UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TÉRCIO DE MORAIS SAMPAIO SILVA

MULTI-CLOUD-AWARE MIDDLEWARE

Recife
2021

TÉRCIO DE MORAIS SAMPAIO SILVA

MULTI-CLOUD-AWARE MIDDLEWARE

Ph.D. Thesis presented to the graduate program in Computer Science of Centre of Informatics of Federal University of Pernambuco as a partial fulfillment of the requirements for the degree of Philosophy Doctor in Computer Science. Major field: Computer Networks and Distributed Systems.

Advisor: Nelson Souto Rosa

Recife
2021

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

S586m Silva, Tércio de Morais Sampaio
Multi-cloud-aware middleware / Tércio de Morais Sampaio Silva. – 2021.
128 f.: il., fig., tab.

Orientador: Nelson Souto Rosa.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIN, Ciência da Computação, Recife, 2021.
Inclui referências.

1. Rede de computadores. 2. Elasticidade. I. Rosa, Nelson Souto (orientador). II. Título.

004.6 CDD (23. ed.) UFPE- CCEN 2021 - 181

TÉRCIO DE MORAIS SAMPAIO SILVA

MULTI-CLOUD-AWARE MIDDLEWARE

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação. Área de Concentração: Redes de Computadores e Sistemas Distribuídos.

Aprovada em: 23/08/2021.

Orientador: Prof. Dr. Nelson Souto Rosa

I dedicate this Ph.D. to my powerpuff girls: Bubbles (Carol), Blossom (Maria Júlia) and Buttercup (Helena).

ACKNOWLEDGEMENTS

There has been a challenging and long way until being here, writing these acknowledgements.

I gratefully acknowledge my wife and daughters ("The Powerpuff girls" in dedication) for love, support, and belief. Without you, none of this would be possible. Thanks for standing by me.

I would like to express my gratitude to my supervisor, Dr Nelson Souto Rosa, for believing and trusting me. For the guidance, dedication and encouragement. I have accumulated a lot of experience that goes beyond doctoral support.

I would like to thank my co-workers at campus Arapiraca-UFAL for supporting me during the doctoral period. My friends Marcelo Oliveira and Rodrigo Pinheiro at campus A. C. Simões-UFAL, my hosts at the LATIM lab and LCCV, respectively.

Thanks to *Research Group on Foundations and Applications of Distributed Systems* (GFADS) sponsored by Dr Nelson Rosa And Dr Fernando Lins.

Thanks to the examination committee for their availability and contribution. Particularly for professor Dr Paulo Maciel for your valuable words in the presentation beginning.

Finally, thanks to everyone who, directly or indirectly, collaborated with this thesis.

ABSTRACT

The idea of collaboration among clouds has emerged to address issues related to single cloud adoption. From the developer's perspective, distributed applications can take advantage of multi-cloud environments to create, extend and integrate their components across cloud domains in a dynamic, automatic and transparent way and improve their quality requirements, such as availability, performance, and scalability. However, the management complexity increases substantially in this scenario, whose responsibility lies with the developer. Despite standardisation efforts, most applications cannot exploit multi-cloud benefits (e.g., elasticity). Furthermore, solutions for interoperability, cloud's administrative boundaries, and the lock-in problem remain as open challenges, concerning at the IaaS level, unaware of what is running on top of it. This work proposes a middleware architecture for distributed applications in multi-cloud environments: Multi-Cloud Aware Middleware (M-CaMid). The architecture combines middleware functionalities with IaaS services for distributed application management. M-CaMid takes advantage of elasticity to provide a cross management that integrates infrastructure and application layers (vertical management) and integrate many clouds (horizontal management), enabling a holistic view of distributed systems and a better application performance and rational usage of cloud resources. Experiments were carried out to assess the performance gains due to M-CaMid. Results show that cross management can improve performance of distributed applications and rational usage of cloud resources for distributed applications in a multi-cloud environment. Thesis' unique contributions are: (*i*) middleware architecture for distributed application management in a multi-cloud environment; (*ii*) cross management that integrates infrastructure and application layers and many clouds; and (*iii*) multi-cloud elasticity that extends single cloud elasticity to the multi-cloud scope.

Keywords: middleware; multi-cloud computing; elasticity.

Resumo

A ideia de colaboração entre nuvens surgiu para tratar de limitações relacionadas à adoção de uma única nuvem. Do ponto de vista do desenvolvedor, aplicações distribuídas podem tirar proveito de ambientes *multi-cloud* para criar, estender e integrar seus componentes em várias nuvens de forma dinâmica, automática e transparente. No entanto, a complexidade do gerenciamento aumenta substancialmente neste cenário, cuja responsabilidade é do desenvolvedor. Apesar dos esforços de padronização dos serviços de nuvem, a maioria das aplicações distribuídas não usufrui dos benefícios de várias nuvens como, por exemplo, escalar seus componentes distribuídos em outras nuvens. Além disso, a maioria das soluções para interoperabilidade, limites administrativos e escalabilidade concentra-se em nível IaaS, ignorando aplicações que rodam acima da camada de infraestrutura. Este trabalho apresenta uma arquitetura de middleware para aplicações distribuídas em ambientes *multi-cloud – Multi-Cloud Aware Middleware* (M-CaMid). A arquitetura combina funcionalidades de middleware com serviços de IaaS para o gerenciamento de aplicações distribuídas em ambientes *multi-cloud*. M-CaMid tira proveito da elasticidade para o gerenciamento transversal que integra camadas de infraestrutura e software (vertical) e múltiplas nuvens (horizontal), permitindo uma visão holística do sistema distribuído e proporcionando melhor suporte ao gerenciamento do desempenho de aplicações distribuídas e uso racional de recursos das nuvens. Experimentos realizados avaliaram o ganho de desempenho de aplicações distribuídas, bem como o uso racional de recursos de infraestrutura, demonstrando os benefícios do uso da elasticidade de múltiplas nuvens.As contribuições desta tese são: (*i*) arquitetura de middleware para o gerenciamento de aplicações distribuídas em ambientes de múltiplas nuvens; (*ii*) gerenciamento transversal, integrando as várias camadas de um sistema distribuído e integrando várias nuvens; e (*iii*) o uso da elasticidade em um ambiente multi-cloud.

Palavras-chave: middleware; computação *multi-cloud*; elasticidade.

List of Figures

List of Tables

List of Acronyms

API	Application Programming Interface
CPU	Central Processing Unit
DMTF	Distributed Management Task Force
DOC	Distributed Object Computing
GICTF	Global Inter-Cloud Technology Forum
IaaS	Infrastructure-as-a-Service
ICT	Information and Communication Technology
IP	Internet Protocol
IT	Information Technology
MAPE-K	Monitor-Analyse-Plan-Execute over a shared Knowledge
NIST	National Institute of Standards and Technology
OASIS	Organisation for the Advancement of Structured Information Standards
OCCI	Open Cloud Computing Interface
OS	Operating System
P2P	Peer-to-Peer
PaaS	Platform-as-a-Service
QoS	Quality of Service
SaaS	Software-as-a-Service
SLA	Service-Level Agreement
SLO	Service-Level Objective
TCP	Transmission Control Protocol
VI	Virtual Image
VM	Virtual Machine
XaaS	Everything-as-a-Service

CONTENTS

1 Introduction

Cloud computing has become one of the main streams of Information and Communication Technology (ICT) in industry and academy with predictions of changing the computing into the fifth utility such as electricity and telephony (Parkhill, 1966; Buyya et al., 2009; Gill et al., 2022). However, because of its youth, many challenges are still open and, as the demand for cloud services grows, new challenges arise. This chapter introduces challenges related to developing and managing multi-cloud distributed applications, contextualising the scenario, and presenting the current issues. Furthermore, this thesis is introduced as a solution for distributed systems based on middleware that can support the development and management of distributed applications by leveraging cloud elasticity in a multi-cloud environment.

1.1 Context

Nowadays, cloud computing is a consolidated paradigm for on-demand services delivery based on virtual provisioning of computational resources, such as processing, storage, networking, and application services. We can observe the growth in the adoption of cloud computing to offer services without requiring significant capital and technical skills for managing the service's infrastructures.

The abstraction of the infrastructure management allows cloud developers and cloud users to focus only on the application's business logic, avoiding knowing about complexities concerning underlying software and hardware infrastructure. Another benefit is the ability to re-size dynamically virtual resources according to the current demand, namely *elasticity*. Cloud users can allocate more resources at workload peak times and release them when it is no longer needed instead of purchasing permanent hardware and software resources to attend to eventual workload demands.

Both academia and industry have coined their definitions of cloud computing in this beginning (Foster et al., 2008; Vaquero et al., 2009; Armbrust et al., 2009; Lenk et al., 2009; Mell and Grance, 2011). However, a more well-defined way to describe cloud computing is characterising it by the following criteria (Armbrust et al., 2009): consuming resource as a service, pay-as-you-go economic model and rapid elasticity and self-service (without human interaction). Cloud Computing promises benefits to its users on supporting of Everything-as-a-Service (XaaS), management complexity reduction, pay-per-use billing, elasticity, heterogeneity, infinite resources illusion, and Quality of Service (QoS).

Elasticity is a crucial cloud property that motivates the consolidation of cloud computing paradigm growth. It refers to the ability for adding and removing resources "on the fly" to adapt according to the variation of the user demand in real-time (Al-Dhuraibi et al., 2018), providing the illusion of infinite resources. By elasticity, it is possible to improve desirable distributed system's requirements, such as availability, scalability, and performance.

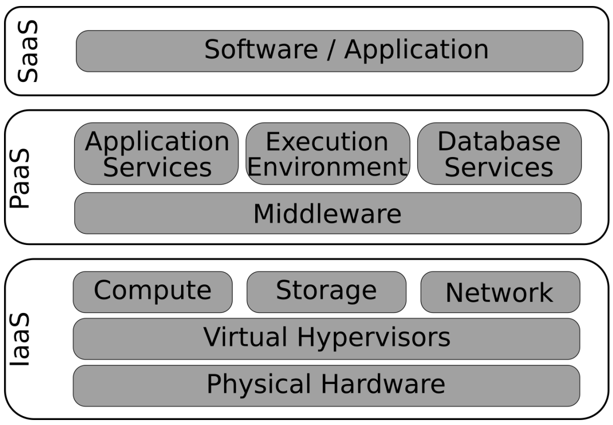
Cloud computing environments are also featured by their service models logically organised in a layers stack (Figure 1): Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). The three layers comprise a natural hierarchy that characterises abstraction levels of delivery models.

SaaS encapsulates the functionality of business processes, delivering the application as a final product. The cloud consumer has limited administrative control in this layer since its interaction is only with the application interface.

PaaS is an environment for running and developing user' applications and data management. It relies on pre-packaged products used to support the application lifecycle. However, with applications' development and configuration flexibility, the cloud consumer is still tied in some aspects of the application environment. For example, the development tools are usually designed for running exclusively on a specific PaaS environment, leading to the lock-in problem.

Finally, IaaS provides hardware virtualisation for computing, network, and data storage resources through virtual hypervisors. Virtual resources can be accessed and managed through IaaS interfaces. The IaaS environment allows high-level control over the configuration and utilisation of its resources. The more abstract the level, the more restrictive is the user control over the resources. A natural consequence of this architecture is service integration among the levels. For example, a PaaS environment can be deployed upon a IaaS environment benefiting from cloud facilities instead of deploying upon the traditional hardware infrastructure.

Figure 1 – Cloud architecture based on service layers.



Source: author (2021).

Regarding infrastructure provisioning, the cloud deployment model can be characterised as private, public or hybrid (Mell and Grance, 2011). Private cloud refers to on-premise infrastructure provisioned for the exclusive use of a single organisation. Public clouds allow public access to their resources, and hybrid cloud combines both deployment models. For example, a private cloud can extend its infrastructure by allocating resources from a public cloud.

As observed in the aforementioned hybrid cloud example, cloud computing has evolved to integrate several services at different abstraction layers and cloud environments. Bittman (2008) has predicted the evolution of cloud in three phases:

- **Monolithic**, which cloud services were based on proprietary/internal architectures - an island of cloud services without integration. The principal purpose of this phase is to take benefit of the on-demand allocation of single virtualised resources without intention for integrating with other services at different layers;

- **Vertical Supply Chain**, which cloud providers leverage services from other cloud providers. In this scenario, higher-level services (e.g., applications at the SaaS layer, namely service provider) are hosted on underlying infrastructure (virtual machines at IaaS layer) of other cloud providers – namely, IaaS provider. Various small companies provide value-added services around deploying, managing, and scaling applications on compute clouds, building many ecosystems around the clouds (Sotomayor et al., 2009; Maximilien et al., 2009b); and

- **Horizontal Federation**, when smaller and medium providers federate their resources horizontally to gain economies of scale and improve their peak capacity (overdraft protection and cloud bursting). Federation of clouds foresees collaboration among clouds in a coordinated way to assure integration at data, processing and security levels. Furthermore, business operators predict that process toward an interoperable federated cross-cloud scenario will begin shortly (Celesti et al., 2010a).

Integrating clouds (private or public clouds) can follow different architectural models. Because of the many models and definitions to characterise groups of cloud sharing resources, this thesis refers to collaboration among cloud in a general way as multi-cloud computing.

1.2 Motivation

Private clouds are usually physical-limited on-premise infrastructure exclusively used by a single organisation when compared to public clouds. As observed by Gill (2021), around 75%of enterprise workloads will run in the private cloud while only 12% intend to reduce their on-premise infrastructure. The main motivations behind these figures are performance, compliance and cost. Usually, these enterprises keep their applications running in on-premises data centres. Besides, enterprises plan to move their private clouds towards multi-cloud architecture. McGillicuddy (McGillicuddy, Shamus, 2021) envisages that most organisations plan to expand their private clouds to more data centre sites.

Cloud computing evolution has been motivated to move towards the third phase. That is, small and large enterprises will opt to use several cloud providers rather than stand on a single one (Maximilien et al., 2009b; Varghese and Buyya, 2018; Takamura et al., 2019; Ravi and Thangarathinam, 2019). Another motivation is that clouds are becoming increasingly geographically distributed to support new applications paradigms (Buyya et al., 2018).

Small and medium organisations can connect their on-premise infrastructure to compose a multi-cloud infrastructure. For example, a small private cloud can extend its infrastructure by sharing resources with other private clouds. This collaboration can address single cloud hurdles, such as hardware limitations, the lock-in problem, and administrative boundaries.

Although public cloud providers advertise the illusion of infinite resources, there is a natural limitation of physical resources (processor, memory, and network), mainly in cases of small and medium cloud providers (Aoyama and Sakai, 2011; Esposito et al., 2013; Ficco et al., 2014). It is desirable that private data centres, with limited resources, can extend their capabilities to another cloud in workload peak times and shrink them when it is no longer needed. Moreover, it is hard to deal with unpredictable workload behaviours without guarantees of resource scalability since the cloud provider meets different user's demand needs (Buyya et al., 2010; Aoyama and Sakai, 2011; Grozev and Buyya, 2014). Complying with QoS agreements is a critical task. Services can become unavailable if the underlying infrastructure is not enough or even fails. Even large providers such as Amazon and Google have experienced some kinds of outage (Budnik, 2013; Pariseau and Jones, 2014; Roumani and Nwankpa, 2019).

From the perspective of small and medium organisations, three main motivations drive the mentioned evolution: cost optimisation, QoS improvement (Petcu, 2014), and the well-known vendor lock-in problem (Opara-Martins et al., 2016), which becomes cloud consumer unable to change or add technologies from another cloud provider. The lock-in problem is the result of proprietary technologies that are incompatible with other providers (Bouzerzour et al., 2020).

Another aspect motivating the adoption of multiple clouds is the increasing software complexity, notably distributed applications (e.g., social networks and weather forecast applications) whose components can run on different devices or even different clouds (Desprez et al., 2010). These applications can require distributed resources that a single private cloud may not be enough to support application's QoS requirements.

The adoption of many clouds considers integrating services at different layers from different cloud providers. Since both cloud provider and consumer needs are diverse, different approaches are found in the literature, being inter-cloud, cloud federation and multi-cloud the most adopted (Buyya et al., 2010; Petcu, 2014; Cuadrado et al., 2014; Toosi et al., 2014; Elhabbash et al., 2019; Ferrer et al., 2019). Although diversity, adopting more than one cloud, aims to face limitations of single cloud solutions. Multi-Cloud computing brings many advantages for cloud consumers. Cloud consumers and applications can extend their capabilities dynamically for improving their services through collaboration with other clouds.

The ability for extending infrastructure resources over cloud domains may improve desirable requirements for distributed applications. Multi-Cloud environments allow application resilience, availability, reliability, performance and fault tolerance. Furthermore, choosing the best solution for cloud consumers needs to aggregate flexibility on the development and management of applications. Cloud consumers can coordinate and distribute application components over diverse cloud domains according to their specific requirements. Besides the classical challenges of distributed systems, such as openness, heterogeneity and scalability (Tanenbaum and Steen, 2007; Coulouris et al., 2013), the multi-cloud computing paradigm has added elasticity as a new desired property for distributed applications.

On the other hand, middleware is in charge of dealing with distribution aspects on supporting quality properties, such as scalability, performance, and availability. Middleware for cloud computing aims to promote scalability, performance, response time and efficient resource optimisation (Chauhan et al., 2017).

The role of middleware systems in cloud computing environments has ranged from a virtualisation management tool to a data format converter (Spring, 2011). At the middleware level, abstractions are necessary for application developers to focus on functional concerns instead of non-functional concerns that must be delegated to the middleware layer (Imai et al., 2016).

Middleware is used to deal with distribution aspects. It aims to provide transparency among distributed applications, abstracting communication details, distribution, and integrating heterogeneous technologies. Furthermore, the middleware's role is beyond the communication capabilities and usually also includes common and specific services such as management and monitoring services (Schantz and Schmidt, 2002). Adopting middleware for cloud applications can bring advantages:

- Middleware can perform the integration among cloud domains in a transparent fashion, treating multiple clouds as a homogeneous environment;
- Developers can focus on business logic issues since middleware leverages the QoS application's requirements by benefiting from cloud services;
- Middleware can combine different types of services from the IaaS layer to meet QoS requirements transparently;
- Middleware can bring flexibility for management and communication of applications, dealing with rapid changes in the environment.

In a nutshell, multi-cloud computing can benefit small and medium companies if they integrate their on-premise infrastructures to share resources and services. Sharing resources and services can mitigate the hardware limitation of single private clouds.

However, organisations must evaluate the trade-off between adapting their infrastructures to meet some standards and keeping autonomy to plan their infrastructure architecture. Autonomy in a cloud platform means that organisations can define their infrastructure regardless of cloud architectures and platforms composing the multi-cloud environment. Furthermore, application performance may demand more resources in a limited environment. From the developer perspective, there is a trade-off between application performance and management of limited resources.

1.3 The Problem

As discussed in the previous section, organisations have strong motivations to shift to multi-cloud computing, including elasticity supported by many clouds. Despite the benefits of multi-cloud computing, both distributed application and elasticity management still needs to address challenges until reaching plenty of functionality. The state-of-the-art presents several shortcomings related to management in the multi-cloud context, such as heterogeneity, networking, lack of control, automatism, scalability, and security.

In a limited infrastructure as a single private cloud, improving performance may be impractical. Even in a multi-cloud environment composed of private clouds, there are hardware limitations compared to public clouds. In this scenario, applications of many purposes belonging to single private clouds are competing for resources. There is a trade-off between application performance and limited resource sharing, and the more performance application requires, the more resource is needed.

Currently, existing works address the mentioned issues. However, most efforts on cloud management focus on infrastructure aspects, prioritising solutions to cloud providers, not for consumer ones (Papazoglou and van den Heuvel, 2011; Petcu, 2014; Varghese and Buyya, 2018). These provider-centric approaches are restrictive and with limited scope, not giving facilities to the cloud user. Furthermore, the elasticity at IaaS level can lead to problems of under-provisioning and over-provisioning (Islam et al., 2012; Mondéjar et al., 2013; Lorido-Botrán et al., 2014) that is harmful to small and medium organisations. On the other hand, client-centric approaches offer more flexibility and control in elasticity management, such as on-the-fly and opportunistic combinations of different cloud services (Singhal et al., 2013).

Regarding distributed application management, configuring and executing applications across many clouds is a challenging task since the developer must deal with heterogeneous cloud technologies to integrate distributed application components (Leite et al., 2016; Saatkamp et al., 2020). Accomplishing communication and distribution management of applications running on different clouds becomes a complex, laborious and error-prone task for application developers. Besides, applications are not developed to take advantage of cloud benefits, such as resource monitoring and elasticity. These applications require mechanisms to hide underlying complexities, such as middleware.

Distributed systems' transparencies (e.g., access, location, fault-tolerance, and scalability) are services provided by middleware. However, traditional middleware is not designed to leverage cloud computing facilities and capabilities. In practice, middleware platforms (Bernstein, 1996; Vinoski, 2002) are developed to non-cloud infrastructures, where resources must be previously defined. These infrastructures behave in a planned fashion, in contrast to the dynamic behaviour of cloud computing.

With the benefits of elasticity, distributed applications can take advantage of multi-cloud computing transparently and automatically. Nevertheless, research challenges of elasticity management remain, such as the management of computational demand and application performance (Imai et al., 2012; Srirama and Ostovar, 2014), and scaling out and in virtual resources (infrastructure and application resources) accurately (Imai et al., 2013; Qu, 2016; Muñoz-Escoí and Bernabéu-Aubán, 2016; Al-Dhuraibi et al., 2018; Pahl et al., 2018).

The efficient use of elasticity requires automatically and timely provisioning services without human intervention (Qu, 2016). Furthermore, to fulfil these characteristics at the application level is a complicated, tiresome and error-prone task for developers (Hoffert et al., 2010; Hamdaqa and Tahvildari, 2012).

This scenario, where distributed applications run in a limited-resource multi-cloud environment, defines some requirements to comply with the distributed application's needs and provides an opportunistic, automatic, and transparent application lifecycle management.

The middleware architecture must be revisited, intending to tailor it to the multi-cloud context. Furthermore, existing cloud middleware solutions (Maximilien et al., 2009b; Abbadi, 2011b,a; Amin et al., 2012; Azeez et al., 2010; Ferretti et al., 2010; Hoffert et al., 2010; Merle et al., 2011; Paraiso et al., 2016) do not draw the cloud capabilities on the well-known Distributed Object Computing (DOC) middleware architecture, proposed by Schantz and Schmidt (2002) composed of the layers infrastructure, distribution, common services and domain-specific services. Although Schantz's model is based on the specific paradigm of distributed objects, most middleware technologies follow many of its principles, even that it is not explicitly declared. Furthermore, DOC middleware architecture distributes middleware's responsibilities through its layers, clearly defining the distribution of the roles.

1.4 Partial Solutions

Multi-cloud elasticity management can be divided into provider-centric and client-centric solutions (Singhal et al., 2013; Bouzerzour et al., 2020). The first one focuses on IaaS resource provisioning, allowing the management by cloud providers, while in the last one, the control focuses on consumer and application needs, shifting more power to cloud clients. Most solutions focus on cloud integration and elasticity at the IaaS level (provider-centric), neglecting the application requirements. In fact, provider-centric managers are unaware of what runs on top of a Virtual Machine (VM). These provider-centric solutions can lead to problems of resource under-provisioning and over-provisioning (Islam et al., 2012; Mondéjar et al., 2013; Lorido-Botrán et al., 2014). They commonly work by moving the whole VM between the clouds. Over time, a VM may become underused, wasting resources and being costly. On the other hand, the client-centric approach offers more flexibility, shifting the infrastructure control to third party entities.

In the last years, the container-based virtualisation approach has gained more attention because it reduces the deployment time and application management complexity and flexibility, enabling a lightweight environment (Rodriguez and Buyya, 2019) compared to hardware virtualisation. As VM, containers are designed to provide portable Operating System (OS) environment to run applications. Containers allow fine-grained sharing of cloud resources and reduce the deployment overhead (Pahl, 2015). However, like VMs, containers are subject to elasticity and demanding management mechanisms.

Since the client-centric approach allows resource management at different abstraction levels (infrastructure, platform and application), they can deliver better management services to application developers, such as on-the-fly and opportunistic combinations of infrastructural resource and application components in a multi-cloud environment at different abstraction levels (Singhal et al., 2013). However, most efforts exploring multi-level abstractions are single cloud solutions (Naskos et al., 2016).

Many efforts can be found in the literature, presenting a solution for many contexts. However, few initiatives intend to cope entire magnitude of elasticity power.

1.5 Research Questions

Applications can take advantage of the elasticity service of many cloud providers to support performance requirements. However, in a scenario with limited-hardware cloud infrastructures composing a multi-cloud system, meeting application performance may be a problem since scaling solutions require additional resources. Elasticity management also requires timely and precise resource management mechanisms to automatically respond to the environment when it changes and avoid over/under-provisioning scarce infrastructure resources.

From the developer perspective, middleware must abstract elasticity management and multi-cloud integration complexities, besides meeting non-functional goals of distributed applications like performance. This challenge implies designing a middleware architecture to explore available cloud resources as more as possible in a multi-cloud environment to meet application performance requirements. This challenge leads to other three ones: *(a)* how to incorporate elasticity mechanisms into a multi-layer middleware architecture along with traditional middleware elements since middleware architecture is not designed to leverage the multi-cloud benefits, specifically the multi-cloud elasticity; *(b)* how to tackle the multi-cloud heterogeneity challenge, since cloud platforms and cloud providers are independents and adopt proprietary management standards; and *(c)* how to explore the elasticity abstraction levels to provide timely and efficient mechanisms to manage infrastructure and application resources.

1.6 Hypothesis

This thesis presents a multi-cloud-aware middleware architecture (M-CaMid) for supporting the development and management of distributed applications running on a multi-cloud environment. The proposed architecture combines middleware functionalities with cloud services provided by the IaaS layer.

The thesis is based on the following hypothesis:

***Hypothesis*:**

> Middleware-based approaches can support the development and run-time management of distributed applications in a limited-resource multi-cloud environment by leveraging cloud elasticity, allowing timely and more precise application management, resource usage maximisation, and multi-cloud elasticity automatically and transparently.

The middleware architecture can meet the requirements mentioned before by supporting the ability to re-configuring the distributed application and its infrastructure dynamically and automatically by taking advantage of the elasticity of many cloud domains with limited infrastructure resources. Managing elasticity across many clouds can improve distributed applications' high availability, scalability, and performance, mitigating the trade-off between performance and resource limitations.

M-CaMid is a distributed architecture built on the DOC middleware architecture (Schantz and Schmidt, 2002). In this model, middleware functionalities are drawn into the layer stack perspective. This view contributes to a better understanding and positioning of the relationship between middleware and cloud computing.

Since cloud providers are distributed and independent, the thesis takes into account a model of the multi-cloud environment for distributed applications composed of three domains: *node*, *cloud*, and *multi-cloud*. A node domain comprises a Virtual Machine (VM) provided by a cloud provider, a set of components of the distributed application running in this VM, and an instance of the middleware (node middleware) for managing the respective domain. A set of node domains in the same cloud provider constitutes a cloud domain managed by a middleware instance (cloud middleware) to orchestrate all nodes into the cloud domain.

The middleware organises its duties according to its role in each domain. For example, for each cloud domain (e.g., cloud provider), the middleware instantiates a management module that controls all application's components and virtual resources (node domains) in the respective cloud domain. Two or more cloud domains establish the multi-cloud domain. The cloud domains must collaborate through their cloud middleware instances to provide seamless management of the distributed application. Each middleware instance must be autonomous concerning its management.

Middleware should also ensure adequate response to events and changes in the environment to accomplish efficient management through a distributed management architecture. Distribution and elasticity management is achieved through dynamic and automatic management mechanisms regardless of the cloud domains. The management design is based on a distributed architecture where management components are implemented according to the respective domain. Each middleware's instance monitors and controls the application's components, running on its respective cloud domain. The instances communicate with each other for sharing monitoring information among them. The monitoring information coming from all cloud domains composes the holistic view of the whole application. Furthermore, each management instance is autonomous on the control of its managed components and can request IaaS resources to other cloud domains.

For example, it can replicate components in other cloud domains to tolerate fails or improve application performance.

Experiments were performed to observe and evaluate the M-CaMid architecture on managing elasticity in a multi-cloud environment to benefit distributed applications' performance. A systematic approach for performance evaluation was adopted (Jain, 1991) to accomplish the experiments. Some steps are defined to conduct the experiments: goals and system boundaries, scenarios, services and outcomes, metrics, parameters and factors and experiment design. Then, the results are presented and discussed.

The experiment's scenario comprises two cloud infrastructures with limited hardware resources where distributed applications are running. A distributed application is submitted to a sudden peak of workload that affects its performance. It is expected M-CaMid steps in the environment to reconfigure both application and infrastructure to reestablish the expected system behaviour pre-defined by the developer.

1.7 Unique Contributions

This thesis contributes to scientific and technological communities by presenting a middleware architecture to support distributed applications' development and run-time management in multi-cloud environments. In short, the unique contributions of this work are:

1. *Multi-Cloud elasticity* extends cross management of a single cloud environment to a multi-cloud environment. Applications components distribution can range from a single VM to many VMs spread in many clouds. M-CaMid manages elasticity in both domains: single cloud and multi-cloud. Furthermore, multi-cloud elasticity integrates single clouds elasticity services to benefit distributed applications yielding precise resource usage and timely response to undesired events through cross-layer management mechanisms;

2. *Cross management*, which defines two management dimensions: vertical (cross-layer) and horizontal (cross-domain), allowing a more precise and transparent multi-grained elasticity: coarse-grained elasticity at the virtual machine level and fine-grained elasticity at application component level. Cross management enables developers to explore elasticity services from many clouds at the same time affords timely application scalability and infrastructure resource rational usage; and

3. *Multi-Cloud middleware architecture* in addition to classical DOC middleware architecture, identifying and locating the new functionalities in the middleware layer stack. The management layer brings mechanisms to integrate application and infrastructure layers. Thus, the middleware is aware of both the application and infrastructure layers. Besides, its hybrid architecture meets both cloud (centralised architecture) and multi-cloud (decentralised architecture) behaviours;

1.8 Thesis Structure

The rest of this document is organised as follow:

- **Chapter 2** presents the basic concepts of cloud computing, multiple clouds paradigms, multi-cloud applications, and multi-cloud middleware. Finally, middleware, its classification and benefits are discussed.
- **Chapter 3** presents multi-cloud principles driving M-CaMid conception, and the requirements M-CaMid must realise to meet these principles.
- **Chapter 4** presents how M-CaMid meets the multi-cloud principles' requirements to reach the thesis' goals. M-CaMid architecture is presented in detail, highlighting the management layer and the elasticity cross management.
- **Chapter 5** details experiments realised to assess M-CaMid. This chapter describes the experiments scenarios and environmental configurations to accomplish the middleware evaluation. Results are presented and discussed, evidencing M-CaMid efficiency.
- **Chapter 6** presents similar efforts on application and elasticity management in the cloud and multi-cloud environments.
- **Chapter 7** summarises the thesis, discussing its results, limitations and benefits, as well as proposing some future works not covered in the thesis.

2 Multi-Cloud Computing Foundations

Cloud computing is a paradigm for provisioning computing as a utility (Parkhill, 1966), shifting the on-premise infrastructure to the Internet, reducing costs and abstracting the underlying infrastructure complexity. Although in its childhood phase, cloud computing has grown up fast with many advances and technology diversity. This chapter presents the basic foundations of cloud and multi-cloud computing, multi-cloud application and multi-cloud middleware to comprehend this work better.

2.1 Cloud Computing

The Cloud Computing paradigm has many definitions because of its application diversity, varying according to the context of its use. Vaquero et al. (2009) compared about 20 definitions showing how the proposed definitions focus on specific characteristics of cloud computing, such as dynamic management, business model, virtualisation, and elasticity. They proposed a more comprehensive definition for Cloud Computing:

> Clouds are a large pool of easily usable and accessible virtualised resources, such as hardware, development platforms or services. These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing optimum resource utilisation. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the infrastructure Provider using customised Service-Level Agreements (SLAs).

Another summarised and widely known definition was published by the National Institute of Standards and Technology (NIST) (Mell and Grance, 2011):

> Cloud computing is a model for enabling 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.

Furthermore, NIST complemented this definition by listing the essential characteristics to describe cloud computing:

- ***On-demand self-service***: A cloud user can consume virtual resources as needed without human interaction, thanks to the service model for resource provision.

- ***Broad network access***: The virtual resources are available over the network, and the cloud consumer can access them through standard mechanisms that abstract aspects of heterogeneity.

- ***Resource pooling***: The computing resources are transparently shared among multiple consumers by using a multi-tenant model. The cloud consumer can use its resources dynamically, assigning and reassigning according to his/her demand.

- ***Rapid elasticity***: Capabilities can be scaled up and down and, in some cases, automatically. This characteristic allows the illusion of unlimited resources that can be adjusted dynamically according to their demand for cloud consumers.

- ***Measured service***: Cloud systems automatically monitor, control and optimise the resources to meter the cloud usage based on pay-per-use or charge-per-use model.

Cloud computing can be classified according to two models: delivery and deployment (Erl et al., 2013). The delivery model is how a cloud provider offers its cloud services to the consumer, while the deployment model defines how the infrastructure is provisioned.

2.1.1 Delivery Model

A cloud delivery model represents a specific combination of Information Technology (IT) resources offered by a cloud provider (Erl et al., 2013). Three basic models are widely established: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). Further models are considered according to the goals in the cloud computing adoption, e.g., Storage-as-a-Service, Security-as-a-Service, and Business-as-a-Service.

The IaaS model comprises of an infrastructure IT resources environment. It can include hardware, network, and OS. These resources are virtualised and packaged into bundles (e.g., VMs). IaaS provides services for managing pre-configured resources with a high level of control and administrative responsibility for their resources. The IaaS model allows more flexibility in the configuration and management of the resources. However, it charges the cloud consumer with more effort, complexity, and responsibility.

The PaaS model is a pre-defined environment for the development and execution of applications. The cloud provider provides a development environment for the cloud consumer. This environment is usually composed of programming languages, libraries and services supported by the providers. The underlying infrastructure is out of the cloud consumer control. The control is only over the deployed application and some configuration settings for application deployment. PaaS providers have their platforms hosted on IaaS environments.

In the SaaS model, software products are available as pubic resources. Cloud consumers can use applications running on a cloud infrastructure. Only application is accessible from the cloud consumer through a thin client interface, such as a web browser. The control over these resources (such as configuration, performance and scalability) is minimal compared to the other models since users access only the application. The access to software development platforms or infrastructures is restricted to their respective providers.

The three models are tightly related regarding the dependency among them. In other words, each model forms an abstraction layer in the cloud delivery architecture in a way that the SaaS layer depends on the PaaS, that in turn, depends on the IaaS layer, as shown in Figure 1.

2.1.2 Deployment Model

A deployment model refers to how the cloud infrastructure is provisioned, primarily characterised by ownership, size and access. The primary standard deployment models are public, community, private, and hybrid (Mell and Grance, 2011). In the public model, the cloud provider allows the public access to its environment. The cloud infrastructure is provisioned for open use by the general public. Since it is public, the public cloud usually has an extensive infrastructure to support many users. This infrastructure is managed and operated by a single owner.

