Automated Application and Resource Management in the Cloud

Nikos Parlavantzas

To cite this version:

Nikos Parlavantzas. Automated Application and Resource Management in the Cloud. Distributed, Parallel, and Cluster Computing [cs.DC]. Universite de Rennes 1, 2020. tel-03369179v2

HAL Id: tel-03369179

https://hal.inria.fr/tel-03369179v2

Submitted on 28 Apr 2022

HABILITATION Á DIRIGER DES RECHERCHES

UNIVERSITÉ DE RENNES 1

Mention : Informatique

École doctorale MathSTIC

présentée par

Nikos Parlavantzas

préparée à l'unité de recherche 6074 - IRISA
Institut de Recherche en Informatique et Systèmes Aléatoires

Automated Application and Resource Management in the Cloud

HDR soutenue le 15 juin 2020 à Rennes

devant le jury composé de :

Rosa Badia
Manager, BSC / Rapporteuse

Gordon Blair
Professor, Lancaster University / Examinateur

Frédéric Desprez
Directeur de recherches, Inria / Rapporteur

Christine Morin
Directrice de Recherche, Inria / Examinatrice

Guillaume Pierre
Professeur, Université de Rennes 1 / Président

Pierre Sens
Professeur, Sorbonne Université / Rapporteur

Contents

1 Introduction 1

1.1 Cloud Computing 1

1.1.1 Concepts 2

1.1.2 Technology and Business Aspects 3

1.2 Research Challenges 4

1.2.1 Providers 4

1.2.2 Customers 5

1.3 Research Activities 6

2 Resource Management for Providers 9

2.1 Resource and Execution Management for SaaS Providers 9

2.1.1 SaaS System Model 10

2.1.2 Service Delivery Mechanisms 11

2.1.3 Qu4DS Prototype 12

2.1.4 Evaluation 13

2.1.5 Discussion 14

2.2 SLA-based Resource Management for PaaS Providers 15

2.2.1 PaaS System Model 16

2.2.2 Resource Sharing Policies 17

2.2.3 Computational Applications 17

2.2.4 Meryn Prototype 20

2.2.5 Evaluation 22

2.2.6 Discussion 24

3 Application Management for Customers 27

3.1 Deploying and Managing Multi-cloud Applications 28

3.1.1 PaaSage Adapter 29

3.1.2 Cost-effective Adaptive Deployment for Multi-cloud Applications . 32

3.1.3 SPL-based Application Management with a Proactive Policy for Vertical Scaling 37

3.1.4 Discussion 39

3.2 A Service-based Framework for Epidemic Simulation Applications 40

3.2.1 DiFFuSE Framework 41

3.2.2 Evaluation 42

3.2.3 Discussion 44

4 Application and Resource Management in Private Clouds 47

4.1 PaaS System Model 48

4.2 Auction Implementation 50

4.3 Adaptation Policies 53

4.3.1 Vertical Scaling Policy 53

4.3.2 Horizontal Scaling Policy 54

4.4 Merkat Prototype 54

4.5 Evaluation 55

4.5.1 Simulations with Large Traces 55

4.5.2 Testbed Experiments 56

4.6 Discussion 58

5 Conclusion 59

5.1 Synopsis 59

5.2 Outlook 60

5.2.1 Beyond Traditional Clouds 60

5.2.2 Improved Decision-Making 63

5.2.3 Decentralised Management 63

Bibliography 65

List of Figures

2.1 Qu4DS architecture 13
2.2 Heterogeneous customer profile with three penalty policies 14
2.3 Safe customer profile with three penalty policies 14
2.4 Meryn architecture 21
2.5 Profit for different workloads and policies 23
2.6 VM usage proportion for different workloads and policies (experiments)—VM usage is calculated as number of VMs multiplied by duration 24
2.7 Workload completion time from the submission of the first job to the completion of the last job (experiments) 24
3.1 Application management architecture 28
3.2 PaaSage architecture 30
3.3 Adapter architecture 31
3.4 Architecture for continuous deployment optimisation 35
3.5 Response time, throughput, and profit with the validator switched off (Scenario 1) and on (Scenario 2) 36
3.6 XIPE architecture 38
3.7 XIPE proactive adaptation 39
3.8 DiFFuSE architecture 41
3.9 Data transfer time for different numbers of workers 44
3.10 Amount of data transferred inside (I) and outside (O) the public cloud, with replication enabled (E) and disabled (D), for different numbers of workers . . 44
4.1 Overview of the market-based approach 49
4.2 Merkat prototype 54
4.3 Performance in terms of (a) total customer satisfaction, and (b) percentage of successfully finished applications for Merkat, FCFS, and EDF—contention increases from left to right. 57

List of Tables

2.1 Percentage of delayed applications (A) and average delay of delayed applications (B) with the optimisation policy (experiments) 25
4.1 Comparison of Merkat, Maui, and Merkat's simulation in CloudSim. 57

1 Introduction

Cloud computing has become a prominent feature of the modern computing landscape. Industry analysts predict that almost 30% of total IT expenditure will shift to the cloud by 2022, while the global cloud market will reach \$330 billion in annual revenues in 2022, attaining a compound annual growth rate of 12.6% between 2018 and 2022 [55, 56]. Cloud computing is a model in which providers own and manage configurable computing resources (e.g., networks, servers, storage, applications, services), and customers access them on demand, typically on a pay-per-use basis. The model brings benefits to both providers and customers [125]. Providers pool and share computing resources among multiple customers, which leads to increased resource utilisation and reduced costs of resource provision. Customers obtain and release resources on demand while paying only for actual resource use, which leads to reduced costs of using resources. To optimise the value that providers and customers draw from the cloud, these actors must continuously manage cloud resources and applications in response to changes in their operating environments.

This document presents my research activities conducted between 2009 and 2019 within the MYRIADS project-team (IRISA and Inria Rennes – Bretagne Atlantique) focused on cloud management. The overall goal of these activities was *to develop methods and tools to assist cloud actors in managing cloud resources and applications in line with their objectives.*

This chapter is structured as follows. First, Section 1.1 presents an overview of cloud computing concepts and related technology and business aspects. Section 1.2 then discusses the main research challenges that guided my research. Finally, Section 1.3 summarizes my research activities and outlines the remainder of this document.

1.1 Cloud Computing

We present next some fundamental aspects of cloud computing in order to establish a common, basic vocabulary.

1.1.1 Concepts

This document adopts the definition of cloud computing provided by NIST [99]:

> *Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*

According to NIST, the essential characteristics of the cloud computing model are as follows:

- *On-demand self-service*—cloud customers can provision computing capabilities, such as server time and network storage, automatically, without requiring human interaction.
- *Broad network access*—capabilities are available over the network and accessed through heterogeneous client platforms, such as mobile devices and workstations.
- *Resource pooling*—provider resources are pooled to serve multiple customers.
- *Rapid elasticity*—capabilities can be rapidly provisioned and released in response to demand.
- *Measured service*—resource usage is continuously monitored, controlled and reported.

The services provided by cloud providers are classified as follows:

- *Software-as-a-Service (SaaS)*—customers access applications running on the provider's infrastructure; customers have no control over these applications or the infrastructure.
- *Platform-as-a-Service (PaaS)*—customers deploy on the provider's infrastructure applications created or acquired by customers; customers have control over the deployed applications, but not over the underlying infrastructure.
- *Infrastructure-as-a-Service (IaaS)*—customers provision fundamental computing resources (e.g., processing, storage, and network resources) in order to deploy and run arbitrary software, including applications and operating systems.

We use the general term *cloud system* (or cloud) to describe a system that offers cloud services. Note that cloud systems may offer services to one another. For example, a SaaS system may use services provided by a PaaS system that in turn uses services provided by an IaaS system.

A *cloud provider* (or provider) is a person or organisation that owns and operates a cloud to provide cloud services. A *cloud customer* (or customer) is a person or organisation that uses a cloud. For example, a cloud customer may deploy *cloud applications* on a PaaS or IaaS system (but not a SaaS).

We distinguish the following models of deploying and using clouds:

- *Private cloud*—a cloud available for exclusive use by a single organisation.
- *Community cloud*—a cloud available for use by a specific community of consumers.
- *Public cloud*—a cloud open to use to the public.
- *Hybrid-cloud*—a combination of multiple, independent clouds enabling data and application portability. Hybrid clouds may be used to support *cloud bursting*; that is, allowing applications that run in a private cloud to burst out into public clouds, when needed. The related concept of *multi-cloud* (or, cross-cloud) refers to a customer using multiple clouds at the same time.

1.1.2 Technology and Business Aspects

The main enabling technology of cloud computing is *virtualisation*, which enables the creation of virtual resources by multiplexing physical resources. A common form of virtualisation creates *virtual machines (VMs)* that emulate physical machines and execute operating systems. VMs are composed of virtual resources (e.g., memory, CPU, storage). VMs and virtual resources are typically the computing resources delivered by IaaS clouds.

Virtualisation brings several technical benefits. First, it facilitates sharing physical infrastructures among multiple users while isolating users from each other. Second, it introduces flexibility in resource management. For example, it enables modifying the resource allocations of VMs in a fine-grained manner (e.g., modifying the memory of a VM) as well as migrating VMs between physical machines. Third, it enables encapsulating applications in isolated packages with customised software stacks.

A core feature of cloud computing is the delivery of computing resources as a service. Customers select cloud services based on the provided qualities of service (QoS) and prices. *QoS* typically covers non-functional service characteristics, such as performance, availability, and reliability. We use the term *Service Level Objective (SLO)* to express requirements on QoS (e.g., the monthly uptime percentage for a service should be greater than 99.99%). Naturally, customers prefer that cloud systems provide strong guarantees on provided QoS. Such guarantees may be included in *Service-level Agreements (SLAs)*, which are contracts that formalize the terms of the provider-customer interaction. SLAs include the SLOs that the provider promises to meet and the price that the customer agrees to pay. The SLAs may also specify penalties paid by the provider when SLOs are violated.

There are various ways in which cloud services are priced. The most popular pricing model is the *pay-per-use* (or pay-as-you-go) model, in which the charge paid by the customer depends on service usage (e.g., paying per second of use of a VM instance). Another model is *subscription* pricing, in which the charge is fixed in advance for a given subscription period. *Static pricing* refers to models in which prices are fixed or change at a slow rate. *Dynamic pricing* refers to models in which prices fluctuate, for example, based on supply and demand. An example is Amazon EC2 Spot Instances pricing, in which customers bid a maximum price and run VM instances as long as the fluctuating spot price is lower than their bid and capacity is available [5].

1.2 Research Challenges

Next I outline the main challenges that have guided my research activities. The challenges are divided into two parts, focusing on automated management on behalf of providers and of customers.

1.2.1 Providers

Cloud providers require automated resource management systems that continuously control the allocation of virtual or physical resources to customer requests. The main challenges in designing such systems are as follows:

Satisfying provider objectives Resource management must deal with the changing environment in which the cloud system operates while effectively pursuing provider objectives. This environment includes fluctuating workloads imposed by customers, varying levels of performance provided by underlying infrastructures (e.g. shared resources, networks), and varying QoS and prices provided by external cloud services. Providers have multiple objectives that must be considered simultaneously (e.g., maintaining SLAs and minimising costs) and are subject to various constraints (e.g., resource capacities). The main challenge is thus ensuring that the objectives are satisfied despite the environment fluidity.

Depending on the type of cloud, provider objectives may take various forms. For public cloud providers, the main objective is optimising profit, which requires some combination of increasing revenues and reducing the costs of using underlying resources. Another important objective is maintaining the SLAs with customers. SLA violations typically reduce short-term profit because of incurred penalties, but they also reduce the long-term attractiveness of the cloud service. For private cloud providers, a typical objective is achieving a resource allocation with desired properties, such as economic efficiency. Economic efficiency here means that the resources are allocated to the customers that value them the most (more precisely, the total value delivered to customers is maximised).

Supporting an extensible set of applications and SLOs To increase the value of their service, providers should respond to the diversity and evolution of customer needs. This is a particular concern for PaaS providers. Indeed, these providers aim to deliver a complete development and hosting environment for applications while the application types popular with their customers are constantly evolving (e.g., web, data analytics, machine learning applications). An important challenge in designing PaaS management systems is thus supporting extensibility with respect to application types.

Different application types are associated with different QoS properties and methods of ensuring that these properties are satisfied. Along with application types, the management system should thus also be able to support associated QoS properties and methods. For example, the system should be able to support HPC applications along with SLOs related to completion time, web applications along with SLOs related to response time, or streaming applications along with SLOs related to latency and throughput.

Supporting scalability with regard to resources and customers Providers are operating a shared infrastructure that serves multiple customers. Depending on the cloud system, the size of this infrastructure may vary from a few machines to the multiple, globally distributed data centres of enterprise cloud providers. Therefore, an important concern for the management system is its scalability; that is, its ability to handle growing numbers of resources and customers. Scalability can be increased, for example, by applying efficient decision-making algorithms or by applying decentralised management structures that involve multiple decision-making units, each having only partial information on the system.

1.2.2 Customers

Cloud customers require automated application management systems that continuously control the deployment of their applications on cloud services. We focus, specifically, on PaaS and IaaS customers since SaaS customers have no direct control over applications. The main challenges in designing such management systems are as follows:

Satisfying customer objectives Modern distributed applications have complex structures imposing multiple constraints on the required underlying infrastructure (e.g., resource requirements of individual components, requirements for sharing resources between components). These constraints are subject to change as the workloads and requirements of end users change. At the same time, an immense range of options is available for hosting an application, comprising different services (e.g., computation, storage) with different characteristics (e.g., VM types) from different cloud providers. The availability, cost, and performance of any selected cloud infrastructure are also subject to change (e.g., network failures, price changes, performance variations due to resource sharing). Ensuring that customer objectives are satisfied despite this complexity and dynamism is thus the central challenge in designing application management systems.

Several concerns arise in overcoming this challenge. First, the management system should be flexible to support different customer objectives. Customer objectives typically include achieving the SLOs of application users as well as reducing the charges paid to IaaS or PaaS providers for application execution. Second, the management system should take into account the potential costs of reconfiguration actions. Indeed, adapting a deployment requires executing reconfiguration actions (e.g., migrating components to different clouds), potentially expensive in terms of time and monetary costs. These costs should be carefully balanced against predicted benefits before adapting the deployment. Third, the management solution should allow customers to deploy an application across multiple clouds. Potential benefits of multi-cloud deployment include avoiding dependence on a single cloud provider, taking advantage of lower resource prices or resource proximity, and enhancing application availability.

Enabling legacy applications to fully exploit clouds The simplest way to migrate a legacy application to the cloud is encapsulating the application in one or more VMs that run in an IaaS cloud. However, this type of migration does not allow realising the full benefits of the cloud in terms of elasticity, performance, and cost savings. Realising those benefits invariably requires restructuring the legacy application to fully exploit cloud services. Application management could then extend to supporting customers in restructuring legacy applications and running them in the cloud. To facilitate this restructuring, customers should be provided with tools, guidelines, and reusable code. The application deployment should then be continually managed to optimise application performance and cost, as described previously. Since failures are common in cloud environments, automated support for dealing with failures should also be included.

1.3 Research Activities

My activities can be structured into three research strands:

1. *Resource management for providers.* The first strand focused on provider-side management. We proposed two automated solutions, one for SaaS providers and one for PaaS providers. In both cases, the provider objective is to increase profit. The SaaS solution uses a single IaaS cloud as the infrastructure. The PaaS solution is extensible with respect to application types and uses as an infrastructure a private IaaS cloud that automatically bursts into public IaaS clouds.

2. *Application management for customers.* The second strand focused on customer-side management. First, we proposed multi-cloud application management solutions for IaaS customers. The considered customer objectives are increasing performance and reducing cost. The applications were assumed to be designed to run in the cloud. Following that, we focused on a particular type of legacy applications, that is, epidemic simulation applications, and proposed a solution for enabling them to fully exploit cloud platforms.

3. *Application and resource management in private clouds.* The previous two research strands studied systems that independently serve the interests of either providers or customers. This strand focused on designing a collaboration structure that allows the customers and the provider to jointly serve their interests. The assumed setting was that of a private PaaS system in which the objective of the provider is to efficiently distribute resources between customers, taking into account selfish customers. Customers have various, customer-specific objectives and run an open-ended set of application types.

Document Outline

The following three chapters discuss each research strand in turn. They present the context of each piece of research work, the main contributions, and how the contributions were evaluated. The final chapter summarizes the key findings and discusses areas for future study.

2 Resource Management for Providers

The runtime operation of cloud systems should be automatically driven not only by QoS and technical goals but also by economic goals. This observation underlies the research described in this chapter. Specifically, the chapter discusses automated resource management solutions for public cloud providers. Unlike most commercial cloud providers, the providers assumed in this work offer strong QoS support. In particular, they support establishing and maintaining SLAs that include QoS expectations (e.g., response time constraints), prices, and penalties to be paid when these expectations are violated. At the same time, being commercial entities, the providers have the objective to increase profitability. To assist such providers with delivering their services, we proposed management solutions that take into account SLAs and infrastructure costs and take actions to increase provider profit while coping with changing workloads.

Specifically, the chapter presents two solutions.

- The first targets SaaS providers and uses resources provided by a single IaaS cloud. This work was performed in the context of André Lage's PhD thesis [81] that I co-supervised with Jean-Louis Pazat (IRISA).
- The second targets PaaS providers, uses resources provided by a private IaaS cloud and bursts into public clouds, if necessary. This work was performed in the context of Djawida Dib's PhD thesis [39] that I co-supervised with Christine Morin (Inria).

We discuss each solution in turn in the following.

2.1 Resource and Execution Management for SaaS Providers

The initial motivation for this work was developing a QoS assurance solution for services deployed on grid infrastructures [83, 85, 86]. The increasing popularity of clouds prompted us to add IaaS systems as a target infrastructure for deploying services, which then led us to consider the economic goals of the service provider. We thus extended the solution to support SLAs specifying both QoS and prices and to actively seek to maximize the provider's profit [81, 82, 84]. The main novelty of this work was the provision of a complete SLA management solution oriented towards increasing the provider profit. Indeed, although a significant amount of research had investigated QoS management for large-scale distributed applications [14, 65, 73], there was little research in explicitly addressing the provider's business objectives. Moreover, although a few systems employed similar QoS and resource management mechanisms [93, 94] or had flexible architectures, potentially enabling them to integrate such mechanisms [24, 45, 54], our work was the first to describe the design of these mechanisms, to integrate them into a real platform, and to perform a detailed evaluation of their impact on the provider profit.

This work has led to two main contributions [81, 82, 83, 84, 85, 86]. First, we proposed an approach for automatically managing cloud resources to increase the profit of SaaS providers. The approach enables processing service workloads reliably and cost-effectively, taking into account operational costs and multiple, simultaneous SLAs with different customers, with each SLA defining expected QoS, prices, and penalties. Second, the work proposed a prototype, called Qu4DS, that applies this approach to support developing and managing SaaS applications.

In the following, I discuss the system model underpinning the approach and the mechanisms provided by the approach. I then provide an overview of the prototype and the evaluation before concluding with a brief discussion.

2.1.1 SaaS System Model

The approach considers a SaaS system that integrates an application and makes it accessible to customers. The application is based on the master/worker pattern with an adjustable number of workers. Each customer has its own SLA (contract) with the provider, specifying a price, the expected levels of QoS for executing customer requests, and the penalties paid by the provider when these levels are violated. To execute a customer request, the SaaS system uses resources provided by a private or public IaaS cloud. Resources are assumed to be homogeneous (i.e., VMs with the same number of cores and memory sizes) and charged per time used. Resources can be shared among workers from different contracts. The provider objective is profit optimisation. The profit takes into account the cost of underlying resources over time as well as the penalties.

2.1.2 Service Delivery Mechanisms

The approach provides mechanisms for creating SLAs, managing resources, and managing the execution of services.

SLA Creation

The proposed approach defines different *SLA types* (i.e., fast, safe, classic, standard) by combining different levels (i.e., high, medium, low) of response time and reliability QoS. The QoS levels are translated to mechanism-level configurations that enable the system to deliver the targeted QoS level. Specifically, response time levels are translated to the number of workers used for service execution. This translation is done by profiling the service with different numbers of workers and classifying response time values as high, medium, and low. Reliability levels are translated to the number of times that faulty workers are replaced. For example, if reliability is high, the system replaces both delayed and crashed workers; if reliability is medium, it only replaces crashed workers.

