threads of other VMs or to perform a management or virtualization task.

In Chapter 3, we characterized Outscale's workload and found that resource creations are bursty, virtual resource lifetimes and co-deployments are skewed, and so are the amounts of hardware resources requested and effectively used.
Snapshot creation bursts stress the orchestrator 6 times a day. Outscale would gain from scaling the orchestrator to the workload. 10% of volumes live more than 5 days. Yet, because long-lived volumes fill up progressively and are snapshotted recurrently, they have a larger footprint on storage backends than the majority of shorter-lived volumes. Predicting the lifetime of volumes could allow to spread the long-lived ones on the storage backends and increase the overcommit ratio of storage space. 75% of images and security groups are used by two VMs or less, but sometimes they are used by thousands of VMs. The implementation of images and security groups must be scalable.
Regarding VMs alone, we compared their utilization at Outscale with Azure and Bitbrains, which are respectively hyperscale and small-scale providers. We found that VMs at Outscale request 13% less vCPU and 16% more RAM than at Azure. Given these differences, additional work is required to evaluate if the output of state of the art VM placement algorithms remains acceptable at Outscale. At Outscale, 70% of VMs never use more than 20% of their requested CPU, and an overcommit of three allows to save 45% of CPU costs. However, CPU overcommit generates involuntary context switches, especially for small VM. Besides, short VM runtimes (90% of VMs run less than 1h) limit the benefit of state of the art VM placement techniques surveyed in Chapter 2. Current techniques place new VMs based on their resource request, monitor the effective utilization, and then use migrations to consolidate the VMs on a minimum number of servers. The orchestrator does not have sufficient time to perform these steps for short-running VMs, hence, their allocation of resources is probably not optimal. This characterization of Outscale's workload led to two publications:

— Loïc Perennou, Mar Callau-Zori, Sylvain Lefebvre, Raja Chiky. *Workload Characterization for a Non-Hyperscale Public Cloud Platform*, short paper, Proceedings of the IEEE International Conference on Cloud Computing (CLOUD '19).

— Loïc Perennou, Mar Callau-Zori and Sylvain Lefebvre. *Understanding Scheduler Workload on Non-Hyperscale Cloud Platform*, poster, Proceedings of the 19th ACM/IFIP Middleware Conference (Middleware '18).

To solve the problem caused by short VM runtimes, we made the hypothesis that it was possible to predict the behavior of VMs at startup because deployments in the cloud were automatized and repetitive.

### 6.1.2 Predictions of VM Runtime to Improve VM Placement

In Chapter 4, we predicted the runtime of VMs at startup using supervised machine learning. We used features proposed in previous works, such as the amount of resource requested, the timing of the API request of VM creation, and the account state and history. We proposed the utilization of features describing the network and ephemeral storage configuration of VMs, server affinity, and text tags, which users attach to VMs to simplify their inventory. With this extensive set of features, we obtained an F1 score of 0.91 on a four-class classification problem, using Outscale's workload trace. This is a significantly better than previous works that obtained F1=0.75 on the same four-class classification problem, with the Azure trace. Initially, the F1 score was lower (0.65), especially for the classes with a minority of items. As this is a symptom of class unbalance, we tried to train the model on a dataset where the majority class was under-sampled, or the minority class was over-sampled. In addition, we proposed to decompose the four-class classification problem into a cascade of binary classification problems. None of these approaches increased model performance. We then tuned the model hyperparameters, and found that tree depth needed to be increased to 75 to reach an optimal score. This showed that our initial model was underfitting the data, rather than suffering from class unbalance. This contribution led to the following publication:

— Loïc Perennou, Raja Chiky, *Applying Supervised machine learning to predict virtual machine runtime for a non-hyperscale cloud provider*, Proceedings of the 11th International Conference on Computational Collective Intelligence (ICCCI '19).

In Chapter 5, we sought to determine what level of prediction error is required to benefit to the placement of VMs. To answer this, we evaluated the sensitivity of a VM placement algorithm from the literature with respect to the prediction error of VM runtime. The evaluated VM placement heuristics, Release-Time Aware Best Fit (RTABF), co-locates VMs that finish simultaneously in order to reduce server utilization time and save energy. In previous works, RTABF was evaluated with the assumption that the runtime of VMs was perfectly known when they start. To complement the evaluation of RTABF and determine the tolerance to prediction error, we analyzed the sensitivity of

RTABF with respect to the error of runtime prediction. We simulated the execution of RTABF for various levels of synthetic prediction error, generated from a Gaussian distribution. We compared RTABF with Any Fit and Best Fit with respect to the energy consumption of servers. In order to compare our results with the original evaluation of RTATF, we simulated the execution of the Google workload trace. Unexpectedly, we found that RTABF does not outperform Best Fit, even when RTABF has access to perfect predictions of VM runtimes. This contribution led to one publication:

— Loïc Perennou and Sylvain Lefebvre. *Runtime Prediction Error Levels for Virtual Machine Placement in IaaS Cloud*, Proceedings of the 9th International Conference on Ambient Systems, Networks and Technologies (ANT '18).

## 6.2 Perspectives

Although we didn't find the RTABF algorithm to be a promising use case for predictions of the behavior of VMs, we still think that predictions can be very useful.

### 6.2.1 Maintenances

The cloud provider needs to make periodic maintenances on servers, to correct a hardware fault (e.g., ECC errors in memory chips), or update a software such as the hypervisor. Each maintenance operation requires to reboot the server. To make maintenance transparent for the user, the orchestrator must perform live migrations of the VMs out of the relevant servers. Hence, maintenances cannot be made simultaneously on all servers, instead, they are scheduled in batches. In this context, there is a conflict between the objective of minimizing the maintenance makespan, and minimizing the cost of migrations. In practice, to make the migration cost acceptable, the orchestrator puts servers in a staging mode preliminary to a maintenance. Servers in staging mode continue to run existing VMs but do not accept new ones. This mode allows to lower the number of VMs running, and thus the migration cost. However, it lengthens the maintenance makespan, and makes a fraction of resources unavailable to users. We argue that the provider could obtain a better tradeoff if the placement of VMs took into account the runtime of VMs and the maintenance schedules. Long-running VMs should go on servers that will not need maintenance in the foreseeable future, whereas short-running VMs should be allowed to run on servers that are in staging mode.

### 6.2.2 First Class Upgrade

Sometimes, the cloud platform lacks the sufficient resources to allow a VM of a given type to start. In this case, the orchestrator has two possible choices: deny the service, or accept the VM on a better server than initially requested (we assume this second option is available). It is not trivial to determine what the optimal choice is. It seems preferable to accept the VM on an upgraded server class, provided that, during runtime of the VM, there will be sufficient resources in the upgraded server class to serve high-end VMs. All else being equal, it is preferable to upgrade the server class for short-running VMs than long-running ones.

### 6.2.3 Overcommitment

The overcommitment of resources generates interferences between VMs when they need to use resources simultaneously. In the long term, we think that we will need to use a combination of existing unsupervised techniques with our proposed supervised approach to minimize interferences. Existing unsupervised machine learning approaches allow to establish representative profiles of VMs, through the clustering of their time series of resource utilization. Once representative profiles are established, we believe that supervised machine learning could be used to predict the profile a VM at startup. Thanks to the successive utilization of unsupervised and supervised machine learning, we believe that it will be possible to optimize the co-location of VMs with compatible resource usage as soon as they start.

# Chapter 7

# Extended Summary in French

## 7.1 Introduction

### 7.1.1 Contexte

Les progrès dans de nombreux secteurs d'activité, tels les transports et la médecine, sont conditionnés par le besoin d'analyser un volume de données qui double tous les deux ans - un phénomène appelé Big Data [4, 3]. Pour réduire les délais et les coûts de traitement des données, le cloud offre un accès sur demande, via le réseau, à des ressources informatiques mutualisées, gérées par un fournisseur et facturées à l'usage [6]. Ainsi, les utilisateurs peuvent ajuster leur consommation à leur besoin. Il existe plusieurs types de services clouds pour lesquels le périmètre de responsabilité du fournisseur varie. Les trois principaux, définis par le *National Institute of Standards and Technology* en 2011, sont présentés en Figure 7.1.

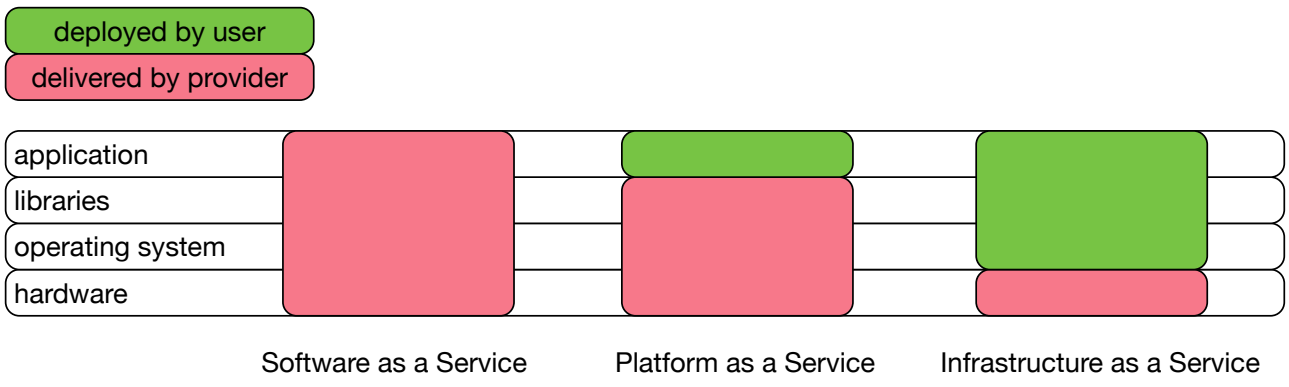



Figure 7.1: Périmètre de responsabilité des trois principaux types de services cloud

Dans cette thèse, nous travaillons sur le modèle IaaS, qui constitue le coeur de métier d'Outscale, notre partenaire industriel. Outscale possède deux actifs : 1) une infrastructure matérielle conçue en partenariat avec CISCO, Intel, Netapp et Nvidia; et 2) un *orchestrateur* propriétaire, TINA OS, qui coordonne la gestion du matériel, et alloue des ressources aux utilisateurs. Comme le montre la Figure 7.2, les technologies de virtualisation sont au coeur de la plateforme, car elles permettent un partage de toute ressource matérielle (serveurs, baies de stockage et réseaux) entre plusieurs utilisateurs pour en optimiser l'utilisation.