The community model is similar to a public cloud, except for a cloud infrastructure provisioned for the exclusive use of a community of cloud consumers grouped in organisations with shared concerns. The community is in charge of managing and operating the cloud infrastructure.

Private cloud is provisioned and exclusively used by a single organisation, as well as, the management and operation is the responsibility of this organisation. In this model, the consumer is also the provider of the cloud. Usually, the private cloud has a small infrastructure. Finally, the hybrid cloud combines two or more different cloud models. For example, a private cloud with a small infrastructure can expand its resources by allocating resources in the public cloud.

Multiple cloud adoption introduced an architecture model composed of two or more cloud infrastructures. For example, two public cloud providers can be integrated into sharing common services with the cloud consumer. The hybrid model is a type of architecture model. This model differentiates from the Inter-Cloud one in the model combination. The hybrid combines different models, while Inter-Cloud combines infrastructures of the same model or not. More details about the Inter-Cloud model will be presented in Section 2.3.

2.2 Cloud Elasticity

Elasticity is a crucial characteristic of cloud computing that refers to the ability of rapidly provisioning resources dynamically to comply with the client's needs, on which the client can later release resources when they are not needed anymore (Mell and Grance, 2011). The system can add and remove resources adapting its infrastructure to meet the load demand variation in real-time (Al-Dhuraibi et al., 2018). On the other hand, the consumer can remove unnecessary resources, minimising the cost of resources utilisation.

Although many authors refer to scalability and elasticity as the same thing, there are differences between them. Scalability refers to the ability to support the system's growth, such as the increase of computing workload or the increase of components geographically distributed (Coulouris et al., 2013; Tanenbaum and Steen, 2007). However, scalability is only a static property which the system's demands grows up only, instead of considering the reduction of the system (Coutinho et al., 2014). If the demand shrinks, the resources become underused. The scalability is also time-independent. That is, it does not concern about how long the system takes to adequate the system's performance, neither how efficient the system supports its growth (Herbst et al., 2013). The efficiency is measured by the number of used resources and how fast the system adapts to meet the demand.

On the other hand, elasticity is the ability to expand and shrink the system's resources dynamically by adding or removing computational resources based on the *ad hoc* demand. The elasticity allows the efficient use of virtual resources through the rapid adaptation of the system to fit the resource amount to the system's needs (Herbst et al., 2013; Han et al., 2014; Naskos et al., 2016). The precision on allocating or deallocating resources aims to address the over-provisioning and under-provisioning. The precision and the time take in adaptation are two critical factors for characterising the elastic efficiency.

Finally, Al-Dhuraibi et al. (2018) reduce the elasticity understanding to the following equation:

$$elasticity = \underbrace{scalability + automation}_{auto-scaling} + optimisation. \tag{1}$$

In this case, elasticity is based on the scalability automatised through an auto-scaling process that optimises optimisation as best and fast as possible.

Both the scalability and elasticity properties have the goal to support the system's performance and availability. However, the dynamic nature of elasticity allows more precise management and cost reduction by renting virtual resources instead of investing in physical resources. In a multi-cloud scenario, the elasticity can be employed to improve availability and performance through deployment across cloud providers as discussed in the Section 1.2.

In practice, the elasticity extends the concept of scalability comprising dynamic properties, such as time and efficiency. To reach these properties, an elastic system must be fast and precise from the consumer perspective. In other words, scalability, autonomy, adaptability (Herbst et al., 2013), and SLA-awareness (Muñoz-Escoí and Bernabéu-Aubán, 2016) are desired properties of the auto-scaling process.

2.2.1 Taxonomy

There are several taxonomies to elasticity. Galante and de Bona (2012) classify elasticity based on four dimensions: the purpose of elasticity, the abstraction level of the elasticity's management, the way the elasticity is managed (manual and automatic), and the method for realising the elasticity.

2.2.1.1 Purpose of elasticity

The elasticity purpose is motivated by specific concerns, such as performance, availability and cost reduction. In short, purposes can be viewed from the cloud provider's perspective, whose focus on better resource usage (for cost reduction on hardware investments) and energy-saving, and provided QoS; and can be viewed from the cloud consumer perspective, whose focus in on the application's QoS, such as availability and performance. In the context of multi-cloud computing, both perspectives focus on cost savings and fault tolerance against natural disasters.

2.2.1.2 Abstraction level

The abstraction level refers to who has the elasticity control. It can be realised by the underlying infrastructure or the application/platform. In the first case, the IaaS provider provides the control, and in the last case, the platform or even the application controls the elasticity. When the application controls the elasticity, the controller is embedded in the application.

2.2.1.3 Elasticity management

Elasticity management refers to elasticity mechanisms that adapt the cloud environment to meet the workload demand. The elasticity management can be manual and automatic. Manually, the user is in charge of monitoring her/his virtual resources and applications, making decisions, and intervening in the environment. Automatically, the monitoring and control are performed by autonomous mechanisms that can be reactive or predictive. The reactive adaptation means the elasticity mechanism triggers actions based on monitoring information compared to a set of thresholds or rules. The predictive management implements mechanisms to forecast future demands and trigger adaptation actions in advance.

2.2.1.4 Realising elasticity

The method for realising cloud elasticity can be replication, redimensioning and migration. The replication task creates copies of a resource and instantiating it in a distinct computing node in a distributed system, improving availability and performance. In cloud computing, it realises horizontal elasticity. Redimentioning consists of reconfiguring a computing node characteristics by adding (scaling-up) or removing (scaling down) nodes' resources, such as CPU, memory and network bandwidth. Resource dimensioning accomplishes vertical elasticity. Migration in cloud computing can be considered vertical elasticity. It consists of moving resources between infrastructure environments. Its goal is to change underlying resources (e.g., node computing) to another with more or less computational resources. For example, a VM can be moved to a physical machine with more (scaling-up) or less (scaling down) computational performance.

Naskos et al. (2016) extend Galante's taxonomy, including more dimensions. They detail the scope dimension by adding the application type category under this scope. The policy dimension is split into decision making and elastic action. The decision making refers to the triggering way (reactive or predictive) and the mechanism (rule-based, control theory, model checking, statistical optimisation, and machine learning). The elasticity action dimension adds other methods in the taxonomy, making explicit the different abstraction levels of elasticity: infrastructure and application. The elasticity action can be classified as: *a*) horizontal scaling; *b*) vertical scaling; *c*) VM live migration; *d*) application reconfiguration; and *e*) application live migration. The first three types (at infrastructure abstraction level) refer to the replication of virtual resources, the re-size of a resource by adding more process (CPU cores) and memory capacities, and migration of resources without stopping its execution. The last two are actions for reconfiguring applications and migrating to another resource without stopping its execution.

Furthermore, Naskos et al. (2016) classify the cloud elasticity based on the scope. The scope refers to the range of cloud providers supported by elasticity. An elasticity tool can support only a single cloud provider or more than one. In the case of more than one cloud provider, the support may be simultaneous or not. The non-simultaneous multi-cloud elasticity is tightly related to the portability of the resources, while the simultaneous one is related to multi-cloud interoperability.

Finally, Al-Dhuraibi et al. (2018) complement the taxonomy, adding the dimension architecture that can be centralised or decentralised. Centralised elasticity management architecture has its control on a single entity. Most solutions adopt the centralised architecture Al-Dhuraibi et al. (2018). In decentralised architecture, many entities realise the elasticity control through application managers or middleware. They are in charge of managing elasticity for different cloud platforms and system components.

2.3 Multi-Cloud Computing

Over time, cloud computing has evolved to a phase where single clouds are not enough, and the need for cloud integration has become apparent. The adoption of many clouds has emerged to solve distribution challenges such as availability, scalability, performance, and security. This section discusses the paradigm of many clouds adoption, existing concepts and their taxonomy.

2.3.1 Definitions and Concepts

Since its beginning, the academy and industry have had few consensuses about definitions of clouds integration and its terminological ambiguities. Many initiatives have been coined terms and definitions based on technical issues. Inter-Cloud (Bernstein et al., 2009), Cloud-of-Clouds (Bessani et al., 2011), Cross-Cloud (Celesti et al., 2010b), Sky Computing (Keahey et al., 2009) and Multi-Cloud (Ferrer et al., 2012) are examples of clouds integration concepts. Each one has its proper definitions and characteristics.

Keahey et al. (2009) propose the *Sky Computing* in which dynamically provisioned distributed domains are built over several clouds combining the ability to trust remote sites with a trusted networking environment. Sky Computing assumes that there is the compatibility of Virtual Image (VI), standard Application Programming Interface (API) for service and trusted networking environments between cloud providers. It must provide an end-user environment that represents a uniform abstraction of the resources (such as a virtual cluster or a virtual site) independent from any particular cloud provider and can be instantiated dynamically.

The term *Inter-Cloud* was coined by Bernstein et al. (2009), who proposed a set of inter-cloud protocols to connect resources provided by different cloud providers, allowing portability and interoperability among them. Aoyama and Sakai (2011) also presents a concept of *Inter-Cloud Computing*, defining it as single cloud systems inter-connected via broadband networks that can mutually request resources from their peers or provide virtual resource capacity to them.

Celesti et al. (2010b) refer to *Cross-Cloud* as a federation of clouds where clouds can cooperate accomplishing trust contexts and providing new business opportunities such as cost-effective assets optimisation, power saving, and on-demand resources provisioning. This initiative introduces an abstract layer to support developing a set of high-level management components on top of conventional cloud computing systems to reach a higher level of sustainability among the participating cloud systems. This proposal exempts cloud providers to agree with standards and protocols.

The Global Inter-Cloud Technology Forum (GICTF) formally defines Inter-Cloud Computing (Global Inter-Cloud Technology Forum., 2010) as:

> A cloud model that, to guarantee service quality, such as the performance and availability of each service, allows on-demand reassignment of resources and transfer of workload through an inter-working of cloud systems of different cloud providers based on coordination of each consumers requirements for service quality with each provider SLA and use of standard interfaces.

As discussed in Section 2.3.3, cloud federation and multi-cloud computing are specialisations in the taxonomy. In this work, multi-cloud computing is adopted because it meets the requirements of our proposal.

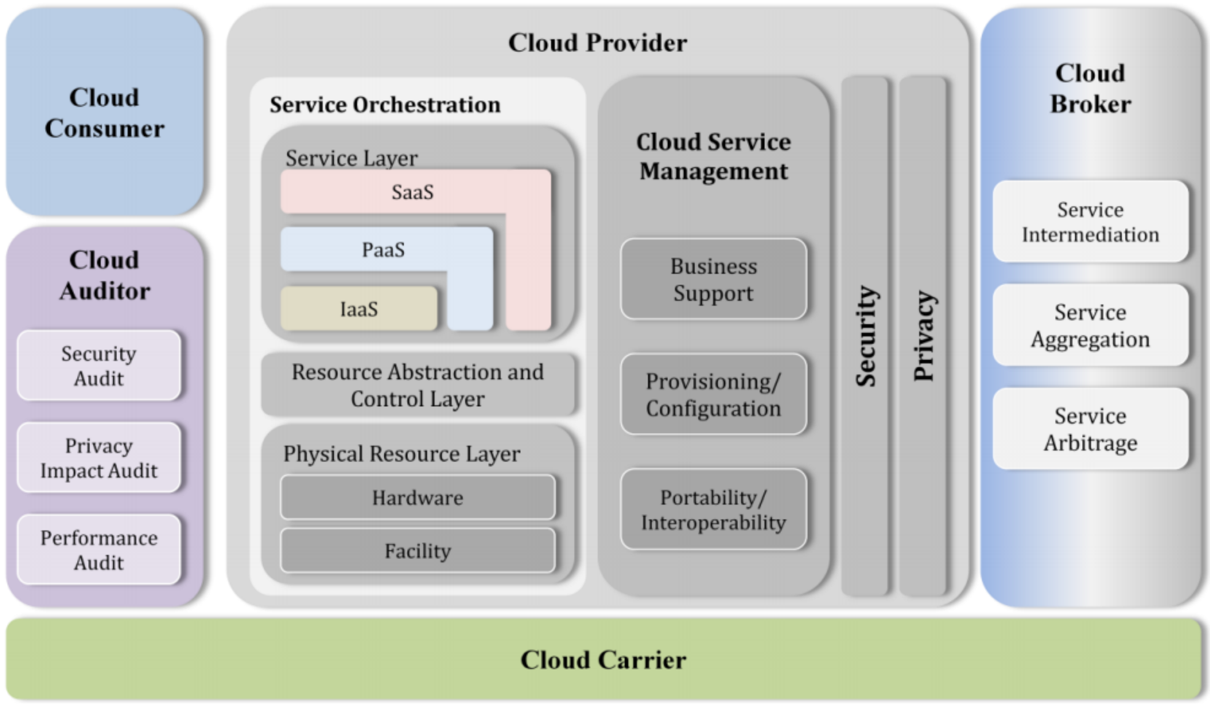
Despite several definitions of many clouds integration, it is possible to identify a set of common characteristics. Firstly, it extends the properties of a single cloud provider for improving the QoS of their resources, services and applications, such as availability, reliability, fault tolerance, and performance, by dynamically expanding their provisions. Secondly, a single cloud can expand its infrastructure by requesting resources from other clouds. Thirdly, cloud providers can also deliver specific services in cooperation with other clouds to provide a complete service.

On the other hand, cloud consumers can explore different aspects of cloud providers to offer better services and applications. Finally, cost reduction is possible for both the cloud client and cloud provider since they can make choices based on the cost-benefit. In all cases, existing solutions promise flexibility and diversity in support of the needs of cloud providers and cloud clients.

2.3.2 Basic Elements

The conceptual reference model of cloud architecture proposed by the NIST (Figure 2) (Liu et al., 2011) includes elements related to multiple clouds usage by identifying its main actors, their activities and functions. Beyond *Cloud consumer* and *Cloud provider* concepts, the model specifies the *Cloud Auditor*, *Cloud Broker* and *Cloud Carrier*.

Figure 2 – The Conceptual Reference Model proposed by NIST.



Source: (Liu et al., 2011).

2.3.2.1 Cloud Auditor

Cloud Auditor is a third-party element that can conduct an independent evaluation of cloud services, information system services, performance and security of the cloud implementation. In the multiple cloud scenario, it can provide certification based on cloud service verification of criteria such as security, performance, and SLA. Although its relevance, Cloud Auditor is out of the scope of this proposal.

2.3.2.2 Cloud Broker

Cloud Broker is in charge of managing the use, performance and distribution of cloud services and negotiating relationships between cloud providers and cloud consumers, allowing intermediation, aggregation and arbitrage of services. *Cloud Broker* can improve the interoperability by access and identity management. It can also support the management of distributed applications on multiple clouds by providing integration among cloud providers.

Some organisations (Edmonds and Nyrén, 2011; Bankston et al., 2012; Palma and Spatzier, 2013) propose cloud standards as an approach to realise cloud interoperability instead of the use of Cloud Brokers. However, even though interoperability standards are defined and broadly adopted, mechanisms still need to realise the standards and manage the interoperability between cloud providers.

In most cases, however, cloud brokers act on behalf of the cloud consumer and cloud provider dealing with infrastructure services and cloud consumer's applications. Cloud Broker can manage the interoperability horizontally at application (interoperability between application components) or infrastructure (cloud providers inter-operation) levels and vertically, supporting the relationships between application and infrastructure levels.

The Cloud Broker's functions can be categorised into provisioning and management. Provisioning functions are related to the selection of cloud providers:

- *Searching* resources that meet cloud consumer requirements. As a result, the function returns a set of cloud candidates that fit well with the consumer's needs;
- *Negotiation* among the cloud resources found by the Broker, the negotiation selects the best fit based on the consumer requirements;
- *Deploying* virtual resources on heterogeneous cloud providers, through a unified interface, allowing automatic deployments, migration of applications and cloud services; and
- *Accessing* to other clouds requires the Broker mechanisms to authenticate and authorise different cloud providers using a universal identity.

Management functions are related to the coordination of infrastructure resources and application lifecycle and distribution management. Management functions play two primary roles:

- *Monitoring* gathers information from the environment for identifying events. Events refer to changes in the environment. Monitors are in charge of analysing information and notifying *Controllers* about changes that can affect environment functioning.
- *control* execute operations based on monitoring information to adjust the environment according to consumer's requirements.

Cloud Broker functions are usually made available through services or libraries (Grozev and Buyya, 2014). The functions can be accomplished by services designed by the own provider, federation or a third-party service provider. The developer must specify an SLA or a set of rules that trigger actions whenever rules' premises are satisfied. For example, a VM or application overload can trigger actions for scaling up resources by elasticity. Libraries are collections of implementations designed to abstract the programmers from differences of management APIs and provide control over the provisioning of resources of cloud providers. Libraries only focus on providing a single interface of cloud services. The developer must programme solutions that use the libraries.

2.3.2.3 Cloud Carrier

Cloud Carrier is in charge of the connectivity and transport of data between Cloud Consumer and Cloud Providers. While cloud computing focuses on the data centre, the *Cloud Carrier* addresses the network connecting data centres and cloud users. In general, the Cloud Carrier may be required to provide dedicated and encrypted connections. Carrier clouds encompass data centres at different network tiers and extensive area networks that connect multiple data centres and cloud users. Links between data centres are used, for instance, for fail-over, overflow, backup, and geographic diversity.

2.3.3 Taxonomy

As discussed in Section 2.3.1 there is little consensus in the terms to define and classify multiple cloud usage. Because of the application diversity, several terms are applied to multiple clouds, such as multi-cloud, inter-cloud, cross-cloud, and cloud-of-clouds. For disambiguation reasons, this work takes inter-cloud as the general term to refer to multiple cloud usage. Federation and multi-cloud are classifications of inter-cloud computing.

The inter-cloud models define the way the cloud providers and cloud consumers *a*) deliver their resources and *b*) delegate the management control. Cloud providers deliver resources for general purposes to support the massive diversity of consumers' requirements. It motivates the usage of more than one cloud that can be in two ways. Subsequent usage refers to the migration of VM or application from one cloud to another motivated by economic reasons, legislation and political reasons, or better services. Simultaneous usage of multiple clouds can improve QoS requirements of distributed applications and support for a wide range of distributed applications. The entities involved in the simultaneous usage of cloud providers require mechanisms to deal with cloud interoperability.

Concerning the management, inter-cloud computing can be *provider-centric* or *client-centric*. Provider-centric depicts a scenario where the cloud provider deals with the interoperability with other ones. Different cloud providers agree with the adoption of standards for their common services constituting cloud federations. Furthermore, they are responsible for extending their functionalities to other clouds by requesting resources exempting the consumer's concern in infrastructure management. Although the benefits, cloud providers do not intend to commoditise their services. Even adopting service standards, they will try to offer differentiated services.

Since most cloud providers have few interests in "put all together", the client-centric model adopts third-party solutions to integrate cloud services. The cloud provider is unaware of clouds' integration. The cloud consumer is in charge of integration among clouds and resources' management. On the one hand, the cloud consumer gets more flexibility to define its infrastructure according to the application needs. On the other hand, this model charges the consumer for infrastructure management.

Third-party brokers aim to provide facilities of cloud integration for cloud consumers. In this sense, cloud brokers provide support for different purposes based on the consumer's needs, such as searching and selecting cloud providers, cloud accessibility, resource management, and application lifecycle management.

Three primary surveys (Grozev and Buyya, 2014; Toosi et al., 2014; Petcu, 2014) propose taxonomies for multiple cloud usage. The authors take into account different criteria to define categories and classify projects. Although divergences and overlapping in terms and categorisations, there is a consensus on two primary models: cloud federation and multi-cloud computing. Recently, Bouzerzour et al. (2020) extend the taxonomy proposed by Toosi et al. (2014) adding new approaches in client-centric and provider-centric categories.

2.3.3.1 Cloud Federation

Cloud Federation model allows resource sharing through federation regulations (Toosi et al., 2014). Cloud providers collectively agree upon standards of operation. The term may be used when describing the inter-operation of two distinct, formally disconnected communications networks that may have different internal structures (Serrano et al., 2011). Federated clouds aim at supporting their users by providing a single interface on which they can transparently handle different cloud providers Kertesz et al. (2013). The role of cloud federation goes beyond interface adaptation. It includes federation services that act as mediators between users and providers (Carlini et al., 2012).

A federation of clouds requires a formal agreement among cloud providers based on trust to collaborate by sharing their resources. Cloud interoperability requires cloud providers to adopt and implement standard interfaces, protocols, formats and architectural components. Cloud members can request resources from other members to make up their resource limitations. On the other hand, the members can make available underused resources to other members. This model is focused on cloud provider interest depicting a provider-centric scenario of interoperability.

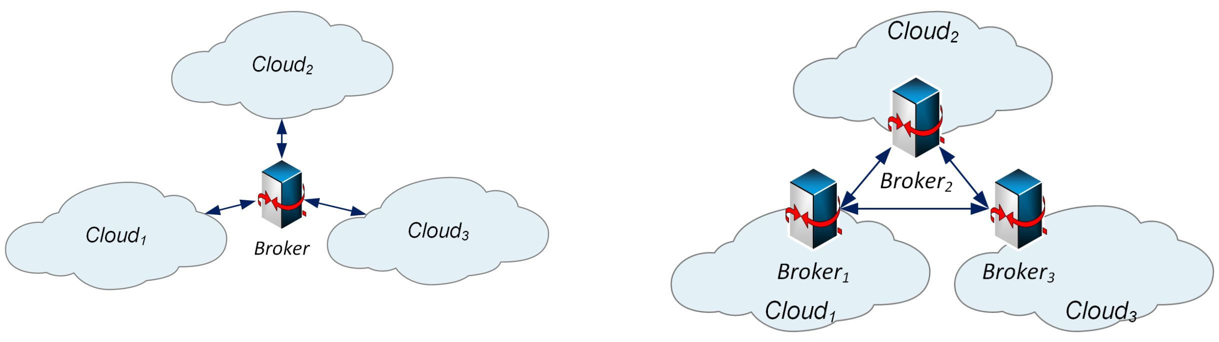
The primary motivation for adopting cloud federation is the ability to expand by elasticity the resource infrastructure meeting QoS requirements such as scalability, performance, availability and fault tolerance. Since cloud members agree on a universal standard for interoperability, cloud consumers are unaware of the underlying infrastructure where their virtual resources are running. For example, a cloud provider can ask for a new VM to other cloud providers in the federation to answer a client request. The client request is transparently forwarded to the other cloud. The federation limits the flexibility and control over cloud resources since the federation can deliver only standard services.

On the other hand, no matter what the federation offers as essential services, many cloud providers offer specific services to aggregate value and differentiate themselves from the other providers. The collaboration among federated clouds usually occurs in two ways (Villegas et al., 2012): vertical or horizontal. Vertical federation or delegation means collaboration among cloud service layers. For example, overloaded SaaS applications can increase their capacity by requesting more resources to underlying layers. In this case, the layers must agree on standard interfaces. Horizontal federation means the collaboration among cloud members at the same service layer. In this case, the federated services are available at the same deployment level, usually IaaS level. Horizontal collaboration is the majority of existing federations (Toosi et al., 2014; Assis and Bittencourt, 2016).

Architecture Cloud federations can be classified based on their architecture. The architecture is defined according to how cloud members manage their collaboration, centralised or decentralised. In a centralised architecture, a central entity performs and facilitates the cloud members collaboration. A cloud broker instance realises every interaction among clouds in charge of connecting both sides of the interaction. Figure 3a shows the relationship among cloud members in a centralised architecture.

Figure 3 – Architectures of cloud federations.

(a) Centralised architecture. **(b)** Decentralised architecture.



Source: author (2021).

Decentralised architectures do have not a central entity to manage the communication among cloud entities and manage distributed resources. All elements in a decentralised architecture have similar roles. The Cloud Federation can have more than one Cloud Broker distributed in the federation. This approach can enhance the performance of the federation management and supports the availability and reliability of the federation services. However, decentralised management is usually more complicated than centralised one. A fully distributed management (Peer-to-Peer (P2P) architecture) is presented in Figure 3b where each cloud member has a cloud broker instance in charge of dealing directly with other cloud members.

2.3.3.2 Multi-Cloud

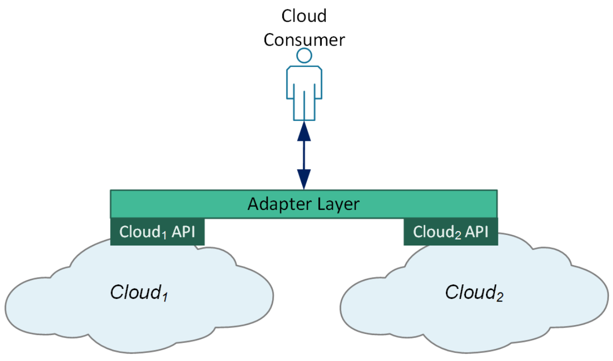
Multi-Cloud is a client-centric model for using multiple and independent clouds. This approach is substantially independent of the cloud provider while the cloud consumer manages the collaboration among them. Unlike cloud federation, in the multiple clouds, cloud provider integration is not spontaneous.

The non-spontaneity on integration by large cloud providers motivates the multi-cloud adoption by cloud consumers. Cloud providers design their services for general purpose needs. Applications must adapt themselves to meet their needs. Although cloud federation offers different services, it is still unaware of what applications need because it usually focuses on IaaS services and not on what is running on top of VMs.

From the cloud consumer perspective, multi-cloud gives more flexibility to the application lifecycle management, such as QoS guarantees. This approach allows more refined tuning of application components aiming the management based on the application needs. However, the development of such applications increases on complexity whose responsibility for managing the collaboration between a cloud application and IaaS service layer is the application.

Architecture Multi-Cloud management functions may be accomplished by an adapter layer responsible for the collaboration and interoperability of cloud applications and cloud providers. The Adapter layer can provide functions as a service or library. In the first case, the functions are implemented by services hosted in-house or in the cloud which developers can specify the application requirements by SLA or defining a set of rules. In the second case, an API is used by the application to interact directly with IaaS providers. APIs facilitates the use of multiple clouds in a uniform way. However, the developer must implement how the multi-cloud management uses the API. Services can use libraries to implement their brokerage functions. Multi-Cloud solutions based on services usually adopt libraries to provide portability and interoperability to cloud consumers. Figure 4 shows the basic architecture of the multi-cloud model. The Adapter Layer interacts with a specific cloud provider's API and provides either uniform services or a uniform API to the cloud consumer.

Figure 4 – Basic architecture of the multi-cloud model.



Source: author (2021).

2.4 Multi-Cloud Applications

Cloud providers deliver resource services for a general purpose instead of meeting specific application needs. On the other hand, classical applications are not designed to take advantage of the cloud benefits. Although cloud computing brings new possibilities for distributed applications, it also brings new challenges up.

Beyond the relationship between cloud consumers and cloud providers, it is necessary to manage distributed application components. The multi-cloud application refers to one with a set of distributed components into a multi-cloud environment (Cuadrado et al., 2014).

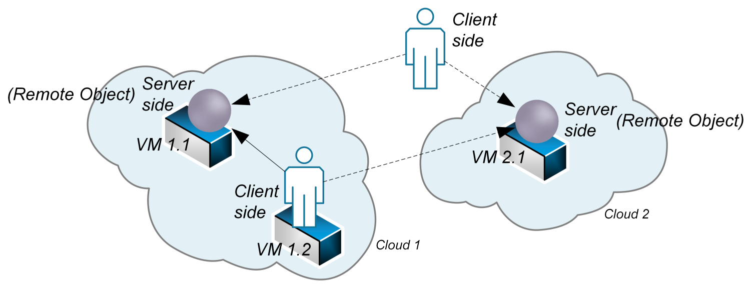
The multi-cloud application brings many advantages. Applications can be managed based on specific aspects defined by the cloud consumer instead of general aspects defined at the infrastructure level by the IaaS provider. The approach allows more flexibility in choosing cloud providers and fine-grained control over the application. However, the onus for managing infrastructure resources and applications' lifecycle lies with the cloud consumer. The application developer must be aware of details of underlying infrastructures to support meeting desirable distribution properties. Furthermore, as a multi-cloud application, the dynamic nature of multi-cloud computing adds more complexity to application management.

2.4.1 Scopes of Multi-Cloud Applications

The environment of distributed applications in the cloud scenario can range from a single VM to a set of resources distributed throughout different clouds. In the simplest case, the entire application runs on top of a VM and distribution occur at the application process level. The application can also be distributed over more than one VM, characterising the distribution at the infrastructure level. In this scenario, the scope of the application is restricted to a single cloud domain. Lastly, the components of an application can be hosted on VMs instances running on different cloud domains, characterising the multi-cloud domain.

Figure 5 shows a distributed application running in different scopes. The client/server application is distributed on two clouds. In $Cloud_1$, $VM_{1.1}$ and $VM_{1.2}$ host client and server sides of the application, respectively. The interactions between client and server sides in this scenario are contained in the cloud domain. $VM_{2.1}$ in $Cloud_2$ hosts another server component (remote object) of the application. Interactions between the client on $VM_{1.2}$ and server hosted on $VM_{2.1}$ take place in a multi-cloud domain.

Figure 5 – Scopes of a distributed object application in the multi-cloud.



Source: author (2021).

2.4.2 Multi-Cloud Application Requirements

Both business and technological contexts can classify application requirements. The first one states strategy's needs about non-technical issues, such as the economy and logistics. For example, the cost of the application deployment may have constraints. The last one refers to aspects of the application functioning (e.g., communication) and non-functional characteristics, such as availability, performance and scalability. Furthermore, the dynamic nature of cloud computing introduces new requirements:

Multi-Cloud elasticity is an essential requirement for assuring applications can be distributed through many cloud providers, leveraging the benefits of multi-cloud computing. The multi-cloud elasticity improves the application support for meeting primary distributed systems' requirements, such as **scalability** and **availability** on the presence of performance degradation and even faults. Furthermore, the elasticity management must be **automatic** to provide transparency to application development, exempting (at least reducing) the human intervention and hiding management complexities on scaling in/out virtual resources in a multi-cloud environment. The automation also improves the efficiency of the application on timely responding to undesirable behaviours.

Distribution transparency hides the complexities of application distribution, such as access, location, scalability and failure. Developers can focus on the business logic, exempting themselves from being aware of distribution management. Since the elasticity is tightly related to distribution aspects, it is also distribution transparency.

Cross interoperability among mechanisms at different layers and clouds can improve collaboration to leverage cloud benefits. Vertical and horizontal interoperability must be covered to realise integration between application and underlying IaaS resources, and among resources at different cloud domains (**multi-cloud interoperability**), respectively. Vertical interoperability allows the integration of management mechanisms at different levels of abstraction (infrastructure and application) for more precise monitoring and control of the application and its infrastructure. Fine-Grained management means more precise management and fewer cost and complexity since it allows the **rational resource usage**, one of the characteristics that differentiate elasticity from scalability. Furthermore, the interoperability must be provided in a seamless way of supporting distribution transparency. The application also must be **cloud agnostic** to accomplish this requirement.

2.4.3 Multi-Cloud Application Management

Rapid changes in the cloud environment feature the dynamic nature of cloud computing. Management mechanisms should assure adequate response that does not affect the application performance and availability. However, cloud providers only manage virtual resources without knowing what is running on top of the virtual resources and how it is running. It is not manageable by the provider's mechanisms. On the other hand, the consumer's lack of control over infrastructure resources limits the decision-making power.

Mechanisms must consider monitoring information from different abstraction levels and cloud domains to perform complete and accurate management, combining them to give a big picture of application health. The management can make decisions about the application controlling based on this monitoring information. The multi-cloud application management benefits from cloud elasticity to meet the most application requirements.

2.5 Distribution Middleware

The role of middleware systems in cloud computing environments has ranged from an interoperability enabler (Bouzerzour et al., 2020) to a data distribution manager (Ömer Köksal and Tekinerdogan, 2017) and cloud-based Internet of Things (IoT) distribution platforms (Ngu et al., 2017). However, similarly to off-cloud middleware systems, it should be noted that the middleware is used to deal with distribution aspects, such as transparency among remote applications, hiding complexities of communication, distribution and integration (de Morais et al., 2013; de Morais and Rosa, 2017).

Middleware platforms facilitate the development of distributed applications by implementing distribution transparencies (e.g., access, location and failure) and providing distributed services (e.g., naming, security, and concurrency) to developers (Bernstein, 1996; Schantz and Schmidt, 2002; Völter et al., 2004). In practice, the middleware hides from application developers the complexity of underlying communication, distribution and concurrency mechanisms, offering general services that support distributed execution of applications. It is an additional software layer that sits between the network services provided by Operating System (OS) and the application layer hosting the client and server application.

Middleware systems are in charge of realising distribution tasks for *i*) operational interaction between application components by message exchange; *ii*) remote interaction between components hosted in different address spaces. The remote style defines how the interaction takes place. For example, remote objects interact with each other by method invocation; *iii*) distribution transparency for application's standpoint. The application is unaware of the type of interactions among application components that can be local or remote interactions; and *iv*) technological independence, supporting the integration of different technologies.

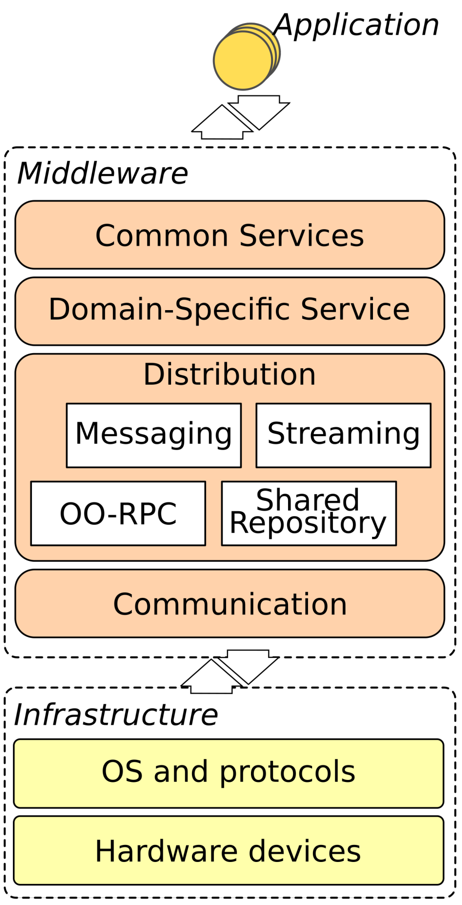
2.5.1 DOC Architecture

The basic middleware architecture can be organised according to its function: communication and distribution. The first one is in charge of encapsulating and enhancing native OS communication and concurrency mechanisms. The distribution layer defines higher-level distribution mechanisms that can be based on the remoting patterns (Völter et al., 2005). Figure 6 shows the basic architecture and the style variations. It is based on the layers of Distributed Object Computing (DOC) middleware architecture proposed by Schantz and Schmidt (2002). In Figure 6, the communication layer interacts directly with the infrastructure, while the distribution layer provides transparency of communication services to distributed applications based on remote style. The application interacts with the middleware through APIs of the distribution layer's components.

In addition to Figure 6, two more layers are placed on top of the distribution layer: common services and domain-specific services. The first one consists of generic services (e.g., security, naming, transaction, concurrency), and the last one performs domain-specific tasks such as telecommunication, e-commerce, e-learn, and cloud services. The interaction between the application layer and the middleware can be direct with the distribution layer or through the services' layer.