Each SLA defines a price that depends on the SLA type and the required duration of the contract. The prices are defined in a cost-based manner; that is, the more resources are required, the higher the prices are. The SLA also defines penalties that are paid by the provider to customers when SLOs are violated. There are two kinds of penalties: *contract rescission* penalties for rescinding ongoing contracts, and *request cancellation* penalties for failing to process individual requests.

Resource Management

Resource management seeks to control the number of ongoing contracts and requests in order to increase the provider profit. The proposed approach separates the assignment of resources to each customer (termed *resource booking*) and the assignment of resources to each customer request (termed *resource allocation*). To assist in these tasks, the approach employs two mechanisms, namely, *under-provisioning* (that is, booking less resources than required) and *contract rescission* (that is, canceling ongoing contracts).

Resource booking is triggered by new contract proposals. It determines the amount of resources to be booked using an under-provisioning policy (e.g., booking 70% of required resources). In case of resource shortage (e.g., in private cloud settings), resource booking decides whether to rescind an ongoing contract and accept the proposal or reject the proposal. The decision takes into account the estimated profit of accepting the proposal and the penalty for contract rescission, aiming to increase the provider profit. Resource allocation is triggered by request arrivals. When there are no available resources, resource allocation decides whether to cancel ongoing requests to free resources for the new request. Similarly, this decision takes into account resource requirements and request cancellation penalties and aims to increase the provider profit.

Execution Management

Execution management seeks to enforce the agreed reliability QoS, thus reducing the payment of penalties. Specifically, the approach relies on detecting worker failures, which can be either worker crashes or delays. *Crashes* represent unsuccessful worker executions (e.g., I/O errors), and *delays* represent executions whose duration exceeds the expected duration according to profiling data. Execution management reacts to such failures through the *replacement* of crashed and delayed workers. Replacement decisions rely on the number of times that workers were replaced, constrained by thresholds, and on whether replacement is expected to satisfy response time constraints.

2.1.3 Qu4DS Prototype

To show that it is feasible to build a management system for SaaS providers following the approach and to enable the evaluation of the approach, we built a research prototype, the *Quality Assurance for Distributed Services (Qu4DS)* framework [53]. The Qu4DS architecture is depicted in Figure 2.1.

Qu4DS includes three main components: SLA management, resource management, and execution management. *SLA management* includes functions to assist in SLA creation, QoS translation, and negotiation. Qu4DS maintains templates for each SLA type based on different levels of QoS. QoS translation relies on automatically profiling service executions. Negotiation relies on a simple negotiation protocol in which customers choose an SLA type (template), set the contract duration and propose it to the service which may accept it or not. *Resource management* controls the amount of resources used for service execution. Finally, *execution management* detects and handles failures and delays during execution.

Qu4DS-enabled applications must follow the master/worker pattern. Developers implement two methods: one method that receives a service request and triggers the deployment of workers, and one method that receives the results from all workers. Qu4DS is implemented in Java and the service interface is implemented as a SOAP Web service. To interact with IaaS clouds, Qu4DS uses a SAGA-based API [58].

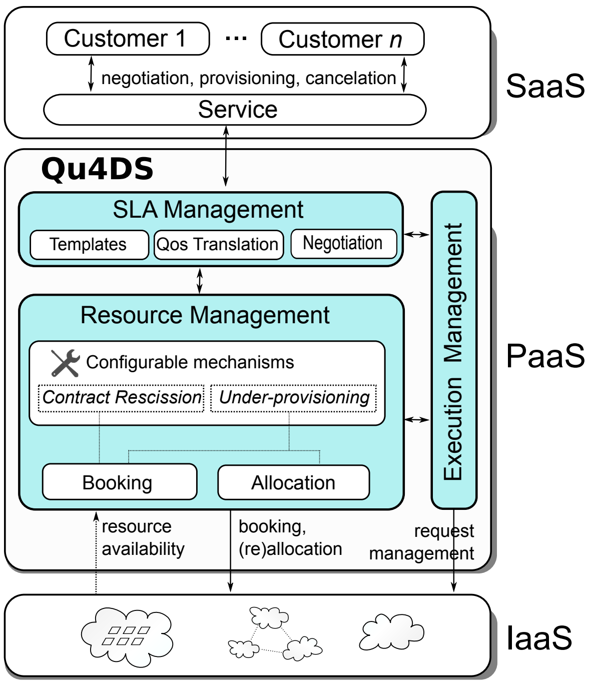


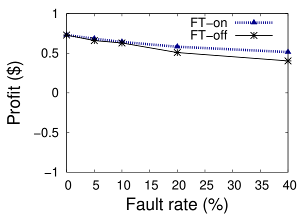
Figure 2.1: Qu4DS architecture

2.1.4 Evaluation

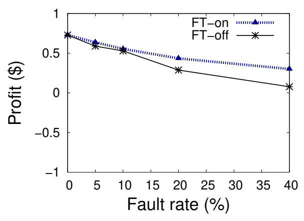
The goal of the evaluation was to analyze the impact of the under-provisioning, contract rescission, and worker replacement mechanisms on the service provider profit. The evaluation relied on experiments with an audio encoding service developed with Qu4DS. The experiments considered various request loads, fault rates, infrastructure capacities, and *customer profiles*, defined as sets of contracts with associated SLA types. The SLA types are based on different combinations of levels of guarantees. For example, the *safe* SLA type defines strong guarantees on reliability and weaker guarantees on response time. The customer profiles were designed to reflect probable real-word workloads.

The results showed that applying the previously mentioned mechanisms increases the provider profit. Specifically, the results showed that under-provisioning increases profit by 20-50%, contract rescission increases profit up to 4 times, and worker replacement increases profit by up to 60%. The experiments were performed in the Grid'5000 [18] infrastructure using a total of 51 nodes. Full details on these experiments can be found in [82]. As an illustration, Figures 2.2 and 2.3 depict the results of fault-tolerance (FT) scenarios. The figures show the generated profit for two customer profiles (heterogeneous and safe) and for three penalty policies (half, full, and double reimbursement). The *heterogeneous* profile contains an equal number of contracts of each SLA type, while the *safe* profile contains only contracts with strong reliability requirements (i.e., of safe SLA type). The *half* reimbursement policy means that the penalty that providers pay is half the contract price.

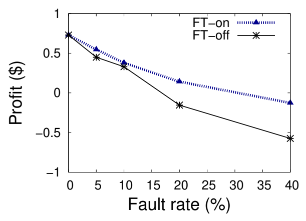
We observe that the higher the fault rate, the lower the profit is since high fault rates increase the chances of incurring SLA violations and penalties. Similarly, as expected, the higher the penalty, the lower the profit is. Moreover, the profit is lower for the heterogeneous customer profile since this contains contracts that are more sensitive to faults. In all cases, using work replacement (FT is on) increases the provider profit.



(a) Half reimbursement

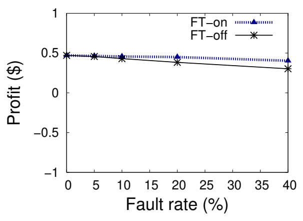


(b) Full reimbursement

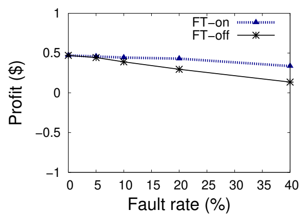


(c) Double reimbursement

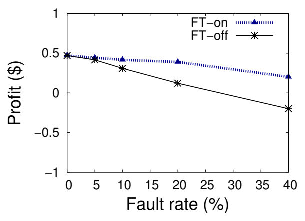
Figure 2.2: Heterogeneous customer profile with three penalty policies



(a) Half reimbursement



(b) Full reimbursement



(c) Double reimbursement

Figure 2.3: Safe customer profile with three penalty policies

2.1.5 Discussion

The main outcome of this work was the development of a complete, SLA-driven solution for managing service execution. Moreover, the work showed the benefit of using simple tactics based on a cost-benefit calculation to increase provider profit. A limitation of the work was the assumption that resources are obtained from a single IaaS cloud, either a private or a public IaaS cloud. Simultaneously managing private and public cloud resources was considered in subsequent work (see Section 2.2). Moreover, the work focused exclusively on master/worker applications. In particular, it focused on master/worker applications whose resource allocation is fixed statically, when the contract is established. My following research looked at supporting extensibility with regard to application types, including support for applications whose resource allocation can change dynamically (Section 2.2 and Chapter 4).

2.2 SLA-based Resource Management for PaaS Providers

After examining SaaS systems, I focused my attention on PaaS systems, which introduce new difficulties in designing effective management tools. Unlike SaaS systems that tightly integrate applications, PaaS systems accept and control customer-supplied applications. PaaS systems have thus less knowledge of application internals, making it more difficult to manage application QoS properties. Although extending the range of accepted applications exacerbates this difficulty, at the same time, it increases the applicability and thus the value of the PaaS system. For this reason, we chose as a requirement of this work PaaS *openness*, that is, the ability to support many application types. Another requirement was supporting cloud bursting, which enables providers to make the most of their private infrastructure, while removing the need for rejecting customer requests in peak periods. Similarly to our previous work (Section 2.1), main features of this work remain the support of QoS guarantees through SLAs and the management objective of increasing provider profit.

Several research efforts had focused on SLA-aware resource management for PaaS systems, attempting to satisfy economic goals such as optimising profit or reducing operational costs [25, 59, 68, 89, 111, 137]. However, unlike our work, such solutions were restricted to a single type of application, such as multi-tier web applications [59], MapReduce applications [111], or workflow applications [25], providing no extensibility with respect to application types. Moreover, most solutions used a single cloud, either a public IaaS cloud [68, 89, 137] or a private cloud [59]. No related system provided both the required extensibility and the capability of simultaneously deploying an application on multiple clouds.

This work led to two main contributions [40, 41, 42]. First, it proposed an SLA-based resource management approach for PaaS systems, integrating a policy for optimising the PaaS provider profit. Second, it proposed an open, cloud-bursting PaaS system, called Meryn, which applies the approach and provides support for batch and MapReduce applications.

In the following, I present the system model underpinning the approach and then describe the main policies suggested by the approach. Next, I outline the application of the approach to a specific type of applications—namely, computational applications—the Meryn prototype, and the evaluation, concluding with a brief discussion.

2.2.1 PaaS System Model

The approach considers an open PaaS system able to host applications of different application types. Applications are hosted on one or more resources, which are assumed to be homogeneous VMs. The resources may be private, owned by the provider, or public, rented from public IaaS clouds, and they are charged per time used. The resources are separated into groups, where each group is dedicated to a specific application type. Each application type-specific group is then composed of a set of private resources and, potentially, public resources, where all resources have similar configuration in terms of software stack and management tools. A key design decision made by the approach is that groups independently decide how to allocate their resources to applications, exchanging, if necessary, resources with other groups and bursting into public clouds. The reason for this decentralisation is threefold. First, it makes it easy to customise resource allocation policies for specific application types, taking advantage of application type-specific knowledge (e.g., whether applications can be scaled horizontally or vertically). Second, it reduces the complexity of developing such policies by decomposing the global problem of allocating resources to all applications of all types into simpler subproblems, solved by individual groups. Third, it removes the need for continuously sending information about the entire system to a central location, allowing fast decisions and supporting scalability to large clouds.

The objective of the PaaS provider is profit optimisation, taking into account the payment of penalties if the levels of QoS promised in SLAs are not met. Independently of the payment of penalties, QoS violations damage the provider's reputation and negatively affect long-term profits. For this reason, the approach imposes a constraint on the provider: the provider has to limit both the number of applications whose QoS guarantees are violated and the level of each QoS violation below predefined *QoS violation thresholds*.

The approach is generic and independent from the types of supported applications and it relies on one background assumption. Namely, for each supported application type, there is a corresponding *QoS translator* plug-in able to translate the resource configurations and, potentially, workload conditions, to provided QoS levels and vice versa. This enables predicting the QoS impact of modifying resources and estimating the required resources for achieving given QoS levels. Such translation capabilities can be based on application-type specific knowledge, performance measurements on the provider platform, workload models, and performance models. Although improving prediction accuracy can improve the provider profit, the approach remains independent of the achieved accuracy level.

2.2.2 Resource Sharing Policies

We developed three policies for sharing the private resources between the different groups and allocating resources to each newly arrived application.

Basic Policy

The basic policy is naive. It statically partitions the private resources between the groups. When a new application arrives for a group and the group has insufficient private resources, the application will be hosted on public resources.

Advanced Policy

The advanced policy enables the transfer of resources from one group to another one. When a new job arrives in a peak period, its corresponding group first checks if it can get available private resources from other groups before renting additional resources from a public cloud.

Optimisation Policy

The optimisation policy aims to find the most cost-effective solution for executing a newly arrived application. Specifically, when the available private resources in all groups are not sufficient for hosting the new application, the policy compares the cost of three options for getting the missing resources: (1) renting them from a public cloud, (2) obtaining them from running applications, or (3) waiting for private resources to become available. The first option is similar to the one used in the advanced policy. The second option, called *donating option*, may have a QoS impact on the affected applications. Note that this option is applicable only to applications that can use variable amounts of resources during their lifetime. The third option, called *waiting option*, may have a QoS impact on the new application.

The policy is implemented in a decentralised, auction-based manner. In essence, the group corresponding to the new application receives bids from the other groups, representing the estimated penalties associated with obtaining resources from these groups using both the donating and waiting options. The corresponding group then selects the option with the lowest bid and compares this with the option of renting public cloud resources. The pseudocode of the optimisation policy is shown in Algorithm 1.

2.2.3 Computational Applications

We applied the approach to *computational applications*, defined as applications running for a finite duration without any human intervention (e.g., batch applications). Computational

Algorithm 1 Optimisation Policy

```
Require: Request $req$ to host a new application
Ensure: Resources for $req$ are allocated
  $R_{req} \leftarrow$ resources required by $req$
  $R \leftarrow$ available resources in current group
  if $(R_{req} \leq R)$ then
    Get $R_{req}$ from current group
    Exit
  Get $R$ from current group
  $R_{missing} \leftarrow R_{req} - R$
  $R_{available} \leftarrow$ available resources in other groups
  if $(R_{missing} \leq R_{available})$ then
    Get $R_{missing}$ from corresponding groups
    Exit
  Get $R_{available}$ from corresponding groups
  $R_{missing} \leftarrow R_{missing} - R_{available}$
  Collect donating bids from all groups and select group $G^i$ with lowest bid, called $donating\_bid$,
  that satisfies QoS violation thresholds
  Collect waiting bids from all groups and select group $G^j$ with lowest bid, called $waiting\_bid$, and
  wait time $wait$ that satisfies QoS violation thresholds
  if $(donating\_bid < waiting\_bid)$ then
    $req_{private\_cost} \leftarrow cost($hosting $req$ on $R_{req}$ private resources$) + donating\_bid$
    $private\_option \leftarrow donating$
  else
    $req_{private\_cost} \leftarrow cost($hosting $req$ on $R_{req}$ private resources$) + waiting\_bid$
    $private\_option \leftarrow waiting$
  $req_{public\_cost} \leftarrow cost($hosting $req$ on $(R + R_{available})$private and $R_{missing}$ public resources$)$
  if $(req_{private\_cost} > req_{public\_cost})$ then
    Rent $R_{missing}$ from corresponding public cloud
    Exit
  if $(private\_option = waiting)$ then
    if ($req$ supports running with partial amount of resources) then
      Start running the new application with available resources
    Wait for $wait$ time
    Get $R_{missing}$ from $G^j$
  else
    Get $R_{missing}$ from $G^i$
```

applications may be *rigid*, that is, applications that require a fixed amount of resources, or *elastic*, that is, applications that can use variable amounts of resources at runtime. The work assumed that the QoS translator plug-in for computational applications relies on a simple, linear speedup performance model, where the application runtime is inversely proportional to the used number of resources.

The following subsections describe how SLAs are defined and how the waiting and donating bids are calculated for computational applications.

Creating SLAs

The SLA for computational applications specifies a *deadline* and a *price*. The deadline is a limit for the completion time of the application. The PaaS system calculates the deadline and price using the QoS translator plug-in. Specifically, the deadline is calculated as the application's predicted runtime and the price as the cost of the resources for this time period.

The system proposes several *SLA classes* (i.e., high, medium, low) to customers by combining different values for deadline and price. For example, the high SLA class has the shortest deadline, corresponding to the minimal possible application's runtime, and the highest price; inversely, the low SLA class has the longest deadline and lowest price. Having multiple SLA classes increases the number of choices given to customers (e.g., best price or best QoS). If the customer does not agree with any proposed SLA class, the customer may propose either the deadline or the price, and the system fills in the missing value, if feasible.

If the system fails to complete the application before its agreed deadline, the provider pays a *penalty*. The penalty depends on the *delay*, that is, the difference between the application's actual completion time and the deadline. We defined three revenue functions that determine how the penalty increases with delay, that is, a *linear*, a *bounded linear* and a *step* revenue function.

Calculating Bids

In this section, we discuss how donating and waiting bids are calculated for computational applications. In essence, the waiting bid corresponds to the estimated penalty of the new application due to waiting private resources to become available. The donating bid corresponds to the sum of the estimated penalties of the applications involved in donating the required resources.

Waiting Bid Using the QoS translator plugin, the group calculates the remaining time for each application to finish its execution and release its resources. The group's waiting time is the earliest such time when the released resources are greater than or equal to the required resources. The group's waiting bid is then the estimated penalty for delaying the new application by this waiting time.

Donating Bid A group can impose on an application three alternative forms of donating resources: total lending, partial lending, and giving. With *total lending*, the application is suspended, it lends all its resources to the new request, and resumes execution once the request terminates. With *partial lending*, the application lends a subset of its resources to the request, continues running with the remaining resources, and gets back the resources once the request terminates. With *giving*, the application gives a subset of its resources to the request and continues running with the remaining resources. Rigid applications support only total lending, while elastic applications support all three forms of donation. Each form of donation impacts the remaining time of the application, possibly incurring a delay and an associated penalty. The remaining time is estimated using the QoS translator plugin and the penalty using the revenue function.

The group's donating bid is calculated as follows. For each group application and each applicable form of donation, we calculate the estimated delay and associated penalty. We then use a heuristic to calculate a subset of applications and associated donation forms that reduces the sum of penalties while providing the required resources and satisfying the QoS violation thresholds. The group's donating bid is then this sum of penalties.

2.2.4 Meryn Prototype

To validate the proposed profit optimisation approach, we developed a prototype, called *Meryn*, a cloud-bursting PaaS system that supports batch and MapReduce applications [40, 42]. The VMs managed by Meryn are partitioned into Virtual Clusters (VCs), containing private VMs and possibly VMs rented from public IaaS clouds. A VC hosts applications of the same type and runs a specific programming framework. For example, Meryn includes a VC that hosts batch applications and runs the Oracle Grid Engine (OGE) framework [109], and a VC that hosts MapReduce applications and runs the Hadoop framework [8].

VCs may exchange private resources among them or burst into public cloud resources according to the policies described in Section 2.2.2. The customers submit their applications through a common and uniform interface, independently of the type of their applications.

The architecture of Meryn is shown in Figure 2.4. The main components are the client manager, a cluster manager for each VC, an application controller for each hosted application, and the resource manager. The *client manager* receives application submission requests, acts as the intermediary between the customer and the corresponding cluster manager, and

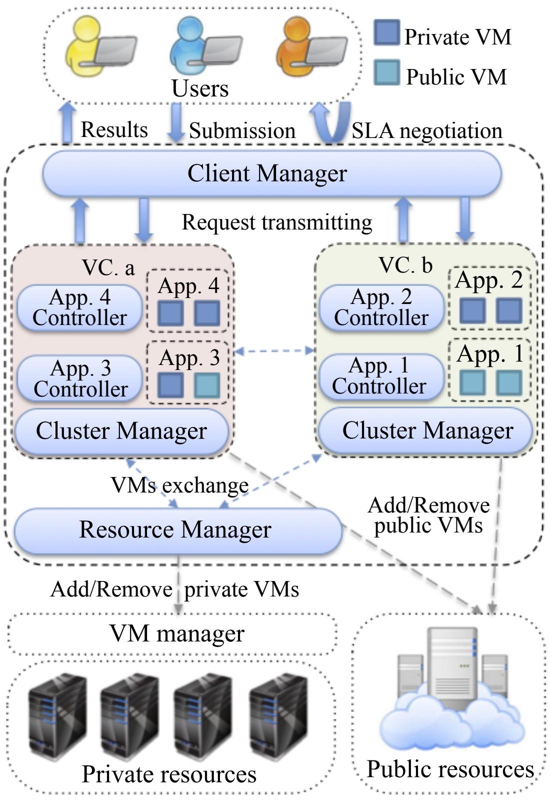


Figure 2.4: Meryn architecture

returns application results. The *cluster manager* consists of a generic part and a framework-specific part. The generic part decides when to exchange resources with other VCs and when to rent additional resources from public clouds. The framework-specific part converts submission requests to requests compatible with the framework; this part is also responsible for proposing SLAs and negotiating them with the customers using the QoS translator plug-in. The *application controller* monitors the execution of its associated application and the satisfaction of its agreed SLA. Finally, the *resource manager* provides support for transferring VMs from one VC to another one. The resource manager interacts with an IaaS-level VM manager, such as OpenStack [108], OpenNebula [107], or Snooze [48].