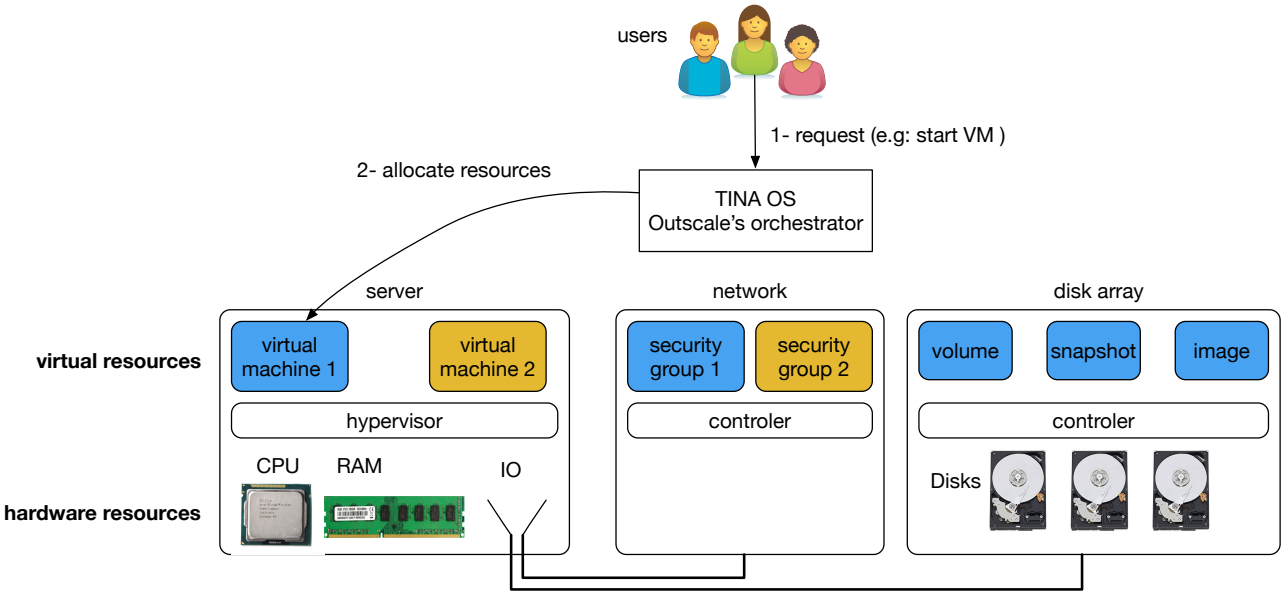



Figure 7.2: Architecture de la plateforme IaaS d'Outscale

### 7.1.2 Contributions

L'objectif de cette thèse est d'aider Outscale à optimiser l'allocation des ressources sur sa plateforme cloud. Outscale cherche à accueillir le plus grand nombre d'utilisateurs possibles pour maximiser le profit généré par la plateforme, tout en veillant à ce que la Qualité de Service (QoS) reste conforme aux attentes. L'algorithme d'allocation de ressources traduit l'expertise des administrateurs en un ensemble de règles. Les performances de l'algorithme dépendent donc du degré de compréhension des caractéristiques de la charge de travail. Dans un premier temps, nous chercherons à développer cette expertise avant de proposer des améliorations de l'algorithme.

### 7.1.3 Caractérisation de la charge de travail

Pour identifier des opportunités d'améliorer l'allocation des ressources, nous caractérisons la charge sur la plateforme grâce à des traces que nous avons collectées. Nous caractérisons l'utilisation conjointe des ressources virtuelles pour identifier les opérations qui stressent l'orchestrateur. Nous caractérisons l'utilisation des ressources matérielles et discutons des implications sur le management de la QoS. Enfin, nous comparons la charge d'Outscale avec la charge d'autres fournisseurs.

### 7.1.4 Amélioration de l'allocation des ressources aux VMs à partir de prédictions de leur durée de vie

Le placement des VMs sur les serveurs est un problème *en ligne*, l'orchestrateur a une connaissance partielle des données du problème [17]. Par exemple, la durée de vie de la VM à placer est inconnue. La migration des VMs est couramment utilisée pour parer à l'imprévu. Les migrations servent à consolider les VMs sur un ensemble minimal de serveurs pour économiser de l'énergie, ou bien pour vider un serveur avant une maintenance [20]. Mais les migrations prennent du temps et consomment des ressources, ainsi elles peuvent dégrader la qualité de service des VMs [23, 22]. Cette thèse apporte deux autres contributions visant à améliorer l'allocation des ressources.

— Nous formulons l'hypothèse que la durée de vie des VMs peut être prédite à leur démarrage. Nous montrons que les étiquettes textuelles (tags) décrivant les VMs peuvent être utilisées pour améliorer la précision des modèles de prédiction.

— Nous cherchons à déterminer quelle précision est nécessaire pour utiliser les prédictions lors de l'allocation de ressources. Pour ce faire, nous analysons la sensibilité d'un algorithme de placement de VMs de la littérature à l'erreur de prédiction de durée de vie. Cet algorithme, Release-Time Aware Best Fit (RTABF), avait précédemment été évalué en considérant des prédictions parfaites.

## 7.2 Etat de l'art

### 7.2.1 Caractérisation de la charge sur les plateformes

Afin d'optimiser l'allocation des ressources de calcul [99, 97], l'allocation des ressources de stockage [94, 95], la fixation des prix [106], la tolérance aux pannes [107, 108], ou le développement de

nouvelles fonctionnalités [110], il est nécessaire d'avoir une connaissance fine de l'utilisation de la plateforme. Concernant les problématiques d'allocation des ressources, la littérature montre que la principale difficulté tient à l'hétérogénéité et la variabilité temporelle des besoins [99, 97]. La Table 7.1 compare les traces d'exécution qui ont servi aux études. La table montre que les travaux s'intéressent principalement au déploiement des machines virtuelles (VMs), mais négligent le déploiement des volumes, instantanés, et groupes de sécurité utilisés conjointement. Par ailleurs, les plateformes à l'étude sont de très grande taille, comme celle d'Azure (2M VMs/mois) [2], ou bien de petite taille [16]. Or, Outscale opère une plateforme de taille intermédiaire ($\sim 100k$ VMs/mois), ce qui pourrait influer sur d'autres aspects de la charge. Enfin, les études précédentes ont caractérisé l'utilisation des ressources matérielles (CPU, RAM, IO) [105], mais pas les interférences entre utilisateurs qui dégradent la QoS fournie.

| trace | entity | hyper scale | state events | virtual machine utilization | | | | | other virtual resources |
|---|---|---|---|---|---|---|---|---|---|
| | | | | CPU | RAM | Disk | Net. | interf. | |
| Google [96] | job | yes | yes | yes | yes | yes | – | yes | – |
| Alibaba [97] | job | yes | yes | yes | yes | yes | yes | yes | – |
| Azure [2] | VM | yes | yes | yes | – | – | – | – | – |
| Eucal. Sys. [92] | VM | – | yes | – | – | – | – | – | – |
| SCERIT-SC [16] | VM | – | yes | – | – | – | – | – | – |
| Bitbrains [15] | VM | – | – | yes | yes | yes | yes | – | – |
| IBM [95] | VM | – | yes | – | – | – | – | – | images |
| Nutanix [94] | VM | – | – | – | – | – | – | – | – |
| Outscale | VM | – | yes | yes | yes | yes | – | yes | yes |

Table 7.1: Comparaison du contenu des traces cloud

### 7.2.2 Placement des VMs

Concernant le problème spécifique du placement des VMs sur les serveurs, il s'agit d'un problème d'optimisation combinatoire [18]. Un tel problème fait intervenir une fonction objectif pour trouver la meilleur solution parmi un ensemble discret de solutions acceptables. Dans la littérature, il existe plusieurs approches pour définir l'objectif ainsi que la méthode d'exploration des solutions. Pour les objectifs, on peut citer le besoin de consolider les VMs sur un minimum de serveurs pour éteindre les serveurs inutilisés et ainsi économiser de l'énergie [14, 41, 136]. D'autres approches maximisent la

profitabilité de l'infrastructure en ajustant les prix des VMs à la demande, et en n'exécutant que les plus rentables [45, 46]. Enfin, certains travaux visent à respecter les accords de services passés sur la QoS fournie [47, 49].

La définition d'une fonction objectif requiert de modéliser l'utilisation de ressources et la QoS des VMs. Certains modèles considèrent que la QoS est acceptable tant que l'utilisation d'un serveur ne dépasse pas un seuil [50, 51]. Ces modèles ont deux limitations. Premièrement, l'utilisation de ressources micro-architecturales, comme les caches CPU ou le bus mémoire, ne peut pas être mesurée. Ensuite, la tolérance aux interférences des applications qui s'exécutent sur les VMs est variable [63]. C'est pourquoi certains travaux s'appuient sur des mesures d'interférences collectées pour différentes configurations de co-localisation des applications [47, 64]. Cependant, ces modèles de performance ne peuvent pour l'instant pas être établis dans un cloud IaaS, car le fournisseur n'a pas connaissance de l'identité de l'application qui s'exécute dans une VM.