Despite the great diversity of middleware models and products, the architecture presented by Schantz and Schmidt (2002) is widely adopted for structuring distributed object computing middleware. Furthermore, the concepts of the object model ideally reflect the characteristics of distributed systems (Puder et al., 2006).

Figure 6 – Middleware basic architecture adapted from (Schantz and Schmidt, 2002).



Source: (Schantz and Schmidt, 2002).

2.5.2 Multi-Cloud Middleware Goals and Challenges

A multi-cloud middleware system should provide support to a distributed application on meeting its requirements (see Section 2.4.2) by leveraging the benefits of multi-cloud computing. The multi-cloud elasticity is a crucial characteristic of cloud computing that can improve the middleware on the support of multi-cloud applications. Therefore, the multi-cloud middleware goals are:

1. Provide multi-cloud elasticity for distributed applications in a transparent way; and
2. Provide multi-cloud interoperability so that distributed applications take advantage of the elasticity in different cloud providers.

However, these goals demand complex management and multi-cloud interoperability. The development of a multi-cloud middleware requires solving these demands, hiding them from the developer. Therefore, some principles must be followed in the design process, such as adaptability and multi-cloud awareness.

2.6 Concluding Remarks

This chapter presented the foundations of multi-cloud computing, encompassing cloud and inter-cloud computing, elasticity, multi-cloud application, and multi-cloud middleware. Some definitions and terms were discussed, and their essential elements and taxonomies were also presented.

The elasticity is presented as a critical characteristic of cloud computing that refers to rapidly provisioning resources dynamically to comply with the client's needs. The management of elasticity can take place at the infrastructure level or application/platform level. Mechanisms of migration and replications can realise its management.

Inter-cloud computing is classified into cloud federation and multi-cloud. The first one is computing or network providers that agree upon standards of operation in a coordinated fashion that aims to support their users by providing a single interface on which they can transparently handle different cloud providers. The last one is a client-centric model for using multiple and independent clouds. This approach is substantially independent of the cloud provider while the cloud consumer manages the collaboration among them.

Finally, the middleware definition and its underlying architecture composed of two primary layers (communication and distribution) were presented. The DOC layer stack was shown as a widely accepted architecture. Two more layers are added to the basic architecture: common services and domain-specific services.

3 MULTI-CLOUD PRINCIPLES

Designing a middleware solution for multi-cloud computing must follow some basic principles. These principles are fundamental statements placed at the beginning of any inference related to multi-cloud design. For multi-cloud middleware design, some fundamental propositions are *decentralisation*, *openness* and *interoperability*, *dynamic behaviour*, and *adaptability*. These multi-cloud principles drive the middleware development, point out design requirements to be addressed, such as multi-cloud awareness, multi-cloud elasticity, and cross-management.

This chapter initially presents some primary multi-cloud middleware goals. Next, it describes the proposed principles by focusing on *what* is the principle, *why* it is necessary to adopt the principle, *where* it could be placed within the middleware structure and *how* it can be achieved.

3.1 MULTI-CLOUD MIDDLEWARE GOALS

Multi-cloud middleware platforms target to provide distribution transparencies for cloud providers and multi-cloud applications, hiding underlying complexities such as communication, distribution and integration (de Morais et al., 2013). It can benefit from multi-cloud computing to support multi-cloud distributed applications in meeting their requirements (see Section 2.4.2) and support interoperation and collaboration between cloud providers. Therefore, the multi-cloud middleware goals are:

1. To manage, in a seamless way, distributed applications components running in multiple clouds;
2. To manage infrastructure resources, improving their rational usage and abstracting management aspects;
3. To provide multi-cloud elasticity for distributed applications in a transparent way; and
4. To support the transparent integration of cloud providers so that distributed applications take advantage of virtual resources in different cloud providers.

However, these goals demand to assume some principles for designing multi-cloud middleware systems. These principles are related to cloud and multi-cloud aspects and their influence on distributed application requirements.

3.2 DECENTRALISATION (***P1***)

The decentralisation principle refers to the component autonomy in a distributed architecture, where each component is independent and responsible for its management. System's architecture follows two models (Tanenbaum and Steen, 2007; Coulouris et al., 2013): centralised and decentralised. Centralised architecture is composed of entities with distinct roles and hierarchical relationships among them. Entities in decentralised architectures have similar roles, establishing a flat relationship among them.

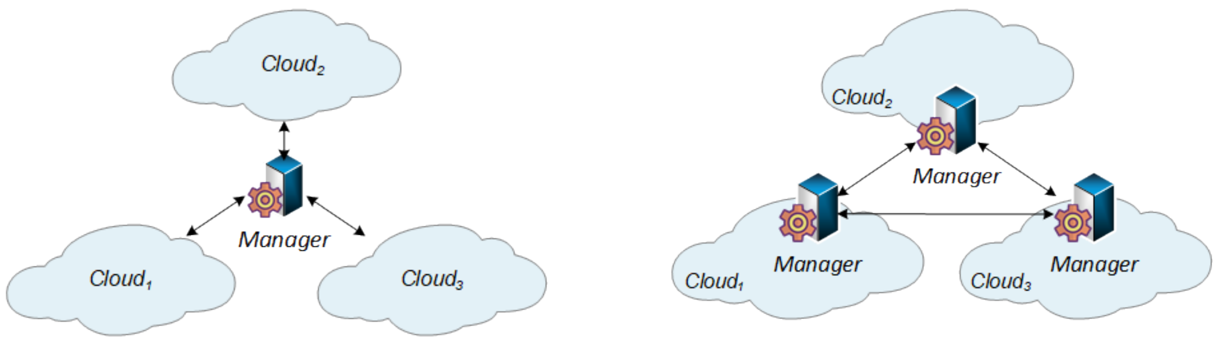
Single cloud architecture follows the centralised model since its resource management tools are grouped into a specific entity – IaaS manager. IaaS manager (server) is the element controlling the resources' life cycle to deliver cloud services to the customer (client). On the other hand, a multiple cloud environment comprises autonomous cloud providers, characterising a decentralised architecture. Cloud providers have similar behaviour from an external cloud view, delivering the same essential cloud services autonomously. Cloud providers can connect their domains, composing a decentralised architecture, such as federated clouds (Section 2.3.3.1).

It is crucial to understand the relationship between cloud architectural elements in the cloud and multi-cloud scenarios. A multi-cloud middleware project can drive its design considering architectural aspects, such as communication models and management architecture. For example, communication models can be chosen according to the system's architecture, such as oriented messaging communication for multi-cloud communication and management strategies.

Management strategies for multiple clouds can vary according to the multi-cloud architectural aspects. As discussed in Section 2.3, management control can be realised by the cloud provider or delegated to a third-party entity. However, the cloud provider management is limited to infrastructure resources because it is unaware of what is running on top of VMs. In both cases, the management can centralise or decentralise its control (Figure 7). Centralised control (Figure 7a) concentrates all management mechanisms into a single entity. While decentralised one (Figure 7b) distributes mechanisms instances through many entities.

Figure 7 – Centralised and decentralised management for multi-cloud environments.

(a) Centralised architecture. **(b)** Decentralised architecture.



Source: author (2021).

3.3 Openness and Interoperability (***P2***)

Openness is the capability of a system to offer services according to standardised rules (Tanenbaum and Steen, 2007). Then, users can enjoy cloud services without ordering them to the service provider. According to (Czaja, 2018), openness has the following characteristics:

- *Independence of desired system behaviour* through intermediate languages and their compilers or interpreters;
- *Unified and standardised communication mechanisms*, such as communication protocols; and
- *Widely offered standard interfaces of access to common resources*.

Nowadays, many cloud platforms are available, offering different kinds of services through application interfaces. Cloud providers are open platforms because they provide public specifications and documentation of their primary services – public interfaces as openness property enablers. With public interfaces, cloud computing has evolved naturally towards a horizontal integration model where cloud environments interoperate.

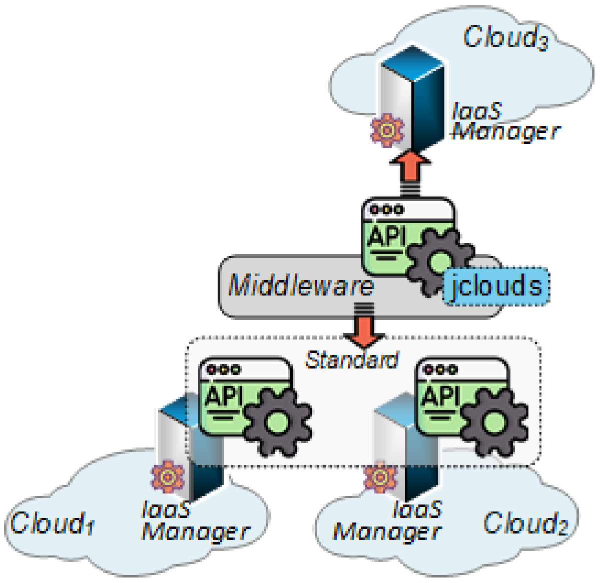
Openness capabilities allow independent systems to be interoperable. The decentralised nature of multi-cloud architecture is tightly related to interoperability since cloud platforms are independents and heterogeneous. Interoperability is the ability of computing systems to coexist and work together (Tanenbaum and Steen, 2007) by information and service exchange, which states that a computing system is open when it can be extended or reimplemented in many ways (Coulouris et al., 2013). In multi-cloud computing, interoperability allows applications to have distributed components deployed on many cloud environments to communicate, integrate and migrate regardless of their location and platform heterogeneity (Bouzerzour et al., 2020).

Before cloud computing, interoperability was a challenge for system heterogeneity and evolution (DeNardis, 2012). As discussed in Chapter 1, there are many standardisation proposals, although the industry showed little interest to adopt them. Furthermore, adopting a standard does not mean addressing the interoperability problem because there is a *pletora* of divergent standards (Nogueira et al., 2016) and few consensus about standard adoption. In addition, even if clouds adopt standards for basic services, commercial providers are always looking for specialised services to add value to their product.

The interoperability charge can be placed in a client-centric perspective, where middleware design can adopt standards defined by wide used cloud platforms, such as (OpenStack (RackSpace, 2010) and OpenNebula (Moreno-Vozmediano et al., 2012)), or mass-adopted libraries that encapsulate heterogeneous cloud services into interfaces – API, such as JClouds® (Apache, 2011).

Figure 8 shows two multi-cloud interoperability scenarios. In the first one, $Cloud_1$ and $Cloud_2$ implement their services following some standard. Thus, a middleware system can access its services through public APIs. In the second one, $Cloud_3$ implements its services following a proprietary pattern. In this case, the middleware can implement its own library to access the cloud services through pattern-proprietary APIs or use libraries that implement basic cloud services access. In this example, the middleware accesses $Cloud_3$ services through the JClouds® library.

Figure 8 – Interoperability scenarios for multi-cloud integration through middleware.



Source: author (2021).

3.4 Dynamic Behaviour (*P3*)

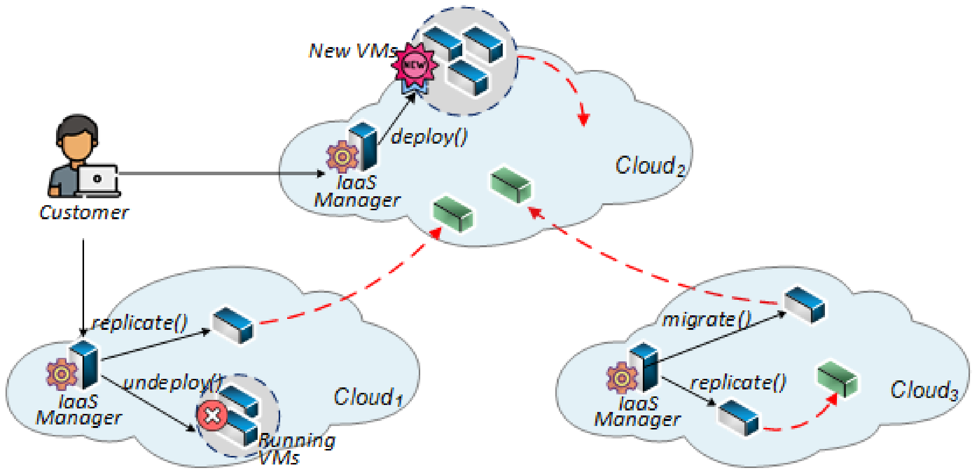
Dynamic behaviour of a system refers to the state pattern changing over time (Coad, 1992). Multi-Cloud computing inherits the dynamic nature of cloud computing. Constantly, virtual resources are deployed and undeployed or resized in a single cloud environment, changing its state.

Cloud computing faces changes and events in the environment according to customer demands. As customer's needs change, the cloud environment state also changes. For example, in a single cloud provider with global access, new customers can request deployments of VMs, while others are requesting VMs shutdown. Any management design for cloud and multi-cloud computing views the system as a dynamic environment. Therefore, a middleware architecture must design efficient management mechanisms to deal with the dynamic nature of multi-cloud computing. Addressing this principle implies on middleware's ability to support system scalability and adaptability.

Figure 9 depicts the dynamic behaviour of a multi-cloud environment. Virtual resources are constantly in change, both inside a single cloud or a multi-cloud context. Inside $Cloud_1$, running VMs are turned off by the IaaS manager. On the other hand, in $Cloud_2$, new VMs are deployed.

In the case of $Cloud_3$ IaaS, the manager replicates a VM. At multi-cloud scope, cloud providers can interact through operation such as migration and replication, characterising the multi-cloud dynamic behaviour: $Cloud_1$ replicates a VM in $Cloud_2$ and $Cloud_3$ migrates its VM to $Cloud_2$. It is worth mentioning that the interoperability principle is allowing the cloud cooperation.

Figure 9 – Events constantly changing the multi-cloud environment state.



Source: author (2021).

Multi-cloud dynamic behaviour can be observed through clouds' monitoring information. However, no cloud provider monitors a multi-cloud environment thanks to its decentralised architectural nature. Each cloud provider is in charge of monitoring the behaviour of its virtual resources. Observing the multi-cloud behaviour is possible through monitoring from multi-cloud participating cloud providers.

In a single cloud, the IaaS manager is in charge of monitoring the environmental state at the infrastructure level, assuring performance and scalability. In a multiple cloud environment, participant cloud providers share behavioural information between them (cloud federation). However, most multi-cloud scenarios are composed of non-voluntary participating clouds that are unaware of other cloud providers – third-party solutions aggregate information from multi-cloud participating cloud providers and provide a holistic view of multi-cloud behaviour.

3.5 ADAPTABILITY (***P4***)

The adaptability principle refers to the capability of a system to adapt itself (autonomy) to keep its components working as close as possible to parameters required by the customer. A system is autonomous when it can configure itself according to high-level policies Kephart and Chess (2003).

According to the decentralisation principle, a cloud provider is an autonomous participant of a multi-cloud environment. Cloud providers can configure the customer's resource system to meet a pre-defined Service-Level Objective (SLO) with minimal human intervention, thanks to the cloud's adaptation ability through elasticity service (Pahl et al., 2018). The elasticity allows clouds to adapt their infrastructure to meet client demands by increasing or shrinking resources. The dynamic nature of cloud and multi-cloud computing establishes an adaptable behaviour of their related elements, such as architectures, technologies and tools.

The adaptability principle intersects with decentralisation and dynamic behaviour principles. Both intersections characterise the importance to assume adaptability as the dynamic behaviour agent of each autonomous (decentralised control) participating cloud in the multi-cloud environment.

Based on the conceptual model defined by de Lemos et al. (2013), an autonomic multi-cloud system comprises:

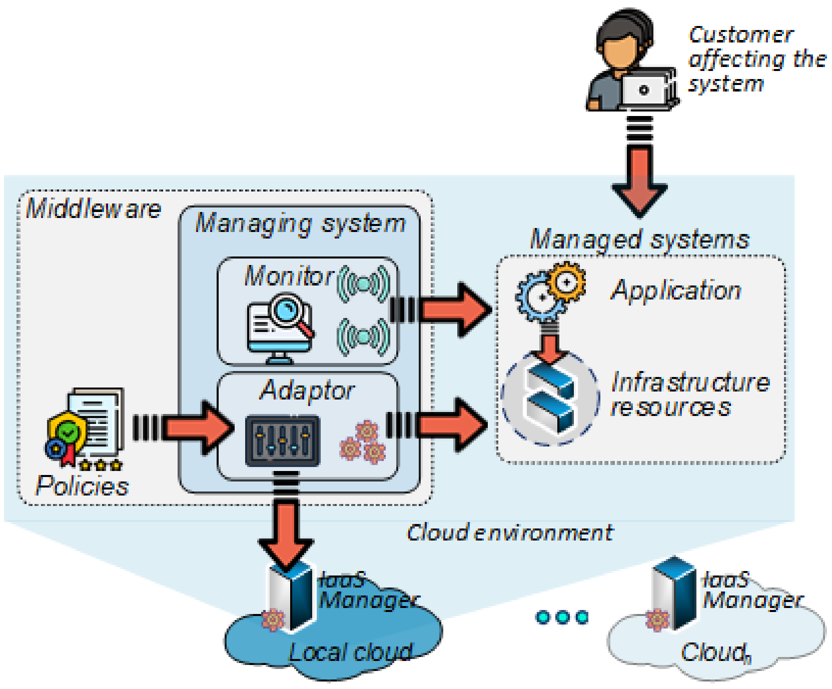
- The environment, representing the external world where a multi-cloud application is running (multi-cloud environment and Internet, from where clients interact with the multi-cloud application);
- The managed system, comprising cloud resources (infrastructure level) and multi-cloud application components (application level). The managed system is the object of adaptation. It is equipped with sensors for gathering environmental data;
- Policies, defining adaptation goals and rules to meet customer SLO;
- The managing system, responsible for realising the adaptation based on the system policies. It comprises mechanisms for monitoring the multi-cloud environment and adapting the managed systems based on analysis of environmental monitoring data.

Figure 10 shows a multi-cloud autonomous environment, where the middleware is in charge of implementing the *managing system*. The environment comprises cloud providers (*Local cloud*, $Cloud_1$, $Cloud_2$, ..., $Cloud_n$) and the Internet where customers access the application. The middleware monitors and controls infrastructure resources and applications through managing systems' mechanisms (*Monitor* and *Adaptor*). Monitoring mechanisms gather data from the environment to support the adaptation process that is performed by the *Adaptor*, based on rules and goals defined in *policies*. The *Adaptor* can request elastic services to the current cloud (*Local cloud*) or to an external cloud ($Cloud_n$)

A multi-cloud managing system (e.g., multi-cloud middleware) can take advantage of cloud elasticity services to support adaptability for multi-cloud systems. The middleware extends elasticity to the multi-cloud context.

From a multi-cloud perspective, the autonomic behaviour of a single cloud can be extended to the multi-cloud environment through voluntary cloud collaboration (federated clouds) or middleware as a third-party system, intermediating communication and supporting interoperability in case of cloud providers are not volunteers. The managing control of adaptation can be centralised or decentralised, as presented in Figure 7. Furthermore, the middleware can apply adaptation to infrastructure and application levels (de Lemos et al., 2013).

Figure 10 – Elements of a multi-cloud autonomic system with *managing system* as part of middleware.



Source: author (2021).

3.6 Concluding Remarks

This chapter presented basic principles to drive a multi-cloud middleware design: decentralisation, interoperability, dynamic behaviour and adaptability. These principles drive the middleware design according to multi-cloud middleware goals. Each principle was discussed in the context of its definition, justifying its adoption and relating it to the cloud and multi-cloud computing context. Besides, this chapter discusses how principles take place in cloud and multi-cloud computing and where the middleware can realise them. Finally, Some principles' relationships were presented throughout the chapter.

Multi-cloud principles are fundamental for multi-cloud design. Following them, the next chapter presents the requirements and architecture of Multi-Cloud aware Middleware (M-CaMid), an object-oriented middleware specially designed for supporting the development and management of distributed applications in multi-cloud environments.

4 Multi-Cloud Aware Middleware – M-CaMid

This chapter presents the M-CaMid architectural design, detailing its components operations and relationships, and M-CaMid management mechanisms to meet multi-cloud requirements. M-CaMid design follows multi-cloud principles discussed in the previous Chapter 3. Section 4.1 introduces M-CaMid goals and requirements, and the rest of this chapter presents details of its design.

4.1 M-CaMid Definition, Goals and Requirements

M-CaMid (Multi-Cloud aware Middleware) is an object-oriented middleware specially designed for supporting the development and management of distributed applications for multi-cloud environments by abstracting underlying distribution complexities inherent to multi-cloud infrastructures. M-CaMid implements mechanisms to support *(i)* communication between the application's components distributed over many cloud domains; and *(ii)* execution and management of distributed applications and their infrastructure resources with minimal human intervention by leveraging the cloud elasticity benefit automatically and transparently.

Aiming afore mentioned M-CaMid goals, the architectural design follows multi-cloud essential principles presented in Chapter 3: (***P1***) decentralisation, (***P2***) openness and interoperability, (***P3***) dynamic behaviour, and (***P4***) adaptability. However, following these principles implies meeting some requirements, such as hybrid architecture, multi-cloud awareness, cross management, and elasticity. The following sections discuss requirements related to each principle.

4.1.1 Decentralisation (*P1*) Requirements

Why The multi-cloud architecture comprises independents and heterogeneous cloud providers, where the control is usually decentralised. On the other hand, a single cloud provider is characterised by its centralised architecture, where its controls are centred in the IaaS manager. In both scenarios, the cloud provider architectural style follows the layered model for delivering cloud services (IaaS, PaaS and SaaS).

Requirements M-CaMid must be multi-cloud aware (***R1***) for controlling both distributed applications and infrastructure, targeting transparency of technology and distribution to hide underlying complexities from developers.

Besides, M-CaMid must tackle the hybrid multi-cloud architecture (***R2***) to deal with both centralised and decentralised architectures of a multi-cloud environment.

Finally, M-CaMid must be aware of different abstraction levels at cloud and multi-cloud scopes. In other words, M-CaMid must combine different abstraction levels (infrastructure and application) at cloud and multi-cloud scopes.

4.1.2 Openness and Interoperability (*P2*) Requirements

Why Although multi-cloud evolution, many cloud platform solutions offer heterogeneous services demanding standard pattern specifications (lock-in problem). A distributed application needs to be cloud-agnostic in a multi-cloud environment to leverage essential cloud services from different cloud applications regardless of cloud technology. Application's parts in various cloud platforms also need to interoperate. For example, a distributed application with components scattered in many clouds platforms must deal with distinct infrastructure organisations, e.g. networking.

A mitigating solution is to place the interoperability in charge of the middleware. A middleware design can adopt standards defined by comprehensive used cloud platforms, such as (OpenStack (RackSpace, 2010) and OpenNebula (Moreno-Vozmediano et al., 2012)), or popular libraries that encapsulate heterogeneous cloud services into interfaces – API, such as JClouds (Apache, 2011).

Requirements Openness and interoperability are tightly related to the decentralisation principle, demanding the multi-cloud aware requirement (***R1***) since M-CaMid must adopt available cloud standards to deal with different cloud abstraction levels.

4.1.3 Dynamic Behaviour (*P3*) Requirements

Why Dynamic behaviour is a mandatory principle for multi-cloud distributed application management since its state is constantly in change. Multi-cloud behaviour is affected by events in the abstraction levels and the cloud and multi-cloud scopes. The dynamic nature of multi-cloud computing requires timely middleware response for realising cloud tasks, such as deploying and undeploying VMs in many clouds. Moreover, there is a trade-off between interoperability and performance. As more the system is interoperable, the worse is its performance. Multi-cloud management requires not only a rapid response but also precision.

Requirements Middleware design must include automatic mechanisms to support environmental changes without human intervention. Its management mechanisms must monitor the system's behaviour at the abstraction levels to decide where acting (infrastructure or application levels) and the best configuration to assure rapid and precise responses to keep the system's stability. The management of abstraction levels (vertical management) and across cloud and multi-cloud environments (horizontal management) are fundamental for assuring rapid middleware response to multi-cloud behaviour changes – cross management (***R3***).

In a multi-cloud environment, the application elasticity depends on the application's ability to deal with different cloud platforms. It requires a middleware to extend elasticity service to the multi-cloud scope. Multi-cloud elasticity (***R4***) allows distributed applications to use elasticity among cloud platforms.

4.1.4 Adaptability (*P4*) Requirements

Why Adaptability is the reason the multi-cloud environment is dynamic, with elasticity being the protagonist of this relation. Management entities must control the adaptation of multi-cloud elements through the elasticity services of clouds (multi-cloud elasticity – ***R4***). The elasticity approach also must consider two management granularity levels: fine-grained (at the application level) and coarse-grained elasticity (at the infrastructure level).

Requirement M-CaMid must support the cross management requirement (***R3***) by designing vertical (cloud layers) and horizontal (cloud scopes) elasticity management. M-CaMid must act as an adaptation agent to support the elasticity management in a multi-cloud environment, allowing automaticity and transparency to elasticity and dealing with different architectural scenarios: centralised into the cloud domain and decentralised in the multi-cloud domain.

4.2 Multi-Cloud Awareness Middleware

M-CaMid must be aware of the multi-cloud environment, dealing with different cloud technologies. Designing multi-cloud aware middleware shifts the interoperability concerns to the middleware level, allowing distributed applications to be agnostic and interoperable.

Being aware means that M-CaMid knows the different scopes of multi-cloud architecture. M-CaMid's approach defines a multi-cloud model characterised by three domains to represent the scopes of a multi-cloud environment: *Node*, *Cloud*, and *Multi-Cloud* domains. The role of M-CaMid is defined according to the domain that it is related.

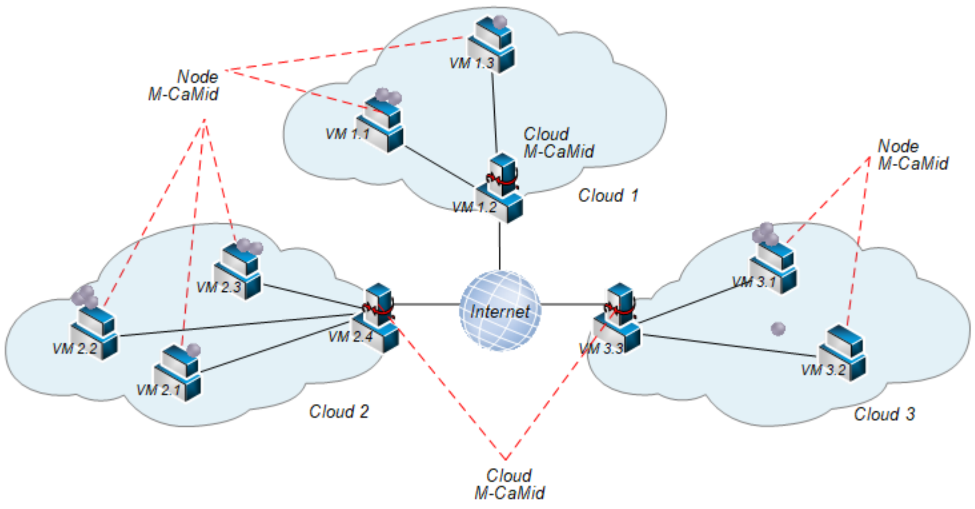
4.2.1 Node Domain

Node domain is the canonical environment, namely *Node*, that comprises the VM provided by a cloud provider, the software stack (OS and the middleware itself), and application's components. For each *node domain*, an instance of the M-CaMid (*Node M-CaMid*) is in charge of managing (monitoring and control) both infrastructure and the application lifecycle, and providing communication at application and middleware level. The *node domain* communicates with its respective *cloud domain* (through the *Node M-CaMid*) for notifying about events and exchanging monitoring information. At application level, it realises the communication among application's components. Figure 11 shows the relationship among *Node M-CaMid* and *Cloud M-CaMid*.

4.2.2 Cloud Domain

A set of *nodes*, on which their VMs are managed by a cloud provider, constitutes a *cloud domain*. The *cloud domain* is composed by the set of *nodes*, the IaaS cloud manager, and a M-CaMid instance. In this domain, the *nodes* can cooperate by sharing its resources under the management

Figure 11 – M-CaMid instances relationship.



Source: author (2021).

of the M-CaMid instance. The M-CaMid (*Cloud M-CaMid*) has the role of supporting the communication among the application's components and orchestrating all *nodes* as a whole, providing a uniform management view of the cloud environment.

4.2.3 Multi-Cloud Domain

Two or more cloud domains running components of the same multi-cloud application establish the *multi-cloud domain*. The whole environment is composed by *nodes* spread over *cloud domains*. The components running into these *nodes* define the multi-cloud application. In this scope, all cloud domains are reachable from each other. Since cloud domains (cloud providers) have independent infrastructures, interoperability occurs at the middleware level in the multi-cloud environment. M-CaMid is in charge of providing transparent interoperation among the cloud domains. Figure 11 shows the relationship among cloud domains through M-CaMid instances (*Cloud M-CaMid*). It can extend the resources of a multi-cloud application to another cloud domain by requesting new resources. Similarly, it can receive requests to lend resources. Each M-CaMid instance of *cloud domain* is in charge of receiving messages and redirecting the proper component endpoint into the cloud domain.

Table 1 summarises the roles of M-CaMid according to the domain.

4.3 Architecture

M-CaMid distributed architecture organises its operations according to the three domains: node, cloud and multi-cloud. The operations distribution allows a clear definition of the architectural components and their roles. Figure 12 shows the distribution of M-CaMid's operations. An operation acts over the multi-cloud environment according to its respective domain. For example,

Table 1 – Domains and M-CaMid functions.

Domain	Function	Description
Node	Monitoring	Monitors OS and Application components.
	Communication	Local management of sending and receiving messages.
	Analysing	Checks data from monitoring and notifies the cloud domain.
	Application management	Manages application components lifecycle: deploying and undeploying.
Cloud	Communication	Intermediates the communication among application components through forwarding pattern; Manages the components' communication on supporting of load balancing and distribution transparency.
	Monitoring	Gathers monitoring information from node domains to analyse the cloud environment.
	Naming	Registries location information of the the application, middleware, cloud and nodes.
	Adaptation	Adapts the cloud environment to meet the application requirements triggering replication tasks.
Multi-Cloud	Communication	Provides communication among cloud domains.
	Brokering	Requests and provides virtual resources to others cloud domains.

Source: author (2021).

monitoring operations (node monitoring, cloud monitoring, and multi-cloud monitoring) execute different tasks in their respective domain.

Because multi-cloud domains have different characteristics and behaviours, M-CaMid considers different architectures for its components in cloud and multi-cloud environments. The hybrid architecture is a requirement discussed in Section 4.1 since the multi-cloud domain can be composed of independent cloud domains. In contrast, a single cloud domain usually is based on a centralised architecture.

Figure 11 shows the relationship between *Cloud M-CaMid* and *Node M-CaMids* into a single cloud (cloud domain) and the relationship among *Cloud M-CaMid* linking many single clouds, composing the multi-cloud domain.

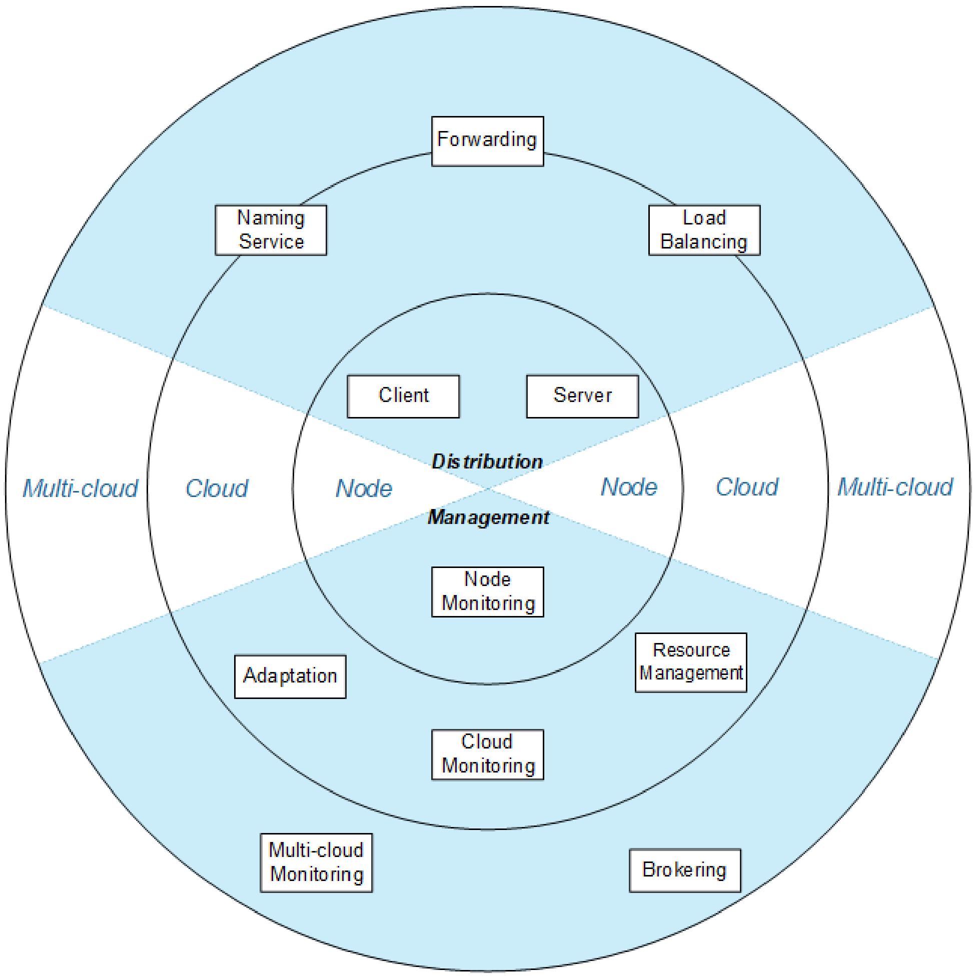
Cloud M-CaMid is the element that integrates a single cloud to a multi-cloud environment. In the scope of cloud domain, *Cloud M-CaMid* plays as a server while *Node M-CaMids* are the clients. *Cloud M-CaMid* manages all *Node M-CaMids* in the cloud domain. In the multi-cloud domain, cloud domains act equally. Each *Cloud M-CaMid* acts as a broker in a decentralised architecture, managing the cloud participation in the multi-cloud environment. *Cloud M-CaMid* is in charge of:

1. Redirecting application's requests to the proper component's endpoint;

2. Balancing the workload among component's replicas, if exists;

3. Monitoring and analysing the cloud environment by gathering information from *nodes* and

Figure 12 – M-CaMid operations arrangement over the multi-cloud domains.



Source: author (2021).

IaaS cloud manager;

4. Stepping in the environment as a response to undesirable events, such as node overload and too high response time of an application's component. As an example, an action to deal with overload may be the replication of an application's component or even the VM replication;

5. Rearranging components distribution among the *nodes* to balance the workload;

6. Interacting with the underlying infrastructure to request IaaS services; and

7. Providing inter-operation with other cloud domains through communication and distribution tasks, featuring the interoperability among cloud domains.

4.3.1 M-CaMid's Components

M-CaMid architecture is based on the well-known layers stack of DOC middleware proposed by Schantz and Schmidt (2002). Moreover, M-CaMid adds a fourth layer in the architecture to group the management components (de Morais et al., 2013). This approach is analogous to the cloud management layer proposed by Lenk et al. (2009), whose vertical layer covers all stack layers providing management services at different abstraction layers. Figure 13 shows the architecture of M-CaMid.