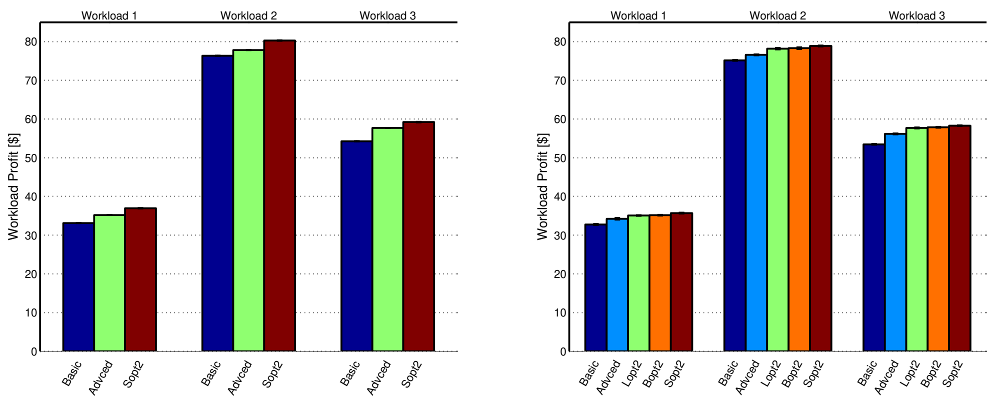
2.2.5 Evaluation

The objective of our evaluation was to compare the generated provider profit with each of the three policies: basic, advanced, and optimisation. For this evaluation, we used both simulations and experiments with the Meryn prototype. The used metrics include the profit of a workload, the completion time of a workload, the number of deadline violations, the average delay, and the VM usage proportions (i.e., the proportion of VMs obtained through the local VC, a different VC, donating applications, waiting for the termination of applications, or the public cloud).

Simulations and experiments used the same setup. The Meryn prototype has the two previously mentioned VCs, one for MapReduce applications and one for batch applications. The private cloud has 100 VMs and the public cloud has unlimited VMs. Simulations used a stripped-down prototype that creates no real VMs and applications and runs in a single machine with limited resources. Experiments used the Grid'5000 [18] testbed; the private cloud had 20 physical nodes and the public cloud had 70 physical nodes and was geographically distant from the private cloud.

We used three workloads, each comprising a batch and a MapReduce workload submitted simultaneously. The batch workloads follow the Lublin workload model [91]. The MapReduce workloads follow the Facebook workload reported in [143]. We defined three SLA classes (high, medium, and low) and randomly assigned classes to the applications in the workload.

We evaluated two variants of the optimisation policy (opt1 and opt2) with different QoS violation thresholds combined with three revenue functions (e.g., *Lopt2* is opt2 with Linear revenue function, *Bopt2* is with Bounded linear, and *Sopt2* is with Step revenue function). The *opt2* variant has higher QoS violation thresholds than opt1 and is thus more aggressive in impacting applications (specifically, it can impact up to 50% of the applications with a delay of up to 100% of the deadline). For illustration, we present a subset of the evaluation results. Full details can be found in [41].



(a) Simulations (b) Experiments

Figure 2.5: Profit for different workloads and policies

Figure 2.5a shows the generated profit in simulations for the three workloads and the basic, advanced and Sopt2 policies. We see that the Sopt2 optimisation policy generates the most profit, that is, from 2.64% to 4.98% more profit than the advanced policy. Figure 2.5b shows the generated profit in experiments for the three workloads and the basic, advanced, Lopt2, Bopt2, and Sopt2 policies. The results are similar to the ones obtained with simulations. The main difference is that with experiments all policies generate less profit than with simulations. This is because of the highly variable overhead of managing VMs in experiments, resulting in more cases of application delays and penalties.

Figure 2.6 shows the VM usage proportions associated with each method of obtaining VMs (i.e., local, VC, donating, waiting, public cloud) for different workloads and policies in experiments. We see that the optimisation policy uses public clouds to a smaller extent than the basic and advanced policies, opting, if possible, for alternative sources of VMs. Figure 2.7 shows the completion times for the three workloads. We see that the optimisation policy produces slightly longer completion times than the basic and advanced policies (an increase of up to 6.63% for workload 2 and the Sopt2 policy). This increase is due to the incurred delays of the impacted applications and the required time for calculating the bids and transferring VMs between VCs.

Table 2.1 focuses on the QoS impact of the optimisation policy on applications and shows that the percentage of impacted applications is between 4% and 8.75% and the average delay of delayed applications is between 39% and 65.52% of the deadline. These values remain below the QoS violation thresholds.

In summary, the results showed that the optimisation policy generates more provider profit than the basic and advanced policies. As expected, the policy causes some applica-

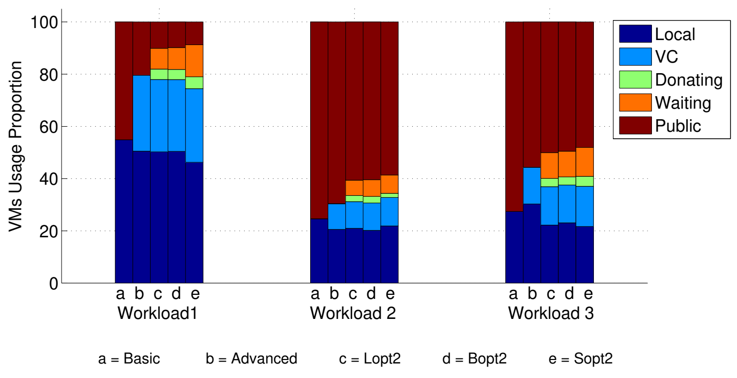


Figure 2.6: VM usage proportion for different workloads and policies (experiments)—VM usage is calculated as number of VMs multiplied by duration

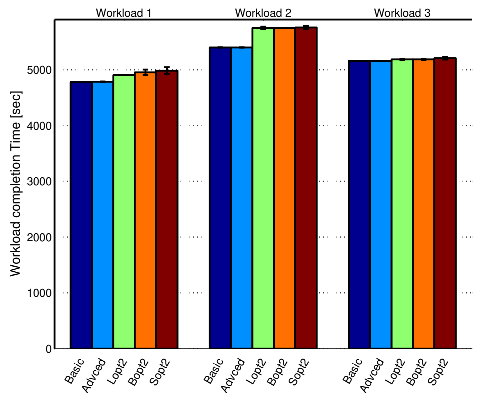


Figure 2.7: Workload completion time from the submission of the first job to the completion of the last job (experiments)

tions to miss their deadlines and slightly increases workload completion times. However, the number of impacted applications does not exceed the predefined thresholds. Moreover, the results of simulations and experiments agree with each other, validating the evaluation.

2.2.6 Discussion

The main novelties of this work were the support for cloud bursting, enabling applications to use simultaneously private and public cloud resources, and the support for multiple application types, such as batch and MapReduce applications. For simplicity, the work only considered the allocation of homogeneous, coarse-grained resources (i.e., VMs with fixed capacities). A more fine-grained, flexible resource allocation model will be described in Chapter 4. Moreover, the work did not consider dynamically adding resources to running applications

Table 2.1: Percentage of delayed applications (A) and average delay of delayed applications (B) with the optimisation policy (experiments)

	workload 1		workload 2		workload 3	
	A	B	A	B	A	B
Lopt2	4 %	46.5 %	4 %	59.07 %	4.5 %	40.65 %
Bopt2	6.5 %	39 %	6.75 %	49.23 %	8 %	46.93 %
Sopt2	7 %	55.63 %	7.5 %	49 %	8.75 %	65.51 %

to improve their QoS. Further research in dynamic application adaptation will be discussed in Chapter 3 and Chapter 4. Natural extensions of this work include taking into account data transfer costs, and adding support for further application types—such as workflow or data streaming applications—allowing a more complete assessment of the usefulness of the approach.

Summary

The chapter presented two automated resource management solutions for cloud providers. The first solution was designed for SaaS providers that manage resources from an IaaS cloud and deliver master/worker-based services. The second solution was designed for PaaS providers that manage resources from a hybrid IaaS cloud and host extensible application types. The next chapter turns to solutions for application management on behalf of cloud customers.

3 Application Management for Customers

In seeking to address every customer need, commercial IaaS providers are delivering an ever-widening range of features, making it increasingly complex to use IaaS clouds. Shielding customers from this complexity is the central theme of this chapter. Specifically, this chapter describes automated application management solutions for customers who want to deploy their applications across one or more IaaS clouds.

The chapter is divided into two parts.

- The first part discusses managing modular applications, composed of independently deployable units communicating over the network. These may be applications designed to run in the cloud or applications that can be easily migrated to the cloud with minimal or no changes. This work was performed in the context of the PaaSage European project (2012–2016) [110] in collaboration with Linh Manh Pham (postdoctoral engineer), Arnab Sinha (postdoctoral engineer), and Christine Morin from Inria, and the PhD thesis of Carlos Ruiz Diaz [122] that I co-supervised with Hector-Duran Limon (University of Guadalajara).

- The second part discusses managing a specific type of applications, namely epidemic simulation applications, with a monolithic structure. Such applications require significant restructuring in order to run in the cloud and thus gain access to large amounts of resources rapidly and cost-effectively. This work was performed in the context of the MIHMES project (2012–2017) [101], funded by ANR under the Investments for the Future Program, and the DiFFuSE ADT project (2017–2018) funded by Inria. I performed this work in collaboration with Linh Manh Pham (postdoctoral engineer), Yvon Jégou, and Christine Morin from Inria, and Sandie Arnoux, Gaël Beauneée, Luyuan Qi, Philippe Gontier, and Pauline Ezanno from INRAE Oniris.

We discuss each part in turn in the following.

3.1 Deploying and Managing Multi-cloud Applications

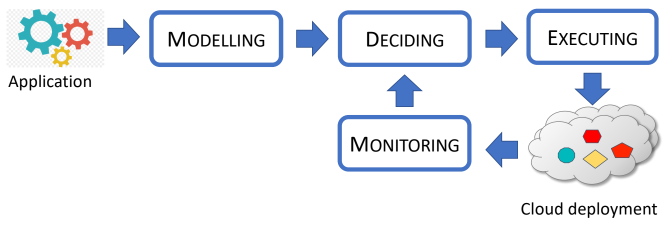


Figure 3.1: Application management architecture

The motivation for this research was reducing the complexity of deploying and managing applications over IaaS clouds. The research assumes that the application is structured as a set of interconnected, deployable components (e.g., web servers, databases) that need to be deployed over one or multiple IaaS clouds. This deployment involves selecting appropriate VM types to host each component together with associated cloud services (e.g., load balancers, storage). Making this selection involves considering an extremely wide range of cloud offerings with different characteristics (e.g., different CPU and memory capacities for VMs) and prices, provided by different cloud providers, accessible through different interfaces. These VM types, cloud services, and components must then be instantiated and configured appropriately to produce the initial application deployment.

At runtime, the application deployment will likely need to be adapted to accommodate environmental changes, such as changes in the workload, in application requirements, or in the availability, performance, and cost of underlying cloud services. This adaptation may involve, for example, increasing the number of web server instances or migrating instances across clouds, and must ensure that the application satisfies customer requirements as well as feasible.

There is a plethora of software platforms aiming to reduce this deployment and management complexity by applying self-adaptation techniques [7, 20, 29, 103, 105]. These platforms typically support the following four tasks (see Figure 3.1): (1) modelling, (2) deciding, (3) executing, and (4) monitoring.

Modelling involves specifying the deployment requirements of the application in a declarative, cloud provider-independent language. For example, this may include specifying the required operating system and geographic location of the VMs hosting individual application components. *Deciding* involves deriving the initial application deployment as well as deciding when and how to adapt the deployment at runtime. *Executing* involves applying the deployment decisions, dealing with the heterogeneity of clouds. Finally, *monitoring* involves collecting information about the running application (e.g., performance data), which is used by the deciding task, forming the typical closed-loop structure of adaptive systems.

My work on automated application management can be separated into three parts:

- development and evaluation of the adapter component for adapting multi-cloud applications, part of the PaaSage open-source application management platform;
- development and evaluation of a cost-effective approach for adaptive application deployment that builds on and specialises the PaaSage platform;
- development and evaluation of an application management platform based on SPLs (Software Product Lines) including a proactive policy for vertical scaling.

Each part is discussed in the following sections.

3.1.1 PaaSage Adapter

This work focused on developing a practical solution for adapting multi-cloud applications and was performed within the PaaSage project (2012–2016), funded by the European Union. Most adaptation solutions that emerged during the lifetime of PaaSage supported only horizontal scaling with a single cloud [7, 28, 29]. Moreover, these solutions relied entirely on developers and operations engineers to design appropriate scaling rules to meet customer objectives. Adaptation in PaaSage, on the other hand, included support for cross-cloud application restructuring based on high-level objectives.

This section first outlines the overall PaaSage architecture and then provides further details on the adapter, the adaptation solution whose development I coordinated within PaaSage.

PaaSage Architecture

The PaaSage project developed a model-based, multi-cloud development and deployment platform that enables developers to access cloud services in a technology-neutral manner while guiding them to configure their applications for best performance [110, 118]. The project was motivated by the heterogeneity of the available clouds and the associated complexity of porting existing applications to a cloud or switching between clouds. The project slogan was "*Define your application once. Deploy to the full spectrum of Clouds*".

To address cloud heterogeneity and facilitate application management, the project adopted a model-driven engineering approach. The project defined a modelling language, called CAMEL (Cloud Application Modelling and Execution Language) [2], which allows developers to model a wide range of application aspects, such as application deployment require-

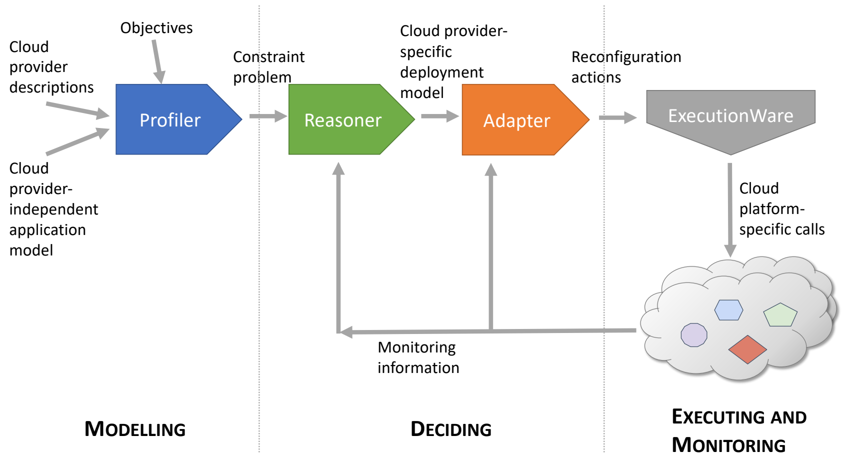


Figure 3.2: PaaSage architecture

ments, service-level objectives, and security considerations. Based on these models, the platform generates an initial application deployment that best satisfies application requirements. When runtime events make the current deployment unsatisfactory (e.g., QoS constraints are violated, or requirements are changed), the platform dynamically adapts this deployment following the models@run.time paradigm [17]. Specifically, the platform maintains runtime models of the deployment, requirements, and environment properties. These models are continually updated through monitoring and form the basis of generating alternative application deployments, detecting deviations between the current deployment and a target deployment, and transforming the current deployment into the target deployment.

The PaaSage platform consists of four main components (see Fig. 3.2):

- *Profiler*: receives a cloud provider-independent application description (e.g., application components and requirements on virtual hardware) and a set of cloud provider descriptions (e.g., VM types and prices for Amazon EC2), all written in the CAMEL language, and formalizes the problem of deploying the application on available cloud resources as a constrained optimisation problem. The optimisation problem includes variables (e.g., the number of component instances), constraints (e.g., two components must be hosted in the same VM) and an optimisation objective (e.g., minimising cost).

- *Reasoner*: solves the optimisation problem using an extensible set of solvers. The platform includes solvers based on mixed integer linear programming, constraint logic programming, and learning automata [37]. The optimisation objective is expressed using a utility function representing the preferences of the application owner among

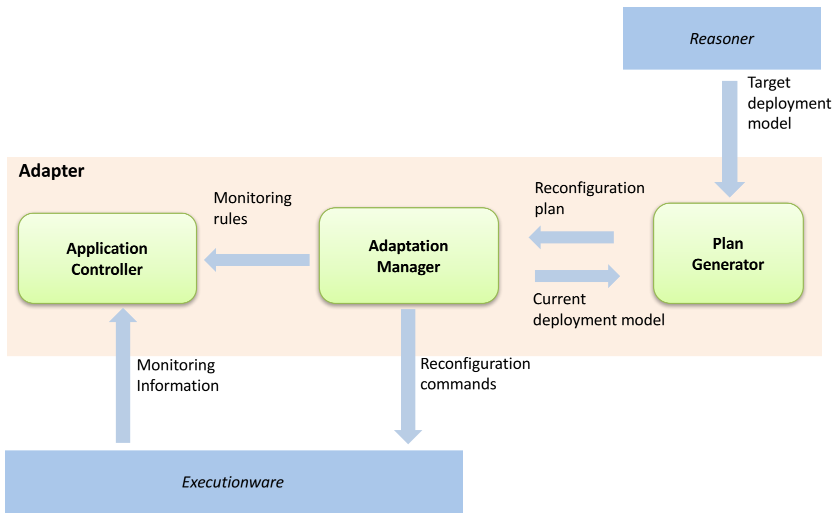


Figure 3.3: Adapter architecture

possible deployments. The reasoner generates a cloud provider-specific deployment model (e.g., application deployment on EC2).

- *Adapter*: receives a proposed deployment model, validates that it is acceptable, and adapts the currently running application deployment to reconcile it with the proposed model. The adapter is also responsible for monitoring and dynamically adapting the application deployment, e.g., when performance deteriorates, or cloud provider prices change.

- *Executionware*: enacts the deployment of application components on selected cloud providers and sets up monitoring of required metrics.

Adapter

The purpose of the adapter is to transform the currently running application configuration into the target configuration in an efficient and safe way [37]. As shown in Figure 3.3, the adapter is composed of three components: the plan generator, the adaptation manager, and the application controller.

The *plan generator* receives as input the current and the target application configurations and generates a plan representing the required reconfiguration actions and their order. The *adaptation manager* performs three tasks: it validates that the reconfiguration plan is acceptable, it applies the plan to the running system in an efficient and safe way, and it maintains an up-to-date representation of the current system state. The *application controller* component monitors the running application and its execution context in order to detect changes that make the current deployment unacceptable. Based on the monitored information, the component triggers the execution of the full reconfiguration process. To handle delays and transient failures, the application controller applies simple fault-tolerance tactics, such as retrying reconfiguration actions or issuing redundant reconfiguration requests.

The adapter brings key features to the PaaSage platform. First, it enables automatic synchronisation of models with the running system, removing the need to manually write low-level reconfiguration scripts. Second, it supports a rich set of reconfiguration actions beyond horizontal scaling, such as large-scale application restructuring and migrating components across clouds. Third, it performs reconfigurations in an efficient and reliable manner through parallel plan execution and handling failures. Finally, in conjunction with the reasoner and the executionware, it enables setting up control loops to monitor and adapt the application deployment.