Une fois que la fonction objectif et le modèle d'utilisation des ressources ont été choisis, il faut déterminer comment explorer au mieux l'ensemble des solutions acceptables. La difficulté provient du fait que le problème est NP-complet, l'ensemble de solutions croît exponentiellement avec le nombre de serveurs, et l'on ne sait pas si il existe une méthode qui garantisse de trouver la solution optimale sans toutes les tester [65]. Deux approches sont mises en oeuvre pour réduire la complexité du problème et accélérer sa résolution. La première approche consiste à décomposer le problème, en divisant l'ensemble des serveurs en clusters, qui peuvent être formés de manière statique [67] ou dynamique [68, 69]. La seconde approche consiste à résoudre le problème de manière itérative. Les heuristiques placent les VMs une par une [35, 55]. Les métaheuristiques partent d'un ensemble de solutions aléatoires et explorent l'entourage des meilleures avec une probabilité plus forte que les moins bonnes [68, 53].

### 7.2.3 Utilisation de l'apprentissage automatique pour le placement des VMs

Avec l'évolution de l'état de la plateforme suite au démarrage/arrêt de VMs ou à un changement d'utilisation, le fournisseur doit ré-optimiser l'allocation des ressources. Pour compenser le temps nécessaire pour rechercher et implémenter une configuration optimale, il est nécessaire d'anticiper les changements. Plusieurs travaux utilisent des techniques d'apprentissage automatique pour prédire l'état du cloud. L'apprentissage automatique cherche à résoudre des problèmes en trouvant des règles dans un ensemble de données, lorsque ces règles ne sont pas explicitement connues [75]. Nous recensons

dans la littérature deux façons d'anticiper la comportement d'une VM. L'on peut prédire l'utilisation de ressources après le démarrage, en se basant sur l'historique d'utilisation depuis le lancement [77, 78, 82], ou bien on peut prédire au démarrage, grâce aux paramètres de lancement de la VM ainsi que l'historique d'utilisation des VMs déjà exécutées [2, 85, 86]. On note que les travaux sur les prédictions au démarrage s'appliquent majoritairement aux grilles de calcul et d'analyses de données, où les paramètres de lancement ne sont pas les mêmes que dans le cloud. Le papier s'appliquant au cloud ne révèle pas les caractéristiques prédictives utilisées. Enfin, à notre connaissance, aucun travail existant n'a étudié la sensibilité de l'algorithme de placement de VMs aux erreurs de prédictions.

## 7.3 Caractérisation de l'utilisation de la plateforme d'Outscale

Optimiser l'allocation de ressources nécessite de prendre en compte la nature de la charge de travail sur la plateforme. Jusqu'ici, la littérature s'est focalisée sur des plateformes de très grande taille (2M+ VMs), ou des plateformes spécialisées de petite taille (10k- VMs). Par ailleurs, les travaux s'intéressent surtout au déploiement et à l'utilisation des VMs et images, mais pas aux autres ressources virtuelles ni aux interférences entre les fils d'exécution des VMs et de l'hyperviseur, qui partagent le CPU. Pour compléter les études précédentes, nous caractérisons la charge sur la plateforme publique, de taille moyenne ( 100k VMs/mois) et à usage général d'Outscale. À cette fin, nous avons collecté, sur la plateforme Européenne, deux trace d'exécution de trois mois entre Août et Octobre 2017. La première trace rapporte les opérations de déploiement des ressources virtuelles, comme le lancement d'une VM, ou la suppression d'un groupe de sécurité. La seconde trace rapport la consommation de ressources matérielles des VMs (CPU, RAM, disque), ainsi que la fréquence des changements de contextes involontaires. Ces derniers mesurent les interférences car ils comptent le nombre de fois où l'hyperviseur a retiré le CPU d'une VM afin d'exécuter une tâche de virtualisation ou une autre VM. Résumons maintenant les caractéristiques de la charge sur la plateforme, et leurs conséquences sur l'allocation des ressources.

### 7.3.1 Déploiement des ressources virtuelles

La caractérisation de l'heure de création des ressources virtuelles nous permet d'observer des pics de charge journaliers pour les VMs, et plus fréquents encore (4h) pour instantanés de volumes. Les ressources de l'orchestrateur doivent donc être élastiques pour lui permettre de recevoir cette charge

variable. L'analyse de la durée de vie des ressources virtuelles et de leurs interdépendances met en lumière des usages variés. Ainsi, 90% des volumes vivent moins de 5 jours. Comme ils ne sont pas remplis de données, ils permettent de sur-allouer l'espace de stockage. En revanche, 8% des volumes vivent plus de 5 jours et servent à plus de 5 instantanés. L'espace de stockage requis pour les volumes à longue durée de vie et leurs instantanés peut croître avec le temps, c'est pourquoi l'orchestrateur doit les répartir sur les différents matériels pour ne pas manquer de ressources. Or, tout comme les VMs, la durée de vie d'un volume n'est pas connue de l'orchestrateur lors de la création.

### 7.3.2 Utilisation des ressources matérielles et interférences entre les VMs

L'analyse de la consommation de ressources par les VMs montre, en accord avec les études précédentes [2], que les utilisateurs demandent souvent plus de ressources que ce qu'ils consomment réellement. Ainsi, 70% des VMs n'utilisent jamais plus de 20% de leur CPU sur une fenêtre de 5 minutes. Pour ne pas gaspiller de ressources, Outscale sur-alloue le CPU. Dans 75% des cas, un coeur physique est partagé par 4 VMs au plus. La caractérisation de la fréquence des changements de contexte involontaires révèle que les interférences CPU varient d'un facteur 100. Comme 90% des VMs vivent moins d'une heure, il paraît compliqué d'atténuer les interférences et d'optimiser la sur-allocation des ressources via les migrations, comme proposé dans la littérature [50]. Cependant, il serait peut-être possible qu'un contrôleur présent sur les serveurs optimise la co-location des fils d'exécution des VMs sur les coeurs CPU.

### 7.3.3 Comparaison de la charge d'Outscale avec d'autres fournisseurs

Nous avons comparé la charge d'Outscale avec celle d'Azure, un fournisseur hyperscale [2], et Bitbrains, qui est spécialisé dans l'hébergement des applications critiques pour l'industrie financière [15]. Chez Outscale, les VMs requêtent en moyenne 13% moins de CPU et 16% plus de RAM que chez Azure, et elles écrivent plus de données sur disque qu'elles n'en lisent alors que c'est l'inverse chez Bitbrains. Comme la performance d'un algorithme d'allocation de ressources dépend de la nature de la charge, nous constatons qu'Outscale a besoin de mener sa propre évaluation des algorithmes de la littérature pour déterminer si leurs performances restent acceptables.

### 7.3.4 Conclusion

Pour identifier des opportunités d'amélioration des algorithmes d'allocation de ressources, nous avons caractérisé la charge de la plateforme d'Outscale grâce à des traces de déploiement des ressources virtuelles et d'utilisation des ressources physiques. Nous avons identifié plusieurs motifs d'utilisation de la plateforme qui impactent la qualité de service. Ainsi, l'orchestrateur doit faire face à des pics de création de VMs et instantanés de volumes, allouer de l'espace de stockage à des volumes dont la durée de vie et donc le taux de remplissage sont hétérogènes, ou bien sur-allouer le CPU des VMs tout en contrôlant le niveau d'interférences. L'étude du cycle de vie des ressources virtuelles confirme que le cloud est majoritairement utilisé pour des besoins temporaires. On peut donc faire l'hypothèse que les déploiements sont répétitifs et donc prévisibles. Dans les chapitres suivants, nous avons voulu améliorer l'allocation des ressources en utilisant les prédictions issues d'un systèmes d'apprentissage automatique.

## 7.4 Prédiction de durée de vie des VMs

Dans la littérature, il a déjà été proposé de placer les VMs en fonction de leur durée de vie [61, 62, 24]. Le problème est que la durée de vie est à priori inconnue de l'orchestrateur. Dans ce chapitre, nous résolvons ce problème en employant des techniques *d'apprentissage automatique*. L'apprentissage automatique vise à trouver la solution d'un problème en évaluant un ensemble de solutions possibles sur un ensemble de données qui encode l'expérience acquise sur le problème [75]. L'apprentissage automatique est utilisé lorsque la solution ne peut pas être déterminées analytiquement. Ici, nous faisons appel à l'apprentissage automatique *supervisé*, car nos données associent la durée de vie des VMs s'étant exécutées dans la trace aux caractéristiques qui ont été, ou auraient éventuellement pu être, connues au démarrage. La nouveauté de notre travail réside dans l'utilisation de nouvelles caractéristiques prédictives extraites de la requête de démarrage de la VM ainsi que l'utilisation d'étiquettes textuelles (*tags*) permettant aux clients de faire l'inventaire des VMs. Les tags ne sont actuellement pas disponibles au démarrage, mais nous voulons évaluer leur contribution à la précision du modèle pour préconiser ou non une éventuelle modification de l'API.

### 7.4.1 Conditions expérimentales

L'une des difficultés associée à l'apprentissage automatique est la recherche des meilleures caractéristiques prédictives. Pour évaluer la contribution des caractéristiques à la précision du modèle, nous comparons trois ensembles de caractéristiques à complétude croissante. Dans le premier ensemble figurent toutes les caractéristiques extraites de la requête de démarrage. Ceci inclue des caractéristiques déjà connues, comme la quantité de ressources [2]et l'horodatage de la requête [123]. Nous incluons quatre nouveaux types de caractéristiques décrivant les contraintes de placement, la configuration réseau de la VM, la configuration de stockage éphémère, ainsi que d'autres paramètres divers. Le second ensemble de caractéristiques, issu de la littérature [86], résume le comportement historique et le statut actuel du compte utilisateur. Le troisième ensemble comprend les caractéristiques extraites des tags. Nous les avons obtenues en concaténant les tags de chaque VM dans un document, puis en indexant les mots et en codant leur présence avec un vecteur.