Transport encapsulates OS services for object communications. The transport layer's components enhance native OS communication and concurrency mechanisms and abstracts away peculiarities of individual OSs;

Distribution the Distribution layer provides APIs and components to automate and extend communication capabilities encapsulated by the Transport layer. This layer establishes high-level operations for remote object programming;

Common Services supports the distribution and management layer by defining higher-level services;

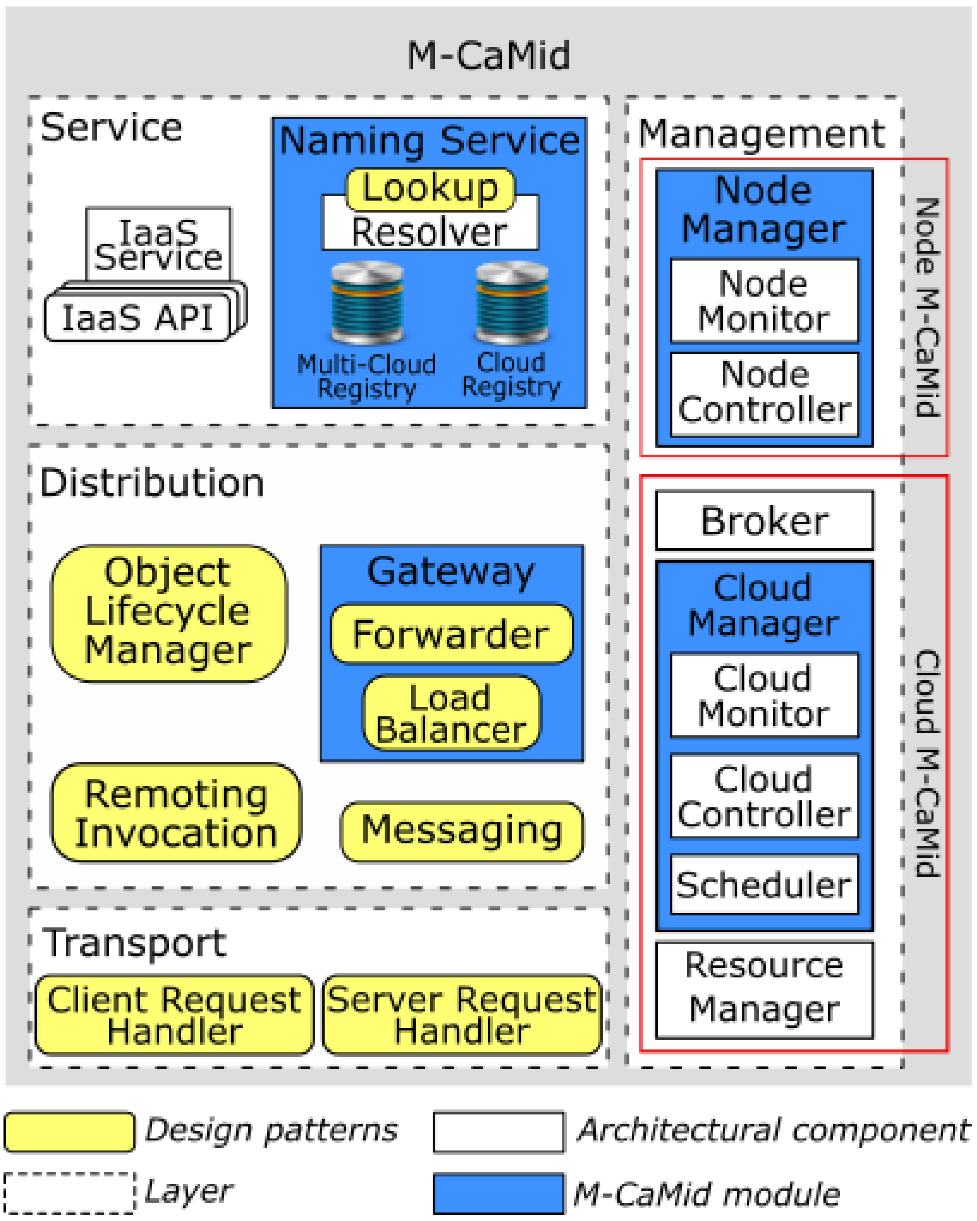
Management delivers a set of mechanisms for monitoring and controlling the application's objects and cloud resources distributed over cloud domains, providing a holistic management view.

4.3.2 Transport Layer

The transport layer works as a wrapper to the operating system. It consists of two remote architectural patterns (Völter et al., 2005), namely *Client Request Handler* and *Server Request Handler* that take responsibility for dealing with all aspects related to the communication inside the middleware, e.g., to establish and configure connections. *Client Request Handler* sends requests and receives replies from the network on the client-side, while *Server Request Handler* has a similar role on the server-side. The proposed Transport layer has been implemented on Transmission Control Protocol (TCP), and its API provided consists of two operations: send and receive.

Figure 14 presents the transport layer architecture in details composed of *Client Request Handler* and *Server Request Handler*. In the client-side, all requests coming from the component *Requestor* in the distribution layer are placed into a *sending* queue. The component *Connector* is responsible for managing the connections. To improve scalability, *Connectors* are managed by a *Thread pool*. For each *Requestor* invocation, a *Connector* is created to handle the communication between the client and server. The invocations are processed concurrently, improving the scalability of client-side. *Connector* creates a connection to the server-side for sending the request message. After the connection is established, *Connector* sends the message to the server-side. The connection is closed at the end of the transmission. *Connector* handles the connection by using the operating system's APIs for networking management.

Figure 13 – M-CaMid architectural elements.



Source: author (2021).

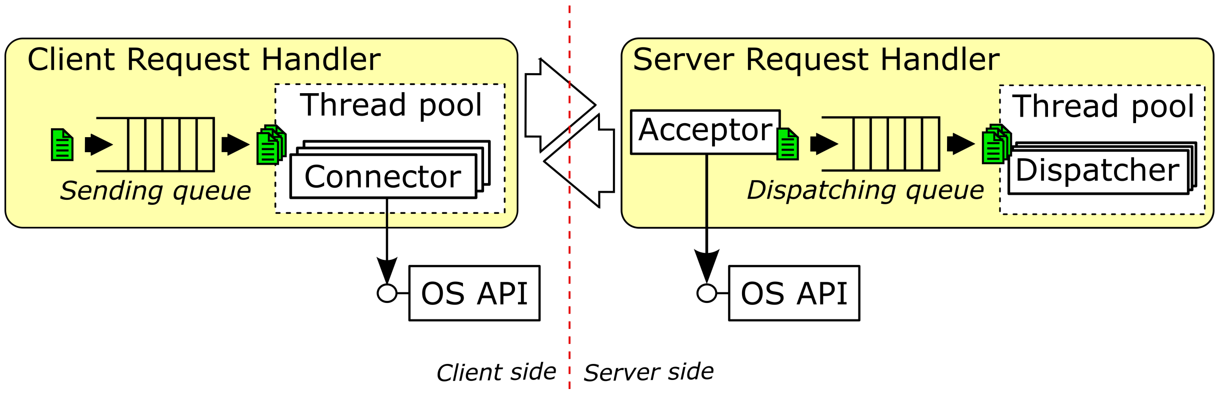
Server Request Handler is a pattern similar to *Client Request Handler*. Its role is to manage the communication on the application's server-side, hiding low-level networking complexity (*OS API*) from the upper layer. In other words, *Server Request Handler* is responsible for establishing connections with the client-side, receive invocations and dispatch them to the proper *Invoker* in the distribution layer. Connections are handled concurrently since the server-side complies with more than one client request. *Server Request Handler* deals with networking operations through *OS APIs*.

For each connection established, *Acceptor* receives an invocation message and lining up it in *Dispatching queue*. *Thread Pool* consumes messages from *Dispatching queue* and creates a *Dispatcher* thread instance for each message. Dispatcher redirects the message to the proper

Invoker. The connection is closed after the transaction ends.

The Transport layer is used for all M-CaMid's components that need to communicate regardless of their scope.

Figure 14 – Transport layer's components.



Source: author (2021).

4.3.3 Distribution Layer

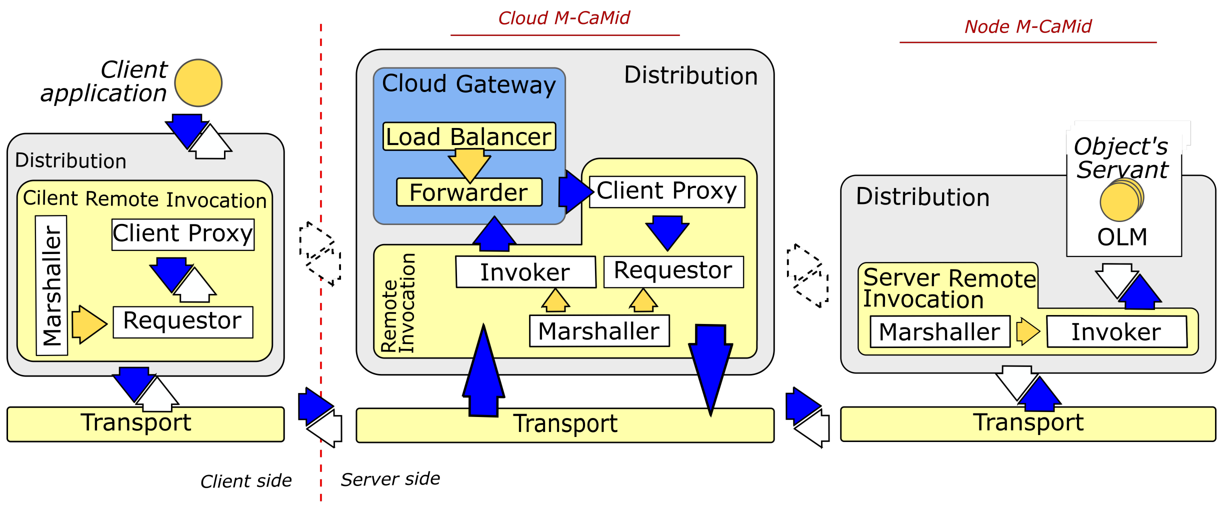
The distribution layer provides APIs and components to automate and extend the communication capabilities encapsulated by the transport layer. In practice, clients invoke local or remote objects without hard-coding dependencies (Schantz and Schmidt, 2002). Furthermore, the distribution layer realises the multi-cloud communication through the components in the *Gateway* (see Figure 13). Thus, the distribution layer plays two main roles: support the remote object invocation and multi-cloud distribution management. Architectural components of distribution layer differs according to the application side: M-CaMid client and server sides. M-CaMid client side is composed of *Client Remote Invocation* (*Client Proxy*, *Marshaler* and *Requestor*). In the server side, there are two M-CaMid instances: Cloud M-CaMid, composed of *Remote Invocation*, *Cloud Gateway* and *Messaging*, and Node M-CaMid, composed of *Server Remote Invocation* and *Object Lifecycle Manager*.

Remote Invocation comprises designed components designed following remoting patterns for communication (Völter et al., 2005): *Client Proxy*, *Marshaller*, *Requestor*, *Invoker*, and *Forwarder*. Remote Invocation is present in all instances, however its components vary according to each instance. Following, Remote Invocation is discussed in the three M-CaMid instances. Figure 15 shows the interaction between the three instances.

4.3.3.1 M-CaMid Client Side

Client Remote Invocation is in charge of receiving the client invocation through *Client Proxy*. This component represents a remote object on the client-side application, making the same interface of the invoked remote object available. The client application invokes methods of *Client*

Figure 15 – Remote invocation in the distribution layer architecture.



Source: author (2021).

Proxy, as the remote object itself. *Client Proxy* is in charge of translating the local invocation into parameters to *Requestor*. The client application is unaware of the object's localisation. *Client Proxy* carries the cloud domain's endpoint where its respective remote object is running. From the client's perspective, the remote object behaves as it is running locally. The client application gets *Client Proxy* by looking up the remote object in *Naming Service* (as described in Section 4.3.4).

Requestor takes the responsibility of sending requests to the server-side by using the transport layer. *Requestor* invokes operations to *Marshaller* to convert remote invocations into byte streams (serialisation) ready to be sent over the network.

4.3.3.2 Cloud M-CaMid

On the server-side, *Cloud M-CaMid* receives remote invocations and redirects them to the proper *Node M-CaMid*. *Cloud M-CaMid* Its Remote Invocation is composed of both *Client* and *Server Remote Invocation* components since its responsibility comprises receiving and forwarding remote invocations.

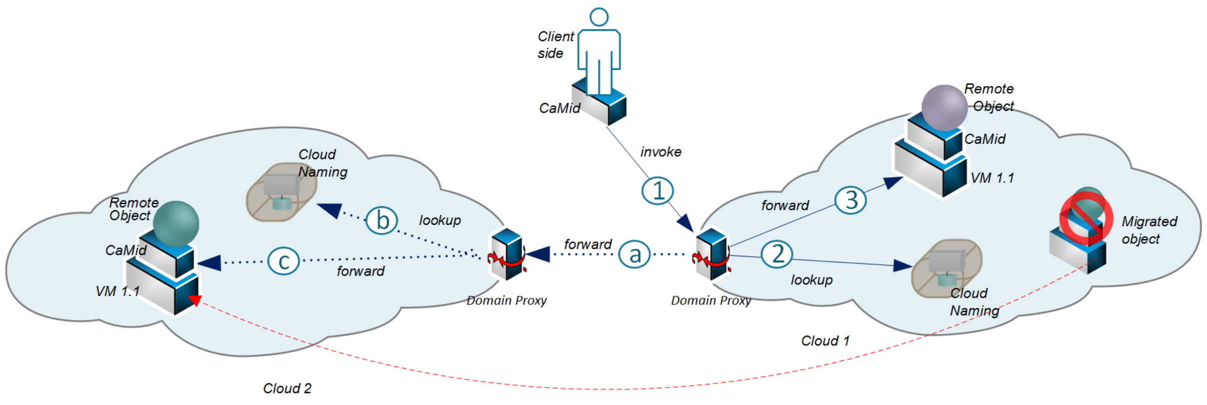
Invoker receives a byte stream from the transport layer and uses it to convert the received byte stream into parameters to build the method call. *Invoker* allows the implementation of remote objects to be independent of details of communication and management. In Cloud M-CaMid, *Cloud Gateway* is implemented as a remote object. It is loaded locally by Invoker that calls the remote method.

Cloud Gateway is composed of *Forwarder* and *Load Balancer*. These components manage the distribution of both cloud and multi-cloud domains. Goals of *Cloud Gateway* are: *(i)* Forward requests to proper remote object's endpoint wherever it is running on; *(ii)* Balance requests among object replicas; *(iii)* Hide details of cloud provider's configuration policies (e.g., network configurations); and *(iv)* Provide location transparency for remote objects.

Component *Forwarder* plays the role of mediator of the communication between a client application and remote objects. It forwards invocations to the proper remote object's endpoint inside or outside the cloud, supporting the location transparency. *Naming Service* provides location transparency to remote objects dealing with the dynamic nature of cloud computing and object visibility on both cloud and multi-cloud domains.

Figure 16 shows the component *Forwarder* in an remote invocation scenario. The component redirects incoming invocations to the proper remote object's endpoint. *Forwarder* ① receives the incoming message and ② gets the target remote object reference in *Naming Service*. Thus, *Forwarder* ③ redirects the invocation to the target location. Suppose the remote object has been recently replicated in another node located in the same cloud domain or another one. In that case, Load Balancer defines which endpoint *Forwarder* ⓐ redirects the invocation message. In the case of multi-cloud replication, *Forwarder* of the current cloud domain (cloud$_1$) redirects the invocation to the new cloud domain (cloud$_2$), which in turn redirects to the proper remote object endpoint.

Figure 16 – *Cloud Gateway* on the remote invocation.



Source: author (2021).

Forwarder implements the pattern Location Forwarder in the two strategies described before. *Forwarder* diverts invocations to another location without awareness of the client side's middleware, in case of replication in the cloud domain; or *Forwarder* notifies the client-side M-CaMid about the new object's location. In the multi-cloud environment, where the Internet is the communication channel, this last strategy is crucial since it avoids the message goes through the Internet twice each time a remote object is invoked. Although the client side's M-CaMid is aware of changes, it remains transparent to the client application layer. Before replicating, the object replica must register its new location on *Naming System*. Information about locations of objects' replicas is held by *Forwarder* and updated by management mechanisms of replication (management layer).

Note that all communication in a cloud domain goes through *Cloud Gateway* that encapsulates its respective nodes. It is worth noting that Cloud M-CaMid runs in a specific node in the cloud domain. Through this node, *Cloud M-CaMid* provides access to cloud and multi-cloud environments.

4.3.3.3 Node M-CaMid

In the same way, as in the Cloud M-CaMid, *Invoker* unmarshals a byte stream from the transport layer into parameters to build the method call. However, instead of requesting *Forwarder's* services, *Invoker* request to *Object Lifecycle Manager* (*OLM*) to create and initiate object servants locally to execute the requested method. *OLM* is in charge of managing remote objects' lifecycle, creating, initiating, destroying and registering objects' servants locally, ensuring that no unnecessary servants are loaded into the memory. *OLM* implements the *Pooling* pattern. This pattern introduces a pool of servants for each remote object to handle remote invocations. Servants are initialised with the state of the remote object they represent. After finalising execution, the servants remove the objects' state (Völter et al., 2005).

4.3.3.4 Messaging

The M-CaMid management components adopt messaging (Hohpe and Bobby, 2004) as a communication pattern to exchange management information. *Messaging* component implements the communication pattern. The messaging allows loosely coupled integration among the management components through asynchronous communication, as shown in Figure 17. *Node M-CaMid* sends monitoring information and alerts to *Cloud M-CaMid* as messages. *Node M-CaMids* writes information to a message channel, while *Cloud M-CaMid* reads ones from that channel.

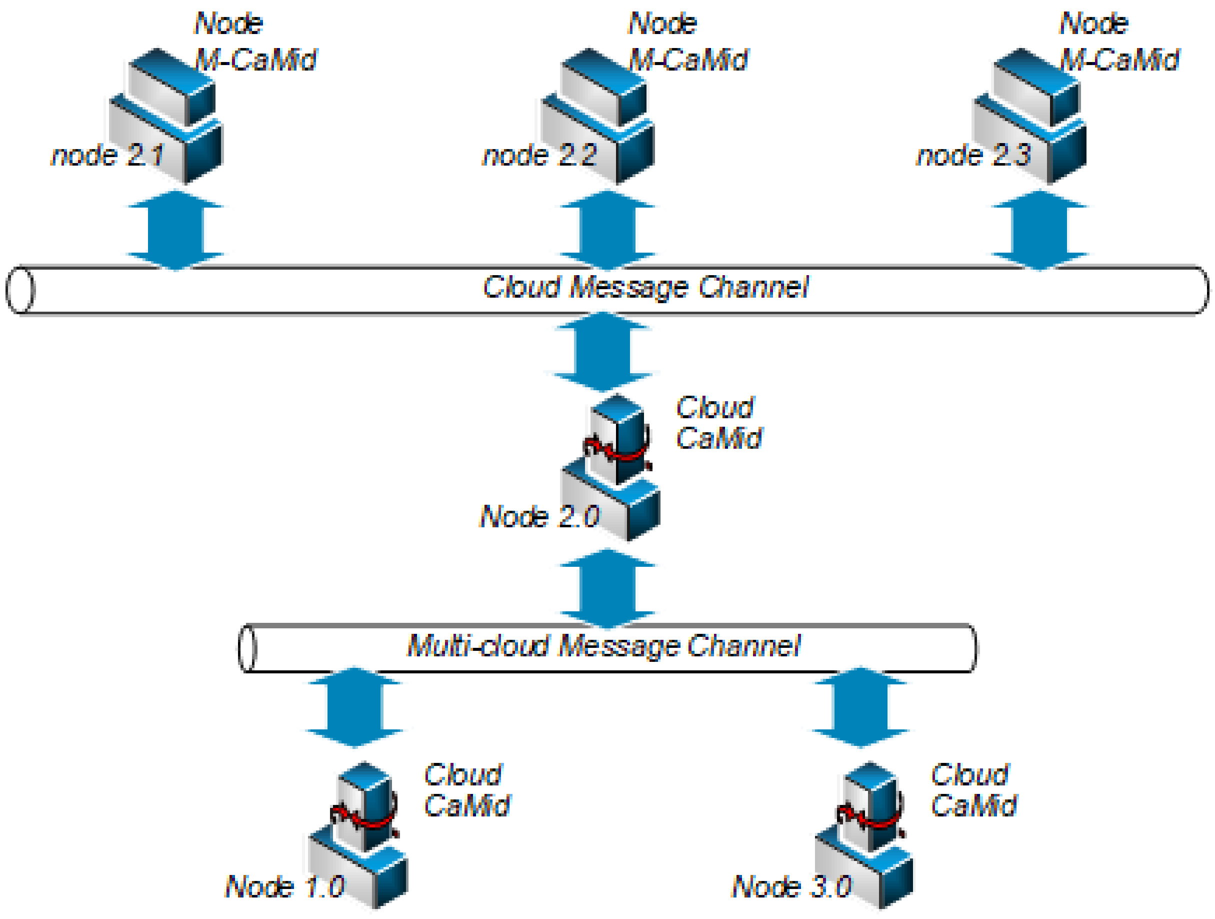
Cloud M-CaMid also uses messaging for requesting information to their respective *Node CaMid* instances and to others *Cloud M-CaMid* instances. Figure 17 shows the integration among the managers.

The messaging system uses two channels for managers integration. *Cloud Message Channel* allows the communication among *Node CaMid* instances and the respective *Cloud CaMid* and *Multi-Cloud Massage Channel* connects the all *Cloud CaMid* instances.

4.3.4 Service Layer

The two services M-CaMid deliveries are naming and IaaS services. These services act in the different application domains. For example, the naming service has components for the cloud domain and multi-cloud domain.

Figure 17 – M-CaMid messaging system.



Source: author (2021).

4.3.4.1 Naming Service

The Naming service provides an intuitive identification of resources and binds them to their respective names regardless of location. In our proposal, we consider three types of resources: the remote object is the canonical entity in the multi-cloud environment; a node is a VM resource provided by the cloud domain and has remote object running; cloud domain represents a cloud provider where remote objects are running. The node's resource location is defined by its local network address, usually an Internet Protocol (IP) address. Remote objects use the node address and a port number (where they are running and listening) to define this endpoint. A cloud domain has a public IP address for multi-cloud communication and a local IP address for communication with its respective nodes. The Naming Service locates both remote objects and cloud domains through their registered names. Even that IP address changes, it is possible to locate resources because Naming Service updates the services registres when changes occur. It allows communication through the Internet and communication inside the cloud domain.

However, the high dynamic cloud behaviour requires naming mechanisms to deal efficiently with sudden changes in resource location and support the multi-cloud interoperability. Furthermore, those mechanisms must provide an intuitive understanding of resource location to application developers and location transparency to the application layer.

The multi-cloud environment is composed of independent cloud domains with their respective namespaces. Thus, locating resources from other cloud domains becomes non-trivial. The Naming service must consider the many namespaces and provide resource visibility to multi-cloud applications overcoming the heterogeneity. The dynamic nature of multi-cloud environments requires scalable and extensible mechanisms for resolving names rapidly and transparently.

M-CaMid considers two basic naming spaces to support these requirements: the cloud domain and the multi-cloud domain. This approach allows the distribution of naming services with more flexibility and scalability as well as autonomic management. Components *Cloud Registry*, *Multi-Cloud Registry*, and *Resolver* compose the naming service in a cloud domain. *Cloud Registry* keeps information of nodes and remote objects of the respective cloud domain, allowing visibility into the cloud domain. In contrast, *Multi-Cloud registry* stores information of other cloud domains (name, endpoint and provider) for multi-cloud communication. Both Registry implementations are data structures based on a hash table, where the *key* is the resource's id, and the *value* is a reference to the resource information.

Resolver is a set of operations to manage the naming in the registries. Operations of *binding* and *looking up* are realised by this component. A new resource in a cloud domain should be registered by M-CaMid. *Resolver* binds new resources and updates information as soon as necessary. For example, unused resources are unbounded.

The registration process is realised by the *binding* operation for new remote objects and nodes. New remote objects running in a node are registered in *Cloud Registry* by *Node Manager*. The *binding* operation registries the remote object, binding its id to a set of information (name, endpoint, node id and a *Client Proxy* object. New nodes in the cloud domain are registered by *Cloud Manager* carrying the node identification, endpoint, and configuration information, such as the number of CPU cores, memory size and operating system.

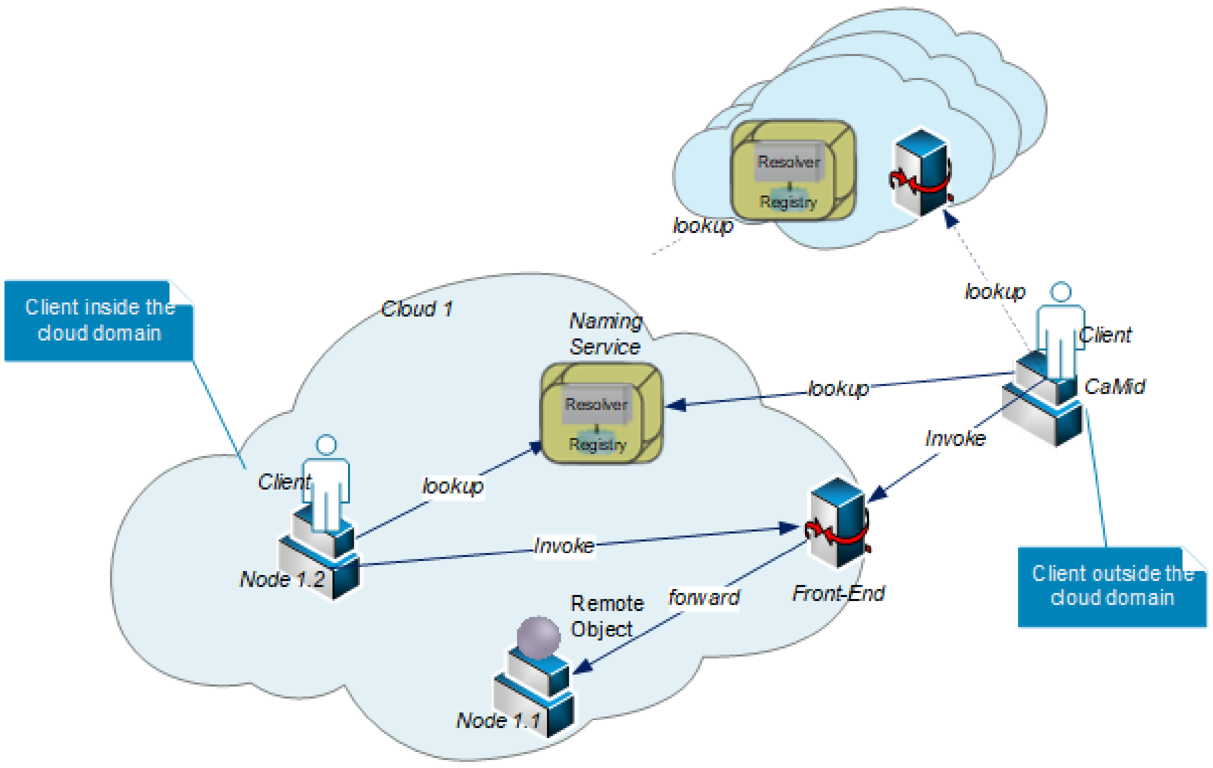
The *lookup* operation (Figure 18) is invoked by clients to look up for a service before invoking any remote object. In the *lookup*, the client sets the naming service's endpoint and remote object's name. *Naming Service* returns *Client Proxy* of the remote object. When a remote object is not found in the respective cloud domain, its *Naming Service* asks the other cloud domains about it. If the service is found in another cloud domain, *Naming Service* returns the correct cloud domain endpoint to the client-side. Otherwise, it returns null.

Multi-Cloud Registry provides information to *Cloud Manager* communicates with others since each cloud domain *Naming Service* has information about all the others. *Cloud Manager* queries information to *Multi-Cloud Registry* for requesting resources outside its cloud domain.

4.3.4.2 IaaS Service

The component *IaaS Service* provides an interface to a set of IaaS's APIs, dealing with the heterogeneity of cloud technologies. M-CaMid can interact with different cloud providers

Figure 18 – Remote object lookup.



Source: author (2021).

transparently. The component uses the open-source library JClouds® (Apache, 2011). JClouds® is a mature library for programming multi-cloud solutions that offer several API abstractions for programming languages, such as Java and Clojure.

4.3.5 Management Layer

The cloud domain management is realised by a *Cloud Manager* and a set of *Node Managers* as shown in Figure 11. *Node Manager* is in charge of monitoring all node's elements (remote objects, guest OS, and middleware components), sending monitoring information to *Cloud Manager* that is responsible for monitoring and controlling all cloud domains. Both managers interact with each other by exchanging information and requesting services. Each *Cloud Manager* manages one cloud domain composed by at least one node that is managed by a *Node Manager*.

The management layer groups components in charge of managing multi-cloud applications. *Node M-CaMid* and *Cloud M-CaMid* are composed of two main components: *Node Manager* and *Cloud Manager*, respectively. Both components include management mechanisms according to their operation domain. M-CaMid implements a distributed management in which its roles are arranged along the *node* (*Node Manager*), *cloud* and *multi-cloud* (*Cloud Manager*) domains.

M-CaMid provides a holistic view of the environment and automatically supports the application management by taking advantage of cloud elasticity. Automation allows timely and efficient responses when changes and undesirable events occur in the environment. M-CaMid management layer (Figure 19) is composed of:

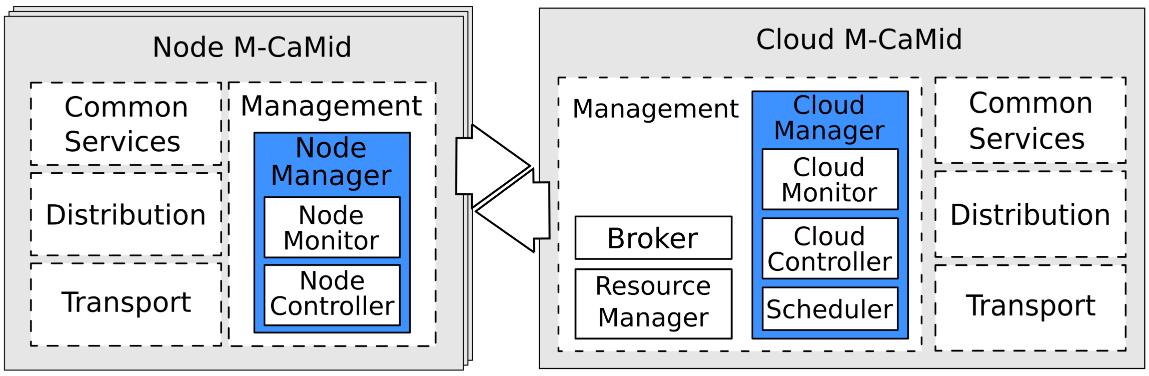
(a) *Node Manager* and *Cloud Manager* that monitor and control the nodes and the cloud environment, respectively;

(b) *Descriptors* structures information about multi-cloud application and infrastructure, where each element of the multi-cloud environment holds a data set defining its attributes and relationships with other parts. *Descriptors* also has pre-defined metrics references used as a reference for monitoring information analysis.

(c) *Broker* acts as a multi-cloud coordinator in charge of dealing with other cloud domains. Through *Brokers*, cloud domains share information about applications and virtual resources to cooperate. For example, a cloud domain can need to replicate some remote object to alleviate its workload. Thus, it asks other cloud domains about available nodes that can host the objects. Cloud domains having enough virtual resources may answer with resource details that are analysed to choose the best-fit; and

(d) *Resource Manager* interacts with the IaaS manager of the cloud infrastructure for requesting cloud services, such as VM deployment and monitoring information. *Resource Manager* uses *IaaS Interface* in the common service layer to deal with the different IaaS managers.

Figure 19 – *Node M-CaMid* and *Cloud M-CaMid* relationship.



Source: author (2021).

Management considers application and infrastructure layers to monitor and control its respective elements, combining monitoring information from different layers and domains to support adaptation. *Node Manager* and *Cloud Manager* collect and analyse monitoring data from nodes and IaaS to support the decision making. In turn, control mechanisms act over the environment for maintaining the application execution.

Cloud Manager handles the multi-cloud elasticity by following scaling strategies comprising a set of rules that *Cloud Manager* can trigger according to monitoring information collected from the *cloud* domain.

Both *Node Manager* and *Cloud Manager* uses *Descriptors* to structure information about the deployment architecture and execution of multi-cloud applications.

4.3.5.1 Multi-Cloud Descriptors

M-CaMid uses descriptors to structure registry information of each element in the multi-cloud environment. Descriptors hold data defining elements' attributes and their relationships. M-CaMid needs information such as identification, placement, configuration, computing requirements and metrics. The description is organised in application and infrastructure descriptors. Table 2 shows the descriptor structure of applications.

Table 2 – Application descriptor.

Element	Attribute	Structure	Description
Object	id	String	Object's identification
	service	String	Service name
	endpoint	[*IP*]:[*port*]	Object's location composed by the local IP address and port (integer)
	node_id	Integer	node where the remote object is running
	interface	$\{o_1, o_2, \ldots \ldots, o_n\}, n > 0$	List of operation
	proxy	Object type	Object's proxy
	sla	$< max\_rt, max\_tp >$	Metrics used by the M-CaMid monitor, where *max_rt* – maximum response time for an invocation (Real) *max_tp* – maximum throughput (Real)
	sla_metrics	$\{< o_i, sla_i >\}, i > 0$	List of tuples relating operations to metrics
	replicas	Integer	The number of running replicas
	is_replica	Boolean	Identifies the object as a replica
	OS_reqs	$< OS\_type, cpus, mem\_size >$	Deployment specification for the object, where *OS_type* is the operating system; *cpus* is the minimum number of required CPUs (Integer); and *mem_size* is the minimum memory size required (Integer)
Application	id	Integer	Application's identification
	name	String	Application's name
	objects	$\{ro_1, ro_2, \ldots \ldots, ro_m\}, m > 0$	List of remote objects

Source: author (2021).

A distributed application is composed of a set of remote objects. Each remote object has a set of attributes that provides information about its relationships and requirements. For example, a remote object has an interface whose operations has SLA metrics to be met by the underlying infrastructure. The application descriptor is relevant to the deployment, execution, and management of the multi-cloud application.

The infrastructure description is composed of the node and cloud descriptors as shown in Table 3. Many of those information feeds *Naming Service* on application initialisation and is used by both node and cloud monitors.

The developer specifies both application and infrastructure descriptors that is accessed by M-CaMid. Each cloud domain has a descriptor specifying its infrastructure specification, and each application also has a descriptor defining its specification. In case of application is distributed

Table 3 – Node and cloud domains specifications.