The benefits of the PaaSage platform, including the adapter, were demonstrated using several use-case applications developed by PaaSage partners, including a flight-scheduling application developed by Lufthansa Systems, and a scientific workflow engine developed by the AGH University of Science and Technology [110]. The PaaSage project was successfully completed in November 2016 with an excellent rating from the reviewers. The PaaSage platform is currently being reused and extended within the Melodic European project (2016–2020) [100] that specifically targets data-intensive applications and supports data-aware application deployment. We used the PaaSage technology to implement the DiFFuSE framework described in Section 3.2. In [77], we describe early work on applying PaaSage to IoT applications in order to support on-demand provision of cloud resources when edge resources are insufficient.

3.1.2 Cost-effective Adaptive Deployment for Multi-cloud Applications

This work was directed at continuously optimising both the performance and the monetary cost of a multi-cloud application through a specific configuration of PaaSage mechanisms. One of the challenges in performing this optimisation is considering not only the predicted benefits of adapting the deployment (e.g., improved performance) but also the predicted costs (e.g., delays for provisioning new virtual machines). For example, when the workload is rapidly changing, the management system should refrain from performing complex, multi-cloud reconfiguration actions. The reason is that the costs are unlikely to be recovered if the application deployment does not remain stable.

Most existing application management solutions do not consider adaptation costs in making adaptation decisions [4, 20, 69, 105, 148]. A limited number of research solutions do consider adaptation costs [74, 75, 90, 128], but these are solutions operated by cloud providers rather than customers, and they do not support multi-cloud application management.

To respond to this challenge, we proposed an approach for adaptive deployment that explicitly considers the adaptation cost in deciding when and how to adapt the deployment [114]. The approach builds on the PaaSage architecture described in Section 3.1.1. The rest of this section describes the approach and then outlines how it was practically implemented and validated.

Continuous Deployment Optimisation

The approach assumes that a running application generates revenue for the application owner depending on the performance obtained by application users (e.g., depending on meeting response time guarantees). In turn, the owner pays cloud providers for the resources used by the application. The goal is then to optimise the profit, which is the difference between revenue and resource cost. The approach assumes that the instantaneous profit can be derived from a profit function that takes as input the current performance characteristics (e.g., response time) and the cost rate of used resources.

The approach proposes a continuous deployment optimisation process composed of two tasks: (1) generating a proposed deployment that optimizes profit under the current runtime conditions, ignoring any reconfiguration costs, and (2) deciding when and how to reconfigure the running system based on the proposed deployment, taking into account reconfiguration costs and benefits. In PaaSage, these tasks are the responsibility of the reasoner and adapter respectively.

Regarding the first task, the reasoner receives as input the profit function and the current application workload and provides to the adapter a proposed deployment model and its predicted, instantaneous profit. Internally, the reasoner maintains a performance model and a cost model, enabling it to estimate the performance and cost for a given application deployment and workload. The reasoner then searches for the deployment that maximizes the profit for the current workload.

Regarding the second task, the adapter produces a reconfiguration plan consisting of the tasks (e.g., creation, update, and deletion of VMs) necessary for transforming the current deployment to the proposed one. Given the proposed reconfiguration plan, the adapter validates that executing this plan is actually beneficial. This validation decision relies on weighing the potential reconfiguration benefits against reconfiguration costs. Benefits depend on the proposed deployment and on the time that the system remains in this deployment. Costs depend on the transient system state during reconfiguration and the time that this reconfiguration lasts.

Concretely, validation is based on the following information: (1) the estimated reconfiguration duration (T_{rc}), (2) the estimated time until a new reconfiguration is initiated (called stability interval or W), (3) the current profit (P_1), (4) the predicted profit during reconfiguration (P_2) and the predicted profit for the proposed configuration (P_3). The adapter performs the reconfiguration if the associated profit is higher than the profit of doing nothing over the stability interval W, i.e.,

$$T_{rc} * P_2 + (W - T_{rc}) * P_3 > W * P_1 \tag{3.1}$$

The reconfiguration duration (T_{rc}) can be estimated using historical information from previous executions of reconfiguration tasks. The stability interval (W) can be approximated as the time during which the application workload remains relatively stable (i.e., varies within a specific workload band), which can be predicted using historical information on workload evolution.

Approach Implementation

The approach was implemented by modifying and extending the reasoner and adapter components (see Fig. 3.4). The profit function that we used is:

$$Profit = \begin{cases} reward - cost & \text{if response time < target} \\ -cost & \text{if response time} \geq \text{target} \end{cases} \tag{3.2}$$

Cost is the cost rate of used resources; *reward* is the revenue when the SLO is satisfied (i.e., the response time is less than the given target).

We have extended the PaaSage reasoner with a solver that relies on manual, off-line profiling of the application to find the best deployment. Specifically, we gather performance samples from multiple executions of the application with representative deployments and workloads and use those samples to predict the performance for similar deployments and workloads. The cost of a deployment is estimated in a straightforward way using the pricing information of cloud providers.

The adapter includes a reconfiguration validator implementing the approach presented earlier, and predictors for the reconfiguration duration and stability interval. The reconfiguration duration is predicted using historical information on past reconfiguration tasks. The stability interval is predicted by applying an autoregressive moving average filter on past measured values of this interval.

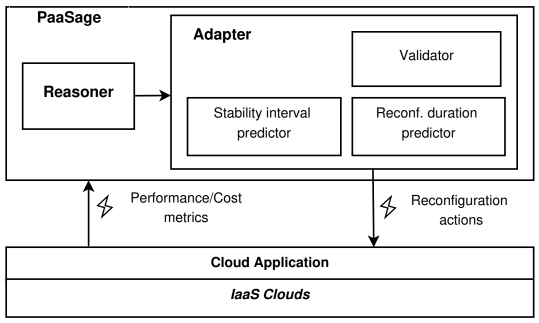


Figure 3.4: Architecture for continuous deployment optimisation

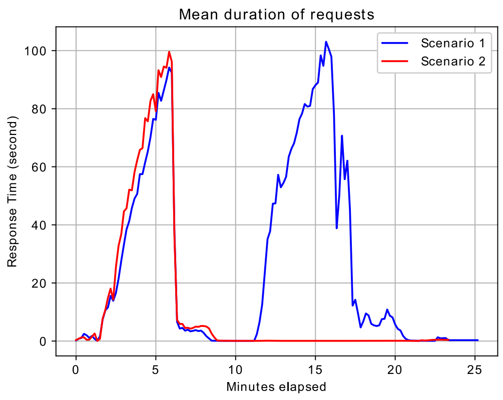
After the initial deployment, the adapter continuously monitors the application workload. If a significant workload change is detected, namely, if the workload varies beyond a specific workload band, the adapter asks the reasoner to provide a new proposed deployment and generates, validates, and applies a new reconfiguration plan. The adapter also triggers the reasoner when SLO requirements are violated or cloud prices are changed.

Evaluation

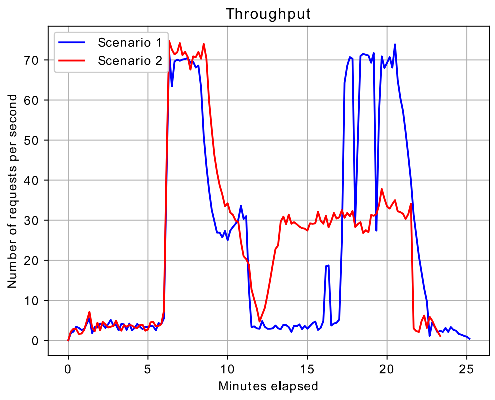
The approach was evaluated with a set of experiments on two cloud environments. The target cloud application was the three-tier RUBiS web application [119] and the workload was based on the EPA-http trace [46]. The experiments showed the practicality of the approach, its effectiveness compared to baseline approaches, its ability to perform multi-cloud reconfiguration actions taking into account changes in cloud pricing, and its effectiveness in optimising profit under various circumstances. More details can be found in [114]. Next, we consider only one of the experiments focusing on the reconfiguration validator.

The EPA workload, comprising an increasing and a decreasing phase, is played twice and an idle period of 3.5 minutes is inserted between the two plays. Scenario 1 is a case when the reconfiguration validator is switched off, whereas the validator is switched on in Scenario 2. Response time, throughput, and profit results for the two scenarios are shown in Fig. 3.5.

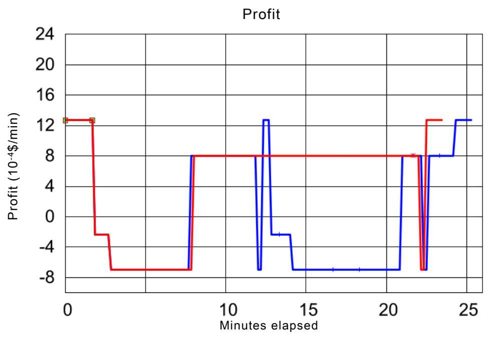
With the first workload increase, both scenarios scale out the application tier, resulting in the same behaviour for the first 10 minutes. The difference occurs at the 11th minute when the workload decreases. Scenario 1 performs a scale-in while Scenario 2 performs no action. Indeed, the turned-on validator in Scenario 2 denies the proposed reconfiguration as the predicted stability interval is small with respect to the reconfiguration interval. As a result, with the second workload increase, Scenario 1 shows worse performance than Scenario 2. The reasoner in Scenario 1 eventually recognizes the increase and suggests a scale-out that brings the response time back to normal at around the 20th minute. When the workload



(a) Response time over time



(b) Throughput over time



(c) Profit over time

Figure 3.5: Response time, throughput, and profit with the validator switched off (Scenario 1) and on (Scenario 2)

drops for the second time, both scenarios perform a scale-in action at around the 23rd minute. This time the adapter in Scenario 2 validates the scale-in as the predicted stability interval is long enough. From the 12th to 22nd minute, Fig. 3.5c shows a significant gain in the profit of Scenario 2 compared to Scenario 1. This experiment demonstrates the effectiveness of reconfiguration validation in optimising the profit.

3.1.3 SPL-based Application Management with a Proactive Policy for Vertical Scaling

This work focused on producing a complete application management solution using a model-driven approach, similar to that followed by PaaSage. It was performed in the context of Carlos Ruiz Diaz's PhD thesis [122], a student at the University of Guadalajara, whom I co-supervised. The main novelty of the work was the integration of a proactive adaptation policy to perform vertical VM scaling. This section first outlines the management approach proposed by this work, and then describes the proactive adaptation policy.

SPL-based Management Approach

The application management approach [121] is similar to that of PaaSage, but relies on a less expressive modelling language. As a result, the approach makes it easier for application owners to specify their deployment requirements, but it supports a more limited range of adaptation scenarios (e.g., it lacks support for distributing application components across different clouds). Specifically, modelling relies on SPL (Software Product Lines) techniques. Although related research had already applied SPL techniques to facilitate application deployment [88, 117], support for dynamic adaptation was lacking.

Following SPL principles, the configuration choices supported by IaaS clouds are represented as feature models; specific application deployments are represented as feature configurations. Models of the deployment are maintained at runtime and are used to facilitate adaptation. The architecture derived from this approach, called XIPE, supports all application management tasks (modelling, deciding, executing, monitoring) and is depicted in Figure 3.6.

The main components are the user interface, the SPL manager, the model manager, the adaptation manager, and the communication component. The *user interface* handles interactions with two types of users, platform administrators and application administrators. Platform administrators define and specialise SPL-based templates that describe features supported by particular IaaS clouds (e.g., based on OpenStack). These features include VM hardware types, operating systems, and supported software stacks. Application adminis-

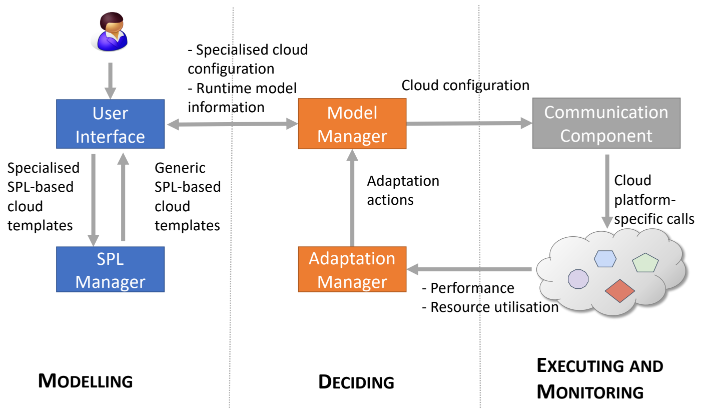


Figure 3.6: XIPE architecture

trators then choose from those features in order to specify a deployment configuration. The *SPL manager* supports creating and maintaining the SPL-based templates. The *model manager* supports creating and maintaining deployment configurations, mapping configurations to components provided by the underlying IaaS cloud, deploying configurations, and synchronising configurations with running applications. The *adaptation manager* adapts the deployment configuration based on monitoring information about the performance and resource usage of the running application. It integrates the adaptation policy described in the following section. Finally, the *communication component* handles the communication with the underlying IaaS cloud. A prototype of the architecture was implemented using the Eclipse modeling tools [44] and the CloudMF framework [50], which was also used in PaaSage.

Proactive Policy for Vertical VM Scaling

A notable aspect of XIPE is its support for proactive, vertical scaling [120]. Indeed, the majority of research and commercial cloud platforms were restricted to horizontal scaling [103], typically triggered in a reactive manner. XIPE enables proactively modifying the allocated memory of a VM before a change in performance occurs with the aim of avoiding SLO violations.

Figure 3.7 illustrates the design of the XIPE proactive adaptation solution. The solution consists of three main modules: monitoring, prediction and execution. The *monitoring module* collects information about the memory utilisation of the VM and the response time (RT) of the application component deployed in the VM. The *prediction module* consists of predictors for RT and memory utilisation that receive as input information from the monitoring module. The predictors rely on Recursive-Least-Square (RLS) filters [66]. The *execution*

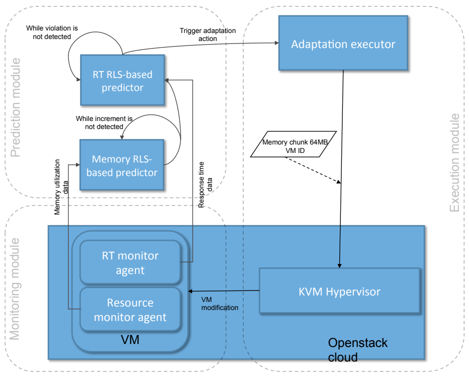


Figure 3.7: XIPE proactive adaptation

module adapts the amount of allocated memory of the VM. The adaptation policy operates as follows. Initially, the system considers only memory predictions. When a memory increase is predicted, the system takes into account RT predictions. When the RT is predicted to violate the SLO specified by the application administrator, adaptation is triggered. After the adaptation, the system returns to the initial state.

The adaptation policy was evaluated using experiments with a web application under a real workload trace in a private cloud environment. The policy was compared to different configurations of the default OpenStack horizontal scaling solution. The results showed that the adaptation policy provides similar or better performance than the OpenStack solutions with significantly reduced resource usage [122].

3.1.4 Discussion

This section presented work aiming at facilitating taking decisions about application deployment. We first presented the adapter tool, part of the PaaSage platform, that enables adaptation of multi-cloud applications. We then discussed an approach for adaptive application deployment that exploits the PaaSage technology. The approach explicitly weighs predicted adaptation costs against adaptation benefits based on predictions about reconfiguration duration and workload evolution. Finally, we presented an application management platform that includes a proactive adaptation policy that modifies the VM memory size.

The work validated the effectiveness of the model-driven approach followed by PaaSage and XIPE in facilitating application deployment and reconfiguration. Moreover, the work validated the effectiveness of the adaptive deployment approach in optimising the application owner profit under various circumstances. A limitation of the work is that it assumes the capability to predict application performance for different deployments and workloads. A large amount of research has recently focused on such predictions using various methods including analytic models and machine learning [6]. Exploiting such research is a natural direction for future work.

3.2 A Service-based Framework for Epidemic Simulation Applications

The motivation for this work was enabling legacy epidemic simulation applications to be deployed in the cloud. Indeed, the cloud is an attractive platform for resource-intensive applications because it enables rapidly obtaining practically unlimited amounts of resources with pay-per-use pricing [43]. Yet, legacy simulators, such as those developed by the BIOEPAR [16] research unit at INRAE (National Institute of Agricultural Research), are typically structured as monolithic applications executed on dedicated, high-end multicore systems and require significant restructuring in order to exploit the cloud.

Restructuring simulation applications to become cloud-based applications presents several challenges. First, the applications must be decomposed into services that can be deployed on separate machines, potentially in different clouds, and independently scaled. Second, automated support for application deployment and management must be provided in order to reduce the complexity exposed to scientists. This should include automated support for elasticity to optimise the application performance and cost. Indeed, simulation applications typically require different amounts of resources at different times. For example, the initialisation phase may require fewer compute resources than the main processing phase, thus making it more efficient to scale up the resource allocation at runtime. Finally, handling failures must be supported, which is essential for avoiding data loss and ensuring the correct and efficient completion of the simulation.

There is a large amount of research work focusing on exploiting clouds to execute epidemic simulations. Such work typically lacks support for elasticity [47, 64, 116, 131], or multi-cloud deployment [3], or fault tolerance [78], or simulation-specific facilities beyond bag-of-tasks execution [13, 133]. In contrast, our work proposed a complete, practical solution that addresses the previously mentioned challenges.

Specifically, we proposed a service-based framework called DiFFuSE that enables simulation applications to fully exploit cloud platforms [113, 115]. DiFFuSE supports structuring simulators as distributed, interacting services and provides mechanisms for managing simulation computations and data and automatically handling failures. To support multi-cloud deployment and elasticity, DiFFuSE relies on the PaaSage framework (see Section 3.1.1). DiF-

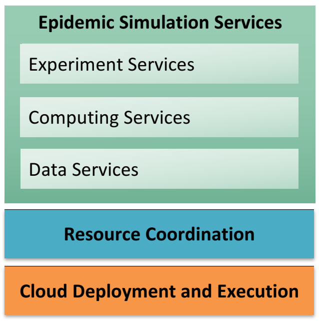


Figure 3.8: DiFFuSE architecture

FuSE has been applied to date to restructure two legacy applications, simulating the spread of two diseases in cattle, namely, bovine viral diarrhea and bovine paratuberculosis.

The rest of this section describes the DiFFuSE framework and outlines its evaluation based on the two case studies.

3.2.1 DiFFuSE Framework

The DiFFuSE framework is structured in three layers (see Figure 3.8). Seen from the bottom up, the layers are increasingly specific to the domain of epidemic simulations. At the bottom is a cloud deployment and execution platform, capable of running any modular application across clouds. Next is the resource coordination layer that supports building services that exchange data using a specific set of provided mechanisms. At the top is the epidemic simulation services layer that supports building epidemic simulation applications as sets of cooperating services. The three layers are outlined in the following.

Epidemic Simulation Services

This layer provides reusable design and code for commonly required functionality in epidemic simulation applications, such as controlling experiments, executing simulation runs, and managing the flow of simulation data. This functionality is provided as a collection of services that can be easily customised and used by developers. The services fall into three main categories.

- `Data Services`: retrieve data from databases and provide these data as well as related meta-data to other services.
- `Computing Services`: perform the main computations of the simulation, either sequentially or in parallel, using single-computer parallelisation (e.g., using OpenMP).

Computing services include services that execute simulation runs (called `worker` services) as well as services that generate and process the final results.

- `Experiment Services`: control the life-cycle of experiments. They track the progress of the computations, they handle service failures, and they provide configuration data about the experiments to other services.

Resource Coordination

This layer enables services to exchange data in a reliable and scalable way. In DiFFuSE, services run as separate processes communicating over the network. Services use, provide, and track the state of *resources*, defined as any named data (e.g., experiment configurations, experiment data). A resource can have multiple providers, each offering the same data. Such replicated resources can be used to support fault-tolerance and improve performance. A service can obtain the data from an available provider and become itself a provider. The layer includes mechanisms to handle different types of failures, including deployment-order failures, service failures and, in particular, worker failures. It handles deployment-order failures through using the nanomsg message library [102], which transparently applies retry and timeout mechanisms, allowing services to be launched in any order. It handles service failures through maintaining links between services and having services check that they periodically receive messages from linked services and take appropriate actions when failures are detected. It handles worker failures through reallocating lost simulation runs to other workers.

Cloud Deployment and Execution

This layer enables hosting the services composing the epidemic simulator on a multi-cloud infrastructure, relying on the PaaSage platform (see Section 3.1.1). PaaSage receives as input CAMEL descriptions of the simulation services and of available cloud providers and generates a deployment on available cloud resources. Moreover, PaaSage dynamically adapts the deployment to react to environment changes, e.g., violations of performance objectives, or changes in cloud prices. It notably triggers elasticity actions (e.g., scaling out a component) based on ECA (Event Condition Action) rules defined in CAMEL.