Comme une approximation de la durée de vie est suffisante pour un algorithme de placement, nous avons formulé le problème en une instance de classification. Nous adoptons le score F1 comme métrique d'évaluation de la précision du modèle, car il est robuste quand les classes ont un effectif hétérogène. Comme modèle, nous optons pour les arbres de décisions aléatoires extrêmes (extremely randomized trees) [120]. Les arbres de décision ont l'avantage d'être interprétables car ils sont construits en partitionnant l'espace des données. Les arbres de décisions extrêmes sont un ensemble d'arbres de décisions, construits en parallèles sur un échantillon aléatoire des données, en considérant des critères de partitionnement aléatoires eux aussi. L'aléa génère des arbres différents qui, une fois combinés, permettent d'obtenir un excellent compromis entre biais et variance, c'est-à-dire qu'ils peuvent correctement modéliser la complexité des données sans pour autant être sensible au bruit. Pour nos expériences, nous avons divisé nos données en un ensemble d'apprentissage (80% des individus tirés au hasard) et de test (le reste).

### 7.4.2 Résultats

Nous présentons ici nos résultats sur la prédiction de la durée de vie des VMs de la plateforme d'Outscale, et les comparons avec ceux précédemment obtenus sur les VMs de Microsoft Azure [2].

La figure 7.3 montre que la précision de la classification, mesurée par le score F1, croît avec

l'utilisation de nouvelles caractéristiques prédictives. Avec les caractéristiques extraites dans la requête de démarrage uniquement, notre modèle atteint un score F1=0.76, soit autant que le modèle de comparaison (ligne horizontale noire). Le score F1 atteint 0.87 en ajoutant des caractéristiques qui résument l'historique et l'état actuel du compte utilisateur, et 0.91 en incluant les tags.

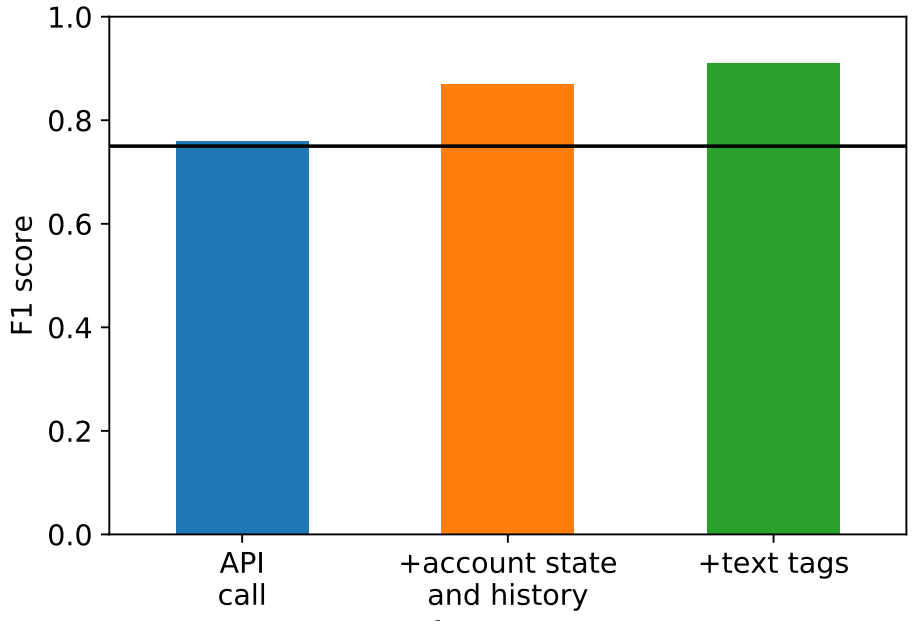



Figure 7.3: Évolution de la précision de la prédiction avec ajout incrémental de caractéristiques prédictives. La ligne horizontale correspond au modèle de comparaison évalué sur la plateforme de Microsoft Azure [2].

La figure 7.4 présente la liste des 20 plus importantes caractéristiques prédictives. L'importance maximale, 0.04, est largement inférieure à 1. Cela démontre donc la nécessité d'utiliser une variété de caractéristiques prédictives. Nous observons aussi, de manière surprenante, que la quantité de CPU et RAM demandées, qui sont les caractéristiques les plus couramment utilisées dans la littérature, apportent en fait peu d'information car elles ne sont pas présentes dans la liste. 4 nouvelles caractéristiques que nous avons proposées apparaissent dans la liste et deux sont en premières places, il s'agit de nombre de groupes de sécurité appliqués à la VM et à la présence de tags.

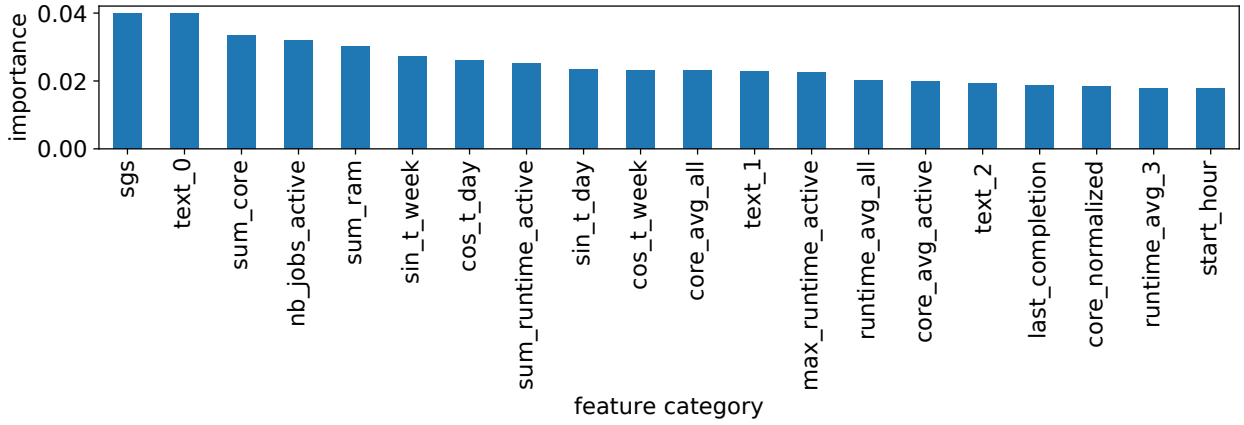



Figure 7.4: Liste des 20 caractéristiques les plus utiles à la prédiction.

### 7.4.3 Conclusion

Nous avons proposé une méthode pour prédire la durée de vie des VMs grâce à l'apprentissage automatique. Nous avons présenté de nouvelles caractéristiques prédictives, extraites de la requête de lancement de VMs et des tags utilisés pour faire l'inventaire des VMs. Grâce à ces caractéristiques, notre modèle obtient une meilleure précision que précédemment reporté dans la littérature (F1=0.91 vs F1=0.74). Les tags sont actuellement fournis après la création de la VM. Pour savoir s'il serait utile de modifier l'API afin de bénéficier du gain de précision, nous devons déterminer quelle est la précision requise par un algorithme de placement. Dans la section suivante, nous présentons notre évaluation de la sensibilité d'un algorithme de placement aux erreurs de prédictions.

## 7.5 Évaluation le a sensibilité de l'algorithme RTABF

Nous avons cherché à déterminer quelle est la précision requise pour qu'un algorithme de placement de VMs s'appuie sur des prédictions de durée de vie. En 2014, Dabbagh et al. ont proposé l'algorithme Release-Time Aware Best Fit (RTABF), qui, en co-localisant les VMs qui s'arrêteront en même temps, permet l'extinction des serveurs inutilisés et minimise la consommation d'énergie [24]. Comme RTABF a été évalué en considérant des prédictions parfaites, nous proposons d'évaluer sa sensibilité aux erreurs de prédiction. Comme dans l'évaluation originale, nous choisissons comme métrique la quantité d'énergie économisée, relativement à l'utilisation d'algorithmes de placement classiques comme Any-

Fit et Best-Fit. Dans le paragraphe suivant, nous présentons chacun de ces algorithmes.

### 7.5.1 Présentation des algorithmes de placement des VMs en ligne

Les algorithmes de placement en ligne sont utilisés pour le placement initial des VMs, pas pour la consolidation des VMs existantes. Les deux algorithmes les plus connus sont Any-Fit (AF), qui choisit un serveur au hasard parmi ceux disposant d'assez de ressources; et Best-Fit, qui choisit le serveur sur lequel le placement de la VM laissera le moins de ressources libres [74, 41, 127].

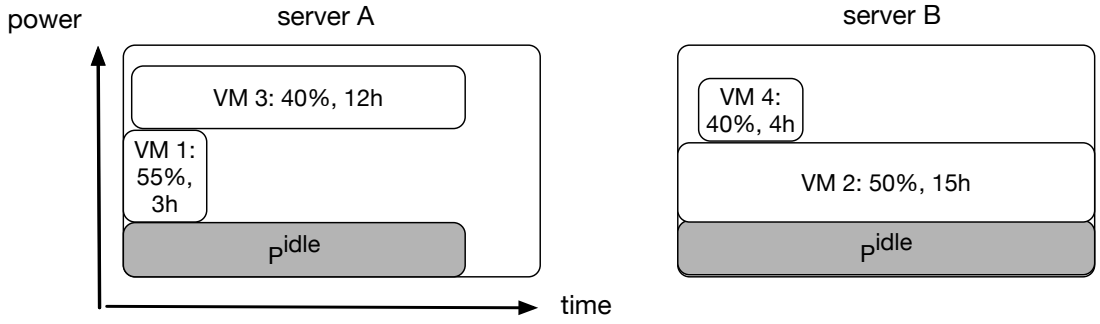
Tandis que AF et BF se basent uniquement sur la quantité de ressources demandée, Release-Time Aware Best Fit (RTABF) prend aussi en compte la durée de vie de la VM. RTABF combine l'objectif de BF avec l'objectif de minimiser le temps d'utilisation des serveurs, et donc l'énergie consommée. Pour cela, RTABF co-localise les VMs qui s'éteindront en même temps. La figure 7.5a compare l'exécution des algorithmes BF et RTABF, et montre comment la prise en compte de la durée de vie peut servir à économiser l'énergie.