Element	Attribute	Sintaxe	Description
Node	id	Integer	Node identification.
	endpoint	$[IP]$:$[port]$	Local network address.
	pub_endpoint	$[IP]$:$[port]$	Endpoint where the Cloud M-CaMid is listening . A not null value means that the node hosts the Cloud M-CaMid (master_node).
	cloud_id	Integer	Identification of the cloud where the node is deployed.
	conf	$< cpus, mem, os >$	Node configuration, where: $cpus$ is the number of CPU cores; mem is the size of memory; and os is the operating system of the VM.
Cloud	id	Integer	Cloud identification.
	provider	String	Name of the cloud provider.
	nodes	$\{n_1, n_2, \ldots \ldots, n_m\}$	Set of nodes (node's ids) belonging to the cloud.
	endpoint	$[IP]$:$[port]$	Endpoint where the Cloud M-CaMid is listening on.
	master_node	Integer	It refers to the node id where the Cloud M-CaMid is deployed.
	nodes_number	Integer	Maximum amount of allocated nodes.

Source: author (2021).

in more than one cloud domain, the application descriptor must be replicated in those cloud domains.

The cloud domains can cooperate at multi-cloud domain by sharing information about distributed application's components running on its respective cloud domain and information about its available resources. Information sharing allows *Cloud Manager* to infer decisions, such as the best way to leverage cloud elasticity. The *Broker* is in charge of cloud interoperability.

4.3.5.2 Node Manager

The Node Manager in *Node M-CaMid* is composed of monitoring and control mechanisms, as shown in Figure 20. The monitoring system collects, processes and analyses data from remote objects, middleware layers and the guest OS (OS of the VM in the node) through the components:

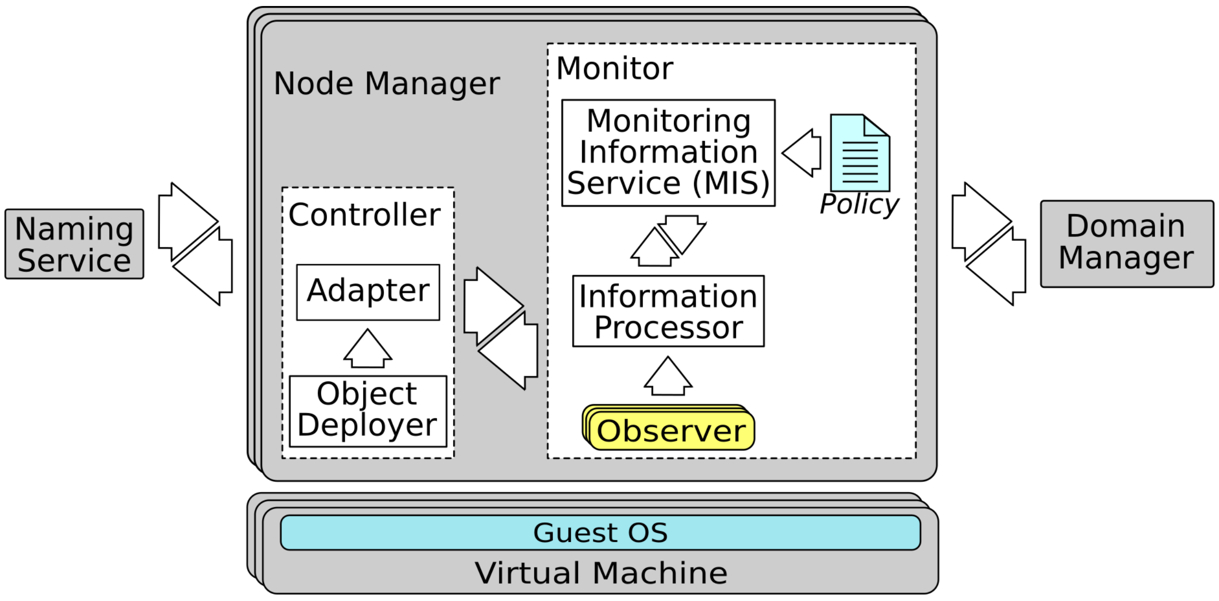
- *Observers* collect information from the remote objects, the middleware layers stack and the guest OS (VM's OS). *Observers* allow different levels of information granularity that can be combined to produce additional information and improve the decision-making process. *Observers* continually collect CPU and memory usages (*System Observers*), and the time consumed to a remote object process incoming invocations (*Object Observer*). Also, *Handler Observers* gather general incoming request data in the node (e.g. incoming request ratio). All those data are then periodically sent to the *Information Processor*.

- *Information Processor* receives raw data to formats and summarises collected data, computing many common statistical values, including lowest and highest values, standard deviation and mean. The Information Processor periodically calls *Monitoring Information Service* (MIS) to analyse the data.

- *Monitoring Information System* (MIS) infers monitoring data coming from different abstraction levels. MIS analyses and relate gathered data from middleware and system observers. Monitoring data is compared to a set of pre-defined reference metrics specified in *Descriptors* to monitor the adequated environmental behaviour and alert about abnormal events. M-CaMid needs information such as identification, placement, configuration, computing requirements and metrics.

- *Object and Node Descriptors* The description is organised in objects and node infrastructure descriptors. Table 2 and Table 3 shows the description of the remote object and node data structures, respectively.

Figure 20 – Architectural elements of *Node Manager*.



Source: author (2021).

Controller in *Node Manager* implements actions to control remote objects and replicas running locally, such as local deployment, object binding and unbinding with the Naming Service's help. *Controller* has two components, namely *Adaptor* and *Object Deployer*.

Adaptor executes actions for setting up objects and replicas locally. It creates and destroys objects, create replicas to run in other application nodes, and registers objects in the *Naming Service*. For example, when an object needs to be replicated, the *Adaptor* makes an object's copy and sends it to the target node domain. In the target node, the *Adaptor* receives the replica and registers it in the *Naming Service*. Next, the *Adaptor* uses the *Object Deployer* to locally record the replica and deploy it in the *Lifecycle Manager*.

The *Object Deployer* is in charge of deploying/undeploying objects and replicas locally. The deployment registers the object locally and requests the *Lifecycle Manager* to execute the object. Conversely, the undeployment unregisters objects locally and stops them in the *Lifecycle Manager*.

Periodically, *MIS* sends information about the general performance of the node to *Cloud Manager* that is responsible for orchestrating the workload balancing in the cloud domain.

4.3.5.3 Cloud Manager

Cloud Manager is in charge of monitoring and control all nodes in a cloud domain. *Cloud Manager* is responsible for:

- *Managing* virtual resources and remote objects, taking actions based on monitoring information gathered from the environment;
- *Interacting* with IaaS mechanisms for using its services; and
- *Cooperating* with other cloud domains for sharing information and requesting services.

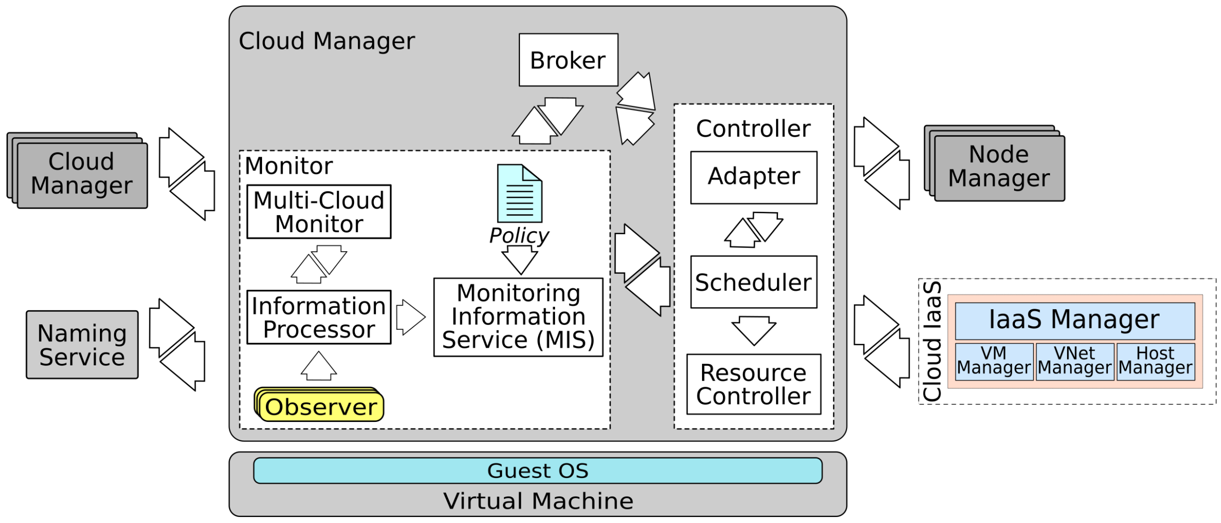
Cloud Manager receives nodes' monitoring information and requests through *Cloud Monitor* and processes it to produce knowledge about the environment. Information feeds *Cloud Manager* that can step in the cloud environment, triggering actions to solve problems or improve performance. *Cloud Manager* also monitors incoming invocations to object's services deployed in the cloud, metering invocation response time and invocations throughput, for example. These measurements allow analysing the service behaviour taking into account the communication between *Cloud M-CaMid* and *Node M-CaMid* during a service invocation. Furthermore, *Cloud Manager* interacts with other ones to exchange information for supporting the multi-cloud resource sharing. Through *Cloud Manager*, it is possible to accomplish four main tasks: sharing performance information with other cloud domains, requesting resources, offering available resources, and executing replication in the multi-cloud environment.

Periodically, *Cloud Monitor* analyses nodes usage to identify idle nodes and redistribute the workload among them. Remote objects can be reallocated without depreciating the overall performance. The workload rearrangement can reduce costs avoiding resource wasting.

Figure 21 presents the components of *Cloud Manager*. *Cloud Monitor* has similar components to *Node Manager*. Each node sends information about its performance. *Observers* in *Cloud Monitor* receive information and alerts from nodes and send it to *Information Processor* to be processed and then analysed by *Monitoring Information Service* (*MIS*). The analysis combines information to identify relationships among them. Based on these relationships, new information can be inferred by *MIS*. Besides, *Observers* gathers services information from *Cloud M-CaMid* distribution layer, such as invocation response time and throughput.

Adaptor performs actions automatically and decides on which plan should be executed to scale the resource. It must decide *what* replicates (remote objects or nodes), *how* to replicate (scaling up and scaling down) and *where* to replicate (in the same cloud or outside the cloud). For example, it can replicate the whole node or just some remote objects to another node. *Adaptor* coordinates actions for replication of remote objects requesting services to *Scheduler* and *Resource Controller*. These decisions compose the adaptation plan, as shown in Table 4.

Figure 21 – Architectural elements of cloud management.



Source: author (2021).

Table 4 – Adaptation plan decisions.

Plan	Decisions
What	remote objects or nodes
How	Scaling up or scaling down
Where	Cloud domain or multi-cloud domain

Source: author (2021).

Scheduler executes adaptation plans built by *Adaptor*. It looks for a list of available nodes, where *Scheduler* can deploy remote objects without degrading the node's performance. If no active node is available (with enough resources to meet the request), *Scheduler* interacts with *Resource Controller* to allocate a new VM and configure it as a new node. *Resource Controller* executes operations at IaaS level and wraps specifics services of the cloud provider, such as allocating, deallocating and cloning VMs.

Broker acts as a multi-cloud coordinator in charge of dealing with other cloud domains. Through *Brokers*, cloud domains share information about multi-cloud applications and virtual resources. *Broker* is in charge of requesting resources to other *Brokers* and selecting them. *Broker* receives the request for new resources from *Adaptor* whenever it is not possible to allocate more resources in the current cloud domain. *Broker* asks for more resources by sending a message to others. In turn, cloud domains with available resources respond through their *Brokers*. After receiving the responses, *Broker* selects the best-fit resources and redirects them to *Adaptor* that selects the best-fit option to execute the replication of the remote object. The target cloud domain can offer resources into a running node or a new node (by instantiating a new VM) if there is not enough resource in the cloud domain.

Node Managers and *Cloud Manager* form a group into a domain, following the group communication paradigm (Coulouris et al., 2013). Each one has attributions as a member, such as joining or leaving the group. The group communication allows reliable communication between the group's members, ensuring that sent messages are delivered to all target members. The group communication paradigm manages the member's participation through four tasks: *(a)* creating and destroying groups and adding and removing members, *(b)* fault detection and change notification (members are notified about changes, such as new member and leaving member) *(c)* message delivery coordination (task delivers messages regardless of the changes in the member participation). *Group Coordination Handler* component is in charge of realising reliable group communication.

4.4 Cross Management

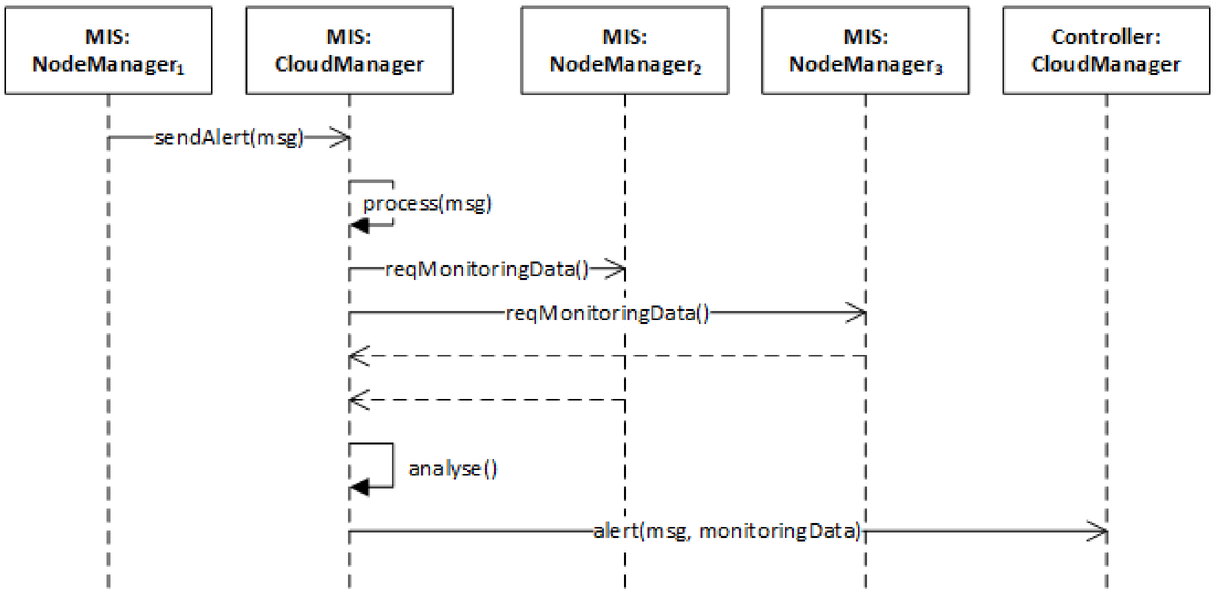
As discussed previously, M-CaMid must integrate the different domains at different levels. Managers in Node and Cloud M-CaMid implement transversal management encompassing many granularity levels that allow more detailed monitoring data to support a more precise action over the environment (vertical management). Different information levels improve a "best fit" resource sharing among domains. Instead of deploying entire nodes, M-CaMid can choose underused nodes to replicate only remote objects. This strategy becomes possible thanks to cooperation among nodes through the Cloud M-CaMid even out of a cloud domain (horizontal management).

M-CaMid defines two management dimensions: vertical and horizontal. The vertical dimension is related to the management granularity level based on the monitoring information analysis. Coarse-grained management takes place at the infrastructure level through the deployment of new VMs. Fine-grained control occurs at the application level, where M-CaMid looks for underused VMs and replicates overloaded remote objects in an existing VM. The vertical dimension refers to how control actions are realised, while the horizontal dimension refers to where the action takes place: same cloud (single cloud environment) or another cloud (multi-cloud environment).

The information analysis in the *Node M-CaMid* periodically checks infrastructure usage data by comparing it to pre-defined metrics. If *MIS* identifies inconsistent values, an alert is generated. For example, the node CPU usage can exceed the maximum allowed value for a while, overloading the node. *MIS* notifies the *Controller* that requests more resource to *Cloud M-CaMid*.

Node CaMid sends to *Cloud M-CaMid* a message with the type of alert and objects' monitoring information to support the *Cloud M-CaMid* decision making (Figure 22). For example, a node's overloading can be caused by a specific remote object with a high response time. Thus, *Cloud M-CaMid* can step in the node only handling the specific object. For a complete analysis, M-CaMid ask other nodes in the cloud for information about resource usage to elect the "best fit" nodes as candidates to assist *Cloud M-CaMid* on environment's adaptation performed by *Controller*. In the absence of available resources in the cloud domain, the *Controller*, through *Broker* component, can request resources outside the cloud.

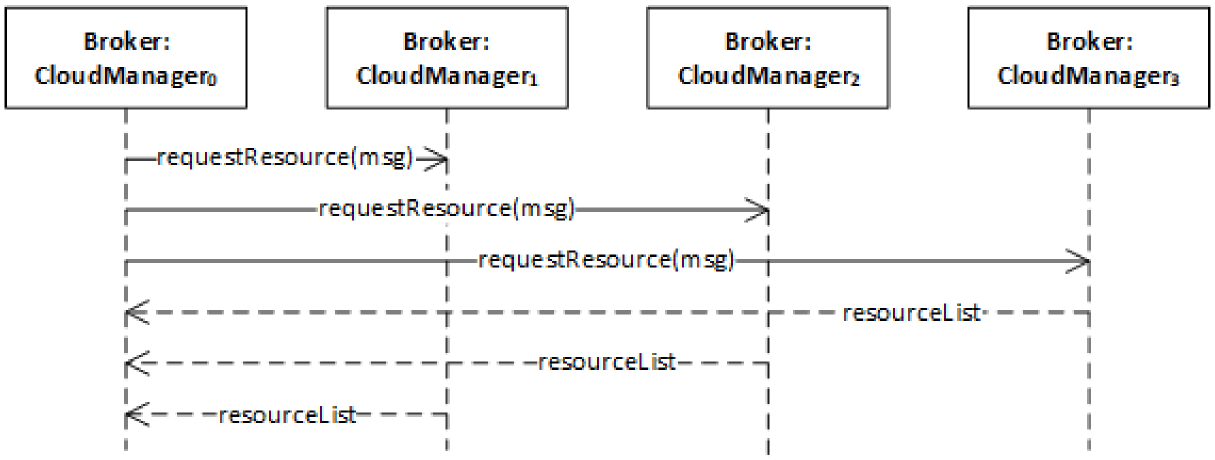
Figure 22 – *Node MIS* and *Cloud MIS* interactions.



Source: author (2021).

Broker can request resources to other cloud domains, send information about its demand (Figure 23 shows the *Brokers*' relationships). Each *Cloud Broker* receiving the request asks the respective *MIS* about available cloud resources. In turn, *MIS* requests monitoring information to its cloud nodes and analyses to select the "best fit" available nodes, as shown in Figure 24. If there is not enough resource, the *Broker* responds with an empty list.

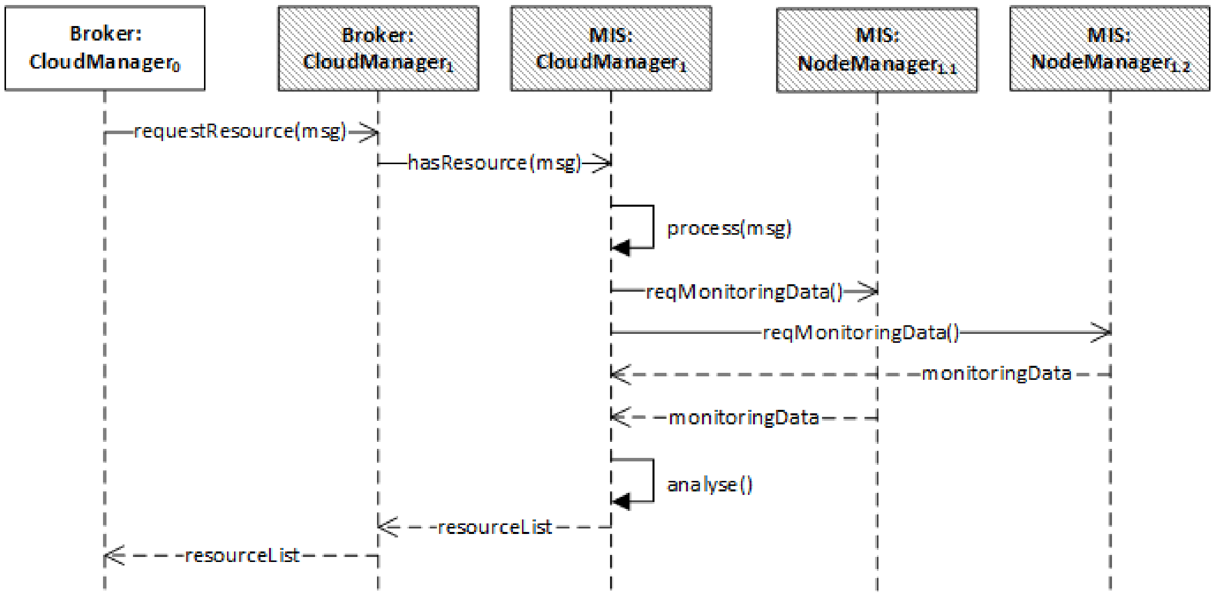
Figure 23 – *Brokers* relationship: *Broker*$_0$ requests resources to others *Brokers*.



Source: author (2021).

Each cloud domain can decide about how and where to adapt its environment based on monitoring information that can be: *(i)* replication of remote objects (fine-grained management) to another existing node or create a new node to support the new demand (coarse-grained management) – vertical management–, and *(ii)* replication can take place inside or outside the cloud domain (horizontal management). *Controller*'s decisions define how M-CaMid can explore multi-cloud elasticity.

Figure 24 – *Broker* locally requesting resources to other cloud domain.



Source: author (2021).

4.5 Managing Elasticity

The main goal of M-CaMid is the multi-cloud application management by exploiting the elasticity benefit among multiple clouds. M-CaMid transparently manages distributed applications and their infrastructure. The elasticity method considers *how* and *where* to scale out and in. Issue *how* refers to the levels of granularity for elasticity: fine-grained and coarse-grained. In the first one, elasticity is applied to remote objects, while in the second one, elasticity is applied to VMs. In both cases, M-CaMid realises a replication task that can be an object or a VM replication, respectively. Replication is the process of copying a resource and instantiating it in a distinct computing node in a distributed system, improving availability and performance. Now, *Cloud Gateway* in the distribution layer also redirects remote invocations to the original resource and its replicas through the Load Balancing mechanism that selects a target endpoint. It uses *Load Balancer* and *Forwarder* to select the target node and redirect the invocation to this node, respectively.

Meanwhile, issue *where* refers to the domain where elasticity can occur: into the current cloud domain or outside. Analysis and decision making about elasticity strategy occur in the *Cloud Manager*, as presented in Section 4.3.5.3.

In *Cloud Manager*, *Monitor* notifies *Controller* about two types of abnormal events. The first one occurs when a remote object's response time overcomes the response time threshold defined in the policy, breaching pre-defined requirements. This type of event is detected by

Cloud Manager, where the total response time in server-side can be measured and analysed by *Monitor* that, in turn, produces an alert to *Controller*. The second one occurs in a node domain when its VM is overloaded, and the system's thresholds (CPU or memory usages) are overtaken. An overloaded VM may occur due to a massive amount of invocations to a remote object. Thus, the service response time also can increase due to the VM overloading.

Regarding *where* elasticity takes place, two strategies are defined: cloud elasticity and multi-cloud elasticity. The first is a default strategy, while the second is triggered if it is impossible to allocate more resources in the current cloud (resource limit has been reached).

4.5.1 Cloud Elasticity

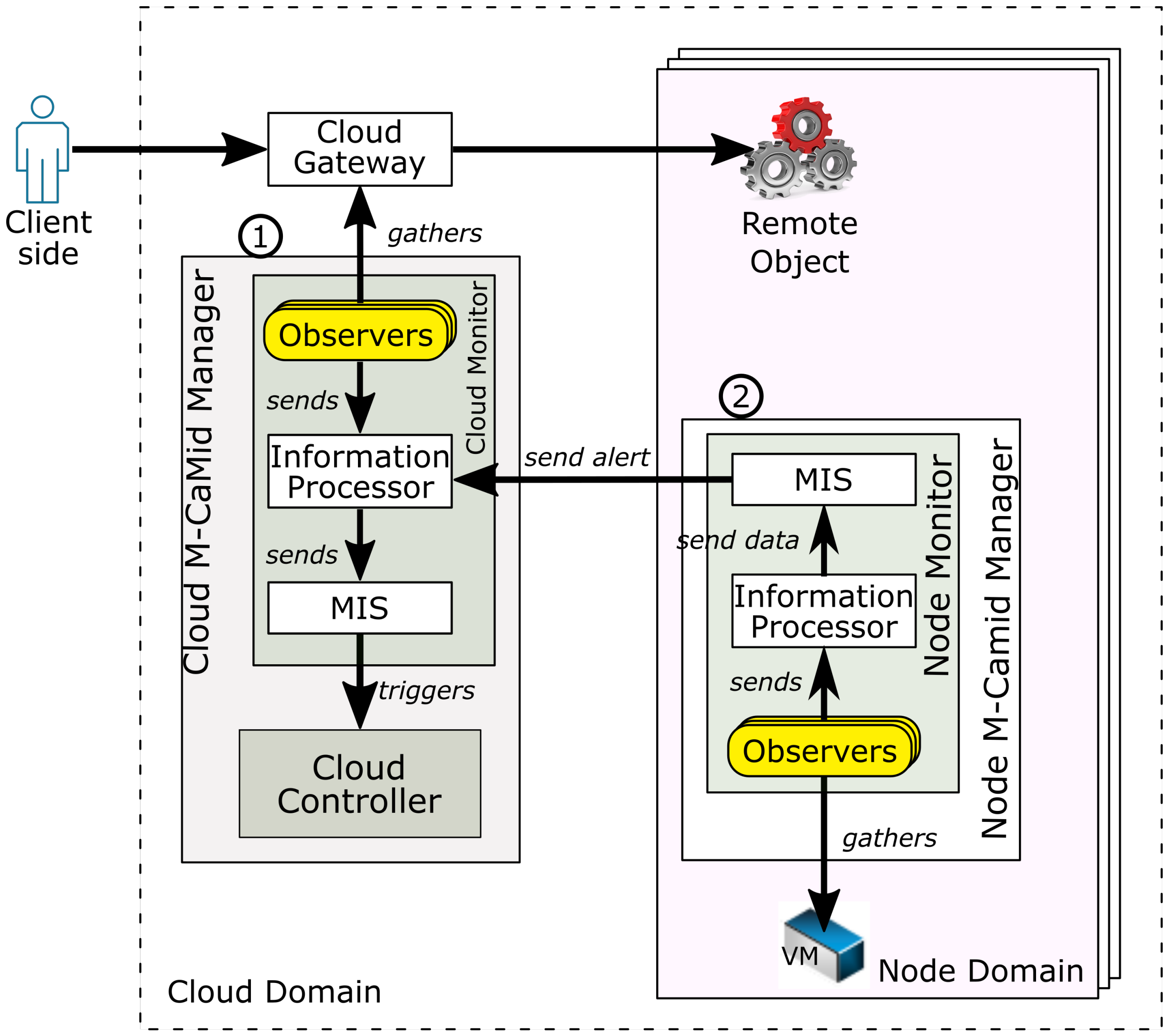
Cloud Manager can generate alerts when response time some service overtakes a pre-defined threshold during a period. *Cloud M-CaMid Monitor*, through *Observers* (Figure 25, ①), measures the response time of remote objects' invocations in *Cloud Gateway*. Information Processor computes statistical measurement, such as mean response time and throughput. *MIS* can identify abnormal events based on statistical data. Since generated alert refers to an application service running in one or more nodes (in case the service has replicas) in the cloud, *Cloud Manager* requests monitoring information (CPU and memory usage) to all related nodes. *Node MIS* in *Node Manager* of each related *Node M-CaMid* sends information to *Information Processor* in the *Cloud M-CaMid* that relates nodes' information to the overloaded service and sends to *MIS*. *MIS* analyses how nodes are affected by service overloading. Nodes with critical performance are candidates to be adapted.

Figure 25 ② also shows another type of event generated by *Node Managers* to notify *Cloud Manager* about system overload (CPU usage). In *Cloud M-CaMid*, *Cloud Manager* receives an alert with event monitoring information and a list of remote objects running in that node. *Information Processor* relates event information to the respective service's throughput and response time collected by *Observers* from in *Cloud Gateway*. *Information Processor* also gets information about available nodes resources in the cloud domain. *MIS* analyses information to identify which remote objects are more affected by node's overload (by comparing response time and throughput thresholds). In both cases, *MIS* analyses information at the infrastructure and application levels to provide a precise analysis to support *Adaptor* on decision making.

Adaptation process is exclusively realised by Cloud M-CaMid as shown in Figure 26. Based on analysed information, *Adaptor* in *Cloud Manager* makes decision about *what* and *how* and *where* to adapt. Adaptation defines the granularity level of elasticity (*what*), actions to be executed (*how*) and *where* executing replication.

Adaptor is in charge of making an adaptation plan, defining *what*, *how* and *where* to do. *Adaptor* defines the granularity level of elasticity: coarse-grained or fine-grained. First, *Adaptor* decides what to do, choosing between two options: scaling up or scaling down. In case of occurrence of a node overload or a service exceeds its response time threshold, *Adaptor* decides to scale up. Scaling up can be accomplished according to the elasticity strategy defined in the

Figure 25 – Event detection and analysis in cloud elasticity process.



Source: author (2021).

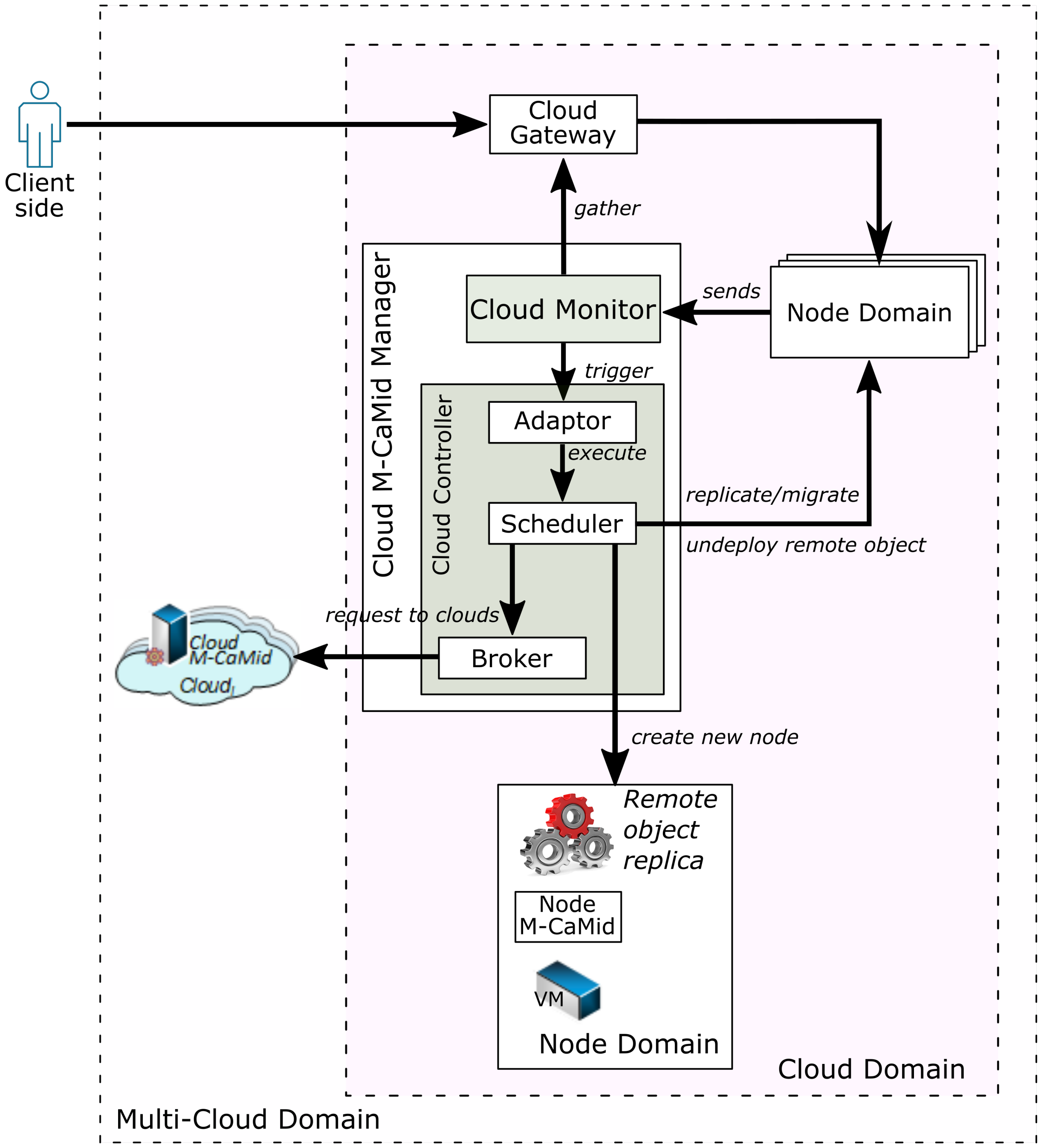
plan (how). Instead, a service implemented by a remote object and its replicas can have response time and throughput under their minimum limits. These minimum limits are defined through controlled tests that combine response time and throughput measurements to state minimum values to keep the service stable. When these measurement values are reached, *Adaptor* decides to scale down, turning some remote objects off. Scaling down does not undeploy nodes because they can be used in the short term to host other affected remote objects.

Regarding how to adapt, *Adaptor* checks a list of available nodes provided by *Information Processor* to find underused nodes. For example, a node with CPU usage under 30% can be an available resource composing the available node list. In case of the list is not empty, *Adaptor* chooses fine-grained elasticity, replicating the most performance affected remote objects in other underused nodes. On the other hand, if the available node list is empty, *Adaptor* must plan to request new nodes to host affected remote objects.

Finally, *Adaptor* plans where adaptation must take place. Suppose the available nodes list is empty and no more resources can be allocated (maximum resource limit has been reached). In that case, Adaptor decides for scaling up outside the cloud domain, requesting resources to other clouds.

Adaptor concludes the adaptation plan by defining what to do (scaling up and down), how to do (fine-grained and coarse-grained elasticity) and where to do (cloud and multi-cloud domains). Table 4 summarises the plan options of *Adaptor*. After, *Adaptor* requests to *Scheduler* to execute adaptation plans.

Figure 26 – Adaptation in cloud and multi-cloud elasticity process.



Source: author (2021).

As explained in Section 4.3.5.3, *Scheduler* is in charge of executing adaptation plan. It maps remote objects to available nodes selected by *Adaptor*. Depending on how to do the adaptation, *Scheduler* can execute replication actions (scaling-up) or deactivate remote objects (scaling down). Scaling up action depends on what must be adapted: remote object through fine-grained elasticity or node through coarse-grained elasticity. In both cases, *Scheduler* notifies both source and target *Node Managers* about which and where remote objects must be replicated. Source *Node Manager* sends an object's copy to the target one that deploys the object locally (*Object Deployer*). The new replica is registered locally and in *Naming Service*. However, when adaptation occurs outside the cloud (multi-cloud domain), *Scheduler* requests to Broker to mediate new resources acquisitions in the multi-cloud domain.