3.2.2 Evaluation

DiFFuSE was used to restructure two legacy simulators: a simulator of the spread of bovine viral diarrhea virus (BVDV), and a simulator of the spread of Mycobacterium avium subspecies paratuberculosis (MAP). The legacy simulators were developed by INRAE and were transformed to cloud-based applications based on DiFFuSE. In total, 6.3% of the original BVDV code and 1.78% of the original MAP code were modified in order to convert them to cloud-based applications. In the final DiFFuSE-based BVDV application, the proportion of the code that is specific to the BVDV spread simulation (rather than reusable code) is around 17.7%. In the final MAP application, this proportion is around 5.93%. These measures provided evidence of the limited effort required for applying DiFFuSE and of the added value of the DiFFuSE reusable code.

Using the BVDV and MAP simulators as reference applications, we conducted a set of experiments to evaluate DiFFuSE in terms of performance, cost effectiveness, failure handling, and elasticity in single and multi-cloud settings. The experiments validated that using DiFFuSE reduces execution time and cost compared to the legacy simulators, thus enabling scientists to perform simulations previously considered as impractical or prohibitively expensive. Moreover, the experiments demonstrated the supported flexibility in deploying DiFFuSE-based applications, which allows customers to make different cost-performance trade-offs by controlling the number and types of used cloud resources. Finally, the experiments validated the elasticity support provided by DiFFuSE as well as its ability to simultaneously exploit diverse VMs from multiple clouds.

Full evaluation results can be found in [115]. Here we cover a single experiment that validated that DiFFuSE helps reduce data transfer times in multi-cloud deployments. In such deployments, components of the simulation application are located at different, geographically distributed data centres. As a result, network characteristics have a strong effect on application performance. In the experiment, data services were deployed in an OpenStack-Grid'5000 private cloud and worker services in the Amazon EC2 public cloud. The workers were configured to be in the same EC2 availability zone in order to reduce network latency. When a worker is initialised, the worker obtains data from data services, which causes traffic of about 2GBs from the private cloud to the public cloud over the wide-area network connection. This would normally be repeated for every worker that joins the simulator with a negative effect on the simulator performance. To reduce this negative effect, we applied the DiFFuSE replication feature, enabling the worker that first obtains the data to provide replicas of the data to other workers. In this way, data transfer traffic was restricted within the boundary of the public cloud.

To evaluate the replication feature, we compared data transfer times for the BVDV simulator when data replication was enabled (E) and disabled (D) for different numbers of workers (4, 8, 12, 16 workers) processing a total of 40 simulation runs. Fig. 3.9 shows that the time is lower when the replication feature is used, regardless of the number of workers. The amount of data transferred inside (I) and outside (O) the public cloud is shown in Fig. 3.10. Regardless of the number of workers, the portion of the traffic inside the public cloud is greatly increased when replication is enabled. This experiment thus validated the usefulness of the replication feature in multi-cloud deployments.

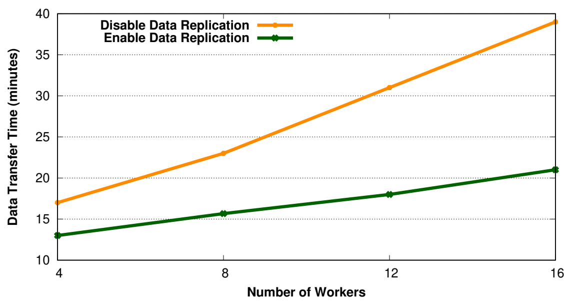


Figure 3.9: Data transfer time for different numbers of workers

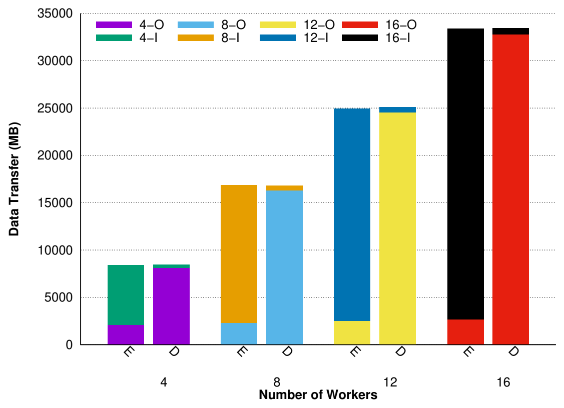


Figure 3.10: Amount of data transferred inside (I) and outside (O) the public cloud, with replication enabled (E) and disabled (D), for different numbers of workers

3.2.3 Discussion

The DiFFuSE framework brings significant advantages in supporting the execution of epidemic simulation applications. These advantages are outlined next: DiFFuSE can exploit variable numbers and types of cloud resources, allowing making different trade-offs between performance and cost. It can automatically handle a wide range of failure and recovery scenarios, reducing the time to complete the simulation. It allows combining resources from multiple clouds, such as private and public clouds. It provides replication support that helps reduce data transfer times in multi-cloud deployments. Finally, it allows elastically modifying resource allocations to adapt to dynamic variations in resource demand.

The DiFFuSE software has been released as open-source software under the CeCILL-B licence and used by epidemiology scientists at INRAE. The scientists appreciated the speedup gains of the DiFFuSE-based applications, while they found code restructuring to be the most difficult part of using DiFFuSE. While working with epidemiologists can sometimes be time-consuming owning to vocabulary differences, the benefits of the collaboration in terms of solving practical problems with tangible economic and societal impacts far outweigh any costs. This work will be continued in two directions. First, we plan to apply DiFFuSE to further applications in collaboration with INRAE scientists and use the feedback to improve the usability and usefulness of the framework. Second, we plan to extend DiFFuSE with support for continuous deployment optimisation. Indeed, the current version supports only static rules that scale in/out services. Continuous deployment optimisation would build on the work discussed in Section 3.1.2, generating good deployments based on historical execution information or performance models.

Summary

This chapter examined work on automated application management on behalf of customers. First, it discussed methods and tools for supporting application reconfiguration across multiple clouds, drawing on a model-driven approach. Then, it concentrated on the special case of epidemic simulation applications and presented a solution for restructuring, deploying, and executing such applications in cloud environments. The next chapter describes a management solution that coordinates the customers and the provider of a private cloud.

4 Application and Resource Management in Private Clouds

To jointly satisfy the objectives of customers and the provider, we need to coordinate their interactions through appropriate rules and incentives. This is a main theme of this chapter. Indeed, in the previous two chapters, we examined cloud management from the perspective of the provider or the customer. In each case, we did not consider how the other party behaves in pursuing its objectives. For example, in Chapter 2, we did not consider how customers negotiate SLAs, or how they react to contract rescissions, request cancellations, or SLO violations. In this chapter, we take a holistic approach that examines how all actors interact and pursue their own objectives within a private PaaS system.

The practical problem that motivated this work arose in the context of a collaboration with EDF (Électricité de France) R&D. This organisation needed to share its private infrastructure among scientists and engineers, who wanted to execute diverse applications (e.g., batch, bag-of-tasks, MapReduce applications) with various performance objectives (e.g., meeting deadlines). Using public cloud resources was excluded owning to security constraints. We identified two key requirements that a resource management system for this organisation should satisfy. First, given that the capacity of the infrastructure is limited, the system should distribute resources to the customers that value them the most. Indeed, traditional resource management systems for clusters (e.g., batch schedulers [130, 142]) rely on priority queues or fair sharing policies, providing no incentive to customers to relax their demand during contention periods, eventually leading to resources not being put to their best use. Second, the system should provide the flexibility to support a variety of applications, customer-defined SLOs, and policies for achieving these SLOs.

To meet these requirements, we proposed a novel application and resource management solution for private PaaS clouds. In our solution, the provider seeks to increase economic efficiency by virtualising the infrastructure and distributing fine-grained virtual resources to customers through a market-based mechanism, namely an auction. Customers seek to satisfy the SLOs of their applications through employing controllers that dynamically adapt the applications' resource demand.

Many resource management systems focused on providing SLO support to applications, typically using dynamic controllers based on performance models [38, 63, 136, 138, 147]. Unlike our solution, these systems target specific application types and lack the flexibility to support a wide range of applications and SLOs. Other related research focused on sharing a cluster between applications requiring different programming frameworks [67, 127, 135]. Although these systems have flexible architectures, which can support various application types, they provide no support for individual, customer-specific SLOs. Moreover, when handling resource contention, these systems rely on priorities or quotas assigned by administrators. As a result, these systems produce efficient allocations only if administrators know the value of resources for all customers at all times or if customers truthfully declare such values. Finally, much research has proposed using markets for sharing the resources of clusters, grids, and clouds [1, 27, 87, 124, 129, 141]. However, these systems did not target a private cloud and provided no adaptation support for meeting customer SLOs.

This work was performed in the context of Stefania Costache's PhD thesis [31] that I co-supervised with Christine Morin (Inria) and Samuel Kortas (EDF R&D). The work made two main contributions [30, 33, 34, 35, 36].

- First, we proposed a market-based approach that allocates virtualised resources to applications while enabling each application to individually and dynamically adapt its resource demand based on customer needs. The approach also provides adaptation policies for vertical and horizontal application scaling.

- Second, we proposed a prototype platform, called Merkat, that applies the approach to support generic and extensible application and resource management for private clouds. The prototype platform was deployed and evaluated within EDF R&D.

The remainder of this chapter is structured as follows. Section 4.1 presents the system model underpinning the approach. Section 4.2 describes the implementation of the auction mechanism, and Section 4.3 describes the proposed adaptation policies. A brief overview of the Merkat prototype is provided in Section 4.4, followed by a discussion of the evaluation in Section 4.5. Finally, Section 4.6 makes concluding remarks.

4.1 PaaS System Model

In our approach, we assume a private PaaS system hosting applications of different types supplied by customers (see Figure 4.1). Each application runs in its own *virtual platform*,

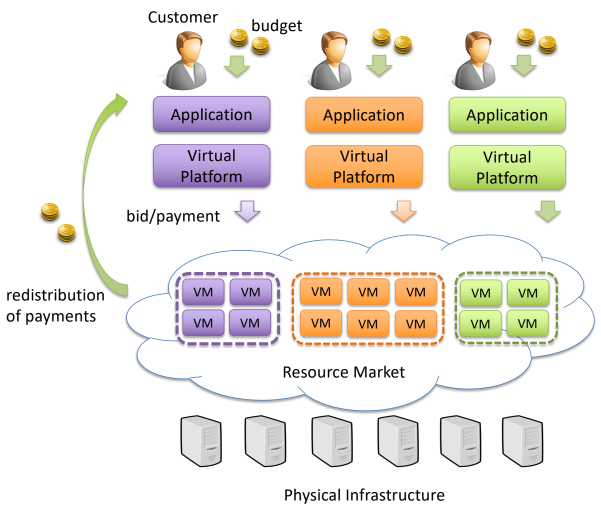


Figure 4.1: Overview of the market-based approach

composed of a set of VMs and an *application controller*. The application controller interacts with the PaaS system to control the application's resource allocation. The interaction between the controller (representing the customer) and the PaaS system (representing the provider) relies on a market-based mechanism, namely an auction. We selected the *proportional-share auction* [87] in which a bidder submits a bid for a resource, receives an amount of resource proportional to this bid, and is charged exactly this bid. The main advantage of such auctions is their simplicity and scalability (the computational complexity of calculating the allocation grows linearly with the number of bids). Moreover, under some theoretical assumptions, proportional-share auctions have been shown to be optimal in terms of economic efficiency among all simple and scalable market mechanisms [104].

Specifically, in our approach, the application controller submits *bids* on CPU and memory resources for the application VMs. Based on these bids, the PaaS system allocates an amount of CPU and memory to each running VM and charges the customer according to the auction rules. Resource allocations are fine-grained (i.e., 30% of one core and 800 MB of RAM) and vary dynamically according to the total resource demand. Payments rely on a *virtual currency*. When a customer registers with the system, the administrator assigns a number of *credits* to the customer's account. When the customer submits an application, the customer allocates an *application budget* to the application. The customer can set the application budget to be renewed automatically to avoid cases in which the application budget is depleted in the middle of the application's execution. Customer payments are redistributed to customers' accounts over a typically long time period.

The benefits of using this market-based mechanism are threefold. First, the mechanism provides incentives to customers to use resources judiciously given that customers are selfish, thus, increasing the economic efficiency of allocations. Second, it provides a generic way to control resource demands, making it easier to support diverse kinds of applications, SLOs, and policies for meeting those SLOs. Finally, supporting fine-grained, dynamic allocation helps to improve the utilisation of the infrastructure.

4.2 Auction Implementation

We describe next how the provider implements the proportional-share auction. This implementation follows four steps: (i) VM bid submission; (ii) VM resource allocation; (iii) VM placement; (iv) price computation.

VM Bid Submission

To provision a VM, the application controller submits a bid (b_{cpu},b_{memory}) with values for each VM resource. The initial bid can be computed based on the current price or past price history, and the bid can be modified to cope with price fluctuations or changes in resource demand.

VM Resource Allocation

The VM Scheduler periodically computes resource allocations for the VMs by considering their bids and resource utilisation limits (a_{max_cpu}, a_{max_memory}) that specify the maximum resource usage for a VM. The resource allocation computation is performed in two steps: (i) the VM Scheduler computes the VM allocation considering the entire infrastructure as a single physical node; (ii) the VM Scheduler corrects the allocations to cope with the fact that the infrastructure capacity is partitioned among nodes.

When considering the entire infrastructure as a single physical node, the VM Scheduler computes the resource allocations as follows. Given a set of bids b_j for VMs $j \in \{1..n\}$, and a resource with a capacity of C units, the resource allocation for j will be:

$$alloc_j = \frac{b_j}{\sum_1^n b_k} \cdot C, \tag{4.1}$$

Since the infrastructure capacity is partitioned between nodes, there are situations when resulting allocations cannot be enforced. For example, consider 3 physical nodes each with a capacity of 100 CPU units (1 CPU unit represents 1% of CPU time). We need to allocate 3 VMs that have a bid of 12 credits, and 2 VMs that have a bid of 30 credits. Using Equation 4.1, the VM Scheduler computes ideal allocations ($alloc_j^{ideal}$) for the VMs as follows: 37.5 CPU units for the first 3 VMs and 93.75 CPU units for the last two VMs. It is practically impossible to enforce these allocations on three physical machines. To solve this issue, the VM Scheduler can place the first 3 VMs on the same physical node, and the last 2 VMs on separate nodes. The VM Scheduler can then recompute the allocations using the capacity of individual nodes in Equation 4.1. In the example, the resulting, real allocations ($alloc_j^{real}$) are: 33.3 CPU units for the first 3 VMs and 100 CPU units for the last 2 VMs. The allocation difference is called *allocation error* and is defined as follows:

$$e_j = \frac{|alloc_j^{ideal} - alloc_j^{real}|}{alloc_j^{ideal}}, \tag{4.2}$$

VM Placement

When VM requests are received the VM Scheduler places them initially on the nodes with the lowest resource utilisation. To minimize the VM allocation error, the VM Scheduler might migrate VMs between nodes. The process of migrating VMs among nodes is called load balancing. As having a high number of migrations leads to a performance degradation for the applications running in the VMs, the load balancing process tries to make a trade-off between the number of performed migrations and the VM allocation error. For example, it may not make sense to migrate a VM when its allocation error is 1%. To select the VMs to be migrated at each scheduling period, the VM scheduler relies on an algorithm based on a tabu search metaheuristic method [57].

Algorithm 2 explains the load balancing process. The algorithm receives the list of current nodes, *nodes*, the list of running VMs, *vms*, the list of VMs to be started at the current scheduling period, *newvms*, and three thresholds: (i) maximum number of iterations performed by the algorithm to obtain a better placement than the current one, N_{iter}; (ii) maximum allocation error supported for the VMs in their current placement, E_{max}; (iii) maximum number of migrations required to reach a better placement, M_{max}. Based on this information the algorithm computes the new placement of VMs on nodes and outputs a migration plan, composed of the VMs to be migrated, and a deployment plan, composed of the VMs to be started.

Price Computation

The resource price is computed as the sum of all bids divided by the total infrastructure capacity. If this price is smaller than a predefined *reserve* price, then the reserve price is used.

Algorithm 2 VM load balancing algorithm

```
ComputePlacement ($nodes, vms, newvms, N_{iter}, E_{max}, M_{max}$)
  for $vm \in newvms$ do
    $node$ ← least loaded node from $nodes$
    $node.vms \leftarrow node.vms \cup \{vm\}$
  $solution_{old} \leftarrow nodes$
  $solution_{best} \leftarrow nodes$
  $nIterations \leftarrow 0$
  $tabu\_list \leftarrow \varnothing$ // list of forbidden moves
  $e_{worse} \leftarrow$ inf
  e = ComputeErrors($nodes$) // based on Equation 4.2 for each VM and each resource (i.e., cpu and memory)
  while $nIterations < N_{iter}$ and $e_{worse} > E_{max}$ do
    $(vm, e_{max})$ ← find $vm$ with $e_{max} = \max_{1 \le i \le n} \max\{e_{i,cpu}, e_{i,mem}\}$
    $source \leftarrow vm.node$
    $destination \leftarrow node$ which minimizes $e_{max}$, $(vm, node) \notin tabu\_list$
    $vm.node \leftarrow destination$
    $source.vms \leftarrow source.vms - \{vm\}$
    $destination.vms \leftarrow destination.vms \cup \{vm\}$
    $tabu\_list \leftarrow tabu\_list \cup \{(vm, source)\}$
    e = ComputeErrors($nodes$)
    $e'_{max} \leftarrow \max_{1 \le i \le n} \max\{e_{i,cpu}, e_{i,mem}\}$
    $nMigrations$ = calculate number of migrations required to reach the current solution from $solution_{old}$
    if $e'_{max} < e_{worse}$ and $nMigrations < M_{max}$ then
      $solution_{best} \leftarrow nodes$
      $e_{worse} \leftarrow e'_{max}$
      $nIterations \leftarrow 0$
    else
      $nIterations \leftarrow nIterations + 1$
  for $node \in solution_{old}$ do
    for $vm \in node.vms$ do
      if $vm.node \neq node$ and $vm \notin newvms$ then
        $migrationPlan \leftarrow migrationPlan \cup \{(vm, vm.node)\}$
  for $vm \in newvms$ do
    $deploymentPlan \leftarrow deploymentPlan \cup \{(vm, vm.node)\}$
  return ($migrationPlan, deploymentPlan$)
```

Customers are charged for each running VM at each scheduling interval with a credit amount equal to the product of the resource price and the resource allocations.

4.3 Adaptation Policies

The application controller applies policies to adapt the application on behalf of the customer. To illustrate this adaptation, we discuss two policies—one for vertical scaling and one for horizontal scaling—that modify the application resource demand to meet customer SLOs under budget constraints. The policies run periodically and use two performance thresholds, upper and lower, as a trigger: when the performance metric traverses a threshold, the policy takes actions that change the application resource demand. The *vertical scaling policy* adapts the bids of each VM and uses suspend/resume mechanisms to avoid high price periods. This policy can be applied for applications with a static number of VMs, such as MPI applications, with customer-provided time constraints. The *horizontal scaling policy* adapts both the number of VMs and their bids. This policy can be applied for elastic applications, such as task-processing frameworks (e.g., Condor, Hadoop).

In general, the policies behave as follows:

- When the SLO is met, the policy reduces the resource demand of the application, thus saving the budget to run other applications.
- When the SLO is not met and the budget allows it, the policy increases the resource demand of the application.
- When the SLO is not met and the budget is insufficient, the policy may suspend the application, or stop the application, or reduce its resource demand.

The policies are outlined next. Full details can be found in [33].