### 7.5.2 Conditions expérimentales

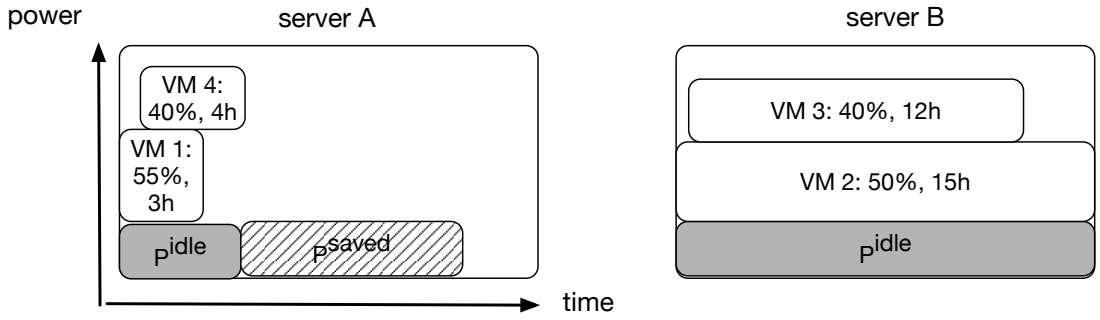
Pour évaluer la capacité de RTABF à économiser de l'énergie en dépit des erreurs de prédiction, nous avons simulé une infrastructure et une charge de travail avec un niveau contrôlable d'erreur de prédiction. Pour l'infrastructure, nous adoptons un modèle de consommation d'énergie linéaire avec la charge CPU des serveurs, en accord avec la littérature [14, 73]. Pour la trace d'exécution, nous avons sélectionné dix échantillons aléatoires de la trace de Google [96] afin d'obtenir des résultats statistiquement fiables à partir de simulations rapides, et aussi afin de comparer nos résultats avec l'évaluation originale de RTABF. Nous avons simulé la présence d'erreur de prédiction en rajoutant à la durée de vie des VMs un bruit Gaussien, et ce pour dix niveaux de bruits différents.

### 7.5.3 Résultats

La figure 7.6 présente la moyenne et la déviation standard des économies d'énergie réalisées par RTABF, relativement à AF et BF, pour différents niveaux d'erreur de prédiction. Indépendamment du niveau d'erreur de prédiction, RTABF économise 25% d'énergie par rapport à AF. Par contre, RTABF n'économise pas d'énergie par rapport à BF, même lorsque l'on ajoute pas d'erreur aux durées de vie des VMs.

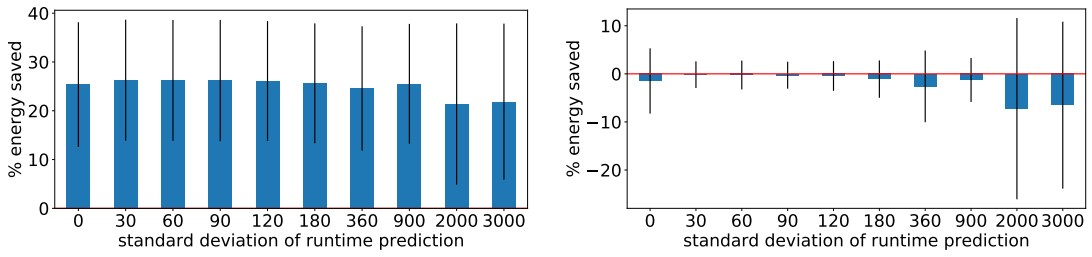



(a) Best Fit




(b) Release-Time Aware Best Fit

Figure 7.5: Comparaison entre Best-Fit(BF) et Release Time-Aware Best-Fit (RTABF). VMs 3 et 4 doivent être placées sur les serveurs A et B, tandis que les VMs 1 et 2 sont déjà en train de s'exécuter. RTABF économise de l'énergie en prenant en compte la durée de vie des VMs, ce qui permet d'éteindre le serveur A plus tôt qu'avec BF.




(a) against Any Fit

(b) against Best Fit

Figure 7.6: Pourcentage d'énergie économisée par RTABF, relativement à AF et BF, sur 10 simulations

Ce résultat négatif pourrait être expliqué par la relation entre la durée de vie des VMs et la fraction de ressources consommée sur l'infrastructure: "La plupart des VMs ont une durée de vie courte, mais contribuent peu à l'utilisation. Ainsi, les 98% de VMs qui durent plus d'une journée consomment moins de 20% des ressources" [135]. Le poids important des longues VMs pourrait empêcher l'algorithme d'éteindre les serveurs pour économiser de l'énergie.

### 7.5.4 Conclusion

Pour étudier l'intérêt de couplet un système prédictif avec un algorithme de placement de VMs, nous avons évalué la sensibilité de l'algorithme RTABF aux erreurs de prédiction. Nous avons simulé le placement d'un flux de VMs d'après la trace de Google pour différents niveaux d'erreurs de prédiction, et rapporté la consommation d'énergie sous RTABF avec celle sous Any-Fit et Best-Fit. Nos résultats montrent que RTABF n'est pas meilleur que Best-Fit, possiblement parce que les VMs de longue durée de vie consomment la plupart des ressources et empêchent donc l'algorithme d'éteindre les serveurs pour économiser de l'énergie. Nous concluons donc que le placement d'une VM avec RTABF n'est pas un cas d'usage viable pour les prédictions de durée de vie, et qu'il faut chercher d'autres cas d'usages.

## 7.6 Conclusion Générale

L'un des principaux défis des infrastructures d'informatique en nuage (cloud computing) est d'offrir aux utilisateurs une performance acceptable, tout en minimisant les besoins en matériel et énergie. Cette thèse CIFRE, menée en collaboration avec Outscale, un fournisseur de services cloud, visait à améliorer l'allocation des ressources de la plateforme grâce à de nouvelles sources d'information pertinentes.

Les caractéristiques de la charge soumise à l'orchestrateur déterminent dans quelle mesure il est possible d'allier performance et économie de ressources. La plateforme d'Outscale et sa charge de travail possèdent des caractéristiques particulières : avec $\tilde{1}$50k machines virtuelles créées par mois, la plateforme est un ordre de grandeur plus grande que des clouds privés, et un ordre de grandeur plus petite que des clouds dits "hyperscale", comme celui de Microsoft Azure. C'est pourquoi nos trois contributions visaient à optimiser l'allocation des ressources en tenant compte des spécificités de la charge de travail.

En premier lieu, nous avons caractérisé la charge d'Outscale d'après des traces d'exécution collectées sur la plateforme et publiées après anonymisation. Nous avons montré que les pics périodiques de déploiement de VMs et instantanés de volumes stressent l'orchestrateur, la possibilité de sur-allouer l'espace de stockage dépend du placement des volumes de longue durée de vie car ils sont plus à risque de se remplir et d'être sauvegardés, la courte durée des VMs complique l'utilisation des techniques de placement basées sur les migrations, et la sur-allocation du CPU génère des interférences que nous avons quantifiées.

En seconde contribution, nous avons proposé un modèle de prédiction de la durée de vie des VMs à partir de caractéristiques prédictives qui sont, ou pourraient être, disponibles à leur démarrage. En plus de caractéristiques déjà connues, comme la quantité de ressources demandée, et l'historique de consommation de l'utilisateur, nous avons utilisé la configuration réseau de la VM et les étiquettes textuelles attachées à la VM pour permettre son inventaire. Notre modèle obtient une précision mesurée par le score F1 de 0.91, tandis qu'un modèle de la littérature évalué sur la trace de Microsoft Azure avait obtenu 0.74.

Enfin, pour savoir si nous pouvions utiliser ces prédictions pour optimiser le placement des VMs, nous avons évalué la sensibilité d'un algorithme de placement de VMs de la littérature aux erreurs de prédiction. Cet algorithme, appelé RTABF, avait préalablement été évalué en considérant des prédictions de durée de vie parfaites. Nous avons donc évalué, via simulation, la sensibilité de RTABF à différents niveaux d'erreur de prédiction. Notre travail invalide l'utilisation des prédictions de durée de vie avec l'algorithme RTABF.

Dans de futurs travaux, nous chercherons à montrer que les prédictions de durée de vie pourraient aider à réduire le nombre de migrations nécessaires lors des maintenances des serveurs. Les serveurs doivent être mis en maintenance par lots pour éviter de rendre indisponible une trop grande fraction des ressources. Au cours de ce processus, les serveurs prévus pour maintenance arrêtent de recevoir de nouvelles VMs pour minimiser le nombre de migrations, mais ils sont donc sous-utilisés. D'autre part, les VMs de longue durée de vie peuvent être migrées plusieurs fois en passant d'un lot à un autre, et donc faire l'expérience d'une qualité de service dégradée. Nous proposons de placer les VMs de longue durée de vie sur des serveurs qui ne nécessiteront pas de maintenance, tandis que les VMs de courte durée de vie pourraient aller sur les serveurs en attente de maintenance afin de continuer à les utiliser.

# Bibliography


[1] F. Zabatta and K. Ying, "A thread performance comparison: Windows nt and solaris on a symmetric multiprocessor," in *Proc. 2nd USENIX Windows NT Symposium*, 1998, pp. 1–11.