Scaling down action deactivate remote objects which service's measures are under minimum limits. M-CaMid prioritises turning remote objects off hosted in other clouds and those hosted in nodes with low CPU usage.

4.5.2 Multi-Cloud Elasticity

The multi-cloud elasticity is triggered when no more resources are available in the current cloud domain. In this scenario, source cloud *Broker* requests more resources to all clouds in the multi-cloud domain through *Broker*, sending information about its demand. Each target cloud domain (through its *Broker*) receives and analyses the request. First, *Broker* checks the node availability, i.e., the number of running nodes and total nodes available in that cloud domain. Without enough reservation, the cloud domain is unfit to meet the demand. Target *Broker* validates a request based on its deployment constraints: VM size (memory and CPU), OS type and maximum resource limit. In case the cloud domain meets constraints, *Broker* sends the resource characteristics to the source cloud domain. The source cloud domain may receive the resource offering of many cloud domains and, thus, decides which one best fits the demand. Once the source cloud domain chooses the best fit resource, its *Broker* sends object's replicas to be deployed in the new cloud domain.

Remote object replication processes among cloud domains are similar to the respective cloud processes. The difference is that *Broker* mediates the process. It is in charge of getting the object's copy and sending it to the target *Broker*. In turn, the target *Cloud Controller* and *Node Controller* are responsible for deploying the remote object's copy deployment in the respective domain. The target node notifies about the operation success or failure. The new object is registered in *Naming Service*. The source cloud domain is also notified, and the new location is registered in its *Naming Service*. The registration is useful to *Forwarder* redirects multi-cloud invocations as described in Section 4.3.3.2.

4.6 Concluding Remarks

This chapter presented requirements for designing the M-CaMid architecture. Principles (Chapter 3) and Requirements drove the M-CaMid architecture design. To be multi-cloud aware, M-CaMid defined three domains in the multi-cloud environment: node, cloud and multi-cloud. The architecture details and operations were discussed regarding distributed application communication and management in a multi-cloud environment. The architecture's layers were presented: transport, distribution, common service and management. Cross management gave how its components interact to manage multi-cloud distributed applications by exploring elasticity in the cloud and multi-cloud domain. Finally, elasticity management was explained, presenting its components and how adaptation is accomplished.

5 Experimental Evaluation

This chapter discusses experiments to evaluate M-CaMid. The main goal is to assess the M-CaMid in managing multi-cloud distributed applications by exploring elasticity at different levels of granularity and scopes. A systematic approach for performance evaluation was adopted (Jain, 1991) to accomplish the experiments. First, some steps are defined to conduct the experiments: goals and system boundaries, scenarios, services and outcomes, metrics, parameters and factors, experiment design. Then, the results are presented and discussed.

5.1 Goals and System Boundaries

The experiments' main goal is to estimate the impact of using M-CaMid over the performance of multi-cloud distributed applications. The experiment's system consists of a multi-cloud environment. Its components are a multi-cloud distributed application, a multi-cloud infrastructure composed of a set of private clouds and M-CaMid. The multi-cloud application follows a client-server architecture, where the client-side can invoke a service implemented by remote objects on the server-side. The service executes the calculation of the Fibonacci sequence. Remote objects can be hosted in a single node, in many nodes in a single cloud, or many clouds.

Since M-CaMid is a client-centric solution, some environment components do not belong to the system under evaluation: tools for IaaS management, low-level communication channel and the Internet. Public clouds are not used in the experiments because their IaaS infrastructure resources are out of the system boundaries.

5.1.1 Scenarios' Description

M-CaMid runs on scenarios where a multi-cloud application executes in a limited-growth infrastructure and is submitted to unpredictable peaks of workloads.

To face this challenge, M-CaMid supports the high availability and scalability of multi-cloud applications. The implemented distributed application follows the client/server architecture, in which a wide range of client applications (client-side) invoke remote objects' methods (server-side). The client-side is deployed on top of a VM outside of cloud domains. The remote object on the server-side delivers as a service the calculation of Fibonacci sequence. Initially, instances of *M-CaMid Naming Service* and *Cloud M-CaMid* are deployed in all cloud domains. *Cloud M-CaMid* instances register themselves in the respective *M-CaMid Naming Service*. Then, a *Node M-CaMid* instance is deployed on a VM in the respective cloud domain. It is worth noting that *Node M-CaMid* deployment can occur in more than one cloud domain. *Node M-CaMid* registers itself and its remote objects in the respective *M-CaMid Naming Service*. After registration, a remote object can be reachable by client-side instances.

The multi-cloud environment is composed of two cloud domains where remote objects are deployed. Both cloud domains have the same hardware configuration but different cloud management technologies: OpenNebula (Moreno-Vozmediano et al., 2012) and OpenStack (RackSpace, 2010). In both cases, the cloud infrastructure comprises a cloud frontend and a host server. The cloud frontend is in charge of managing hardware resources and works as an interface for cloud users, and the host server shares its resource for VMs. The OpenNebula and OpenStack frontend instances run on machines core i5 2.44GHz with 4 GB of memory, and their respective host servers are octa-core Xeon 2.93 GHz machines with 20 GB of memory.

VMs for hosting M-CaMid instances (Cloud and Node M-CaMid) are configured with one CPU core and 1 GB of memory. Initially, two VMs were deployed to execute the experiments: a server-side application node (Node M-CaMid) and a Cloud M-CaMid VM. As the experiment progresses, Cloud M-CaMid may deploy new VMs.

5.2 M-CaMid's Services and Outcomes

M-CaMid provides a set of services for managing communication and distribution of multi-cloud applications. M-CaMid must keep the multi-cloud application working properly, assuring its properties. At the same time, it manages the underlying infrastructure, providing efficient use of its resources.

It is expected that M-CaMid supports the remote object invocation assuring its correct completion within an acceptable period (response time). Otherwise, the invocation can fail or take a time higher than the pre-established threshold. The undesirable requests must be detected by M-CaMid that acts over the environment through the elasticity. Elasticity management comprises monitoring and controlling resources through replication, migration and load balancing in different domains. Undesirable results are short reaction time by M-CaMid facing unwanted events (e.g., overload) and efficient workload distribution among application's nodes. The desirable results are low overhead and error rate in remote invocation when M-CaMid's services are activated.

5.3 Experiments

Three experiments evaluate the M-CaMid operations for managing multi-cloud distributed applications:

1. *Management reaction time*: evaluates the automatic control of M-CaMid on dealing with undesired environment's behaviour. It observes how fast M-CaMid steps in the environment to keep it stable. The experiment evaluates M-CaMid's performance measuring the reaction time and response time with M-CaMid management turned off and turned on. Both scenarios are analysed by comparing each other. In the scenario turned on, the reaction time is measured in two elasticity levels (VM and application level).

2. *Elasticity resource management*: assesses the rational usage of infrastructure (virtual machines) considering coarse- and fine-grained management. It evaluates M-CaMid management by measuring the resource usage of VMs (CPU usage), observing resource over- and under-provisioning aspects. The experiment scenario takes place at two management levels: coarse- and fine-grained management.

3. *M-CaMid overhead*: assesses the overhead introduced by M-CaMid's node monitor on the application performance. The overhead is measured in two scenarios: node monitor turned off, and node monitor turned on. This experiment is limited to the monitoring system because a growing workload stresses M-CaMid. In this case, if the M-CaMid controller is on, the manager triggers the elasticity to fix the overload problem, affecting the experiment results.

5.4 Reaction Time

The experiment's primary goal is to assess how fast M-CaMid automatic management intervenes in the environment to reestablish a multi-cloud application's regular operation. It intends to observe the time spent by M-CaMid's strategies to reconfigure the environment under undesired behaviours. Metric reaction time is measured in milliseconds data units. Reaction time is measured when M-CaMid management is configured to adopt different levels (coarse- and fine-grained control) in the single cloud and multi-cloud domains. The experiment analyses differences among strategies in both domains. Fine-grained management is expected to take less time to reconfigure the environment regardless of the domain—furthermore, the experiment analyses differences in the same management strategy in both domains.

5.4.1 Hypotheses

In the experiment null hypotheses ($H0_{1..6}$), there is no difference in the reaction time (RT) when cross management strategy assumes the configurations turned off (off), coarse-grained (cg) or fine-grained (fg) levels, whether at a single cloud (sc) or multi-cloud (mc) domains.

$H0_1 \ : RT_{off\_sc} \cong RT_{cg\_sc}$

$H0_2 \ : RT_{cg\_sc} \cong RT_{fg\_sc}$

$H0_3 \ : RT_{cg\_sc} \cong RT_{fg\_mc}$

$H0_4 \ : RT_{cg\_mc} \cong RT_{fg\_mc}$

$H0_5 \ : RT_{cg\_sc} \cong RT_{cg\_mc}$

$H0_6 \ : RT_{fg\_sc} \cong RT_{fg\_mc}$

On the other hand, the alternatives hypotheses ($H1_{1..6}$) are:

$H1_1 \ : RT_{off\_mc} < RT_{cg\_sc}$

$H1_2 \ : RT_{cg\_sc} < RT_{fg\_sc}$

$H1_3 \ : RT_{cg\_sc} < RT_{fg\_mc}$

$H1_4 \ : RT_{cg\_mc} < RT_{fg\_mc}$

$H0_5 \ : RT_{cg\_sc} < RT cg\_mc$

$H0_6 \ : RT_{fg\_sc} < RT fg\_mc$

5.4.2 Metrics, Parameters and Factors

The metric adopted to assess the mean reaction time (RT) is the time taken by M-CaMid to conclude management actions to restore the multi-cloud application's regular operation. This period comprises the undesired event detection (application's response time violation) until the multi-cloud application regular operation reestablishment (response time under the maximum limit).

The system parameters are constants. Thus, we keep the exact configuration of VMs, cloud management configurations, and hardware specifications (Section 5.4.4). The cloud and multi-cloud domains are simulated in private environments to avoid distortions on measuring caused by the Internet's speed fluctuations.

The workload parameters (factors) vary to analyse M-CaMid behaviour while it steps in the environment. Section 5.4.4 describes the workload in details. The workload parameters are the following:

- **Number of clients** (*NC*): number of simultaneous users (client-side instances);
- **Cross management** (*CM*): it can be turned off, turned on and set to coarse- or fine-grained management ($CM = \{off, cg, fg\}$); and
- **Domain** (*D*): it indicates the domain where replication takes place (single cloud or multi-cloud) ($D = \{sc, mc\}$).

5.4.3 Treatment

The treatment applied to this experiment is the M-CaMid cross management approach that manages the elasticity at different granularity levels and domains. The absence of treatment is called cross management off in a single cloud domain, while the treatment is called management in the single cloud and multi-cloud domains.

M-CaMid cross management impacts are analysed by comparing multi-cloud application performance without treatment, with M-CaMid cross management acting exclusively at the infrastructure and application levels. Besides, treatments are compared to validate the use of multiple strategies and domains.

The experiment's control object is a multi-cloud distributed application whose server-side is a remote object that implements the Fibonacci sequence in two situations: without M-CaMid management ($CM = off$) in a single cloud ($D = sc$) and with coarse-grained management ($CM = cg$) in a single cloud ($D = sc$). The experiment submits the object control to a workload to observe its behaviour as the workload varies.

The experimental object is the same as the control object but with M-CaMid management configured to fine-graned management. The experiment also submits the same workload as the control object. Both object behaviours are observed at single cloud and multi-cloud domains.

5.4.4 Experimental Design

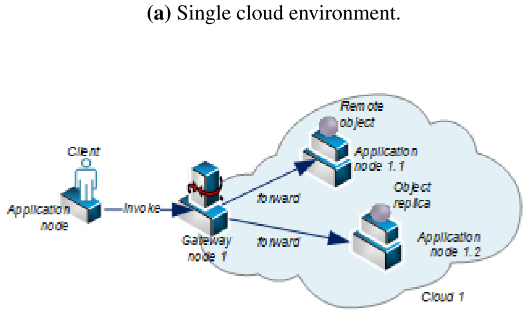
A client-server application implemented atop M-CaMid and two cloud technologies, namely OpenNebula (Moreno-Vozmediano et al., 2012) and OpenStack (RackSpace, 2010), were utilized in this experiment. The OpenNebula was configured with an IaaS manager (frontend) and a host for running VMs. The frontend runs on a machine core i5 2.44GHz with 4 GB of memory, and the host is an octa-core Xeon 2.93 GHz machine with 20 GB of memory. Each VM is configured with one CPU core and 1 GB of memory. Three VMs were deployed to execute the experiments: a server-side application and its replicas. OpenStack was configured in a similar way to OpenNenbula.

Figure 27 shows the basic configurations used in all scenarios. In the first scenario (S_1), which works as a base case of response time behaviour, cross management is turned off. In Scenario S_2, cross management is set to trigger the coarse-grained strategy, deploying a new VM when necessary. Scenario S_3 describes how existing underused VMs can support new application's replicas without the need for new VM deployments. These three scenarios were deployed in a single cloud environment, as shown in Figure 27a.

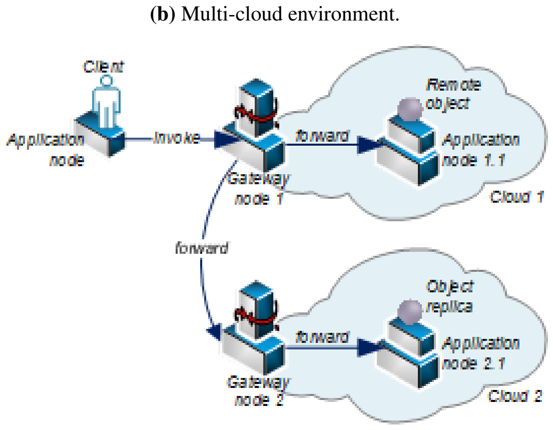
In the multi-cloud scenarios (S_4 and S_5), shown in Figure 27b, the current cloud domain do not support more resources to new application's replicas and become necessary to request further resources in another cloud domain. In these scenarios, it is up to the target cloud domain to decide the cross management level that *Adaptor* must trigger. In scenario S_4, the target cloud deploys a new VM (coarse-grained level). In contrast, in scenario S_5, a new application's object replica is created in an existing VM in the target cloud domain.

The experiments use a workload pattern having a periodic unexpected workload increase. In this way, it is expected that the response time suddenly increases and exceeds a pre-defined maximum response time. The execution starts with 100 clients that continuously invoke a method to the remote object that runs on the server-side. After 60 seconds, the number of clients increases to 300 and lasts for 200 seconds. During this time, the server-side becomes overloaded, leading to an abrupt increase in the response time. After that, the number of clients decreases to 100, and the response time returns to the same level as before. The same pattern repeats until the

Figure 27 – Basic configurations used in all scenarios.



(a) Single cloud environment.



(b) Multi-cloud environment.

Source: author (2021).

Table 5 – Evaluation scenarios.

Scenario	Cross management strategy	Domain
S_1	OFF	sc (single cloud)
S_2	cg (coarse-grained)	sc (single cloud)
S_3	fg (fine-grained)	sc (single cloud)
S_4	cg (coarse-grained)	mc (multi-cloud)
S_5	fg (fine-grained)	mc (multi-cloud)

Source: author (2021).

end of the experiment.

Each client continuously makes requests, and the time interval between requests has a Gaussian distribution whose mean value and standard deviation are 50ms and 12.5ms, respectively. Additionally, two thresholds were defined to trigger the creation/release of VMs. If the *response time* becomes higher than 1300ms (*highest response time*), a new VM is necessary. Meanwhile, if the *response time* becomes lower than 500ms (*lowest response time*), there is an excess of resources being allocated, and a VM needs to be released. All mentioned thresholds were empirically defined through several initial experiments, where the system ran in different scenarios and under diverse operational conditions.

5.4.5 Analysis

This experiment compares the data of experimental and control objects to check the null hypothesis rejection. First, this section discusses the observed scenarios behaviours. It then analyses the cross management impact over the M-CaMid reaction time by denying null hypotheses (Section 5.4.1).Table 6 summarises statistical results of the experiments for each scenario presented in Table 5.

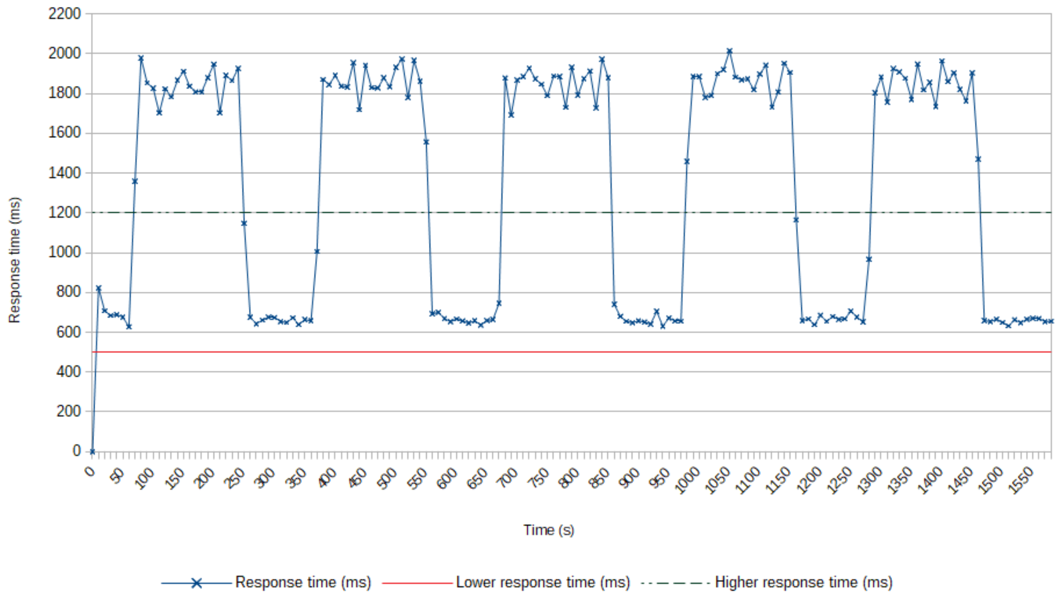
Table 6 – Average reaction time and standard deviation in different experiment's scenarios.

Scenario	Reaction time (s)	Standard deviation (s)
S_1	182	4.472
S_2	110	7.071
S_3	22	4.472
S_4	122	8.365
S_5	36	5.477

Source: author (2021).

Figure 28 shows the behaviour of metric *response time* in scenario S_1. As mentioned before, this scenario works as the base case in which the cross management is disabled, and the *response time* is only impacted by the changes defined by the workload pattern described before.

Figure 28 – *Response time* without elasticity management.



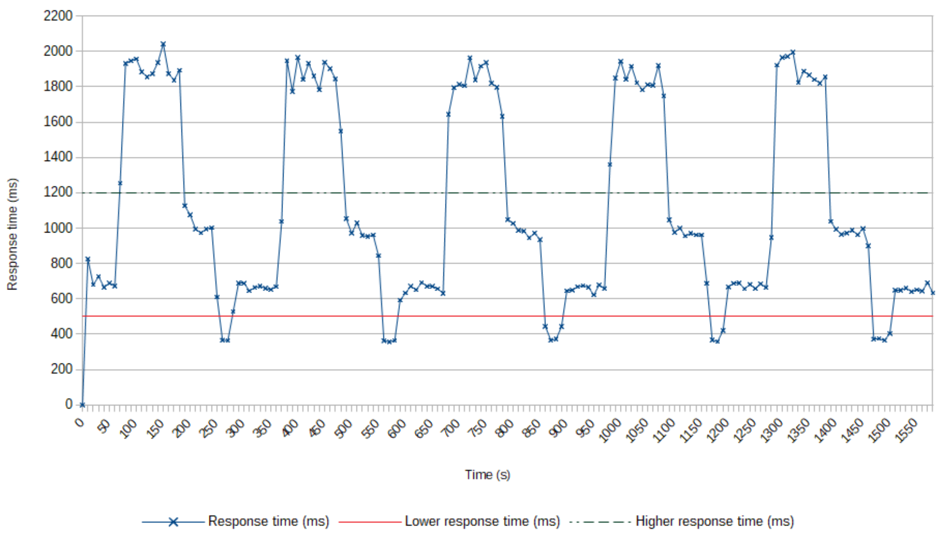
Source: author (2021).

Figure 29 shows the behaviour of *response time* in scenarios in which the M-CaMid cross management is enabled in a single cloud. Figure 29a shows the coarse-grained management level in action as defined in scenario S_2. It is possible to observe that *higher response time* and *lower response time* violations trigger the coarse-grained strategy deploying a new VM to host an object's replica when the *higher response time* is reached. Meanwhile, M-CaMid releases the VM when the *response time* is below *lower response time*. The creation and releasing of VMs occur in a single cloud.

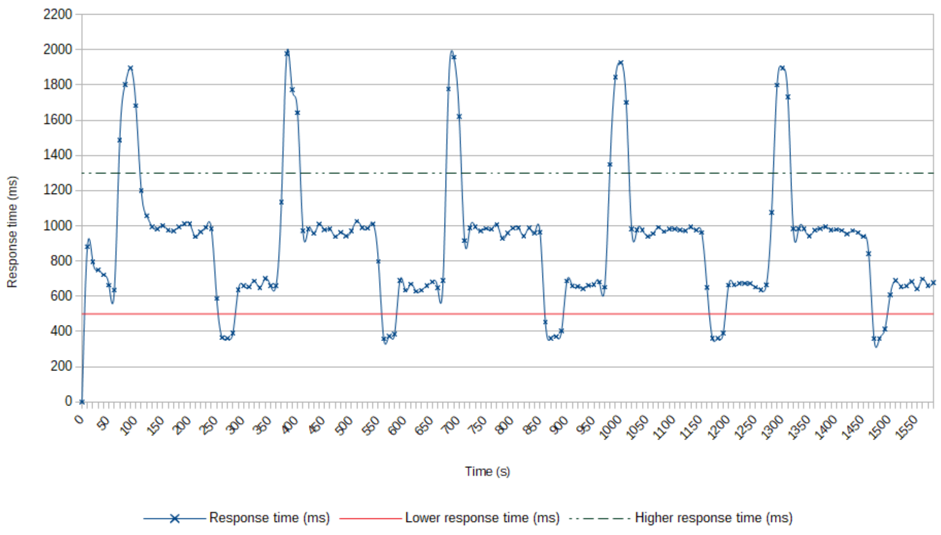
The null hypothesis $H0_1$ proposes no significant difference between the mean response time in S_1 and the mean response time in S_2. In scenario S_1, the meantime between the abnormal event detection and the response time reestablishment was $182s$ with a standard deviation of $4.472s$, while scenario S_2, it was 110 with a standard deviation of 7.071.

Figure 29 – *Response time* with elasticity management (single cloud).

(a) Coarse-grained elasticity.



(b) Fine-grained elasticity.



Source: author (2021).

The null hypothesis $H0_1$ is rejected in the statistical t-test since the result is significant at $p < 0.05$: t-value is 19.243, and p-value is < 0.00001. Thus, the experiment can assert a significant performance gain when M-CaMid manages a multi-cloud application. Furthermore, the effect size for the t-test is Cohen's $d = 12.17$, expressing a hugely significant difference.

In scenario S_3, Figure 29b, M-CaMid steps in the cloud replicating a remote object in an underused VM belonging to the same cloud. In this scenario, it is possible to note the mean response time remains above *higher response time* for a shorter period than in scenario S_2 ($22s$ with a standard deviation of $4.4721s$, and $110s$ with a standard deviation of $7.071s$, respectively). This difference happens because starting a new VM with a coarse-grained management strategy takes longer than finding an existing underused running VM and replicating the remote object.

However, the existence of an underused VM is necessary to execute the fine-grained strategy.

The null hypothesis $H0_2$ proposes no significant difference between the mean response time in S_2 and the mean response time in S_3. In scenario S_2, the meantime between the abnormal event detection and the response time reestablishment was $182s$ with a standard deviation of $4.472s$, while scenario S_2, it was $110s$ with a standard deviation of $7.071s$.

The null hypothesis $H0_2$ is rejected in the statistical t-test since the result is significant at $\alpha = 0.05$: t-value is 19.243, and p-value is < 0.00001. Thus, the experiment asserts a significant performance gain when M-CaMid manages a multi-cloud application. Furthermore, the effect size for the t-test is Glass's delta $\delta = 16.1$, expressing a very significant difference.

The null hypothesis $H0_3$ says no significant difference between mean response time in scenarios S_2 e S_5. The experiment intends to reject this hypothesis to show that the fine-grained management level takes less time even in the multi-cloud domain when compared to the coarse-grained level in a single cloud domain. In Scenarios S_2 and S_5, M-CaMid takes $110s$ ($standarddeviation = 7.071$) and $36s$ ($standarddeviation = 5.477$) to execute the coarse-grained and fine-grained levels, respectively.

The statistical t-test of $H0_3$ results in the hypothesis rejection with $t - value = 18.5$ and $p - value < 0.00001$. The result is significant at $p < 0.05$. It concludes that even if the object replication occurs in another cloud, the taken time to perform the replication is less than deploying a new VM in the same single cloud. The t-test effect size is Cohen's $d = 11.7$, ratifying the null hypothesis rejection.

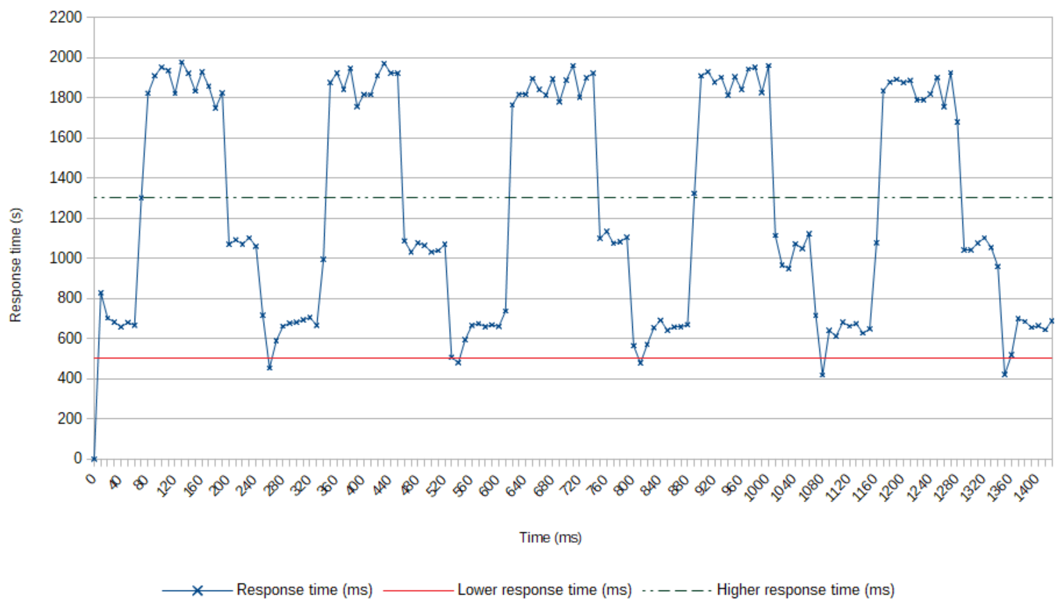
The same comparison between Scenarios S_4 and S_5 is analysed. The null hypothesis $H0_4$ argues no significant difference when M-CaMid cross management performs both management levels in a multi-cloud domain. The t-test result rejects the null hypothesis, presenting a significant difference between the samples at $p < 0.05$ with $t - value = -6.532$, $p - value = 0.000091$. Once more, the result validates that deploying a new VM costs more than replicating a remote object, even in the multi-cloud domain. The Cohen's d effect size reinforces the rejection with $d = 2.8$.

By comparing the behaviour of coarse-grained level in the single cloud (Figure 29a) and multi-cloud (Figure 30a) domains (scenarios S_2 and S_4), it is possible to note that it performs better in a single cloud with *reaction time* of $110s$ (*standard deviation* = 7.072) and $122s$ (*standard deviation* = 8.365) in scenarios S_2 and S_4, respectively, i.e., the coarse-grained management in a single cloud is 10.909% faster than in a multi-cloud environment. This difference has its origin in three facts: different cloud technologies have distinct deployment methods, the VM size differs from cloud to cloud technologies, and the time taken to request, analyse and select another cloud domain. The chosen cloud domain also spends time searching and assigning resources to perform the best-fit approach to meet the new demand.

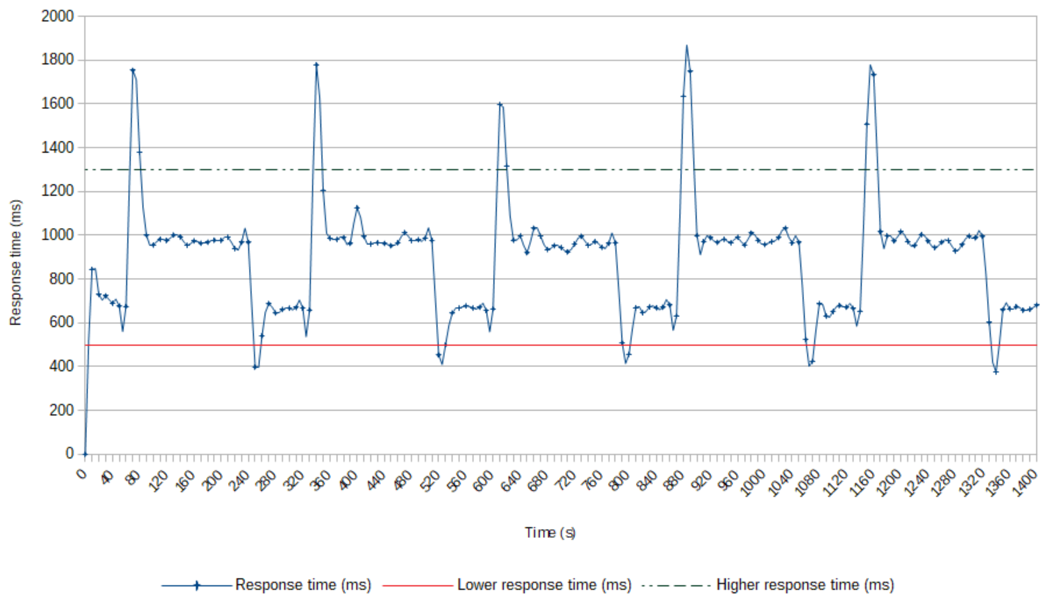
The statistical t-test of $H0_5$ says that there is no significant difference between scenarios S_2 and S_4 results in the hypothesis rejection with $t - value = -2.449$ and $p - value = 0.02$. The result is significant at $p < 0.05$ with a significance level $\alpha = 0.05$. $H0_5$ is rejected with an

Figure 30 – *Response time* with elasticity management (multi-cloud).

(a) Coarse-grained elasticity.



(b) Fine-grained elasticity.



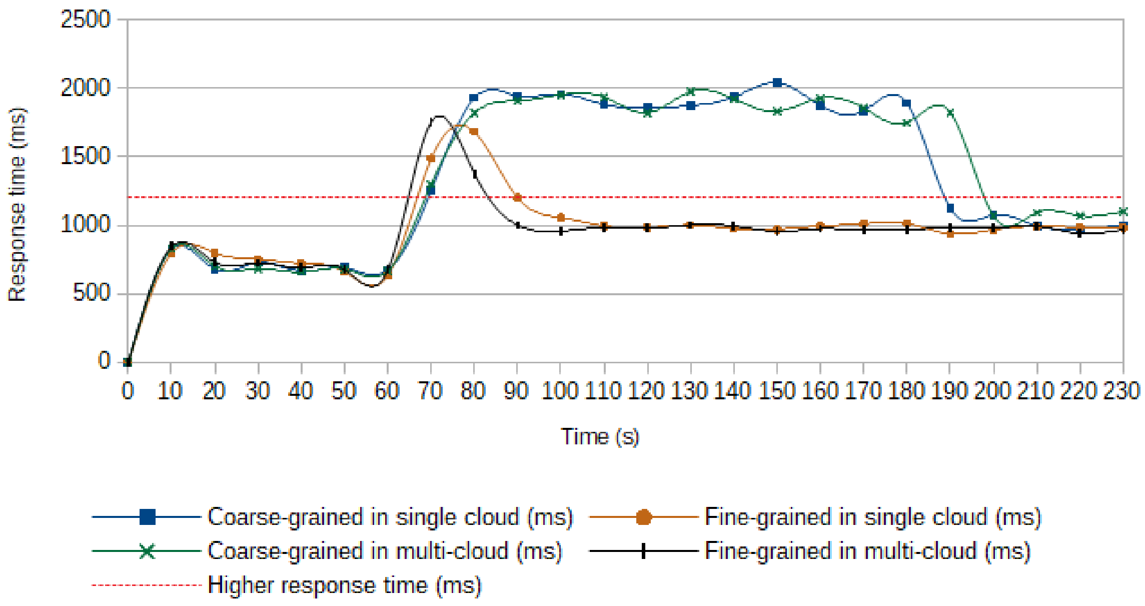
Source: author (2021).

effect size is Cohen's $d = 1.549$, ratifying the null hypothesis rejection. The difference between S_2 and S_4 is because there are delays related to the multi-cloud communication and the cross management actions performed in the chosen cloud domain. However, the effect size is much smaller compared to the other t-tests.

A similar comparison between the performance of the fine-grained level in single (Figure 29b) and multi-cloud (Figure 30b) environments (scenarios S_3 and S_5) shows that it performs 80% better in the single one. Also, this difference is because of the network delay and cross management operations delay in the target cloud. These values are absolutes regardless of the domains. Thus, the experiment rejected the hypothesis $H0_6$. t-value is -4.42719 and p-value is 0.001103. The result is significant at $p < 0.05$ with a Cohen's d impact size of $d = 2.8$.

Figure 31 compares the *response time* behaviours in the different scenarios, zooming in a eight minutes interval. In this figure, it is possible to observe how fast M-CaMid cross management reacts when the application is overloaded underloaded, i.e., the response time becomes higher than the *higher response time* and less than the *lower response time*.

Figure 31 – Response time behaviour with elasticity management at different scenarios.



Source: author (2021).

As expected, the fine-grained approach is faster than the coarse-grained one, whatever the environment. The performance of each method is better in the single cloud than in the multi-cloud environment because of network and management delays performed in the case of two single cloud negotiation and management (multi-cloud domain). Table 7 summarises the null hypothesis statistical t-test values.

Table 7 – Experiment t-tests results.

Hypothesis	Null hypothesis	*t-value*	*p-value*	Cohen's d
$H0_1$	rejected	19.243	< 0.00001	12.17
$H0_2$	rejected	19.243	< 0.00001	16.1
$H0_3$	rejected	18.5	< 0.00001	11.7
$H0_4$	rejected	-6.532	< 0.00001	12.162
$H0_5$	rejected	-2.449	0.02	1.549
$H0_6$	rejected	-4.42719	0.001103	2.8

Source: author (2021).

It is worth observing that although rejection of the null hypotheses $H0_5$ and $H0_6$, Cohen's impact sizes are too small compared to the other ones. It means that both null hypotheses t-test are much closer to no rejection than the other hypotheses analysis results.