4.3.1 Vertical Scaling Policy

This policy generates the resource bids for the next period based on the application performance metrics, the current allocations, and the budget. The policy decreases the resource bids if the performance metric drops below the lower threshold (e.g., the application is expected to finish sooner than its deadline), or if the allocation per VM for one resource reaches the maximum. If the performance metric is above the higher threshold (e.g., the application is expected to miss the deadline), the policy increases the resource bids. If the current budget is not enough to meet the SLO, the policy takes a decision based on the application and SLO types. Specifically, if it is advantageous to suspend the application execution (e.g., in the case

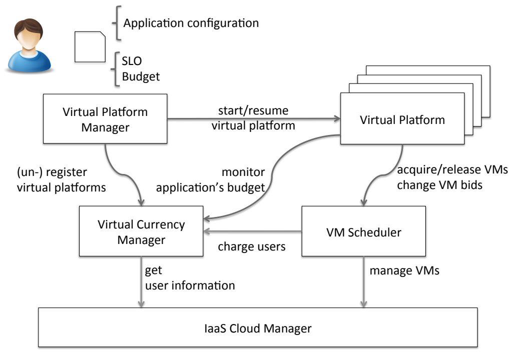

Application configuration
SLO
Budget
Virtual Platform Manager
start/resume virtual platform
Virtual Platform
(un-) register virtual platforms
monitor application's budget
acquire/release VMs change VM bids
Virtual Currency Manager
charge users
VM Scheduler
get user information
manage VMs
IaaS Cloud Manager

Figure 4.2: Merkat prototype

of best-effort or deadline-driven batch applications), the policy suspends the application, resuming it when the price drops. If it is not advantageous to suspend the application execution (e.g., for interactive applications, or applications for which the customer might be satisfied with partial results delivered at a deadline), the policy recomputes the bids in a way that favors the resource (CPU, memory) with a small allocation (i.e., the resource that represents a bottleneck in the application's progress).

4.3.2 Horizontal Scaling Policy

This policy generates the number of VMs and the bids for the next time period based on the performance metrics, the current resource prices, and the budget. The policy decreases or increases the number of VMs depending on whether the SLO is met is not. The policy ensures that the resulting number of VMs is less than or equal to an upper bound on the number of VMs. This upper bound is calculated as the maximum number of VMs permitted by the application budget so that all VMs receive their maximum resource allocation (i.e., $a_{max\_cpu}, a_{max\_memory}$) at the current resource prices.

4.4 Merkat Prototype

To validate our proposed resource and application management approach, we built a prototype, called Merkat, whose architecture is shown in Figure 4.2. Merkat is composed of three main services: The *VM Scheduler* is in charge of allocating resources to running VMs and computing their node placement using Algorithm 2. The *Virtual Currency Manager* applies virtual currency distribution policies and manages customer and application budgets. The *Virtual Platform Manager* acts as the entry point for customers to use the system.

To submit an application, the customer uses a *virtual platform template* that specifies the configuration of the VMs (e.g., VM disk images), the type of application controller, budgeting information, and adaptation policy parameters, such as desired SLO (e.g., execution time limits). Based on this information, the Virtual Platform Manager creates the virtual platform and deploys the application controller, responsible for further application management based on the adaptation policies. The Virtual Platform Manager registers the virtual platform with the Virtual Currency Manager to allow charging the application for its resource usage.

VMs are allocated using an IaaS Cloud Manager that provides interfaces to start, delete and migrate VMs, manage their storage and network, and keep information about infrastructure users. Merkat is implemented in Python and relies on the Twisted [134] and ZeroMq [145] libraries. It uses OpenNebula as the IaaS Cloud Manager [107]; the default scheduler of OpenNebula is replaced by Merkat's VM Scheduler.

4.5 Evaluation

We conducted a set of experiments to evaluate the extent to which the approach supports customers in meeting their SLOs. We also wanted to validate the flexibility of the approach in supporting different application types and SLOs. To that end, we performed both simulations and testbed experiments described in turn next.

4.5.1 Simulations with Large Traces

To test Merkat with large workloads, we used simulation since this would have been impractical to do in a real testbed. Specifically, we implemented the algorithms of Merkat in CloudSim [23], an event-driven simulator implemented in Java. The applications are batch applications composed of a fixed number of tasks running in parallel. To express the diversity of customer requirements, we defined three types of customers:

- *Full deadline customers:* The customer wants the results by a specific deadline; if the application fails to terminate before the deadline, the customer is unsatisfied.
- *Partial deadline customers:* The customer wants the results by a specific deadline; the customer values partial results at the deadline.
- *Full performance customers:* The customer wants the results as soon as possible, but can also accept a bounded delay.

Each customer type is associated with a utility function that measures the value obtained by the customer (called *customer satisfaction*) as a function of the application execution time.

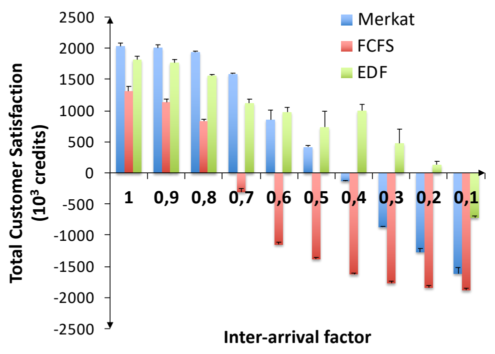
Each type is also associated with an adaptation policy, which specialises the vertical scaling policy described in Section 4.3.1 to reflect customer requirements. The workload was based on the HPC2N trace [112], containing a log of applications submitted to a Linux cluster with 120 nodes. This trace was chosen because of the information regarding memory requirements of applications. Since the trace had no information on deadlines, we assigned synthetic deadlines to applications, between 1.5 and 10 times the application execution time. We assigned budgets inversely proportional to this deadline factor.

The experiments showed that Merkat can accommodate different types of customers while providing good customer value and with limited VM management overhead (migration and suspend/resume operations). Full results can be found in [33]. Here we focus on the comparison of the total satisfaction provided by Merkat compared to two traditional scheduling policies, namely, FCFS (First Come, First Served) and EDF (Earliest Deadline First). FCFS is the policy typically applied by IaaS cloud managers. EDF is a policy typically applied to minimise the number of missed deadlines. It is important to note that cloud systems cannot practically apply EDF or similar algorithms without limiting their support to a predefined set of customer goals (e.g., in the case of EDF, meeting deadlines). We chose EDF in order to compare our flexible approach with a specialised, deadline-driven system. Figure 4.3(a) describes the results of this comparison. It considers full-deadline customers and the associated Merkat policy. Figure 4.3(b) describes the number of applications that successfully finished their execution before their deadline.

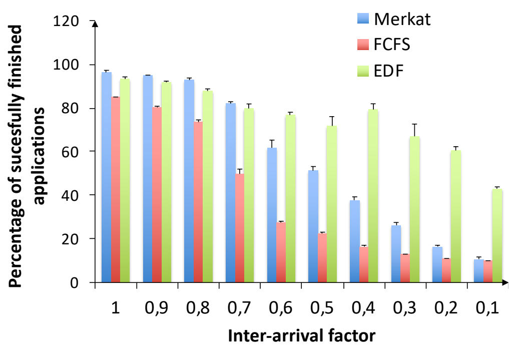
We observe that (i) the system outperforms FCFS in all cases, as FCFS does not consider application valuation or SLO in its decisions; (ii) when the contention is not high, the system outperforms EDF in terms of customer satisfaction; and (iii) when the system is highly loaded, EDF performs better. The performance gap between the mechanism and EDF in case of high contention can be explained as follows. With EDF, the central scheduler takes consistently good scheduling decisions; it sorts applications by their deadline and executes the application with the smallest deadline first. Thus, more applications with smaller deadlines get to run on time, providing higher satisfaction. With Merkat, applications do not consistently take the best allocation decisions since they adapt independently with only limited information. This leads to a performance degradation, the result of allowing applications to behave selfishly (i.e., the price of anarchy [104]).

4.5.2 Testbed Experiments

For the experiments on a real testbed, we used three applications with different SLOs and associated policies. Specifically, we used an MPI application with a vertical scaling policy that attempts to meet customer deadlines, the Condor framework with a horizontal scaling pol-



(a) Total satisfaction



(b) Finished applications

Figure 4.3: Performance in terms of (a) total customer satisfaction, and (b) percentage of successfully finished applications for Merkat, FCFS, and EDF—contention increases from left to right.

Platform	% Met Deadlines	Satisfaction	Migrations/hour
Merkat	82.5%	764330	310
Maui	58.1%	-178581	-
Merkat/CloudSim	94%	950554	62

Table 4.1: Comparison of Merkat, Maui, and Merkat's simulation in CloudSim.

icy that attempts to regulate the number of queued tasks, and the Torque framework with a horizontal scaling policy that attempts to regulate the wait time in the queue per task. The experiments validated the flexibility of Merkat in supporting different applications, SLOs, and policies. They also showed that the applications can adapt to changes in operating conditions (e.g., price increases in contention periods) as well as to changes in customer requirements (e.g., changes in deadlines) [33].

In a particular experiment, we compared Merkat with Maui, a popular batch scheduler in HPC environments [98]. Specifically, we deployed Merkat and Maui on a cloud of 10 compute nodes of Grid'5000 and ran 160 MPI applications over a time interval of 7 hours. The applications had 1 to 8 processes, each process in one VM. We considered full-deadline customers, and generated the workload based on a Lublin [91] model. In Merkat, each application ran with the vertical scaling policy. Maui used FCFS and backfilling to schedule applications.

Table 4.1 summarizes the results of this experiment. The results showed that, when using Merkat, the number of applications that finish before their deadline increases by 24.1 percentage points compared to using Maui. Using Merkat also increased the customer satisfaction compared to Maui (for Merkat, the total satisfaction was 764330 credits, compared to a negative satisfaction of -178581 credits for Maui). These results provided evidence of the effectiveness of Merkat in managing real clouds. To further validate the results, we compared Merkat with its CloudSim-based simulation on the same workload trace. We found a difference of 11 percentage points in the number of applications that meet their deadlines, with more applications meeting their deadlines in the simulation. We also see a difference in the total satisfaction and the number of migrations. The reason for the difference is that Merkat/CloudSim relies on a simple application model in which performance interference is not accurately modelled. When running real applications on Merkat, Merkat's algorithms have to cope with higher variability in performance metrics, which leads to higher variability in bids and resource allocations, and thus a higher number of migrations.

4.6 Discussion

This chapter described a solution for application and resource management in private PaaS clouds. In this solution, the provider operates an auction that allocates fine-grained virtual resources to customers. Customers operate applications that autonomously adapt their resource demand in a decentralised manner, seeking to satisfy their QoS objectives. A straightforward way to continue this work is to support additional resource types (e.g., network and storage resources), application types, SLOs, and policies for meeting these SLOs, including policies that consider the costs of reconfiguration. A limitation of the work is that developing adaptation policies remains a complex and ad hoc process, mainly due to the price dynamicity and uncertainty. It would thus be interesting to provide improved support for developing such policies (e.g., based on the bidding strategies for EC2 spot pricing [146]) or even to investigate alternative auction mechanisms that simplify the policy design space [26, 80].

The Merkat prototype was deployed in an EDF R&D testbed, but we unfortunately obtained few data on its usability and acceptance by users. The most widely deployed PaaS systems for private clouds, developed in the same period as Merkat or later, use traditional methods to handle contention, such as priorities or quotas [21, 67, 135]. Experience with earlier market-based resource management systems showed that users, when given the choice, may prefer traditional methods to markets for simplicity reasons, even if this choice results in lower utility [10]. It thus remains unclear whether markets are appropriate in private cloud settings, where customers belong to the same organisation and simpler methods to solve contention are feasible. Market-based approaches are certainly useful in completely decentralised systems with no single locus of control, a point revisited in the next chapter.

5 Conclusion

This final chapter revisits some key findings of my research and identifies directions for future work.

5.1 Synopsis

The document has separated my research activities into three strands. The first strand focused on designing provider-side management systems, producing solutions for SaaS and PaaS providers. The main insight of this work was that SLAs simplify decision-making because they link QoS levels (e.g., response time values) to economic consequences (prices, penalties). As a result, decision-making can be decomposed into predicting the QoS impact of different management actions, translating this impact into monetary terms, and performing an economic cost-benefit analysis. Another insight was that building an automated SLA-driven management system requires a flexible architecture that supports a wide range of management actions (e.g., adding/removing resources, cancelling contracts) and enables timely response to environmental changes (e.g., fluctuating workloads, failures).

The second strand focused on designing customer-side management systems, producing methods and tools for adapting multi-cloud applications. These tools were notably integrated within the PaaSage open-source platform and exploited by several academic and industrial partners. The main insight of this work was that beneficial application management requires considering reconfiguration costs as well as benefits. To reason about reconfigurations, it is useful to maintain abstract representations of the running system and generate explicit reconfiguration plans. The effectiveness of applying economic analysis to such reconfiguration plans was also confirmed by the work. Another focus in this strand was facilitating the deployment of epidemic simulation applications, producing the DiFFuSE framework built on PaaSage. The DiFFuSE framework was successfully used by epidemiology scientists at INRAE to migrate legacy simulators to the cloud. An insight gained from this work was that legacy simulators can be decomposed into separate services with limited code changes; this decomposition then makes it easy to exploit cloud platform features, such as independent service scaling.

The final strand focused on designing a framework for resource and application management in private clouds. This research took the idea of applying economic reasoning a step further: it transformed the private cloud into an economy where applications selfishly attempt to achieve their own goals through participating in an auction. The main insight of this work was that the market-based approach is a natural and effective way to coordinate interactions among applications and the platform, promoting extensibility with respect to application types and increasing total value.

An important part of my research activities was developing working software systems. Software development was useful not only for experimentally validating the results (typically using Grid'5000 deployments), but also for generating, testing, and improving solution ideas in early development phases. Building software can be tedious and time-consuming, particularly when multiple development teams are collaborating, as was the case, for instance, with the PaaSage platform. Nevertheless, producing usable prototypes is invaluable. It enables broadening the impact of the work as well as obtaining helpful feedback from real users, as was the case, for instance, with the DiFFuSE framework.

5.2 Outlook

Cloud computing is continually evolving to accommodate a widening range of application types, programming models, underlying infrastructures, economic actors, and pricing models, posing new challenges and opening up exciting research opportunities [22]. I plan to continue investigating automated management solutions for domains beyond traditional clouds through applying new or improved management approaches. In the following, I present the research directions that I intend to pursue together with some ongoing work. These directions fall into three categories: management for domains beyond traditional clouds, improved decision-making, and support for decentralised management.

5.2.1 Beyond Traditional Clouds

High-performance, real-time, embedded systems Modern real-time embedded systems such as those monitoring and reacting to evolving physical environments (e.g., ground radar systems, surveillance drones) are increasingly expected to cope with dynamic workloads, difficult to predict in advance. These systems must execute dynamic combinations of applications, each having varying resource needs, while satisfying stringent timing requirements. The resource management problem posed by these systems becomes thus similar to that posed by cloud systems; namely, allocating resources to dynamic workloads in line with application QoS requirements and provider objectives.

We are working on resource management for high-performance embedded systems in the context of Baptiste Goupille-Lescar's PhD thesis that I co-supervise with Eric Lenormand (Thales Research & Technology) and Christine Morin (Inria). The thesis is funded by a CIFRE contract with Thales Research & Technology. In this work, applications require real-time guarantees, and the provider objective is to increase the utilisation of the computing platform. Research challenges include the heterogeneity of the resources, and the dynamism and variability of the workload combined with the real-time application requirements. To address those challenges, we are developing a QoS-aware resource management approach that maintains a continually updated runtime model of the platform architecture and applications, allowing accurate predictions of future resource availabilities. Preliminary results show that the approach leads to reduced application latencies compared to basic solutions while allowing gradual performance degradation in overload scenarios [60, 61].

Function-as-a-Service (FaaS) The FaaS model, the core element of *serverless computing*, is a recent addition to cloud technologies that enables customers to deploy functions on the provider's infrastructure without being concerned with provisioning or operating servers. The functions are executed on demand and customers are charged only for the amount of time that their functions are running. The benefits of FaaS include fully automatic scaling, a general programming model, and reduced operational costs, leading to a rising popularity in industry. The model is supported by all major cloud providers as well as several open-source software platforms [51, 79, 106]. The serverless computing model and FaaS are sparking new research into extending FaaS platforms to support additional application types (e.g., data analytics) and underlying infrastructures (e.g., edge devices) [11, 72].

One limitation of current FaaS platforms is that they lack support for managing the QoS experienced by FaaS customers. To overcome this limitation, we have initiated work to develop an automated resource management solution for FaaS [19]. This work is performed in the context of Yasmina Bouizem's PhD thesis that I am co-supervising with Djawida Dib (University of Tlemcen) and Christine Morin (Inria). The provider objectives chosen for this solution are satisfying the performance and availability requirements of customers while reducing energy consumption. Challenges in developing this solution include implementing fault-tolerance mechanisms, such as active replication, and taking into account the performance, availability, and energy consumption objectives in a coordinated manner.

Fog Computing The rapid growth in the number of Internet-connected devices has triggered the emergence of diverse IoT (Internet of Things) applications and use cases, such as autonomous cars, smart cities, and video surveillance. These applications impose stringent requirements on the underlying infrastructure in terms of available bandwidth and low-latency computation. Traditional cloud infrastructures cannot meet these requirements as they rely on distant data centres accessible over large-scale networks, placing economic and physical limits on bandwidth and latency [126]. This has led to growing industrial and research interest in *fog computing*, sometimes referred to as *edge computing*. Fog computing is an extension of the traditional cloud computing model in which compute, storage, and network capabilities are distributed closer to users along a cloud-to-thing continuum [95].

Increasing the proximity to users promises multiple benefits, including highly responsive services, reduced bandwidth consumption, support for user mobility, and enhanced privacy. However, realising these benefits poses several research challenges. First, compared to traditional cloud computing, fog computing exploits a wider heterogeneity of resources (e.g., sensors, mobile devices, gateways, micro data centres, cloud data centres) and network links (e.g., wireless access technologies, backbone links). This diversity makes it more difficult to define common interfaces and abstractions for accessing these resources and models for reasoning about their capabilities. Importantly, this diversity also expands the range of possible application deployments, making it more difficult to make application placement decisions. In the context of the FogCity project [52], we have initiated work on a fog resource management scheme that takes into account fog node heterogeneity. The scheme is based on a bargaining game and seeks to satisfy the QoS preferences of customers (e.g., application delay and location preferences) while maximising node utilisation.

Second, compared to traditional cloud execution environments, fog environments are more dynamic and subject to unpredictable changes. This results from user and device mobility (e.g., personal devices, cars, drones), resource volatility (e.g., battery shortage), and topology changes (e.g., fog nodes joining, leaving, failing). The higher rate of environment change requires faster management decisions, motivating further research towards improved decision-making approaches and decentralised decision-making structures.

Finally, fog computing will involve a complex interplay of economic actors. Although the fog ecosystem has not yet stabilised, one possible scheme comprises infrastructure providers, service providers, end users, and edge resource owners [76]. In this scheme, infrastructure providers own and manage large-scale infrastructures of devices, networks, and clouds. Service providers rent resources from infrastructure providers and create applications (e.g., video surveillance) used by end users. Edge resource owners (e.g., individuals, companies) own small-scale infrastructures of edge resources and rent these to infrastructure providers.

Coordinating the complex interactions between such actors and automatically managing the multiple associated contracts in line with actors' objectives is a challenging area that requires further research.

5.2.2 Improved Decision-Making

Effective decision-making is an essential ingredient of all management systems. In the work described in this document, decision-making typically relied on simple heuristics (e.g., Section 2.1.2, Section 4.3) or metaheuristics (e.g., Section 4.2) combined with performance, cost, and workload models (e.g., Section 2.2.1, Section 3.1.2). The drawback of such techniques is the complexity of designing appropriate heuristics and of manually tuning them to handle changes in operating conditions (e.g., new application types, new workload types).

Machine learning techniques, such as deep reinforcement learning, have the potential to alleviate this drawback. Indeed, these techniques enable taking decisions based on dynamically collected data without requiring a priori modelling or expert knowledge, while automatically learning and adapting to changing environments. Machine learning techniques are increasingly being applied to cloud management problems with promising results [9, 96, 123, 139]. Nevertheless, to enable widespread deployment of these techniques in real-world settings, many challenges remain to be addressed, including exploring the performance, stability, and generalisability of different learning algorithms [62], improving their interpretability [97], integrating them into practical cloud management systems, and evaluating their effectiveness.

5.2.3 Decentralised Management

Current and emerging cloud and fog ecosystems encompass independent interacting systems operated by actors with distinct, potentially conflicting interests (e.g., application owners, service providers). A challenge that arises in such decentralised systems is global manageability [71], that is, coordinating the interactions of the independent systems in order to ensure global properties, such as maximising total value or fairness.