[2] 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*, ser. SOSP '17. New York, NY, USA: ACM, 2017, pp. 153–167. [Online]. Available: http://doi.acm.org/10.1145/3132747.3132772

[3] D. Bernstein, N. Vidovic, and S. Modi, "A cloud paas for high scale, function, and velocity mobile applications - with reference application as the fully connected car," in *2010 Fifth International Conference on Systems and Networks Communications*, Aug 2010, pp. 117–123.

[4] V. Marx, "The big challenges of big data," *Nature*, no. 498, pp. 255–260, June 2013.

[5] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile networks and applications*, vol. 19, no. 2, pp. 171–209, 2014.

[6] P. Mell, T. Grance *et al.*, "The nist definition of cloud computing," 2011.

[7] B. Varghese and R. Buyya, "Next generation cloud computing," *Future Gener. Comput. Syst.*, vol. 79, no. P3, pp. 849–861, Feb. 2018. [Online]. Available: https://doi.org/10.1016/j.future.2017.09.020

[8] "Gartner cloud revenue forecast." [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2018-09-12-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-17-percent-in-2019

[9] W. Vogels, "Beyond server consolidation," *Queue*, vol. 6, no. 1, pp. 20–26, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1348583.1348590

[10] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management*, vol. 23, no. 3, pp. 567–619, Jul 2015. [Online]. Available: https://doi.org/10.1007/s10922-014-9307-7

[11] A. Verma, M. Korupolu, and J. Wilkes, "Evaluating job packing in warehouse-scale computing," in *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, Sept 2014, pp. 48–56.

[12] C. Isci, J. E. Hanson, I. Whalley, M. Steinder, and J. O. Kephart, "Runtime demand estimation for effective dynamic resource management," in *2010 IEEE Network Operations and Management Symposium - NOMS 2010*, April 2010, pp. 381–388.

[13] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: Improving resource efficiency at scale," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, June 2015, pp. 450–462.

[14] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Gener. Comput. Syst.*, vol. 28, no. 5, pp. 755–768, May 2012. [Online]. Available: http://dx.doi.org/10.1016/j.future.2011.04.017

[15] S. Shen, V. v. Beek, and A. Iosup, "Statistical characterization of business-critical workloads hosted in cloud datacenters," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2015, pp. 465–474.

[16] D. Klusáček and B. Parák, "Analysis of mixed workloads from shared cloud infrastructure," in *Job Scheduling Strategies for Parallel Processing*, D. Klusáček, W. Cirne, and N. Desai, Eds. Cham: Springer International Publishing, 2018, pp. 25–42.

[17] S. Challita, F. Paraiso, and P. Merle, "A Study of Virtual Machine Placement Optimization in Data Centers," in *7th International Conference on Cloud Computing and Services Science (CLOSER)*, ScitePress, Ed. Porto, Portugal: INSTICC, Apr. 2017, pp. 343–350.

[18] Z. Mann, "A taxonomy for the virtual machine allocation problem," *International Journal of Mathematical Models and Methods in Applied Sciences*, vol. 9, pp. 269–276, 2015.

[19] V. Kherbache, E. Madelaine, and F. Hermenier, "Planning Live-Migrations to Prepare Servers for Maintenance," in *Euro-Par 2014: Parallel Processing Workshops*, vol. 8806, no. 0302-9743, Porto, Portugal, Aug. 2014, pp. 498 – 507. [Online]. Available: https://hal.inria.fr/hal-01096040

[20] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, "Entropy: A consolidation manager for clusters," in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, ser. VEE '09. New York, NY, USA: ACM, 2009, pp. 41–50. [Online]. Available: http://doi.acm.org/10.1145/1508293.1508300

[21] E. Feller, C. Rohr, D. Margery, and C. Morin, "Energy Management in IaaS Clouds: A Holistic Approach," INRIA, Research Report RR-7946, Apr. 2012. [Online]. Available: https://hal.inria.fr/hal-00692236

[22] Y. Wu and M. Zhao, "Performance modeling of virtual machine live migration," in *2011 IEEE 4th International Conference on Cloud Computing*, July 2011, pp. 492–499.

[23] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation," in *IEEE International Conference on Cloud Computing*. Springer, 2009, pp. 254–265.

[24] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Release-time aware vm placement," in *2014 IEEE Globecom Workshops (GC Wkshps)*, Dec 2014, pp. 122–126.

[25] J. Koomey, S. Berard, M. Sanchez, and H. Wong, "Implications of historical trends in the electrical efficiency of computing," *IEEE Annals of the History of Computing*, vol. 33, no. 3, pp. 46–54, March 2011.

[26] C. Lameter, "Numa (non-uniform memory access): An overview," *Queue*, vol. 11, no. 7, pp. 40:40–40:51, Jul. 2013. [Online]. Available: http://doi.acm.org/10.1145/2508834.2513149

[27] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung, and L. Smith, "Intel virtualization technology," *Computer*, vol. 38, no. 5, pp. 48–56, 2005.

[28] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, Oct. 2003. [Online]. Available: http://doi.acm.org/10.1145/1165389.945462

[29] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," in *Proceedings of the Linux symposium*, vol. 1. Dttawa, Dntorio, Canada, 2007, pp. 225–230.

[30] L. Youseff, R. Wolski, B. Gorda, and C. Krintz, "Paravirtualization for hpc systems," in *International Symposium on Parallel and Distributed Processing and Applications*. Springer, 2006, pp. 474–486.

[31] F. Rodríguez-Haro, F. Freitag, L. Navarro, E. Hernánchez-sánchez, N. Farías-Mendoza, J. A. Guerrero-Ibáñez, and A. González-Potes, "A summary of virtualization techniques," *Procedia Technology*, vol. 3, pp. 267–272, 2012.

[32] M. Mahalingam, D. G. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks." *RFC*, vol. 7348, pp. 1–22, 2014.

[33] G. Lettieri, V. Maffione, and L. Rizzo, "A study of i/o performance of virtual machines," *The Computer Journal*, vol. 61, no. 6, pp. 808–831, 09 2017. [Online]. Available: https://dx.doi.org/10.1093/comjnl/bxx092

[34] Q. Zhu and T. Tung, "A performance interference model for managing consolidated workloads in qos-aware clouds," in *2012 IEEE Fifth International Conference on Cloud Computing*, June 2012, pp. 170–179.

[35] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubramanian, K. Talwar, L. Uyeda, and U. Wieder, "Validating heuristics for virtual machines consolidation," 01 2011.

[36] 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 *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2015, pp. 1–10.

[37] D. Guyon, A.-C. Orgerie, and C. Morin, "Glenda: Green label towards energy proportionality for iaas data centers," in *Proceedings of the Eighth International Conference on Future Energy Systems*, ser. e-Energy '17. New York, NY, USA: ACM, 2017, pp. 302–308. [Online]. Available: http://doi.acm.org/10.1145/3077839.3084028

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

[39] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, Dec. 2008. [Online]. Available: http://doi.acm.org/10.1145/1496091.1496103

[40] W. Fang, X. Liang, S. Li, L. Chiaraviglio, and N. Xiong, "Vmplanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers," *Comput. Netw.*, vol. 57, no. 1, pp. 179–196, Jan. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2012.09.008

[41] H. Jin, T. Cheocherngngarn, D. Levy, A. Smith, D. Pan, J. Liu, and N. Pissinou, "Joint host-network optimization for energy-efficient data center networking," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, May 2013, pp. 623–634.

[42] F. P. Tso, K. Oikonomou, E. Kavvadia, and D. P. Pezaros, "Scalable traffic-aware virtual machine management for cloud data centers," in *2014 IEEE 34th International Conference on Distributed Computing Systems*, June 2014, pp. 238–247.

[43] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *2010 Proceedings IEEE INFOCOM*, March 2010, pp. 1–9.

[44] "What are your spot instance options on aws, azure, and google?" date accessed: 2018-01-30. [Online]. Available: http://sixninesit.com/what-are-your-spot-instances-options-on-aws-azure-and-google/

[45] A. Nadjaran Toosi, F. Khodadadi, and R. Buyya, "Sipaas: Spot instance pricing as a service framework and its implementation in openstack," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 13, pp. 3672–3690, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.3749

[46] Q. Zhang, Q. Zhu, and R. Boutaba, "Dynamic resource allocation for spot markets in cloud computing environments," in *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, Dec 2011, pp. 178–185.

[47] C. Delimitrou and C. Kozyrakis, "Qos-aware scheduling in heterogeneous datacenters with paragon," *ACM Trans. Comput. Syst.*, vol. 31, no. 4, pp. 12:1–12:34, Dec. 2013. [Online]. Available: http://doi.acm.org/10.1145/2556583

[48] L. Tomás and J. Tordsson, "Cloud service differentiation in overbooked data centers," in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, Dec 2014, pp. 541–546.

[49] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes, "Cpi2: Cpu performance isolation for shared compute clusters," in *Proceedings of the 8th ACM European Conference on Computer Systems*, ser. EuroSys '13. New York, NY, USA: ACM, 2013, pp. 379–391. [Online]. Available: http://doi.acm.org/10.1145/2465351.2465388

[50] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing sla violations," in *2007 10th IFIP/IEEE International Symposium on Integrated Network Management*, May 2007, pp. 119–128.

[51] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers," in *2011 Proceedings IEEE INFOCOM*, April 2011, pp. 71–75.

[52] J. Xu and J. Fortes, "A multi-objective approach to virtual machine management in datacenters," in *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ser. ICAC '11. New York, NY, USA: ACM, 2011, pp. 225–234. [Online]. Available: http://doi.acm.org/10.1145/1998582.1998636

[53] Y. Gao, H. Guan, Z. Qi, Y. Hou, and L. Liu, "A multi-objective ant colony system algorithm for virtual machine placement in cloud computing," *J. Comput. Syst. Sci.*, vol. 79, no. 8, pp. 1230–1242, Dec. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.jcss.2013.02.004

[54] Q. Zheng, R. Li, X. Li, and J. Wu, "A multi-objective biogeography-based optimization for virtual machine placement," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2015, pp. 687–696.