5.5 Resource Management

One of the M-CaMid goals is the rational usage of infrastructure resources through the cross management method. Resource management experiment assesses if M-CaMid reaches its goals by observing how management leverages infrastructure to maximise resource usage.

It is expected M-CaMid leverages underused VMs (fine-grained management - fg) instead of creates new VMs (coarse-grained control - cg). In this scenario, there are underused VMs in cloud domain which M-CaMid must analyse the possibility of object replication instead of new VMs deployments. The resource usage is evaluated through metrics related to VM processor usage.

The experiment evaluates the amount of VMs and the CPU percentage used by the server-side application in two scenarios: when M-CaMid cross management is configured with coarse-grained (S_1) and fine-grained (S_2) levels. Results of both scenarios are compared to decide which one is more efficient. Furthermore, the experiment calculates the efficiency of M-CaMid using infrastructure resources EFF (Equation (1)).

5.5.1 Hypotheses

Based on the previous description, The experiment can define its hypotheses. Thus, null hypotheses advocate no difference between the M-CaMid cross management strategies regarding resource usage:

$H0 : CPU\_usage_{cg} \cong CPU\_usage_{fg}$

Alternative hypotheses argues that fine-grained level uses better available resources than the coarse-grained level:

$H1 : CPU\_usage_{cg} < CPU\_usage_{fg}$

5.5.2 Metrics, Parameters and Factors

The metric adopted to assess resource usage efficiency (EFF) is the proportion of resources used and resources available – Equation (1). The resource usage is measured in CPU percentage with value in a 0..1 range. Furthermore, CPU usage is related to the mean application's response time. Even if the CPU usage is at 1 (100%), the average response time can be within the pre-defined limits. It means that, probably, the application is using the CPU as much as possible. On the other hand, if the average response time increases out of bounds while the CPU usage remains at 1, it means a performance depreciation. The mean response time (RT) is measured in seconds.

$$EFF = \frac{\sum CPU\_usage}{\sum CPU\_available} \tag{1}$$

The system parameters are constants. Thus, we keep the exact configuration of VMs, cloud management configurations, and hardware specifications (Section 5.4.4). The cloud domain is simulated in private environments to avoid distortions on measuring caused by the Internet's speed fluctuations. Besides, available underused resources are set with a small CPU workload (≈ 0.25).

The workload parameters (factors) vary to analyse the resource usage by M-CaMid while it steps in the environment. Section 5.5.4 describe in details the workload. The workload parameters are the following:

Number of clients - NC : number of simultaneous users (client-side instances); and

Cross management - CM : it is set to coarse-grained and fine-grained management levels ($CM = cg, fg$).

5.5.3 Treatment

The treatment applied to this experiment is the M-CaMid cross management approach that manages the elasticity at different granularity levels. The reference treatment is the coarse-grained level in a single cloud domain, while the treatment is the fine-grained level in a single cloud domain.

M-CaMid cross management fine-grained level impacts on the resource usage are assessed by comparing to the coarse-grained level. Coarse-grained management is the usual elasticity level adopted in cloud providers.

The experiment's control object is a multi-cloud distributed application whose server-side is a remote object that implements the Fibonacci sequence in two situations. M-CaMid is set to coarse-grained level ($CM = cg$) in a single cloud. The experiment submits the object control to a workload to observe its behaviour as the workload varies and triggers cross management.

The experimental object is the same as the control object but with M-CaMid management configured to fine-grained management. The experiment also submits the same workload as the control object. Both objects behaviours are observed at a single cloud.

5.5.4 Experimental Design

The experiment system follows the same configuration as defined in Section 5.5.4. Each VM is configured with one CPU core and $1GB$ of memory. Three VMs were deployed to execute the experiments: a server-side application and its replicas. One of the VMs is the available underused resource (AR), with a Gaussian distribution whose average workload of 0.25 and a standard deviation of 0.05.

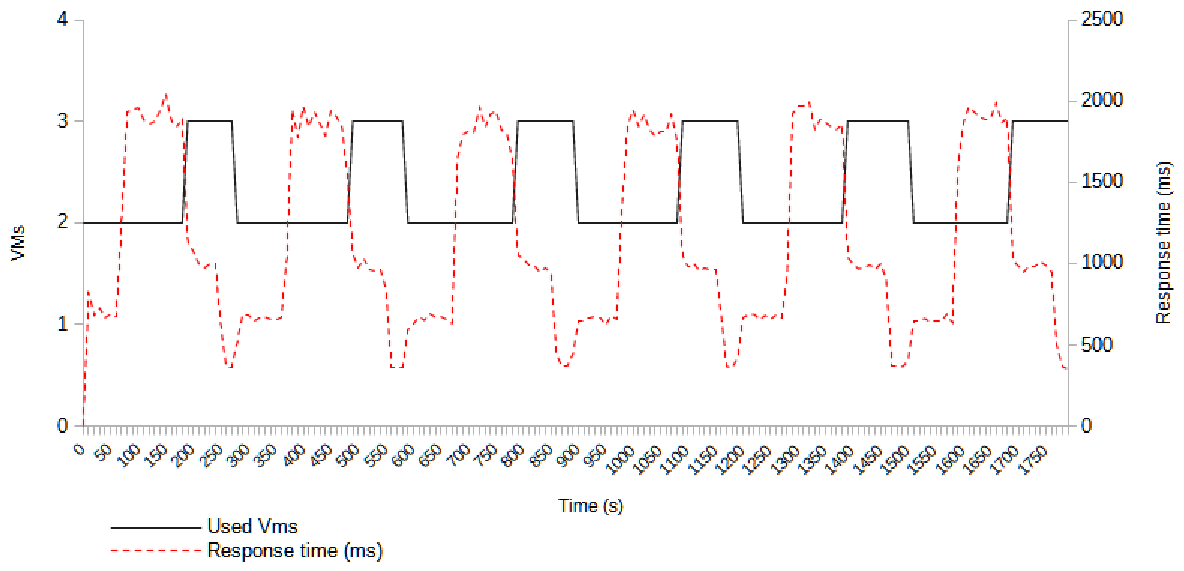
The number of clients (NC) increases and decreases to generate different workloads to trigger cross management control to step in the environment and observe the relation between used resources and available resources.

The experiment starts with 100 clients invoking remote objects. Each client continuously invokes a remote object in time intervals of a Gaussian distribution whose mean value and standard deviation are $50ms$ and $12.5ms$, respectively. Additionally, two thresholds were defined to trigger cross management. If the *response time* becomes higher than $1300ms$ (*highest response time*), new resource is necessary.

5.5.5 Analysis

The analysis discusses the resource usage during the experiment by comparing it with the application response time measurement in scenarios S_2 and S_3. Figure 32 shows resource usage by M-CaMid coarse-grained management. The graph overlaps the measurements, showing when M-CaMid add a new node, the application response time decreases. Similarly, when the response time is under the lower limit, the node is put away, increasing the response time again but still within limits.

Figure 32 – Node usage in coarse-grained strategy.

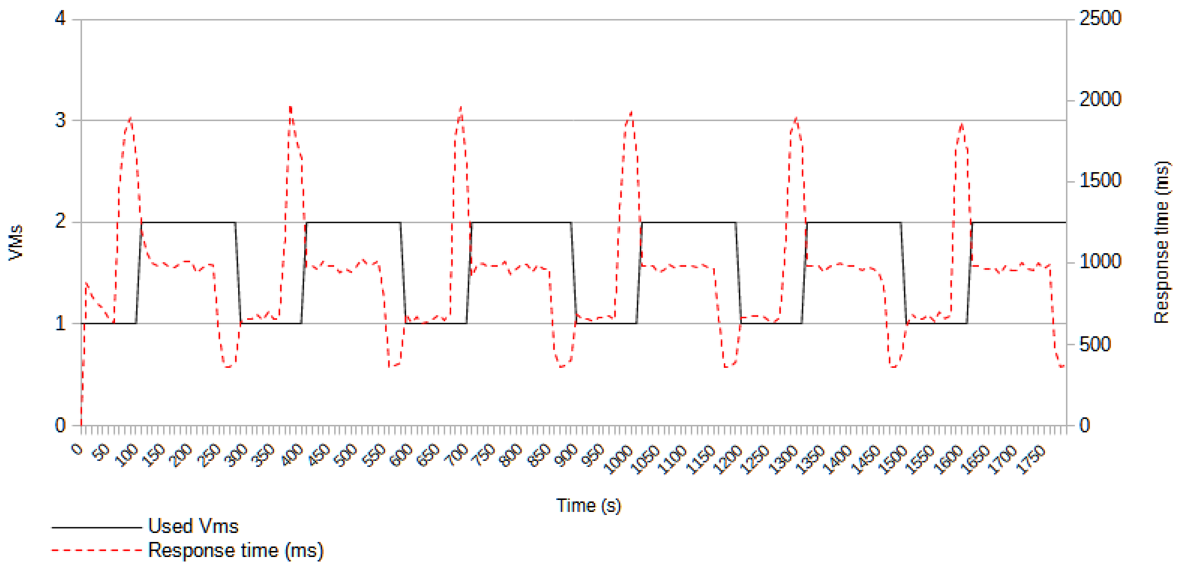


Source: author (2021).

The same behaviour occurs when M-CaMid executes fine-grained management. However, the reaction time is shorter, and the amount of the used node is smaller. Figure 33 shows used nodes by fine-grained strategy along the experiment time. A fine-grained approach takes advantage of underused resources.

Regarding CPU usage, Figure 34 shows available CPU percentage (dashed line) and used CPU by M-CaMid coarse-grained strategy. Note that there are no utilising resources as occurs in traditional infrastructure elasticity. Instead, it deploys new resources regardless of underused resource availability. On the other hand, M-CaMid fine-grained strategy leverages underused resources to avoid resource-wasting, as shown in Figure 35. M-CaMid keeps a rational number

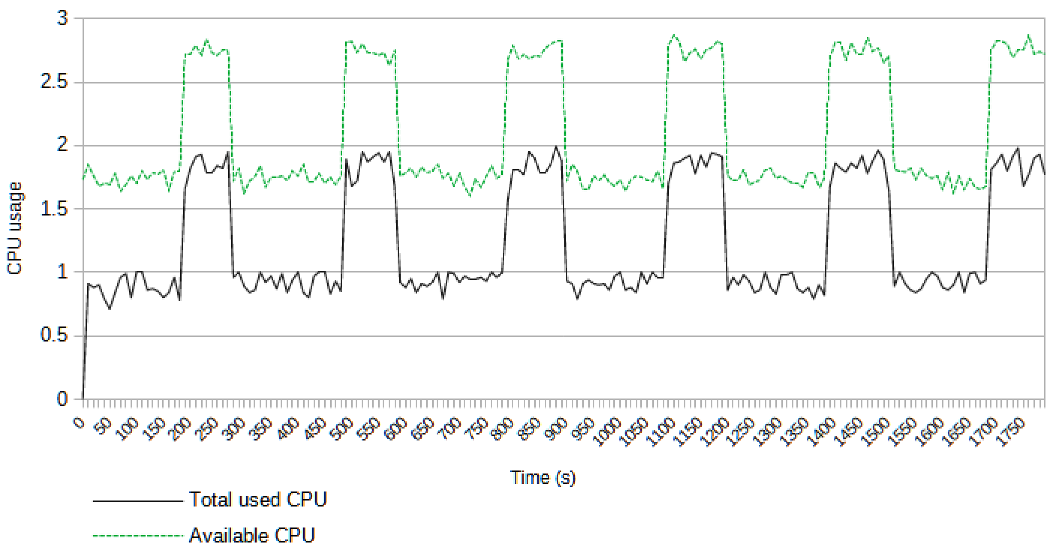
Figure 33 – Node usage in fine-grained strategy.



Source: author (2021).

of nodes whenever possible.

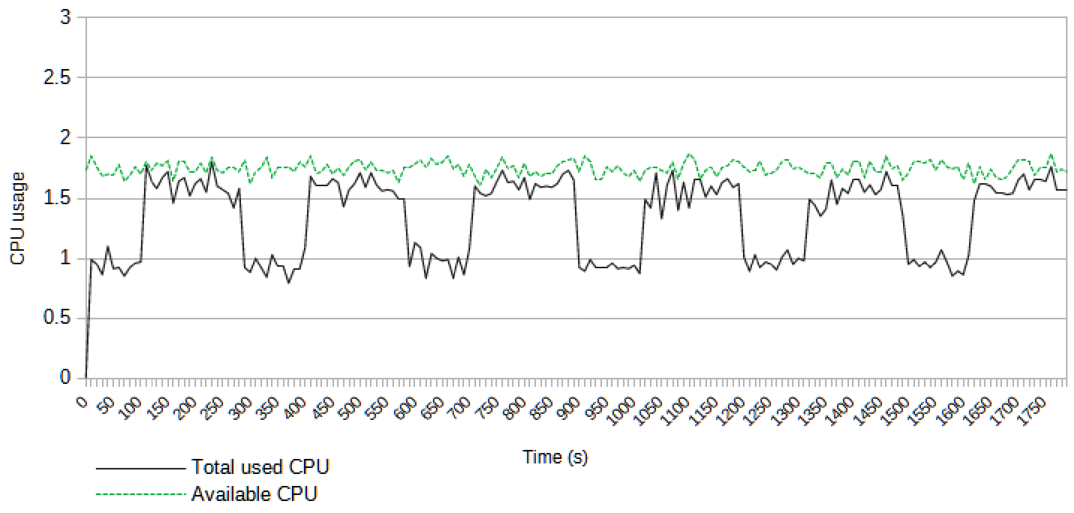
Figure 34 – CPU usage in coarse-grained strategy.



Source: author (2021).

The CPU usage average in both strategies is presented in Figure 36. It compares the total used and total not used CPU by the server-side application, considering coarse-grained and fine-grained cross management levels. Coarse-grained wastes about 0.532 of 3 CPUs, and fine-grained does not use only 0.08 from 2 VMs. The coarse-grained efficiency is $EFF = 0.194$ and fine-grained has $EFF = 0.954$. Table 8 summarises the results of rational usage of CPU resources.

Figure 35 – CPU usage in fine-grained strategy.



Source: author (2021).

Table 8 – CPU usage efficiency (EFF) in coarse and fine-grained management strategies (scenarios S_1 and S_2, respectively).

Scenario	**Adaptation strategy**	**Underused CPU ratio**	**Used CPU ratio**	**Available CPU ratio**	***EFF***
S_2	cg (coarse-grained)	2.204	0.532	2.736	0.194
S_3	fg (fine-grained)	0.08	1.656	1.736	0.954

Source: author (2021).

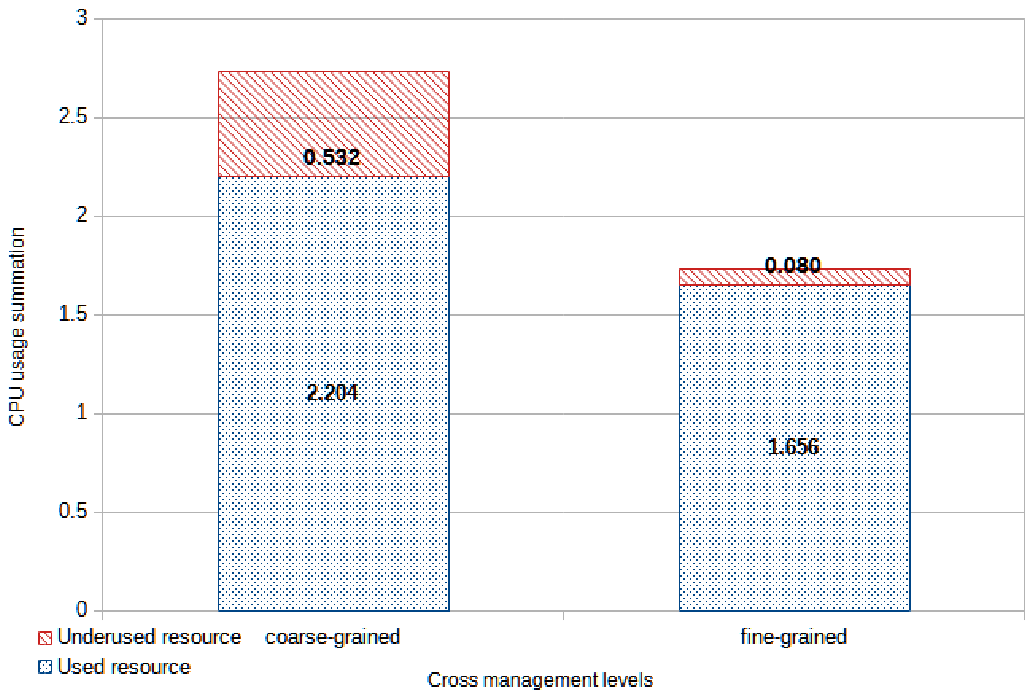
With these results, the null hypothesis $H0$ can be rejected because the result is significant at $p < 0.05$ with $t - value = -6.833$ and $p - value = 0.000067$. The effect size Cohen's $d = 4.49$. Thus, the experiment shows the efficiency of M-CaMid using cloud resources is satisfactory.

5.6 Performance overhead

This experiment assesses the overhead introduced by M-CaMid in the multi-cloud application performance and its infrastructure. It intends to observe the impact on invocations' response time when the experiment submits a multi-cloud application to a variable workload. This investigation also defines levels of workload supported by M-CaMid.

As outcomes, it is expected M-CaMid manages remote objects and their infrastructures during the multi-cloud application execution without significant overhead introduction. The overhead can be expressed by the metric response time measured when the client-side application invokes a successively remote object (RO). This metric evaluates the overhead over the multi-cloud application. The infrastructure overhead is expressed by VM CPU usage (VM). The experiment occurs in two scenarios: M-CaMid with cross management turned OFF (OFF) and turned ON (ON).

Figure 36 – CPU percentage used by M-CaMid cross management.



Source: author (2021).

5.6.1 Hypotheses

In the experiments null hypotheses ($H0_{RO..VM}$), there are a significant difference among response times ($H0_{RO}$) and significant differences among CPU usage ($H0_{VM}$) when M-CaMid management is OFF or ON.

$H0_{RO}\ : RO_{OFF} < RO_{ON}$

$H0_{VM}\ : VM_{OFF} < VM_{OFF}$

On the other hand, the alternatives hypotheses ($H1_{1..3}$) are:

$H1_{RO}\ : RO_{OFF} \cong RO_{ON}$

$H1_{VM}\ : VM_{OFF} \cong VM_{OFF}$

5.6.2 Metrics, Parameters and Factors

As aforementioned, response time and CPU usage are metrics to access how M-CaMid management affects multi-cloud applications and VM performances, respectively, by introducing workload overhead.

The system parameters are fixed, keeping the exact configuration of VM, cloud management configurations, and hardware specifications (Section 5.4.4). The experiment carries only on the cloud domain since it is expected that VMs have similar behaviour regardless of the cloud technology. Furthermore, M-CaMid switches off parameters for triggering actions because the experiment aims to stress the system to observe the elements' behaviours.

Some factors affect the M-CaMid performance. The workload parameters (factors) vary to analyse M-CaMid behaviour. They are:

1. **Number of clients**: number of simultaneous users (client-side instances);
2. **Number of invocations**: number of successive remote invocations performed by each user;
3. **Cross management**: it can be turned on or turned off.

Table 9 shows factors and their respective values assumed in the experiment.

Table 9 – Factors and values of M-CaMid overhead.

Factor	Values
Simultaneous clients	$100, 200, 300, 400, 500, 600, 700, 800, 900, 1000$
Invocation interval	Gaussian distribution: mean value $100ms$ standard deviation $12.5ms$
Cross management	turned off and turned on

Source: author (2021).

5.6.3 Treatment

The treatment applied to this experiment is the M-CaMid approach that monitors and controls the multi-cloud application environment. The absence of treatment is called cross management off (OFF), while the treatment is called management on (ON).

M-CaMid cross management impacts are analysed by comparing response time measures of a multi-cloud application under different workload levels with (ON) or without (OFF) treatment.

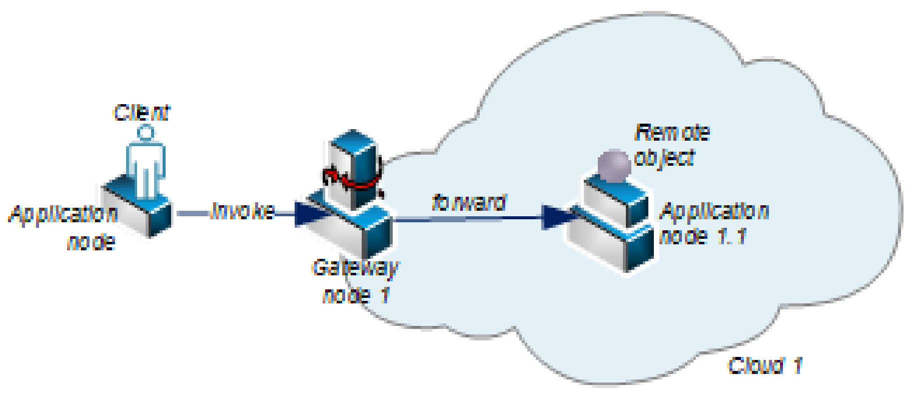
The experiment's control object is a multi-cloud application whose server-side is a remote object that implements the Fibonacci sequence without M-CaMid cross management (OFF). The experiment submits the object control to a workload to observe the system behaviour as the workload varies.

The experimental object is the same as the control object but with M-CaMid management turned on (ON). The experiment also submits the same workload as the control object. Both objects behaviours are observed at a single cloud domain since the same behaviour is expected regardless of cloud technology.

5.6.4 Experimental Design

Figure 37 shows the basic configurations used in the experiment. The experiment system comprises the client-server application implemented atop M-CaMid and OpenNebula (Moreno-Vozmediano et al., 2012) as cloud infrastructure. The OpenNebula was configured with an IaaS manager (frontend) and a host for running VMs. Frontend runs on a machine core i5 2.44GHz with 4 GB of memory, and the cloud host (where the frontend deploys VMs) is an octa-core Xeon 2.93 GHz machine with 20 GB of memory. Each VM is configured with one CPU core and 1 GB of memory.

Figure 37 – M-CaMid scenario for M-CaMid overhead experiment.



Source: author (2021).

Both M-CaMid's monitors (node and cloud) are configured to gather monitoring data in an interval of 10s. Common real-world monitoring systems time intervals include 1, 5, 15, and 60 minutes. The experiment set M-CaMid time interval as 10s because its goal is to observe M-CaMid's behaviour under stressful workloads. Furthermore, in the experimental object, the thresholds (lowest and highest response times) are set with values out of the range to prevent M-CaMid management from adding or removing infrastructure resources and biasing the experiment result.

The experiment submits M-CaMid to a growing workload. The number of users increases, assuming values as described in Table 9. The M-CaMid behaviour is observed through response time (milliseconds) and throughput measurements (request per second).

5.6.5 Analysis

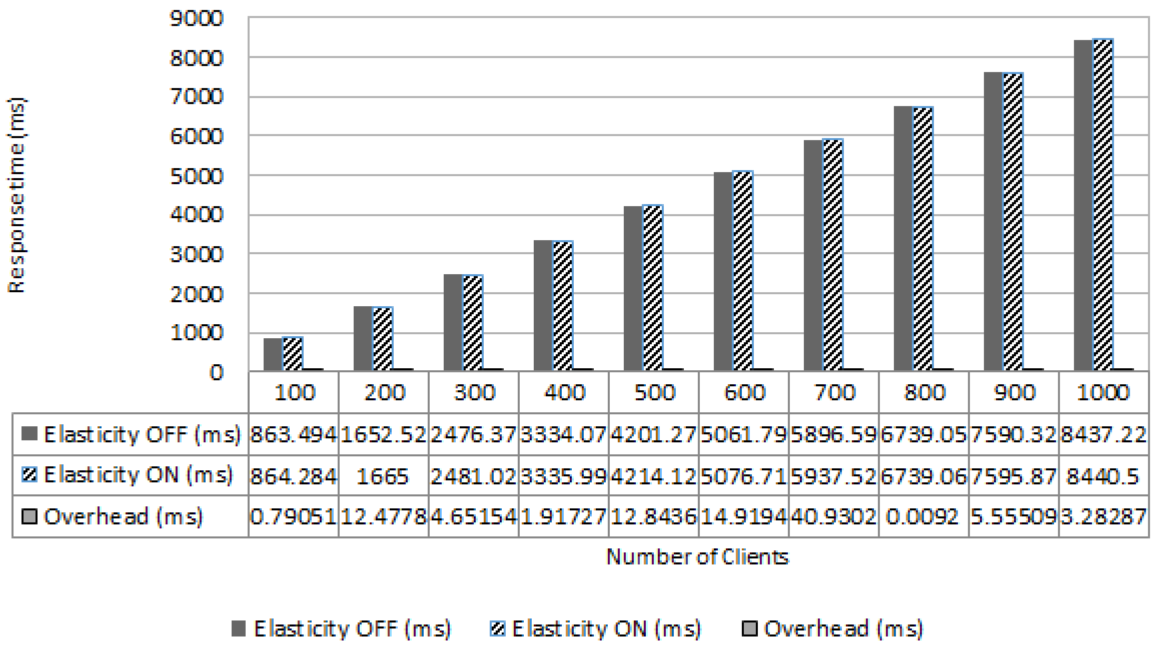
The study analysis compares the experimental object's measurements and control object trials to reject the null hypotheses. The experiment assesses the M-CaMid performance impact over the multi-cloud application response time.

Section 9 described the applied factor values. A t-test compares the mean values of two data sets: with M-CaMid management turned off and with M-CaMid management turned on. T-test analyses if their differences are not significant.

The experiment goal is not to reject the null hypothesis since the experiment intends to show no significant difference between the means. The null hypothesis claims the mean response times are equivalents with M-CaMid management ON than with M-CaMid management off ($H0_{RO} : RO_{OFF} < RO_{ON}$). The experiment calculates the mean response time with the two M-CaMid configurations for each number of clients. The assumed test's confidence is 95%.

Figure 38 shows the mean response time when the M-CaMid management is ON and OFF. As the application is the same in both cases, the difference between mean response times is the overhead of the M-CaMid management. The mean overhead is 3.067% even when the application server is submitted to a heavy workload, i.e., 1000 clients. The lowest and highest overheads are 1.626% and 5.603%, respectively. In this way, it is intuitive to observe that cross management has a minimal impact on the multi-cloud application's performance.

Figure 38 – M-CaMid overhead impact over application response time.



	100	200	300	400	500	600	700	800	900	1000
■ Elasticity OFF (ms)	863.494	1652.52	2476.37	3334.07	4201.27	5061.79	5896.59	6739.05	7590.32	8437.22
▨ Elasticity ON (ms)	864.284	1665	2481.02	3335.99	4214.12	5076.71	5937.52	6739.06	7595.87	8440.5
□ Overhead (ms)	0.79051	12.4778	4.65154	1.91727	12.8436	14.9194	40.9302	0.0092	5.55509	3.28287

Source: author (2021).

Regarding the statistical test, Table 10 shows the results. The t-value is -0.00745 and the p-value is 0.497065. The result (mean difference) is not significant at $p < 0.05$. The null hypothesis can not be rejected.

Table 10 – T-student test statistics values.

α	Difference of means	p-value	t-value
0.05	-8.85	0.497065	-0.497065

Source: author (2021).

5.7 CONCLUDING REMARKS

This chapter presented the evaluation of M-CaMid. The primary goal was to show the performance of M-CaMid supporting the execution of multi-cloud distributed applications. A systematic approach to performance evaluation was adopted to accomplish the experiments. Initially, some steps were defined to conduct the experiments: goals, hypotheses, scenarios, metrics, parameters and factors, and experiment design. Finally, results were presented and discussed through statistical tests.

6 Related Work

Nowadays, many challenges related to multi-cloud computing are still open, despite efforts from the academy and industry. Since multi-cloud computing has issues in an extensive area, many solutions support different issues, features and viewpoints, such as interoperability, dynamic management, fault tolerance, elasticity, multi-tenancy, and security. Many research efforts in cloud computing focus on elasticity management from different points of view regarding multi-cloud application management. This work discusses some researches based on the achieved goals: a) ***multi-cloud awareness***: cloud middleware is aware of more than one cloud and provides transparency to upper application layers to deal with different cloud management technologies; b) ***architecture***: cloud middleware architecture meets requirements of cloud and multi-cloud architectures; and c) ***cross management***: cloud middleware looks at elasticity management from different dimensions, such as granularity levels (coarse and fine-grained elasticity) and coverage (single cloud and multi-cloud domains). Furthermore, some relevant solutions are discussed despite not being classified into the goals mentioned earlier.

6.1 Multi-Cloud Awareness

Multi-Cloud awareness requires cloud interoperability. Interoperability approaches can range from standards, API and blueprints to middleware systems. Many organisations proposed cloud standards for addressing the interoperability, such as e.g., Distributed Management Task Force (DMTF) (Bankston et al., 2012), Open Cloud Computing Interface (OCCI) (Edmonds and Nyrén, 2011) and Organisation for the Advancement of Structured Information Standards (OASIS) (Palma and Spatzier, 2013). However, there is a little advance towards a universal standard because the proposed standards tend to be divergent. Even when standards are adopted, cloud providers are still trying to differentiate their services by using proprietary technologies (Nelson, 2009). Furthermore, choosing one standard may imply on a *lock-in problem* of standard (Nogueira et al., 2016). Another problem with standard adoption is the fact they are limited to the IaaS level (Lewis et al., 2013) because of the greater diversity of platforms and applications. On the other hand, some initiatives proposed libraries to abstract developers from the heterogeneous cloud technologies by providing common API. JClouds$^{®}$ (Apache, 2011) is the most popular library. As standards, current libraries do not comprise all cloud technologies. Usually, middleware-based solutions use libraries to deal with heterogeneous cloud technologies.

Blueprint proposals were introduced for driving the development of multi-cloud systems. Bernstein et al. (2009) suggest a set of protocols and formats to address interoperability and portability issues. A visionary approach, named Cloud Blueprint, was proposed by Papazoglou and van den Heuvel (2011), whose goal is to break the cloud's monolithic architecture and let developers choose and intermix the lowest costs and best quality PaaS/SaaS offerings by cloud providers. To realise this approach, they propose three language components: a definition language, constraint language and manipulation language. The method converts all cloud stack layers into general-purpose commodities to attract partners and build a proper cloud ecosystem.

Middleware-based solutions are usually based on brokers to tackle the multi-cloud interoperability. Altocumulus (Maximilien et al., 2009a,b) is one of the seminal works on middleware for cloud interoperability. The focus of Altocumulus solving the application interoperability problem within and between clouds. Altocumulus facilitates deployment, management, and migration of applications across multiple clouds. They propose a meta-model for defining cloud-agnostic representations for portability, deployment and configuration to realise these facilities. The middleware platform is composed of components in three categories: tools, API and knowledge. The tools are a dashboard for user interaction and monitoring activities, an image repository, and a core APIs tester for testing multi-cloud APIs. Although the work refers to elasticity, there is no reference to how it is realised. Besides, replication is not present as a facility.

mOSAIC (Petcu et al., 2011, 2013) is a project to support resource interoperability and portability at both federation and multi-cloud models. It is a complete and integrated solution for developing, deploying, and managing applications for multi-cloud environments. mOSAIC can support interoperability thanks to its deployable broker that provides integration between cloud providers, executing operations, such as cloud search and selection. Although mOSAIC supports multi-cloud models, its management acts at the infrastructure level only and application scalability management is not designed.

Sotiriadis and Bessis (2016) propose Inter-Cloud Mediation Service, a multi-cloud bridge system for developers to build collaborative environments for distribution and management of cloud services using the RESTFul protocol. The interoperability between cloud providers is by a broker based on OCCI-compliant open standards. Its architecture is based on the OpenStack cloud platform (RackSpace, 2010). The multi-cloud collaboration occurs through the cloud service registration that allows cloud users to register, search, and use cloud services. The Inter-cloud mediator is the core component that offers cloud APIs to users. Furthermore, the Complex Event Processing Service (CEP) allows developers to define rules and patterns to react to event occurrences. Inter-Cloud Mediation Service is multi-cloud aware through OpenStack cloud platform, limiting its interoperability.

M-CaMid adopts the broker pattern to allow multi-cloud collaboration between distributed application components. Its primary purpose is to provide multi-cloud elasticity for supporting distributed application performance and availability.

As a design decision, M-CaMid adopts third-party API libraries and the most popular cloud platforms OpenNebula (Moreno-Vozmediano et al., 2012) and OpenStack (RackSpace, 2010) since they deliver integration services with many public clouds and other platforms. Since M-CaMid should comply with the cloud awareness requirement, API libraries were the best alternative to avoid the lock-in problems besides meeting the client-centric approach (Bouzerzour et al., 2020). M-CaMid architecture uses the JClouds library to interact with different underlying infrastructure cloud technologies.

6.2 Middleware Architecture

Due to multi-cloud environment complexity, middleware architecture must be taken as a relevant aspect. As discussed before, cloud and multi-cloud domains components interact in different ways. While a single cloud has centralised management behaviour, a multi-cloud environment comprises independent members with similar roles. Most solutions adopt the centralised architecture. Intercloud (Buyya et al., 2010; Calheiros et al., 2012), mOSAIC (Petcu et al., 2011, 2013), soCloud (Paraiso et al., 2016) and *elastic component model* Pokahr and Braubach (2015) are examples of centralised architecture.

In the decentralised architecture, the control is distributed, where the processes are logically similar. CloudSNAP (Mondéjar et al., 2013) is a decentralised web deployment platform that promises to move any web application to the cloud. It allows a decentralised deployment environment and distributed mechanisms, like load balancing, fault tolerance, dynamic activation, persistence and replication. The CloudSNAP architecture is based on P2P overlays and interception patterns. However, its distributed architecture, management of P2P infrastructure and application lifecycle follow a centralised logic. Although CloudSNAP claims to provide elasticity, the paper only presents no elasticity method.

Zhang et al. (2013) adopt the proxy approach to design a fully distributed management for a large scale server environment in the cloud. The decentralised control solution is designed to offer both overload protection and resource efficiency for the back-end servers while achieving service differentiation based on SLA. Its management defines two control roles: the *sub-controller* and *proxy dispatcher*. The management is spread among all servers and proxies, which perform resource management and traffic control automatically and independently. The management audits local resource consumption such as CPU capacity, memory usage, bandwidth availability etc. Its main component is the Proxy Dispatcher that regulates the request admission rate to each back-end server. The approach focuses on the multi-cloud domain at the infrastructure level.

Similarly, CloudSNAP (Mondéjar et al., 2013) is a P2P-based platform that allows a management environment with a set of distributed mechanisms for deployment, load balancing, fault tolerance, dynamic activation, persistence and replication. Although the architecture is distributed, the management of P2P infrastructure and application lifecycle follows a centralised logic. M-CaMid hybrid architecture allows management autonomy between cloud domains (decentralised architecture). However, inside the cloud domain, M-CaMid follows a centralised architecture, enabling a full management view.

CLAMBS (Cross-Layer Multi-Cloud Application Monitoring and Benchmarking as-a-Service) (Alhamazani et al., 2019) intends to monitor individual application components deployed in a multi-cloud environment. Monitoring agents are deployed in clouds to monitor applications hosted in many cloud layers (IaaS, PaaS and SaaS). Furthermore, CLAMBS allows centralised and distributed architecture configuration. Although CLAMBS is a monitoring solution, it is worth mentioning because of its architectural similarity with M-CaMid regarding cross-layer and cross-cloud integration.