A well-known approach to addressing the global manageability challenge is applying economic and pricing mechanisms [49]. These mechanisms specify the rules and incentives that govern interactions among self-interested agents, which trade services with each other. We have already applied auction mechanisms to coordinate the interactions among customers and the provider in private clouds (Chapter 4) and among application type-specific resource groups in a PaaS implementation (Section 2.2). Although there is an extensive literature on applying economic and market-based mechanisms in cloud resource management [32, 92], the mechanisms have seen limited adoption in real, commercial clouds. The main reason is arguably the complexity of such mechanisms for human users [70]. This may be of less concern when automated agents are involved, potentially employing sophisticated decision-making techniques (Section 5.2.2).

Further research is clearly needed in using economic mechanisms for decentralised management in cloud and fog deployments. An important requirement is selecting appropriate mechanisms for a given setting. Such settings may include, for instance, selling fine-grained IaaS resources for short time intervals [12], leasing edge resources to fog infrastructure providers [76], allocating micro data centre resources to low-latency applications [132], or placing functions on fog-based FaaS platforms [15]. Selecting mechanisms will require considering not only their economic properties (e.g., economic efficiency, truthfulness), but also their technical properties (e.g., scalability). For instance, typical auction mechanisms employ a central, trusted auctioneer that collects and processes all bids, creating scalability, reliability, and trust challenges. Addressing such challenges may require turning towards distributed auction mechanisms [140, 144]. Finally, much work remains to be done in building practical implementations of such mechanisms and participating agents, and demonstrating their reliable and robust operation in real cloud/fog deployments.

Bibliography

1. D. Abramson, R. Buyya, and J. Giddy. "A Computational Economy for Grid Computing and Its Implementation in the Nimrod-g Resource Broker". *Future Gener. Comput. Syst.* 18:8, 2002, pp. 1061–1074. ISSN: 0167-739X. DOI: 10.1016/S0167-739X(02)00085-7.
2. A. P. Achilleos, K. Kritikos, A. Rossini, G. M. Kapitsaki, J. Domaschka, M. Orzechowski, D. Seybold, F. Griesinger, N. Nikolov, D. Romero, and G. A. Papadopoulos. "The Cloud Application Modelling and Execution Language". *Journal of Cloud Computing* 8:1, 2019, p. 20. ISSN: 2192-113X. DOI: 10.1186/s13677-019-0138-7.
3. D. Agarwal and S. K. Prasad. "AzureBOT: A Framework for Bag-of-Tasks Applications on the Azure Cloud Platform". In: *2013 IEEE International Symposium on Parallel and Distributed Processing, Workshops and Phd Forum*. 2013, pp. 2139–2146.
4. A. Almeida, F. Dantas, E. Cavalcante, and T. Batista. "A Branch-and-Bound Algorithm for Autonomic Adaptation of Multi-Cloud Applications". In: *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. 2014, pp. 315–323. DOI: 10.1109/CCGrid.2014.25.
5. *Amazon EC2 Spot Instances*. URL: https://aws.amazon.com/fr/ec2/spot/ (visited on March 9, 2020).
6. M. Amiri and L. Mohammad-Khanli. "Survey on Prediction Models of Applications for Resources Provisioning in Cloud". en. *Journal of Network and Computer Applications* 82, 2017, pp. 93–113. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2017.01.016.
7. *Apache Brooklyn*. URL: https://brooklyn.apache.org/ (visited on March 9, 2020).
8. *Apache Hadoop*. URL: https://hadoop.apache.org/ (visited on March 9, 2020).
9. H. Arabnejad, C. Pahl, P. Jamshidi, and G. Estrada. "A Comparison of Reinforcement Learning Techniques for Fuzzy Cloud Auto-Scaling". In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 2017, pp. 64–73. DOI: 10.1109/CCGRID.2017.15.

10. A. AuYoung, P. Buonadonna, B. N. Chun, C. Ng, D. C. Parkes, J. Shneidman, A. C. Snoeren, and A. Vahdat. “Two Auction-Based Resource Allocation Environments: Design and Experience”. In: *Market-Oriented Grid and Utility Computing*. John Wiley & Sons, Ltd, 2009. Chap. 23, pp. 513–539. ISBN: 978-0-470-45543-2. DOI: 10.1002/9780470455432.ch23.

11. I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, and P. Suter. “Serverless Computing: Current Trends and Open Problems”. en. In: *Research Advances in Cloud Computing*. Ed. by S. Chaudhary, G. Somani, and R. Buyya. Springer, Singapore, 2017, pp. 1–20. ISBN: 978-981-10-5026-8. DOI: 10.1007/978-981-10-5026-8_1.

12. O. A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. “The Rise of RaaS: The Resource-as-a-Service Cloud”. *Commun. ACM* 57, 2014, pp. 76–84.

13. A. D. Benedictis, M. Rak, M. Turtur, and U. Villano. “A Framework for Cloud-Aware Development of Bag-of-Tasks Scientific Applications”. *International Journal of Grid and Utility Computing* 7:2, 2016, pp. 130–140. ISSN: 1741-847X.

14. S. Benkner and G. Engelbrecht. “A Generic QoS Infrastructure for Grid Web Services”. *Advanced International Conference on Telecommunications / Internet and Web Applications and Services* 0, 2006, p. 141. DOI: 10.1109/AICT-ICIW.2006.16.

15. D. Bermbach, S. Maghsudi, J. Hasenburg, and T. Pfandzelter. “Towards Auction-Based Function Placement in Serverless Fog Platforms”. *CoRR* abs/1912.06096, 2019. arXiv: 1912.06096.

16. *BioEpAR*. URL: https://www6.angers-nantes.inrae.fr/bioepar (visited on March 9, 2020).

17. G. Blair, N. Bencomo, and R. B. France. “Models@ Run.Time”. *Computer* 42:10, 2009, pp. 22–27. ISSN: 1558-0814. DOI: 10.1109/MC.2009.326.

18. R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and I. Touche. “Grid’5000: A Large Scale and Highly Reconfigurable Experimental Grid Testbed”. *The International Journal of High Performance Computing Applications* 20:4, 2006, pp. 481–494. DOI: 10.1177/1094342006070078.

19. Y. Bouizem. “An Approach for Energy-Efficient, Fault Tolerant FaaS Platforms”. In: *19th ACM/IFIP International Middleware Conference*. Rennes, France, 2018, pp. 1–2.

20. A. Brogi, J. Carrasco, J. Cubo, F. D'Andria, E. Di Nitto, M. Guerriero, D. Pérez, E. Pimentel, and J. Soldani. "SeaClouds: An Open Reference Architecture for Multi-Cloud Governance". In: *Software Architecture: 10th European Conference, ECSA 2016, Copenhagen, Denmark, November 28 – December 2, 2016, Proceedings*. Ed. by B. Tekinerdogan, U. Zdun, and A. Babar. Springer International Publishing, 2016, pp. 334–338. ISBN: 978-3-319-48992-6. DOI: 10.1007/978-3-319-48992-6_25.

21. B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes. "Borg, Omega, and Kubernetes". *Queue* 14:1, 2016, pp. 70–93. ISSN: 1542-7730. DOI: 10.1145/2898442.2898444.

22. R. Buyya, S. N. Srirama, G. Casale, R. Calheiros, Y. Simmhan, B. Varghese, E. Gelenbe, B. Javadi, L. M. Vaquero, M. A. S. Netto, A. N. Toosi, M. A. Rodriguez, I. M. Llorente, S. D. C. D. Vimercati, P. Samarati, D. Milojicic, C. Varela, R. Bahsoon, M. D. D. Assuncao, O. Rana, W. Zhou, H. Jin, W. Gentzsch, A. Y. Zomaya, and H. Shen. "A Manifesto for Future Generation Cloud Computing: Research Directions for the Next Decade". *ACM Computing Surveys* 51:5, 2018, 105:1–105:38. ISSN: 0360-0300. DOI: 10.1145/3241737.

23. R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya. "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms". *Software Practice and Experience* 41:1, 2011, pp. 23–50.

24. R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, and R. Buyya. "The Aneka Platform and QoS-Driven Resource Provisioning for Elastic Applications on Hybrid Clouds". *Future Generation Computer Systems* 28:6, 2012, pp. 861–870. ISSN: 0167739X. DOI: 10.1016/j.future.2011.07.005.

25. Z.-G. Chen, K.-J. Du, Z.-H. Zhan, and J. Zhang. "Deadline Constrained Cloud Computing Resources Scheduling for Cost Optimization Based on Dynamic Objective Genetic Algorithm". In: *Evolutionary Computation (CEC), 2015 IEEE Congress On*. IEEE. 2015, pp. 708–714.

26. S. Chichin, Q. B. Vo, and R. Kowalczyk. "Truthful Market-Based Trading of Cloud Services with Reservation Price". In: *Proceedings of the 2014 IEEE International Conference on Services Computing*. SCC '14. IEEE Computer Society, USA, 2014, pp. 27–34. ISBN: 978-1-4799-5066-9. DOI: 10.1109/SCC.2014.13.

27. B. N. Chun and D. E. Culler. "REXEC: A Decentralized, Secure Remote Execution Environment for Clusters". In: *Proceedings of the 4th International Workshop on Network-*

Based Parallel Computing: Communication, Architecture, and Applications. CANPC '00. Springer-Verlag, London, UK, UK, 2000, pp. 1–14. ISBN: 3-540-67879-4.

28. *Cloud Foundry*. URL: http://www.cloudfoundry.org/ (visited on March 9, 2020).

29. *Cloudify*. URL: https://cloudify.co/ (visited on March 9, 2020).

30. S. Costache, N. Parlavantzas, C. Morin, and S. Kortas. "Themis: Economy-Based Automatic Resource Scaling for Cloud Systems". In: *2012 IEEE 14th International Conference on High Performance Computing and Communication*. 2012, pp. 367–374. DOI: 10.1109/HPCC.2012.56.

31. S. Costache. "Market-Based Autonomous and Elastic Application Execution on Clouds". Theses. Université Rennes 1, 2013.

32. S. Costache, D. Dib, N. Parlavantzas, and C. Morin. "Resource Management in Cloud Platform as a Service Systems: Analysis and Opportunities". *Journal of Systems and Software* 132, 2017, pp. 98–118. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2017.05.035.

33. S. Costache, S. Kortas, C. Morin, and N. Parlavantzas. "Market-Based Autonomous Resource and Application Management in Private Clouds". *Journal of Parallel and Distributed Computing* 100, 2017, pp. 85–102. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2016.10.003.

34. S. Costache, N. Parlavantzas, C. Morin, and S. Kortas. "An Economic Approach for Application QoS Management in Clouds". In: *Euro-Par 2011: Parallel Processing Workshops*. Ed. by M. Alexander, P. D'Ambra, A. Belloum, G. Bosilca, M. Cannataro, M. Danelutto, B. Di Martino, M. Gerndt, E. Jeannot, R. Namyst, J. Roman, S. L. Scott, J. L. Traff, G. Vallée, and J. Weidendorfer. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 426–435. ISBN: 978-3-642-29740-3.

35. S. Costache, N. Parlavantzas, C. Morin, and S. Kortas. "Merkat: A Market-Based SLO-Driven Cloud Platform". In: *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*. IEEE, 2013, pp. 403–410. ISBN: 978-0-7695-5095-4. DOI: 10.1109/CloudCom.2013.59.

36. S. Costache, N. Parlavantzas, C. Morin, and S. Kortas. "On the Use of a Proportional-Share Market for Application SLO Support in Clouds". In: *Euro-Par 2013 Parallel Processing*. Ed. by F. Wolf, B. Mohr, and D. an Mey. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 341–352. ISBN: 978-3-642-40047-6.

37. S. Crompton, K. Figiela, G. Horn, F. Griesinger, D. Hope, T. Kirkham, K. Kritikos, M. Malawski, N. Parlavantzas, C. Pérez, L. Pouilloux, D. Romero, C. Sheridan, P. Silva, and A. Sinha. *D3.1.2 - Product Upperware*. Research Report. PaaSage: Model Based Cloud Platform Upperware, 2015.

38. C. Delimitrou and C. Kozyrakis. "Quasar: Resource-Efficient and QoS-Aware Cluster Management". In: *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '14. ACM, Salt Lake City, Utah, USA, 2014, pp. 127–144. ISBN: 978-1-4503-2305-5. DOI: 10.1145/2541940.2541941.

39. D. Dib. "Optimizing PaaS Provider Profit under Service Level Agreement Constraints". Theses. Université Rennes 1, 2014.

40. D. Dib, N. Parlavantzas, and C. Morin. "Meryn: Open, SLA-Driven, Cloud Bursting PaaS". In: *Proceedings of the First ACM Workshop on Optimization Techniques for Resources Management in Clouds*. ORMaCloud '13. ACM, New York, New York, USA, 2013, pp. 1–8. ISBN: 978-1-4503-1982-9. DOI: 10.1145/2465823.2465825.

41. D. Dib, N. Parlavantzas, and C. Morin. "SLA-Based PaaS Profit Optimization". *Concurrency and Computation: Practice and Experience* 29:21, 2017, e4251. ISSN: 1532-0626. DOI: 10.1002/cpe.4251.

42. D. Dib, N. Parlavantzas, and C. Morin. "SLA-Based Profit Optimization in Cloud Bursting PaaS". In: *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2014, pp. 141–150. ISBN: 978-1-4799-2784-5. DOI: 10.1109/CCGrid.2014.78.

43. H. A. Duran-Limon, J. Flores-Contreras, N. Parlavantzas, M. Zhao, and A. Meulenert-Peña. "Efficient Execution of the WRF Model and Other HPC Applications in the Cloud". *Earth Science Informatics* 9:3, 2016, pp. 365–382. ISSN: 1865-0481. DOI: 10.1007/s12145-016-0253-7.

44. *Eclipse Modeling Project*. URL: https://www.eclipse.org/modeling/ (visited on March 9, 2020).

45. J. Ejarque, M. De Palol, Í. Goiri, F. Juliá, J. Guitart, R. M. Badialan, and J. Torres. "Exploiting Semantics and Virtualization for SLA-Driven Resource Allocation in Service Providers". *Concurrency Computation Practice and Experience* 22:5, 2010, pp. 541–572. ISSN: 15320626. DOI: 10.1002/cpe.1468.

46. *EPA-HTTP Trace*. URL: http://ita.ee.lbl.gov/html/contrib/EPA-HTTP.html (visited on March 9, 2020).

47. H. Eriksson, M. Raciti, M. Basile, A. Cunsolo, A. Froberg, O. Leifler, J. Ekberg, and T. Timpka. "A Cloud-Based Simulation Architecture for Pandemic Influenza Simulation". In: *AMIA Annual Symposium Proceedings*. 2011, pp. 364–73.

48. E. Feller, L. Rilling, and C. Morin. "Snooze: A Scalable and Autonomic Virtual Machine Management Framework for Private Clouds". In: *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*. CCGRID '12. IEEE Computer Society, Washington, DC, USA, 2012, pp. 482–489. ISBN: 978-0-7695-4691-9. DOI: 10.1109/CCGrid.2012.71.

49. D. F. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini. "Economic Models for Allocating Resources in Computer Systems". In: *Market Based Control of Distributed Systems. World Scientific*. Press, 1996, pp. 156–183.

50. N. Ferry, F. Chauvel, H. Song, A. Rossini, M. Lushpenko, and A. Solberg. "CloudMF: Model-Driven Management of Multi-Cloud Applications". *ACM Trans. Internet Technol.* 18:2, 2018, 16:1–16:24. ISSN: 1533-5399. DOI: 10.1145/3125621.

51. *Fission - Serverless Functions for Kubernetes*. URL: https://fission.io/ (visited on March 9, 2020).

52. *FogCity - QoS-Aware Resource Management for Smart Cities*. URL: https://team.inria.fr/myriads/projects/fogcity/ (visited on March 9, 2020).

53. A. L. Freitas, N. Parlavantzas, and J.-L. Pazat. "An Integrated Approach for Specifying and Enforcing SLAs for Cloud Services". In: *Proceedings of the 5th International Conference on Cloud Computing (CLOUD)*. IEEE, 2012, pp. 376–383. ISBN: 978-1-4673-2892-0. DOI: 10.1109/CLOUD.2012.135.

54. A. G. Garćia, I. B. Espert, and V. H. Garćia. "SLA-Driven Dynamic Cloud Resource Management". *Future Generation Computer Systems* 31:0, 2014, pp. 1–11. ISSN: 0167-739X. DOI: http://dx.doi.org/10.1016/j.future.2013.10.005.

55. *Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17.5% in 2019*. URL: https://www.gartner.com/en/newsroom/press-releases/2019-04-02-gartner-forecasts-worldwide-public-cloud-revenue-to-g (visited on March 9, 2020).

56. *Gartner Says 28 Percent of Spending in Key IT Segments Will Shift to the Cloud by 2022*. URL: https://www.gartner.com/en/newsroom/press-releases/2018-09-18-gartner-says-28-percent-of-spending-in-key-IT-segments-will-shift-to-the-cloud-by-2022 (visited on March 9, 2020).

57. F. Glover, M. Laguna, et al. *Tabu Search*. Vol. 22. Springer New York, 1997.

58. T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, G. von Laszewski, C. Lee, A. Merzky, H. Rajic, and J. Shalf. "SAGA: A Simple API for Grid Applications - High-Level Application Programming on the Grid". *Computational Methods in Science and Technology: special issue "Grid Applications: New Challenges for Computational Methods"* SC05, 2005, 8(2).

59. H. Goudarzi and M. Pedram. "Multi-Dimensional SLA-Based Resource Allocation for Multi-Tier Cloud Computing Systems". In: *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing*. CLOUD '11. IEEE Computer Society, 2011.

60. B. Goupille-Lescar, E. Lenormand, C. Morin, and N. Parlavantzas. "Communication-Aware Prediction-Based Online Scheduling in High-Performance Real-Time Embedded Systems". In: *ICA3PP 2018 - 18th International Conference on Algorithms and Architectures for Parallel Processing*. LNCS. Springer, Guangzhou, China, 2018, pp. 1–15.

61. B. Goupille-Lescar, E. Lenormand, C. Morin, and N. Parlavantzas. "Introducing Dynamic Adaptation in High Performance Real-Time Computing Platforms for Sensors". In: *ANDARE 2017 - 1st Workshop on AutotuniNg and aDaptivity AppRoaches for Energy efficient HPC Systems*. Portland, OR, United States, 2017.

62. A. Haj-Ali, N. K. Ahmed, T. L. Willke, J. Gonzalez, K. Asanovic, and I. Stoica. "Deep Reinforcement Learning in System Optimization". *CoRR* abs/1908.01275, 2019. arXiv: 1908.01275.

63. R. Han, L. Guo, M. M. Ghanem, and Y. Guo. "Lightweight Resource Scaling for Cloud Applications". In: *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*. 2012, pp. 644–651. DOI: 10.1109/CCGrid.2012.52.

64. M. Haris and M. S. Manzoor. "Spatiotemporal Study of Dengue Virus Infection via Cloud Based Framework". *International Journal of Advanced Research in Computer Science and Software Engineering* 6:6, 2016, pp. 155–161.

65. P. Hasselmeyer, B. Koller, L. Schubert, and P. Wieder. "Towards SLA-Supported Resource Management". In: *Proceedings of the International Conference on High Performance Computing and Communications (HPCC)*. Springer, 2006, pp. 743–752. ISBN: 3-540-39368-4.

66. S. Haykin. *Adaptive Filter Theory*. 4th. Prentice Hall, Upper Saddle River, NJ, 2002.

67. B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center". In: *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*. NSDI'11. USENIX Association, Boston, MA, 2011, pp. 295–308.

68. R.-H. Hwang, C.-N. Lee, Y.-R. Chen, and D.-J. Zhang-Jian. "Cost Optimization of Elasticity Cloud Resource Subscription Policy". *IEEE Transactions on Services Computing* 7:4, 2014, pp. 561–574.

69. A. Iordache, C. Morin, N. Parlavantzas, E. Feller, and P. Riteau. "Resilin: Elastic MapReduce over Multiple Clouds". In: *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. 2013, pp. 261–268. DOI: `10.1109/CCGrid.2013.48`.

70. D. Irwin, P. Shenoy, P. Ambati, P. Sharma, S. Shastri, and A. Ali-Eldin. "The Price Is (Not) Right: Reflections on Pricing for Transient Cloud Servers". In: *2019 28th International Conference on Computer Communication and Networks (ICCCN)*. 2019, pp. 1–9. DOI: `10.1109/ICCCN.2019.8846933`.