[55] F. Hermenier, J. Lawall, and G. Muller, "Btrplace: A flexible consolidation manager for highly available applications," *IEEE Transactions on Dependable and Secure Computing*, vol. 10, no. 5, pp. 273–286, Sept 2013.

[56] A. Rai, R. Bhagwan, and S. Guha, "Generalized resource allocation for the cloud," in *Proceedings of the Third ACM Symposium on Cloud Computing*, ser. SoCC '12. New York, NY, USA: ACM, 2012, pp. 15:1–15:12. [Online]. Available: http://doi.acm.org/10.1145/2391229.2391244

[57] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, "Server workload analysis for power minimization using consolidation," in *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, ser. USENIX'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 28–28. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855807.1855835

[58] Z. Gong and X. Gu, "Pac: Pattern-driven application consolidation for efficient cloud computing," in *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Aug 2010, pp. 24–33.

[59] H. Lin, X. Qi, S. Yang, and S. Midkiff, "Workload-driven vm consolidation in cloud data centers," in *2015 IEEE International Parallel and Distributed Processing Symposium*, May 2015, pp. 207–216.

[60] X. Li, A. Ventresque, J. O. Iglesias, and J. Murphy, "Scalable correlation-aware virtual machine consolidation using two-phase clustering," in *2015 International Conference on High Performance Computing Simulation (HPCS)*, July 2015, pp. 237–245.

[61] Y. Zhao, H. Liu, Y. Wang, Z. Zhang, and D. Zuo, "Reducing the upfront cost of private clouds with clairvoyant virtual machine placement," *The Journal of Supercomputing*, Jan 2019. [Online]. Available: https://doi.org/10.1007/s11227-018-02730-4

[62] S. Srinivasan, U. Bellur, and R. Badrinath, "Debunking the myth that tight packing is energy conserving," in *Proceedings of the 17th International Conference on Distributed Computing and Networking*. ACM, 2016, p. 20.

[63] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: ACM, 2011, pp. 248–259. [Online]. Available: http://doi.acm.org/10.1145/2155620.2155650

[64] R. Xu, S. Mitra, J. Rahman, P. Bai, B. Zhou, G. Bronevetsky, and S. Bagchi, "Pythia: Improving datacenter utilization via precise contention prediction for multiple co-located workloads," in *Proceedings of the 19th International Middleware Conference*, ser. Middleware '18. New York, NY, USA: ACM, 2018, pp. 146–160. [Online]. Available: http://doi.acm.org/10.1145/3274808.3274820

[65] Z. Mann, "Approximability of virtual machine allocation: Much harder than bin packing," in *9th Hungarian-Japanese Symposium on Discrete Mathematics*, 2015.

[66] A. Verma, P. Ahuja, and A. Neogi, "pmapper: Power and migration cost aware application placement in virtualized systems," in *Middleware 2008*, V. Issarny and R. Schantz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 243–264.

[67] M. Kesavan, I. Ahmad, O. Krieger, R. Soundararajan, A. Gavrilovska, and K. Schwan, "Practical compute capacity management for virtualized datacenters," *IEEE Transactions on Cloud Computing*, vol. 1, no. 1, pp. 1–1, Jan 2013.

[68] E. Feller, C. Morin, and A. Esnault, "A case for fully decentralized dynamic vm consolidation in clouds," in *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, Dec 2012, pp. 26–33.

[69] F. Quesnel, A. Lèbre, and M. Südholt, "Cooperative and reactive scheduling in large-scale virtualized platforms with dvms," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1643–1655, June 2012. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.2848

[70] Y. O. Yazir, C. Matthews, R. Farahbod, S. Neville, A. Guitouni, S. Ganti, and Y. Coady, "Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis," in *2010 IEEE 3rd International Conference on Cloud Computing*, July 2010, pp. 91–98.

[71] M. Khelghatdoust, V. Gramoli, and D. Sun, "Glap: Distributed dynamic workload consolidation through gossip-based learning," in *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, Sept 2016, pp. 80–89.

[72] J. O. Gutierrez-Garcia and A. Ramirez-Nafarrate, "Collaborative agents for distributed load management in cloud data centers using live migration of virtual machines," *IEEE Transactions on Services Computing*, vol. 8, no. 6, pp. 916–929, Nov 2015.

[73] C. Mastroianni, M. Meo, and G. Papuzzo, "Probabilistic consolidation of virtual machines in self-organizing cloud data centers," *IEEE Transactions on Cloud Computing*, vol. 1, no. 2, pp. 215–228, July 2013.

[74] S. S. Masoumzadeh and H. Hlavacs, "A cooperative multi agent learning approach to manage physical host nodes for dynamic consolidation of virtual machines," in *2015 IEEE Fourth Symposium on Network Cloud Computing and Applications (NCCA)*, June 2015, pp. 43–50.

[75] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015. [Online]. Available: http://science.sciencemag.org/content/349/6245/255

[76] N. R. Herbst, N. Huber, S. Kounev, and E. Amrehn, "Self-adaptive workload classification and forecasting for proactive resource provisioning," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '13. New York, NY, USA: ACM, 2013, pp. 187–198. [Online]. Available: http://doi.acm.org/10.1145/2479871.2479899

[77] Z. Gong, X. Gu, and J. Wilkes, "Press: Predictive elastic resource scaling for cloud systems," in *2010 International Conference on Network and Service Management*, Oct 2010, pp. 9–16.

[78] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes, "AGILE: Elastic distributed resource scaling for infrastructure-as-a-service," in *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*. San Jose, CA: USENIX, 2013, pp. 69–82. [Online]. Available: https://www.usenix.org/conference/icac13/technical-sessions/presentation/nguyen

[79] J. Xue, F. Yan, R. Birke, L. Y. Chen, T. Scherer, and E. Smirni, "Practise: Robust prediction of data center time series," in *2015 11th International Conference on Network and Service Management (CNSM)*, Nov 2015, pp. 126–134.

[80] S. Di, D. Kondo, and W. Cirne, "Google hostload prediction based on bayesian model with optimized feature combination," *Journal of Parallel and Distributed Computing*, vol. 74, no. 1, pp. 1820 – 1832, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731513002128

[81] A. Khan, X. Yan, S. Tao, and N. Anerousis, "Workload characterization and prediction in the cloud: A multiple time series approach," in *2012 IEEE Network Operations and Management Symposium*, April 2012, pp. 1287–1294.

[82] F. Qiu, B. Zhang, and J. Guo, "A deep learning approach for vm workload prediction in the cloud," in *2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, May 2016, pp. 319–324.

[83] K. LaCurts, J. C. Mogul, H. Balakrishnan, and Y. Turner, "Cicada: Introducing predictive guarantees for cloud networks," in *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)*. Philadelphia, PA: USENIX Association, 2014. [Online]. Available: https://www.usenix.org/conference/hotcloud14/workshop-program/presentation/lacurts

[84] N. J. Yadwadkar, G. Ananthanarayanan, and R. Katz, "Wrangler: Predictable and faster jobs using fewer resources," in *Proceedings of the ACM Symposium on Cloud Computing*,

ser. SOCC '14. New York, NY, USA: ACM, 2014, pp. 26:1–26:14. [Online]. Available: http://doi.acm.org/10.1145/2670979.2671005

[85] J. Liu, H. Shen, and H. S. Narman, "Ccrp: Customized cooperative resource provisioning for high resource utilization in clouds," in *2016 IEEE International Conference on Big Data (Big Data)*, Dec 2016, pp. 243–252.

[86] E. Gaussier, D. Glesser, V. Reis, and D. Trystram, "Improving backfilling by using machine learning to predict running times," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: ACM, 2015, pp. 64:1–64:10. [Online]. Available: http://doi.acm.org/10.1145/2807591.2807646

[87] R. McKenna, S. Herbein, A. Moody, T. Gamblin, and M. Taufer, "Machine learning predictions of runtime and io traffic on high-end clusters," in *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, Sept 2016, pp. 255–258.

[88] C. Canali and R. Lancellotti, "Automatic virtual machine clustering based on bhattacharyya distance for multi-cloud systems," in *Proceedings of the 2013 International Workshop on Multi-cloud Applications and Federated Clouds*, ser. MultiCloud '13. New York, NY, USA: ACM, 2013, pp. 45–52. [Online]. Available: http://doi.acm.org/10.1145/2462326.2462337

[89] R. Zhang, R. Routray, D. M. Eyers, D. Chambliss, P. Sarkar, D. Willcocks, and P. Pietzuch, "Io tetris: Deep storage consolidation for the cloud via fine-grained workload analysis," in *2011 IEEE 4th International Conference on Cloud Computing*, July 2011, pp. 700–707.

[90] A. Lebre, J. Pastor, A. Simonet, and M. Südholt, "Putting the next 500 vm placement algorithms to the acid test: The infrastructure provider viewpoint," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 1, pp. 204–217, Jan 2019.

[91] M. C. Calzarossa, L. Massari, and D. Tessera, "Workload characterization: A survey revisited," *ACM Comput. Surv.*, vol. 48, no. 3, pp. 48:1–48:43, Feb. 2016. [Online]. Available: http://doi.acm.org/10.1145/2856127

[92] R. Wolski and J. Brevik, "Using parametric models to represent private cloud workloads," *IEEE Transactions on Services Computing*, vol. 7, no. 4, pp. 714–725, Oct 2014.

[93] D. Milojičić, I. M. Llorente, and R. S. Montero, "Opennebula: A cloud management tool," *IEEE Internet Computing*, vol. 15, no. 2, pp. 11–14, March 2011.

[94] I. Cano, S. Aiyar, and A. Krishnamurthy, "Characterizing private clouds: A large-scale empirical analysis of enterprise clusters," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*, ser. SoCC '16. New York, NY, USA: ACM, 2016, pp. 29–41. [Online]. Available: http://doi.acm.org/10.1145/2987550.2987584

[95] C. Peng, M. Kim, Z. Zhang, and H. Lei, "VDN: Virtual machine image distribution network for cloud data centers," in *INFOCOM '12*.

[96] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format + schema," Google Inc., Mountain View, CA, USA, Technical Report, Nov. 2011, revised 2014-11-17 for version 2.1. Posted at https://github.com/google/cluster-data.

[97] C. Lu and al., "Imbalance in the cloud: An analysis on alibaba cluster trace," in *2017 IEEE International Conference on Big Data (Big Data)*, 2017.

[98] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, March 2015, pp. 171–172.

[99] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of the Third ACM Symposium on Cloud Computing*, ser. SoCC '12. New York, NY, USA: ACM, 2012, pp. 7:1–7:13. [Online]. Available: http://doi.acm.org/10.1145/2391229.2391236

[100] S. Di, D. Kondo, and W. Cirne, "Characterization and comparison of cloud versus grid workloads," in *2012 IEEE International Conference on Cluster Computing*, Sept 2012, pp. 230–238.

[101] 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)*. Boston, MA: USENIX Association, 2018, pp. 533–546. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/amvrosiadis

[102] I. S. Moreno, P. Garraghan, P. Townend, and J. Xu, "Analysis, modeling and simulation of workload patterns in a large-scale utility cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 208–221, April 2014.

[103] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, "Towards characterizing cloud backend workloads: Insights from google compute clusters," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 4, pp. 34–41, Mar. 2010. [Online]. Available: http://doi.acm.org/10.1145/1773394.1773400

[104] F. Dumont and J.-M. Menaud, "Synthesizing realistic cloudworkload traces for studying dynamic resource system management," in *Revised Selected Papers of the Second International Conference on Cloud Computing and Big Data - Volume 9106*, ser. CloudCom-Asia 2015. Berlin, Heidelberg: Springer-Verlag, 2015, pp. 29–41. [Online]. Available: https://doi.org/10.1007/978-3-319-28430-9_3

[105] R. Birke, L. Y. Chen, and E. Smirni, "Multi-resource characterization and their (in)dependencies in production datacenters," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–6.

[106] C. Kilcioglu, J. M. Rao, A. Kannan, and R. P. McAfee, "Usage patterns and the economics of the public cloud," in *Proceedings of the 26th International Conference on World Wide Web*, ser. WWW '17. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2017, pp. 83–91. [Online]. Available: https://doi.org/10.1145/3038912.3052707

[107] X. Chen, C. Lu, and K. Pattabiraman, "Failure analysis of jobs in compute clouds: A google cluster case study," in *2014 IEEE 25th International Symposium on Software Reliability Engineering*, Nov 2014, pp. 167–177.

[108] B. Schroeder, E. Pinheiro, and W.-D. Weber, "Dram errors in the wild: A large-scale field study," *Commun. ACM*, vol. 54, no. 2, pp. 100–107, Feb. 2011. [Online]. Available: http://doi.acm.org/10.1145/1897816.1897844

[109] E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure trends in a large disk drive population," in *5th USENIX Conference on File and Storage Technologies (FAST 2007)*, 2007, pp. 17–29.

[110] K. He, A. Fisher, L. Wang, A. Gember, A. Akella, and T. Ristenpart, "Next stop, the cloud: Understanding modern web service deployment in ec2 and azure," in *Proceedings of the 2013 Conference on Internet Measurement Conference*, ser. IMC '13. New York, NY, USA: ACM, 2013, pp. 177–190. [Online]. Available: http://doi.acm.org/10.1145/2504730.2504740

[111] P. Leitner and J. Cito, "Patterns in the Chaos: A Study of Performance Variation and Predictability in Public IaaS Clouds," *ACM TOIT '16*.

[112] I. Habib, "Virtualization with kvm," *Linux Journal '08*.

[113] P. Domingos, "A few useful things to know about machine learning," *Commun. ACM*, vol. 55, no. 10, pp. 78–87, Oct. 2012. [Online]. Available: http://doi.acm.org/10.1145/2347736.2347755

[114] C. E. Brodley and P. E. Utgoff, "Multivariate decision trees," *Machine Learning*, vol. 19, no. 1, pp. 45–77, 1995. [Online]. Available: https://doi.org/10.1007/BF00994660

[115] B. Leo, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*, 1984.

[116] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, Mar 1986. [Online]. Available: https://doi.org/10.1007/BF00116251

[117] W.-Y. Loh, "Fifty years of classification and regression trees," *International Statistical Review*, vol. 82, no. 3, pp. 329–348, 2014. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/insr.12016

[118] G. Brown, *Ensemble Learning*. Boston, MA: Springer US, 2010, pp. 312–320.

[119] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[120] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, no. 1, pp. 3–42, Apr 2006. [Online]. Available: https://doi.org/10.1007/s10994-006-6226-1

[121] R. E. Schapire, *Explaining AdaBoost*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 37–52. [Online]. Available: https://doi.org/10.1007/978-3-642-41136-6_5

[122] M. Allahyari, S. Pouriyeh, M. Assefi, S. Safaei, E. D. Trippe, J. B. Gutierrez, and K. Kochut, "A brief survey of text mining: Classification, clustering and extraction techniques," *arXiv preprint arXiv:1707.02919*, 2017.

[123] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling using system-generated predictions rather than user runtime estimates," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 789–803, June 2007.

[124] T. Kopinski, S. Magand, U. Handmann, and A. Gepperth, "A pragmatic approach to multi-class classification," in *2015 International Joint Conference on Neural Networks (IJCNN)*, July 2015, pp. 1–8.

[125] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler *et al.*, "Api design for machine learning software: experiences from the scikit-learn project," *arXiv preprint arXiv:1309.0238*, 2013.

[126] C. X. Ling and V. S. Sheng, *Class Imbalance Problem*. Boston, MA: Springer US, 2010, pp. 171–171. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_110

[127] P. Svärd, W. Li, E. Wadbro, J. Tordsson, and E. Elmroth, "Continuous datacenter consolidation," in *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, Nov 2015, pp. 387–396.

[128] P. Minet, E. Renault, I. Khoufi, and S. Boumerdassi, "Analyzing traces from a google data center," in *2018 14th International Wireless Communications Mobile Computing Conference (IWCMC)*, June 2018, pp. 1167–1172.

[129] W. Lin, W. Wu, H. Wang, J. Z. Wang, and C.-H. Hsu, "Experimental and quantitative analysis of server power model for cloud data centers," *Future Generation Computer Systems*, 2016.

[130] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 732–794, 2016.

[131] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2. ACM, 2007, pp. 13–23.

[132] S. Wang, Z. Liu, Z. Zheng, Q. Sun, and F. Yang, "Particle swarm optimization for energy-aware virtual machine placement optimization in virtualized data centers," in *Parallel and Distributed Systems (ICPADS), 2013 International Conference on*. IEEE, 2013, pp. 102–109.

[133] "Cisco ucs power calculator." [Online]. Available: https://ucspowercalc.cloudapps.cisco.com

[134] A. Orgerie, L. Lefèvre, and J. Gelas, "Chasing gaps between bursts: Towards energy efficient large scale experimental grids," in *2008 Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies*, Dec 2008, pp. 381–389.

[135] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Towards understanding heterogeneous clouds at scale: Google trace analysis," *Intel Science and Technology Center for Cloud Computing, Tech. Rep*, vol. 84, 2012.

[136] F. Farahnakian, P. Liljeberg, and J. Plosila, "Energy-efficient virtual machines consolidation in cloud data centers using reinforcement learning," in *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, Feb 2014, pp. 500–507.

Loïc PÉRENNOU

# Virtual Machine Experience Design: A Predictive Resource Allocation Approach for Cloud Infrastructures

Résumé : L'un des principaux défis des fournisseurs de services cloud est d'offrir aux utilisateurs une performance acceptable, tout en minimisant les besoins en matériel et énergie. Dans cette thèse CIFRE menée avec Outscale, un fournisseur de cloud, nous visons à optimiser l'allocation des ressources en utilisant de nouvelles sources d'information. Nous caractérisons la charge de travail pour comprendre le stress résultant sur l'orchestrateur, et la compétition pour les ressources disponibles qui dégrade la qualité de service. Nous proposons un modèle pour prédire la durée d'exécution des VMs à partir de caractéristiques prédictives disponibles au démarrage. Enfin, nous évaluons la sensibilité aux erreurs d'un algorithme de placement des VMs de la littérature qui se base sur ces prédictions. Nous ne trouvons pas d'intérêt à coupler note système prédictif avec cet algorithme, mais nous proposons d'autres façons d'utiliser les prédictions pour optimiser le placement des VMs.

Mots clés : Infrastructure à la demande, placement de machine virtuelle, plateforme cloud, orchestrateur

Abstract : One of the main challenges for cloud computing providers remains to offer trustable performance for all users, while maintaining an efficient use of hardware and energy resources. In the context of this CIFRE thesis lead with Outscale, a public cloud provider, we perform an in-depth study aimed at making management algorithms use new sources of information. We characterize Outscale's workload to understand the resulting stress for the orchestrator, and the contention for hardware resources. We propose models to predict the runtime of VMs based on features which are available when they start. We evaluate the sensitivity with respect to prediction error of a VM placement algorithm from the literature that requires such predictions. We do not find any advantage in coupling our prediction model and the selected algorithm, but we propose alternative ways to use predictions to optimize the placement of VMs.

Keywords : Infrastructure as a Service, virtual machine placement, cloud platform, orchestrator