71. B. Jennings and R. Stadler. "Resource Management in Clouds: Survey and Research Challenges". *Journal of Network and Systems Management* 23:3, 2015, pp. 567–619. ISSN: 1573-7705. DOI: `10.1007/s10922-014-9307-7`.

72. E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar, J. E. Gonzalez, R. A. Popa, I. Stoica, and D. A. Patterson. *Cloud Programming Simplified: A Berkeley View on Serverless Computing*. 2019. arXiv: `1902.03383 [cs.OS]`.

73. Jose Antonio Parejo and Pablo Fernandez and Antonio Ruiz-Cortés and José María García. "SLAWs: Towards a Conceptual Architecture for SLA Enforcement". In: *Proceedings of the 2008 IEEE Congress on Services – Part I (SERVICES)*. Vol. 0. IEEE Computer Society, 2008, pp. 322–328. DOI: `10.1109/SERVICES-1.2008.80`.

74. G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu. "Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures". In: *2010 IEEE 30th International Conference on Distributed Computing Systems*. 2010, pp. 62–73. DOI: `10.1109/ICDCS.2010.88`.

75. G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu. "A Cost-Sensitive Adaptation Engine for Server Consolidation of Multitier Applications". In: *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*. Middleware '09. Springer-Verlag New York, Inc., Urbanna, Illinois, 2009, 9:1–9:20.

76. D. Kim, H. Lee, H. Song, N. Choi, and Y. Yi. "Economics of Fog Computing: Interplay among Infrastructure and Service Providers, Users, and Edge Resource Owners". *IEEE Transactions on Mobile Computing*, 2019, pp. 1–1. DOI: `10.1109/TMC.2019.2925797`.

77. T. Kirkham, A. Sinha, N. Parlavantzas, B. Kryza, P. Fremantle, K. Kritikos, and B. Aziz. "Privacy Aware On-Demand Resource Provisioning for IoT Data Processing". In: *Internet of Things. IoT Infrastructures*. Ed. by B. Mandler, J. Marquez-Barja, M. E. Mitre Campista, D. Cagáňová, H. Chaouchi, S. Zeadally, M. Badra, S. Giordano, M. Fazio, A. Somov, and R.-L. Vieriu. Vol. 170. Springer International Publishing, Cham, 2016, pp. 87–95. ISBN: 978-3-319-47074-0. DOI: 10.1007/978-3-319-47075-7_11.

78. D. Król, M. Orzechowski, J. Kitowski, C. Niethammer, A. Sulisto, and A. Wafai. "A Cloud-Based Data Farming Platform for Molecular Dynamics Simulations". In: *UCC 2014*. 2014, pp. 579–584. DOI: 10.1109/UCC.2014.89.

79. *Kubeless - The Kubernetes Native Serverless Framework*. URL: https://kubeless.io/ (visited on March 9, 2020).

80. D. Kumar, G. Baranwal, Z. Raza, and D. P. Vidyarthi. "A Systematic Study of Double Auction Mechanisms in Cloud Computing". *J. Syst. Softw.* 125:C, 2017, pp. 234–255. ISSN: 0164-1212. DOI: 10.1016/j.jss.2016.12.009.

81. A. Lage Freitas. "Autonomous Service Execution Driven by Service-Level Agreements". Theses. INSA de Rennes, 2012.

82. A. Lage Freitas, N. Parlavantzas, and J. Pazat. "Cloud Resource Management Driven by Profit Augmentation". *Concurrency and Computation: Practice and Experience* 29:4, 2017, e3899. DOI: 10.1002/cpe.3899.

83. A. Lage Freitas, N. Parlavantzas, and J.-L. Pazat. "A QoS Assurance Framework for Distributed Infrastructures". In: *Proceedings of the 3rd International Workshop on Monitoring, Adaptation and Beyond - MONA '10*. ACM Press, New York, New York, USA, 2010, pp. 1–8. ISBN: 978-1-4503-0422-1. DOI: 10.1145/1929566.1929567.

84. A. Lage Freitas, N. Parlavantzas, and J.-L. Pazat. "Cost Reduction through SLA-Driven Self-Management". In: *2011 IEEE Ninth European Conference on Web Services*. Ieee, 2011, pp. 117–124. ISBN: 978-1-4577-1532-7. DOI: 10.1109/ECOWS.2011.23.

85. A. Lage Freitas, J.-L. Pazat, and N. Parlavantzas. "Ensuring Resource-Level Quality for Services on Grids". In: *2010 6th World Congress on Services*. 2010, pp. 168–169. DOI: 10.1109/SERVICES.2010.75.

86. A. Lage-Freitas, N. Parlavantzas, and J.-L. Pazat. "A Self-Adaptable Approach for Easing the Development of Grid-Oriented Services". In: *Proceedings of the IEEE International Conference on Computer and Information Technology (CIT)*. {B}radford, UK, 2010. DOI: 10.1109/CIT.2010.56.

87. K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. Huberman. “Tycoon: An Implementation of a Distributed, Market-Based Resource Allocation System”. *Multiagent and Grid Systems* 1:3, 2005, pp. 169–182.

88. A. F. Leite, V. Alves, G. N. Rodrigues, C. Tadonki, C. Eisenbeis, and A. C. M. A. de Melo. “Automating Resource Selection and Configuration in Inter-Clouds through a Software Product Line Method”. In: *2015 IEEE 8th International Conference on Cloud Computing*. 2015, pp. 726–733. DOI: 10.1109/CLOUD.2015.101.

89. P. Leitner, W. Hummer, B. Satzger, C. Inzinger, and S. Dustdar. “Cost-Efficient and Application SLA-Aware Client Side Request Scheduling in an Infrastructure-as-a-Service Cloud”. In: *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference On*. 2012, pp. 213–220. DOI: 10.1109/CLOUD.2012.21.

90. J. Li, M. Woodside, J. Chinneck, and M. Litiou. “Adaptive Cloud Deployment Using Persistence Strategies and Application Awareness”. *IEEE Transactions on Cloud Computing* 5:2, 2017, pp. 277–290.

91. U. Lublin and D. G. Feitelson. “The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs”. *Journal of Parallel and Distributed Computing* 63:11, 2003, pp. 1105–1122. ISSN: 0743-7315. DOI: http://dx.doi.org/10.1016/S0743-7315(03)00108-4.

92. N. C. Luong, P. Wang, D. Niyato, Y. Wen, and Z. Han. “Resource Management in Cloud Networking Using Economic Analysis and Pricing Models: A Survey”. *IEEE Communications Surveys Tutorials* 19:2, 2017, pp. 954–1001. ISSN: 2373-745X. DOI: 10.1109/COMST.2017.2647981.

93. M. Maćias, O. Rana, G. Smith, J. Guitart, and J. Torres. “Maximizing Revenue in Grid Markets Using an Economically Enhanced Resource Manager”. *Concurrency and Computation: Practice and Experience* 22:14, 2010, pp. 1990–2011. ISSN: 15320626. DOI: 10.1002/cpe.1370.

94. M. Maćias and J. Guitart. “SLA Negotiation and Enforcement Policies for Revenue Maximization and Client Classification in Cloud Providers”. *Future Generation Computer Systems*, 2014. ISSN: 0167739X. DOI: 10.1016/j.future.2014.03.004.

95. R. Mahmud, R. Kotagiri, and R. Buyya. “Fog Computing: A Taxonomy, Survey and Future Directions”. In: *Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives*. Ed. by B. Di Martino, K.-C. Li, L. T. Yang, and A. Esposito. Springer Singapore, Singapore, 2018, pp. 103–130. ISBN: 978-981-10-5861-5. DOI: 10.1007/978-981-10-5861-5_5.

96. H. Mao, M. Alizadeh, I. Menache, and S. Kandula. “Resource Management with Deep Reinforcement Learning”. In: *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. HotNets ’16. ACM, Atlanta, GA, USA, 2016, pp. 50–56. ISBN: 978-1-4503-4661-0. DOI: 10.1145/3005745.3005750.

97. H. Mao, P. Negi, A. Narayan, H. Wang, J. Yang, H. Wang, R. Marcus, r. addanki, M. Khani Shirkoohi, S. He, V. Nathan, F. Cangialosi, S. Venkatakrishnan, W.-H. Weng, S. Han, T. Kraska, and D. Alizadeh. “Park: An Open Platform for Learning-Augmented Computer Systems”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 2490–2502.

98. *Maui Cluster Scheduler*. URL: https://en.wikipedia.org/wiki/Maui_Cluster_Scheduler (visited on March 9, 2020).

99. P. Mell and T. Grance. *The NIST Definition of Cloud Computing*. Tech. rep. 800-145. National Institute of Standards and Technology, 2011.

100. *Melodic*. URL: https://melodic.cloud/ (visited on March 9, 2020).

101. *MIHMES*. URL: https://www6.inrae.fr/mihmes (visited on March 9, 2020).

102. *Nanomsg*. URL: https://nanomsg.org/ (visited on March 9, 2020).

103. A. Naskos, A. Gounaris, and S. Sioutas. “Cloud Elasticity: A Survey”. In: *Algorithmic Aspects of Cloud Computing: First International Workshop, ALGOCLOUD 2015, Patras, Greece, September 14-15, 2015. Revised Selected Papers*. Ed. by I. Karydis, S. Sioutas, P. Triantafillou, and D. Tsoumakos. Springer International Publishing, Cham, 2016, pp. 151–167. ISBN: 978-3-319-29919-8. DOI: 10.1007/978-3-319-29919-8_12.

104. N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA, 2007. ISBN: 0-521-87282-0.

105. E. D. Nitto, P. Matthews, D. Petcu, and A. Solberg. *Model-Driven Development and Operation of Multi-Cloud Applications: The MODAClouds Approach*. 1st. Springer Publishing Company, 2017. ISBN: 3-319-46030-7.

106. *OpenFaaS - Serverless Functions Made Simple*. URL: https://docs.openfaas.com/ (visited on March 9, 2020).

107. *OpenNebula – The Open Source Cloud Management Platform Developed for the Enterprise*. URL: https://opennebula.org/ (visited on March 9, 2020).

108. *OpenStack*. URL: http://www.openstack.org/ (visited on March 9, 2020).

109. *Oracle Grid Engine*. URL: https://www.oracle.com/technetwork/oem/grid-engine-166852.html (visited on March 9, 2020).

110. *PaaSage*. URL: https://paasage.ercim.eu/ (visited on March 9, 2020).

111. B. Palanisamy, A. Singh, and L. Liu. “Cost-Effective Resource Provisioning for MapReduce in a Cloud”. *IEEE Transactions on Parallel and Distributed Systems* 26:5, 2015, pp. 1265–1279. ISSN: 1045-9219. DOI: 10.1109/TPDS.2014.2320498.

112. *Parallel Workloads Archive*. URL: https://www.cse.huji.ac.il/labs/parallel/workload/.

113. N. Parlavantzas, L. M. Pham, C. Morin, S. Arnoux, G. Beaunée, L. Qi, P. Gontier, and P. Ezanno. “A service-based framework for building and executing epidemic simulation applications in the cloud”. *Concurrency and Computation: Practice and Experience* 32:5, 2020. DOI: https://doi.org/10.1002/cpe.5554.

114. N. Parlavantzas, L. M. Pham, A. Sinha, and C. Morin. “Cost-Effective Reconfiguration for Multi-Cloud Applications”. In: *2018 26th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, Cambridge, 2018, pp. 521–528. ISBN: 978-1-5386-4975-6. DOI: 10.1109/PDP2018.2018.00088.

115. L. M. Pham, N. Parlavantzas, C. Morin, S. Arnoux, L. Qi, P. Gontier, and P. Ezanno. “DiFFuSE, a Distributed Framework for Cloud-Based Epidemic Simulations: A Case Study in Modelling the Spread of Bovine Viral Diarrhea Virus”. In: *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, Hong Kong, 2017, pp. 304–313. ISBN: 978-1-5386-0692-6. DOI: 10.1109/CloudCom.2017.41.

116. R. C. Price, W. Pettey, T. Freeman, K. Keahey, M. Leecaster, M. Samore, J. Tobias, and J. C. Facelli. “SaTScan on a Cloud: On-Demand Large Scale Spatial Analysis of Epidemics”. In: *Online Journal of Public Health Informatics; 2(1): Ojphi.V2i1.2910*. 2010.

117. C. Quinton, D. Romero, and L. Duchien. “SALOON: A Platform for Selecting and Configuring Cloud Environments”. *Softw. Pract. Exper.* 46:1, 2016, pp. 55–78. ISSN: 0038-0644. DOI: 10.1002/spe.2311.

118. P. Rohou. *D10.1.5-Final Report*. Research Report. PaaSage: Model Based Cloud Platform Upperware, 2016.

119. *RUBiS*. URL: https://projects.ow2.org/view/rubis/ (visited on March 9, 2020).

120. C. Ruiz, H. A. Duran-Limon, and N. Parlavantzas. “An RLS Memory-Based Mechanism for the Automatic Adaptation of VMs on Cloud Environments”. en. In: *Proceedings of the 2017 Workshop on Adaptive Resource Management and Scheduling for Cloud Computing - ARMS-CC ’17*. ACM Press, Washington, DC, USA, 2017, pp. 17–23. ISBN: 978-1-4503-5116-4. DOI: `10.1145/3110355.3110358`.

121. C. Ruiz, H. A. Duran-Limon, and N. Parlavantzas. “Towards a Software Product Line-Based Approach to Adapt IaaS Cloud Configurations”. en. In: *Proceedings of the 9th International Conference on Utility and Cloud Computing - UCC ’16*. ACM Press, Shanghai, China, 2016, pp. 398–403. ISBN: 978-1-4503-4616-0. DOI: `10.1145/2996890.3007893`.

122. C. A. Ruiz Diaz. “A SPL-Based Approach for the Configuration and Adaptation of IaaS Deployments”. Theses. University of Guadalajara, 2018.

123. F. Samreen, Y. Elkhatib, M. Rowe, and G. S. Blair. “Daleel: Simplifying Cloud Instance Selection Using Machine Learning”. In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. 2016, pp. 557–563. DOI: `10.1109/NOMS.2016.7502858`.

124. T. Sandholm and K. Lai. “Dynamic Proportional Share Scheduling in Hadoop”. In: *15th Workshop on Job Scheduling Strategies for Parallel Processing*. 2010.

125. T. Sandholm and D. Lee. “Notes on Cloud Computing Principles”. en. *Journal of Cloud Computing* 3:1, 2014, p. 21. ISSN: 2192-113X. DOI: `10.1186/s13677-014-0021-5`.

126. M. Satyanarayanan. “The Emergence of Edge Computing”. *Computer* 50:1, 2017, pp. 30–39. DOI: `10.1109/MC.2017.9`.

127. M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. “Omega: Flexible Scalable Schedulers for Large Compute Clusters”. In: *Proceedings of the 8th ACM European Conference on Computer Systems*. Eurosys’13. 2013.

128. U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh. “A Cost-Aware Elasticity Provisioning System for the Cloud”. In: *2011 31st International Conference on Distributed Computing Systems*. 2011, pp. 559–570. DOI: `10.1109/ICDCS.2011.59`.

129. J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya. “Libra: A Computational Economy-Based Job Scheduling System for Clusters”. *Softw. Pract. Exper.* 34:6, 2004, pp. 573–590. ISSN: 0038-0644. DOI: `10.1002/spe.581`.

130. G. Staples. “TORQUE Resource Manager”. In: *Proceedings of ACM/IEEE Conference on Supercomputing*. 2006.

131. P. Sukcharoen, S. Pumma, O. Mongkolsermporn, T. Achalakul, and X. Li. “Design and Analysis of a Cloud-Based Epidemic Simulation Framework”. In: *2012 9th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*. 2012, pp. 1–4.

132. A. Tasiopoulos, O. Ascigil, I. Psaras, S. Toumpis, and G. Pavlou. “FogSpot: Spot Pricing for Application Provisioning in Edge/Fog Computing”. *IEEE Transactions on Services Computing*, 2019, pp. 1–1. ISSN: 2372-0204. DOI: `10.1109/TSC.2019.2895037`.

133. L. Thai, B. Varghese, and A. Barker. “A Survey and Taxonomy of Resource Optimisation for Executing Bag-of-Task Applications on Public Clouds”. *Future Generation Computer Systems* 82, 2018, pp. 1–11. ISSN: 0167-739X.

134. *Twisted*. URL: `https://github.com/twisted/twisted` (visited on March 9, 2020).

135. V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O’Malley, S. Radia, B. Reed, and E. Baldeschwieler. “Apache Hadoop YARN: Yet Another Resource Negotiator”. In: *Proceedings of the 4th Annual Symposium on Cloud Computing*. SOCC ’13. ACM, Santa Clara, California, 2013, 5:1–5:16. ISBN: 978-1-4503-2428-1. DOI: `10.1145/2523616.2523633`.

136. Z. Wang, Y. Chen, D. Gmach, S. Singhal, B. Watson, W. Rivera, X. Zhu, and C. Hyser. “AppRAISE: Application-Level Performance Management in Virtualized Server Environments”. *IEEE Transactions on Network and Service Management* 6:4, 2009, pp. 240–254.

137. L. Wu, S. Garg, and R. Buyya. “SLA-Based Resource Allocation for Software as a Service Provider (SaaS) in Cloud Computing Environments”. In: *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium On*. 2011.

138. J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif. “Autonomic Resource Management in Virtualized Data Centers Using Fuzzy Logic-Based Approaches”. *Cluster Computing* 11:3, 2008, pp. 213–227.

139. N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, B. Smith, and R. H. Katz. “Selecting the Best VM across Multiple Public Clouds: A Data-Driven Performance Modeling Approach”. In: *Proceedings of the 2017 Symposium on Cloud Computing*. SoCC ’17. Association for Computing Machinery, Santa Clara, California, 2017, pp. 452–465. ISBN: 978-1-4503-5028-0. DOI: `10.1145/3127479.3131614`.

140. S. Yang, D. Peng, T. Meng, F. Wu, G. Chen, S. Tang, Z. Li, and T. Luo. “On Designing Distributed Auction Mechanisms for Wireless Spectrum Allocation”. *IEEE Trans. Mob. Comput.* 18:9, 2019, pp. 2129–2146. DOI: 10.1109/TMC.2018.2869863.

141. C. S. Yeo, S. Venugopal, X. Chu, and R. Buyya. “Autonomic Metered Pricing for a Utility Computing Service”. *Future Gener. Comput. Syst.* 26:8, 2010, pp. 1368–1380. ISSN: 0167-739X. DOI: 10.1016/j.future.2009.05.024.

142. A. B. Yoo, M. A. Jette, and M. Grondona. “SLURM: Simple Linux Utility for Resource Management”. In: *Job Scheduling Strategies for Parallel Processing*. Springer. 2003, pp. 44–60.

143. M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. “Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling”. In: *Proceedings of the 5th European Conference on Computer Systems*. EuroSys ’10. ACM, Paris, France, 2010, pp. 265–278. ISBN: 978-1-60558-577-2. DOI: 10.1145/1755913.1755940.

144. A. Zavodovski, S. Bayhan, N. Mohan, P. Zhou, W. Wong, and J. Kangasharju. “DeCloud: Truthful Decentralized Double Auction for Edge Clouds”. In: *39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019, Dallas, TX, USA, July 7-10, 2019*. 2019, pp. 2157–2167. DOI: 10.1109/ICDCS.2019.00212.

145. *ZeroMQ*. URL: http://www.zeromq.org/ (visited on March 9, 2020).

146. L. Zheng, C. Joe-Wong, C. W. Tan, M. Chiang, and X. Wang. “How to Bid the Cloud”. *SIGCOMM Comput. Commun. Rev.* 45:4, 2015, pp. 71–84. ISSN: 0146-4833. DOI: 10.1145/2829988.2787473.

147. X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. Mckee, C. Hyser, D. Gmach, R. Gardner, T. Christian, and L. Cherkasova. “1000 Islands: An Integrated Approach to Resource Management for Virtualized Data Centers”. *Cluster Computing* 12:1, 2009, pp. 45–57. ISSN: 1386-7857.

148. P. Zoghi, M. Shtern, M. Litoiu, and H. Ghanbari. “Designing Adaptive Applications Deployed on Cloud Environments”. *ACM Trans. Auton. Adapt. Syst.* 10:4, 2016, 25:1–25:26. ISSN: 1556-4665. DOI: 10.1145/2822896.