Join the First Idle Queue (JFIQ) (Desmouceaux et al., 2021) introduces a unified, centralised-monitoring-free architecture for supporting load balancing and auto-scaling, promising to reduce operational overhead and increase response time performance. JFIQ monitoring system is distributed in application nodes, allowing decision making *in loco*. The application-level elasticity management decides to accept or reject a request according to its workload. In case of rejection, the request is redirected to the next node. JFIQ design supports only the single cloud domain.

The hybrid architecture of M-CaMid meets the principles' requirement of multi-cloud computing. Inside a single cloud domain, its monitoring system follows a decentralised architecture, while its controller is centralised, providing integrated actions to reconfigure the environment components. The decentralised monitoring system distributes workloads among nodes and notifies the centralised controller about undesired events. Regarding multi-cloud domain, M-CaMid adopts a decentralised architecture since cloud domains are independents. Cooperation through message exchange realises their integration.

6.3 Cross Management

Elasticity dimensions can characterise the management approach. This work considers two dimensions: the elasticity abstraction level, i.e. its granularity (coarse-grained and fine-grained), and coverage (cloud and multi-cloud domains). Furthermore, it is worth mentioning the implementation approach, which can be programmatic. Elasticity implementation is built in the code or the middleware layer, providing more transparency to developers and decoupling the application from infrastructure.

Elasticity programming can be applied to parallel programming through languages and programming platforms. Dustdar et al. (2012) propose the Simple-Yet-Beautiful Language (SYBL) to implement elasticity directives and runtime functions inside cloud-based applications. Directives allow specifying runtime properties related to resources, quality and cost. Developers can implement elasticity by using SYBL runtime functions to get and set properties, check a program's cost, and execute external scripts. Cloudine framework (Galante and Erpen De Bona, 2015) is a framework for programming elasticity in scientific applications. The approach explores cloud elasticity, in which its elasticity controller is embedded in the application source code. The controller allows the allocation and deallocation of resources by the application without external mechanisms. Its monitoring system uses data from VM and application events. However, the elasticity control is embedded in the source code, and it takes place at IaaS level.

Middleware platforms can provide API for programming elasticity. ElasticRMI (Jayaram, 2013) is an object-oriented middleware that enables application developers to configure elasticity parameters to handle remote method invocation workloads. Developers can combine resource utilisation metrics and fine-grained application-specific information to manage elasticity. The middleware provides a high-level programming framework that addresses elasticity at the level of classes and objects. ElasticRMI middleware targets distributed applications that run on top of IaaS layer. Its runtime system handles all low-level mechanics for instantiating elastic objects, monitoring and balancing workload, and adding and removing additional objects if necessary. The interaction between middleware and the IaaS layer's is realised by Apache Mesos. However, all configuration of elasticity metrics is defined inside the code at development time.

JCLOUDSCALE (Zabolotnyi et al., 2015) is a Java-based middleware for building elastic IaaS applications. It aims is to facilitate developers to implement cloud applications as local, multi-threaded applications without even being aware of the cloud deployment and elasticity. The middleware takes over the management of the underlying physical distribution. The IaaS resource management is accomplished through a declarative programming model. JCLOUDSCALE uses aspect-oriented programming (via AspectJ) to inject remoting and cloud management code into target applications. Hence, all elasticity policy configurations should also be programmed, adding complexity to development. Any change in the elasticity policy requires recompiling the code. JCLOUDSCALE manages the application elasticity at object (application) and IaaS level. The middleware takes over remote objects monitoring, forwarding information into a complex event processing (CEP) engine. All management is centralised on the client-side. Although JCLOUDSCALE supports application portability between cloud providers, it is not designed to interoperate in multi-cloud environments.

The elasticity programming approach requires development experience, besides being code invasive. However, it can be viable for parallel programming since parallelism depends on how implementations are designed. Although some of the above solutions presented here are middleware-based, they provide API to programming elasticity inside the application, being code invasive and without management transparency.

Middleware platforms allow transparent elasticity management to developers. Some middleware-based solutions focus on abstracting underlying management complexity. JadexCloud (Braubach et al., 2011) is an infrastructure for developing, deploying and managing distributed applications for running on top of private IaaS layers. Applications are composed of agent-like autonomous entities interacting via services. JadexCloud supports building scalable and robust enterprise applications through high modular and independent acting modules that can be replaced or restarted if unexpected errors occur. Its main principle is distribution transparency. Although it claims to manage dynamically distributed applications, many functions are still manually executed in this version.

Abbadi (2011b,c) proposes a conceptual and hybrid solution for middleware systems. Abbadi adopts a cloud stack composed of three layers: the physical layer, which contains the features of the physical infrastructure of the cloud (servers, storage and network); the virtual layer, which represents virtual resources hosted at the physical layer; and the application layer that includes user applications. In this structure, the middleware serves as "glue" between resources of the many cloud layers, whilst its primary responsibility is to provide a set of self-managed services for the layers mentioned above. These services automatically manage the applications running in the cloud and all their dependencies and include: adaptability, resilience, scalability, availability, reliability, security services. Although modelling cross-layer management, the solution does not target the multi-cloud domain, and it is limited to a conceptual model.

Yangui et al. (2011) propose a framework that adds facilities for monitoring and configuring service-based applications deploying them into a cloud using a scalable micro-container. The framework allows fine-tuning monitoring and reconfiguration at different granularities levels of application. The services don't need to be designed to be monitored or reconfigured. Instead, they are encapsulated in a new composite added of non-functional service of monitoring and reconfiguration delivering the same functionality as the original. This component transformation is applied dynamically by adding new features for monitoring and configurations. Although fine-grained management, it is not clear about at which level the elasticity takes place.

Pokahr and Braubach (2015) propose an autonomic management system (Monitor-Analyse-Plan-Execute over a shared Knowledge (MAPE-K) approach) based on a vision of elastic component-based applications. This vision is supported by a component-based programming model for allowing developers to specify application functionality in a cloud-enabled way and a runtime environment that supports the specification and automatic management of non-functional attributes, such as application response times. The solution is based on Jadex (Pokahr et al., 2010), which supports monitoring non-functional requirements and provides load balancing and instance management services. In other words, scalability occurs at the component level. The underlying idea is that an application has to offer non-functional properties that can be measured at runtime and which will be used to validate if and to what degree the user-defined non-functional requirements are satisfied. The authors do not detail if elasticity also takes place at IaaS level, i.e., whether the solution interoperates with IaaS layer or not.

Suprex (Aslanpour et al., 2017) is a cost-aware scaling mechanism based on the MAPE-K concept, focusing on the execution phase of the execution phase. The approach mitigates the oscillation of under and over-provisioning by maximising the billing time usage of a surplus VM. For example, since the VM is billed per hour, one surplus VM running for $2h10min$ is charged as $3h$ wasting $50min$ of usage. The approach sends the surplus VM to quarantine instead of releasing it. In case of application overload in this period, the VM is recovered to meet the new demand. It also avoids the delay caused by the instantiation of a new VM. Suprex is an executor mechanism in the MAPE-K cycle. The proposal's architecture is centralised in the application provider, where incoming requests are forwarded to the web application. Its elasticity management takes place at IaaS level.

As presented before, Join the First Idle Queue (JFIQ) (Desmouceaux et al., 2021) introduces a unified, centralised-monitoring-free architecture for supporting load balancing and auto-scaling. The application-level elasticity management decides to accept or reject a request according to its workload. In case of rejection, the application instance redirects the request to the next one.

Load balancer organises these instances in a round-robin structure. It sends a request to the first application instance that can accept or reject according to its workload. The last instance decision is indicative of upscaling of application instances. M-CaMid monitors application components in a similar way. However, the decision making control is centralised in the M-CaMid gateway, which has a broad view of the cloud environment and can extend elasticity to the multi-cloud environment.

Mazidi et al. (2021) propose a MAPE-K control loop for autonomic resource scaler to minimising costs for users. It presents a weighted ensemble prediction model using single prediction models structured in a decision tree. Cloud applications are structured in layers: web, app and database. Which one with its own particular VM configuration. Its monitoring system covers both infrastructure and application layers. However, scaling actions occurs only at the infrastructure.

Recently, OS virtualisation, namely containerisation, has been popular in industry and academy. Elastic container platforms, such as Kubernetes (Cloud Native Computing Foundation, 2014), Docker in Swarm mode (Hykes, 2013) and Apache Mesos (Hindman et al., 2011), support containers' lifecycle management and container-level elasticity. For example, Kubernetes (Cloud Native Computing Foundation, 2014), the most popular platform, orchestrates distributed applications taking care of their scaling and failover. However, Kubernetes does not provide application-level services, such as middleware platforms. Another limitation is incompatibilities among optional and alternative Kubernetes features that cause system dependencies (Truyen et al., 2020).

Containers platforms can be classified as infrastructure solutions. Furthermore, container-level elasticity services are limited to a single cloud domain in most platforms. Kratzke (2017) propose a control process to scale containers across public and private clouds. Its elasticity service executes container migration that can be added to the execution phase of a MAPE-K loop. However, the management is limited to the container level, neglecting aspects of applications running in the container and underlying infrastructure (cloud VMs). In the multi-cloud context, M-CaMid supports elasticity by accomplishing replication besides integrating application and infrastructure levels.

Previous elasticity management solutions target the single cloud environment. Most of them can be extended to the multi-cloud domain because they allow application management independent of cloud providers. However, their designs coverage only the cloud domain, lacking communication and integration among clouds.

Regarding multi-cloud elasticity, few works proposed solutions with a multi-cloud coverage. soCloud (Paraiso et al., 2016) is a service-oriented component-based PaaS for managing portability, elasticity, provisioning, and high availability across multiple clouds. soCloud operates elasticity at IaaS and PaaS levels, in the same manner, referring to resources through abstractions. The scaling of resources occurs at most at IaaS level, on which the middleware can allocate resources for the application as needed. It also monitors the resources looking for the underused ones and re-size them as necessary. Monitoring information comes from the application's parts deployed in many clouds and is analysed by the centralised component *Workload Manager*. Multi-cloud elasticity is used to deal with fail-overs. In case of the application fails, the *Controller* component decides to instantiate a new application. soCloud considers cloud and multi-cloud domains as management granularity levels, where a cloud is in the coarse-grained level, and a VM is in the fine-grained level.

Parlavantzas et al. (2018) propose an approach for dynamically deploying applications over multiple clouds to increase the application's owner profits. This approach continuously optimises the deployment generating a deployment plan that optimises gain under the current runtime conditions and deciding when and how to reconfigure the entire application. Unlike M-CaMid, this approach does not support distributed applications, and continuous optimisation takes advantage of only the VM elasticity level.

Kirthica and Sridhar (2018) propose a residue-based horizontal scaling approach for multi-cloud provisioning. The technique splits the resource demand to be met by external clouds, which each one replies to, providing the maximum available IaaS resource. The Cloud Inter-operation Tool (CIT) requests resources to clouds one by one until there is no residue of demand. CIT is a provider-centric approach which deals with rational resource usage of hardware resources by fitting the best resource demand slice in cloud hardware offering. M-CaMid has a similar system to manage resources. However, M-CaMid supports more precise management through fine-grained management. Moreover, M-CaMid is client-centric, shifting control to third parties.

Table 11 summarises cloud middleware solutions and their characteristics.

Table 11 – Summary of related work.

Middleware	Interoperability	Architecture	Cross management		
			Implementation	**Coverage**	**Elasticity abstraction level**
Autocumulus (Maximilien et al., 2009a,b)	☑	Centralised	n/a[1]	Multi-Cloud	n/a[1]
mOSAIC (Petcu et al., 2011, 2013)	☑	Centralised	Middleware	Multi-Cloud	IaaS
Inter-cloud Mediation Service (Sotiriadis and Bessis, 2016)	☑	Centralised	Middleware	Multi-Cloud	IaaS
Zhang et al. (2013)	☑	Decentralised	n/a[1]	Multi-Cloud	n/a[1]
CloudSNAP (Mondéjar et al., 2013)		Centralised	Middleware	Cloud	IaaS
CLAMBS (Alhamazani et al., 2019)	☑	Hybrid	Middleware	Multi-Cloud	n/a[1]
JFIQ (Desmouceaux et al., 2021)		decentralised	Middleware	Cloud	IaaS
SYBL (Dustdar et al., 2012)		n/a[1]	Programming	Cloud	Application
ElasticRMI (Jayaram, 2013)		Centralised	Programming	Cloud	Application
JCloudScale (Zabolotnyi et al., 2015)		Centralised	Programming	Cloud	Application
Cloudine framework (Galante and Erpen De Bona, 2015)		Centralised	Programming	Cloud	IaaS
JadexCloud (Braubach et al., 2011)		Centralised	Middleware	Cloud	Application
Abbadi (2011b,c)		n/a[1]	Middleware	Cloud	IaaS
Yangui et al. (2011)		Centralised	Middleware	Cloud	Application
Pokahr and Braubach (2015)		Centralised	Programming	Cloud	Application
Suprex (Aslanpour et al., 2017)		Centralised	Middleware	Cloud	IaaS
Mazidi et al. (2021)		Centralised	Middleware	Cloud	IaaS
Intercloud (Buyya et al., 2010; Calheiros et al., 2012)	☑	Centralised	Middleware	Multi-Cloud	IaaS
soCloud (Paraiso et al., 2016)	☑	Centralised	Middleware	Multi-Cloud	IaaS
Parlavantzas et al. (2018)	☑	Centralised	Middleware	Multi-cloud	IaaS
CIT (Kirthica and Sridhar, 2017, 2018)	☑	Centralised	Middleware	Multi-Cloud	IaaS
M-CaMid (de Morais et al., 2013; de Morais and Rosa, 2017)	☑	Hybrid	Middleware	Multi-Cloud	IaaS and application

[1] not applicable

Source: author (2021).

6.4 Discussion

Many works propose solutions for the management of distributed applications in a multi-cloud environment. However, most of them concern with tackling the interoperability issue or resource provisioning and management without considering multi-cloud elasticity mechanisms.

Multi-Cloud application management solutions provide mechanisms to support the elasticity in the cloud and multi-cloud domains. However, most elasticity management proposals target the cloud elasticity (Mondéjar et al., 2013; Jayaram, 2013; Zabolotnyi et al., 2015; Galante and Erpen De Bona, 2015). Furthermore, elasticity requires timely and efficient management. The presented works manage elasticity at infrastructure-level providing coarse-grained elasticity (Mondéjar et al., 2013; Galante and Erpen De Bona, 2015; Aslanpour et al., 2017; Buyya et al., 2010; Calheiros et al., 2012; Paraiso et al., 2016), or an application-level providing fine-grained elasticity (Jayaram, 2013; Zabolotnyi et al., 2015; Pokahr and Braubach, 2015; Yangui et al., 2011; Abbadi, 2011b,c).

Recently, OS virtualisation, namely containerisation, has been popular in industry and academy. Elastic container platforms, such as Kubernetes (Cloud Native Computing Foundation, 2014), Docker in Swarm mode (Hykes, 2013) and Apache Mesos (Hindman et al., 2011), support containers' lifecycle management and container-level elasticity. For example, Kubernetes (Cloud Native Computing Foundation, 2014), the most popular platform, orchestrates distributed applications taking care of their scaling and failover. However, Kubernetes does not provide application-level services, such as middleware platforms. Another limitation is incompatibilities among optional and alternative Kubernetes features that cause system dependencies (Truyen et al., 2020).

Container-based virtualisation is a growing approach that reduces infrastructure deployment and overhead, but most container platforms are still limited to the infrastructure layer and a single cloud domain. M-CaMid cross management integrates application and infrastructure layers management through monitoring and control mechanisms that support multi-cloud elasticity. M-CaMid elasticity method can also be applied to containers once they are infrastructure resources managed by developers.

Regarding multi-cloud elasticity, few solutions were proposed (Buyya et al., 2010; Calheiros et al., 2012; Paraiso et al., 2016; Kirthica and Sridhar, 2018). However, none of them provides fine-grained elasticity to multi-cloud applications.

Regarding implementation, some solutions are code invasive since they propose to developers to program management tasks or insert annotations inside the code (Jayaram, 2013; Zabolotnyi et al., 2015; Galante and Erpen De Bona, 2015; Pokahr and Braubach, 2015). Other ones suggest a middleware layer to abstract underlying infrastructure aspects (Mondéjar et al., 2013; Braubach et al., 2011; Pokahr and Braubach, 2015; Abbadi, 2011b,c; Yangui et al., 2011; Aslanpour et al., 2017; Buyya et al., 2010; Calheiros et al., 2012; Paraiso et al., 2016).

M-CaMid is a multi-cloud middleware architecture that proposes multi-cloud application management taking advantage of cloud elasticity of different cloud providers. In other words, M-CaMid extends the elasticity to the multi-cloud environment. The management architecture follows a hybrid control: for cloud domain management, it is centralised, and for the multi-cloud environment, the control is decentralised to allow independence between M-CaMid cloud domain managers. Furthermore, M-CaMid integrates the infrastructure and application layers through the cross management to provide precise elasticity, extending this integration to a multi-cloud environment.

From the developer perspective, we advocate that a multi-cloud middleware must *a*) be based on the multi-cloud model, and its broker component must be independent of cloud providers; *b*) support distributed communication between components of a multi-cloud application regardless of the scope (node, cloud or multi-cloud); *c*) support the management at application and infrastructure layers, allowing the integration of management mechanisms aiming more precise resource management; *d*) deliver elasticity at infrastructure and application level (coarse-grained and fine-grained, respectively); *e*) transparently extend the elasticity to a multi-cloud domain.

6.5 Concluding Remarks

This chapter presented the related work about single cloud and multi-cloud application management. It presented solutions to provide interoperability between cloud providers as an essential requirement for multi-cloud computing. Next, proposals related to the multi-cloud application management were discussed regarding their main characteristics related to architecture, coverage and elasticity management. Finally, the works were discussed, highlighting the unique contributions of M-CaMid.

7 Conclusion

This chapter summarises this thesis. First, it presents the thesis's general thoughts. Next, it highlights the thesis' unique contributions. Then, it discusses the future works.

7.1 General Considerations

M-CaMid can reconfigure dynamically and automatically the system structure (application and infrastructure) in response to undesirable events that threaten the application's performance requirements. M-CaMid cross management that considers two dimensions: *(a)* vertical: integrates application and infrastructure management layers (cross-layer), allowing more precise and transparent management, through the multi-grained elasticity: coarse-grained elasticity at VM level, and fine-grained elasticity at application component level; and *(b)* horizontal: integrates domains of a multi-cloud environment (cross-domain): node, cloud and multi-cloud.

After an exploratory literature review, some challenges related to multi-cloud application development and management were identified and motivated this work. It observed several initiatives on cloud and multi-cloud management. However, most of the solutions target cloud infrastructure management, neglecting applications running on top of the IaaS infrastructure. Some solutions take into account application requirements but execute elasticity at the IaaS level, a coarse-grained elasticity. Furthermore, few works tackle the multi-cloud elasticity. However, anyone proposes an elasticity cross management. Finally, any one of the works presented a middleware solution based on the classical DOC middleware architecture.

Besides the thesis motivation, the literature review produced the thesis background. The basic concepts of the cloud and multi-cloud computing paradigm were introduced, discussing the many definitions and terms that identified the essential elements and defined the multi-cloud taxonomy. The application's components distribution in this scenario can range from a single VM to many cloud platforms.

Some basic principles were discussed since they were introductory statements to multi-cloud design. M-CaMid design treated four seminal principles: decentralisation, interoperability, dynamic behaviour, and adaptability. Furthermore, this work assumed layered architectural style as a principle following the DOC middleware model. These principles drove the M-CaMid design.

Decentralisation and interoperability are tightly related principles because cloud technologies are independent and heterogeneous.

Decentralisation is a multi-cloud principle that states all clouds are at the same level of resource control and fully independent. Accepting this principle implies M-CaMid needs to adopt a multi-cloud management approach to meet the multi-cloud aware requirement. Furthermore, interoperability means that many clouds can interoperate. M-CaMid must meet multi-cloud aware requirements since cloud technologies are very heterogeneous.

Dynamic behaviour and adaptability are principles supported by the elasticity benefit. However, these principles require timely and precise automatic resource management. Therefore, M-CaMid must implement multi-cloud elasticity to meet them.

M-CaMid met all requirements by proposing a cross management approach that established two management dimensions: vertical and horizontal. In the vertical view, the architecture is composed of vertical layers, namely, infrastructure, distribution and service layers, following the DOC layers stack (layered architectural style as principle). The infrastructure layer serves as a wrapper to underlying cloud and communication mechanisms; the distribution layer implements distribution transparencies. The service layer provides the traditional naming service and security service. Also, M-CaMid extended the DOC model by adding a fourth transverse layer in charge of managing multi-cloud distributed application and their underlying IaaS infrastructure.

The horizontal view considered the M-CaMid distribution throughout the multi-cloud environment: node, cloud and multi-cloud domain. The management components were distributed according to their responsibilities in the respective field.

The M-CaMid transparently took benefit of cloud facilities and allowed multi-cloud management at application and infrastructure levels. The cross management considered *how*, and *where* to scale out and in both application parts and infrastructure. *How* refers to the levels of granularity for the elasticity: fine-grained and coarse-grained. In the first one, the elasticity is applied to the remote object, while in the second one, the elasticity is employed to VM. *Where* referred to the domain where the elasticity can take place: into the current cloud domain or outside that.

The experiment's design of M-CaMid follows a systematic approach. The investigation adopts a method to evaluate the M-CaMid cross management. As a result, it is expected the experiments show how the cross management strategies adapt the environment timely and efficiently to keep the distributed application working properly.

Some experiments were carried out aiming to assess if M-CaMid reaches the required goals. All experiments were performed under a scenario where M-CaMid was submitted to unpredictable peaks of workloads. The experiment's goals were to observe how M-CaMid supports the high availability and scalability of multi-cloud distributed applications.

Three experiments assessed M-CaMid: *(i)* management reaction time evaluated the automatic control of M-CaMid on dealing with undesired environment's behaviour; *(ii)* elasticity resource management assessed the rational usage of infrastructure (virtual machines) considering coarse- and fine-grained management; and *(iii)* M-CaMid overhead assessed the overhead introduced by M-CaMid node monitor on the application performance.

M-CaMid presented satisfactory results improving the management of distributed applications in a multi-cloud environment. Statistical tests ratified the experimental results.

Finally, state of the art is discussed, listing some related initiatives to address the multi-cloud challenges. The related work shows the unpublished contributions of the M-CaMid proposal. The

M-CaMid approaches the multi-cloud elasticity through a cross-management strategy, allowing a more precise and transparent multi-grained management: fine-grained management at the application component level and coarse-grained control at the VM level.

7.2 Contributions

M-CaMid realised unique contributions presented in the Chapter 1 of this thesis:

1. M-CaMid architecture extended the classical DOC model by the introduction of the management layer to place the new functionalities related to cloud and multi-computing;

2. Cross management allowed a timely and efficiently adaptation of multi-cloud distributed applications, giving the ability to M-CaMid to manage resources in a decentralised and interoperable environment; and

3. In a highly dynamic and adaptable multi-cloud environment, multi-cloud elasticity together cross management took advantage of many cloud technologies' elasticity service thanks to M-CaMid meets the multi-cloud aware ability.

Furthermore, this work has accomplished others contributions. First, ongoing and future research directions can adopt the full implementation of multi-cloud middleware architecture as a basis for research and technological development. M-CaMid architecture provides elementary distribution middleware services, enabling middleware to be a tool "foundation" toward future work in many multi-cloud areas. Second, the principles' definition contributes to drive multi-cloud middleware projects and identify essential middleware requirements. Decentralisation, openness and interoperability, dynamic behaviour and adaptability principles were introduced as fundamentals statements driving middleware development. Finally, the extensive adoption of remoting patterns in designing all aspects of the middleware architecture reinforces M-CaMid architecture as a development foundation for multi-cloud middleware.

7.3 Limitations and Lessons Learned

During experiments, some threats were identified. Limited hardware capacities hampered experiments with more complex distributed applications and environments. It also limited cloud technologies adoption in two types: OpenNebula and OpenStack. Furthermore, shared resources in cloud infrastructure can bias experiments' results since cloud hardware configuration for deployment is transparent to the client. M-CaMid development can draw some lessons: (i) interoperability standards is a facility for multi-cloud middleware development. However, it is so far to be fully reached since cloud providers design their services aiming at exclusivity; (ii) Managing multi-cloud elasticity requires new load balancing strategies, mainly the distributed ones. Forwarding requests across cloud domains to equalise workload among cloud domains is a complex task because of the autonomous nature of cloud providers (decentralisation principle);

(iii) the broad purpose of cloud computing may change middleware design goals as new areas emerge, such as Internet of things (IoT), mobile cloud computing, big data computing and edge computing; and (iv) many elasticity management solutions focus on specific approaches, neglecting elasticity dimensions and wasting its full benefits.

7.4 Research Directions

M-CaMid is a not finished project. Many significant advances in the multi-cloud area are happening, such as adopting Artificial Intelligence (AI) and deep learning technologies to improve adaptation, automatic deployment and load balancing. Also, M-CaMid must extend to support decentralised cloud management in mobile, edge computing, and the increasing use of containers.

As future work, we intend to improve M-CaMid cross management with predictive strategies for adaptation. Thus, using hybrid adaptation approaches (reactive and predictive) can support the multi-cloud elasticity by applying the best method based on management knowledge.

Reactive and predictive methods can also improve load balancing. The main goal is to ally distributed load balancing techniques and adaptation methods since M-CaMid follows the decentralised multi-cloud principle.

Another future work is to introduce cost-aware aspects to cross management decision making because M-CaMid can deal with different cloud providers with varying price policies.

Finally, M-CaMid cross management intends to be not technology restrictive, initially extending its support to containers technologies. Container is an evolving technology that needs attention and has challenges in the multi-cloud computing area.

REFERENCES

Abbadi, I. (2011a). Middleware services at cloud virtual layer. In Conference on Computer and Information Technology (CIT), 2011 IEEE 11th International, pages 115 –120.

Abbadi, I. M. (2011b). Middleware services at cloud application layer. In A. Abraham, J. L. Mauri, J. F. Buford, J. Suzuki, and S. M. Thampi, editors, Advances in Computing and Communications, volume 193 of Communications in Computer and Information Science, pages 557–571. Springer Berlin Heidelberg. 10.1007/978-3-642-22726-4_58.

Abbadi, I. M. (2011c). Self-managed services conceptual model in trustworthy clouds' infrastructure. In Workshop on Cryptography and Security in Clouds. IBM, Zurich. http://www.zurich.ibm.com/ cca/csc2011/program.html.

Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., and Merle, P. (2018). Elasticity in Cloud Computing: State of the Art and Research Challenges. IEEE Transactions on Services Computing, **11**(2), 430–447.

Alhamazani, K., Ranjan, R., Jayaraman, P. P., Mitra, K., Liu, C., Rabhi, F., Georgakopoulos, D., and Wang, L. (2019). Cross-Layer Multi-Cloud Real-Time Application QoS Monitoring and Benchmarking As-a-Service Framework. IEEE TRANSACTIONS ON CLOUD COMPUTING, **7**(1), 48–61.

Amin, M. B., Khan, W. A., Awan, A. A., and Lee, S. (2012). Intercloud message exchange middleware. In Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication, ICUIMC '12, pages 79:1–79:7, New York, NY, USA. ACM.

Aoyama, T. and Sakai, H. (2011). Inter-cloud computing. Business & Information Systems Engineering, **3**(3), 173–177.

Apache (2011). Jclouds. `https://jclouds.apache.org/`. accessed: 30-05-2015.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2009). Above the clouds: A berkeley view of cloud computing. Technical report, University of California at Berkeley.

Aslanpour, M. S., Ghobaei-Arani, M., and Nadjaran Toosi, A. (2017). Auto-scaling web applications in clouds. J. Netw. Comput. Appl., **95**(C), 26–41.

Assis, M. R. M. and Bittencourt, L. (2016). A survey on cloud federation architectures: Identifying functional and non-functional properties. Journal of Network and Computer Applications, **72**, 51–71.

Azeez, A., Perera, S., Gamage, D., Linton, R., Siriwardana, P., Leelaratne, D., Weerawarana, S., and Fremantle, P. (2010). Multi-tenant soa middleware for cloud computing. In Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, pages 458 –465.

Bankston, J. K., Burkhart, N., Cohen, J., Davis, J., and Ericson, G. (2012). Cloud infrastructure management interface - common information model (cimi-cim). Technical Report DSP0264, Distributed Management Task Force - DMTF.

Bernstein, D., Ludvigson, E., Sankar, K., Diamond, S., and Morrow, M. (2009). Blueprint for the intercloud - protocols and formats for cloud computing interoperability. In Internet and Web Applications and Services, 2009. ICIW '09. Fourth International Conference on, pages 328 –336.

Bernstein, P. A. (1996). Middleware: A model for distributed system services. Commun. ACM, **39**(2), 86–98.

Bessani, A., Correia, M., Quaresma, B., André, F., and Sousa, P. (2011). Depsky: Dependable and secure storage in a cloud-of-clouds. In Proceedings of the Sixth Conference on Computer Systems, EuroSys '11, pages 31–46, New York, NY, USA. ACM.

Bittman, T. (2008). The Evolution of the Cloud Computing Market. url: http://blogs.gartner.com/thomas_bittman/2008/11/03/the-evolution-of-the-cloud-computing-market/. Last access: 12/05/2018.

Bouzerzour, N. E. H., Ghazouani, S., and Slimani, Y. (2020). A survey on the service interoperability in cloud computing: Client-centric and provider-centric perspectives. SOFTWARE-PRACTICE & EXPERIENCE, **50**(7), 1025–1060.

Braubach, L., Pokahr, A., and Jander, K. (2011). Jadexcloud - an infrastructure for enterprise cloud applications. In Proceedings of the 9th German conference on Multiagent system technologies, MATES'11, pages 3–15, Berlin, Heidelberg. Springer-Verlag.

Budnik, U. (2013). Lessons learned from recent cloud outages. http://www.rightscale.com/blog/enterprise-cloud-strategies/lessons-learned-recent-cloud-outages. Last access 07-19-2015.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems-the International Journal of Grid Computing-theory Methods and Applications, **25**(6), 599–616.

Buyya, R., Ranjan, R., and Calheiros, R. (2010). Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In C.-H. Hsu, L. Yang, J. Park, and S.-S. Yeo, editors, Algorithms and Architectures for Parallel Processing, volume 6081 of Lecture Notes in Computer Science, pages 13–31. Springer Berlin / Heidelberg.

Buyya, R., Srirama, S. N., Casale, G., Calheiros, R., Simmhan, Y., Varghese, B., Gelenbe, E., Javadi, B., Vaquero, L. M., Netto, M. A. S., Toosi, A. N., Rodriguez, M. A., Llorente, I. M., Vimercati, S. D. C. D., Samarati, P., Milojicic, D., Varela, C., Bahsoon, R., Assuncao, M. D. D., Rana, O., Zhou, W., Jin, H., Gentzsch, W., Zomaya, A. Y., and Shen, H. (2018). A manifesto for future generation cloud computing: Research directions for the next decade. ACM Comput. Surv., **51**(5), 105:1–105:38.

Calheiros, R. N., Toosi, A. N., Vecchiola, C., and Buyya, R. (2012). A coordinator for scaling elastic applications across multiple clouds. Future Generation Computer Systems, **28**(8), 1350 – 1362. <ce:title>Including Special sections SS: Trusting Software Behavior and SS: Economics of Computing Services</ce:title>.

Carlini, E., Coppola, M., Dazzi, P., Ricci, L., and Righetti, G. (2012). Cloud federations in contrail. In M. Alexander, P. DÃ¢â‚¬â„¢Ambra, A. Belloum, G. Bosilca, M. Cannataro, M. Danelutto, B. Di Martino, M. Gerndt, E. Jeannot, R. Namyst, J. Roman, S. Scott, J. Traff, G. VallÃƒÂ©e, and J. Weidendorfer, editors, Euro-Par 2011: Parallel Processing Workshops, volume 7155 of Lecture Notes in Computer Science, pages 159–168. Springer Berlin Heidelberg.

Celesti, A., Villari, M., and Puliafito, A. (2010a). Design of a cloud naming framework. In Proceedings of the 7th ACM international conference on Computing frontiers, CF ’10, pages 105–106, New York, NY, USA. ACM.

Celesti, A., Tusa, F., Villari, M., and Puliafito, A. (2010b). How to enhance cloud architectures to enable cross-federation. In Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, pages 337–345.

Chauhan, M. A., Babar, M. A., and Benatallah, B. (2017). Architecting cloud-enabled systems: a systematic survey of challenges and solutions. Software: Practice and Experience, **47**(4), 599–644. SPE-15-0007.R3.

Cloud Native Computing Foundation (2014). Kubernetes. `https://kubernetes.io/`. Last acess aug 23 2021.

Coad, P. (1992). Object-oriented patterns. Commun. ACM, **35**(9), 152–159.

Coulouris, G., Dollimore, J., and Kindberg, T. (2013). Distributed Systems: Concepts and Design. Addison-Wesley, 5th edition edition.

Coutinho, E. F., de Carvalho Sousa, F. R., Rego, P. A. L., Gomes, D. G., and de Souza, J. N. (2014). Elasticity in cloud computing: a survey. annals of telecommunications - annales des télécommunications, pages 1–21.

Cuadrado, F., Navas, A., Duenas, J., and Vaquero, L. (2014). Research challenges for cross-cloud applications. In Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on, pages 19–24.

Czaja, L. (2018). Introduction to Distributed Computer Systems - Principles and Features, volume 27. Springer.

de Lemos, R., Giese, H., Müller, H. A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N. M., Vogel, T., Weyns, D., Baresi, L., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Desmarais, R., Dustdar, S., Engels, G., Geihs, K., Göschka, K. M., Gorla, A., Grassi, V., Inverardi, P., Karsai, G., Kramer, J., Lopes, A., Magee, J., Malek, S., Mankovskii, S., Mirandola, R., Mylopoulos, J., Nierstrasz, O., Pezzè, M., Prehofer, C., Schäfer, W., Schlichting, R., Smith, D. B., Sousa, J. P., Tahvildari, L., Wong, K., and Wuttke, J. (2013). Software Engineering for Self-Adaptive Systems: A Second Research Roadmap, pages 1–32. Springer Berlin Heidelberg, Berlin, Heidelberg.

de Morais, T. and Rosa, N. S. (2017). Towards an application level elasticity by middleware. In 2017 IEEE Symposium on Computers and Communications (ISCC), pages 730–735.

de Morais, T., Liberalquino, D., and Rosa, N. (2013). Cloud-aware middleware. In Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on, pages 780–787.

DeNardis, L. (2012). The social stakes of interoperability. Science, **337**(6101), 1454–1455.

Desmouceaux, Y., Enguehard, M., and Clausen, T. H. (2021). Joint monitorless load-balancing and autoscaling for zero-wait-time in data centers. IEEE Transactions on Network and Service Management, **18**(1), 672–686.

Desprez, F., Krämer-Fuhrmann, O., and Yahyapour, R. (2010). Cloud computing - introduction to the special theme. ERCIM News, **2010**(83), 12–13.

Dustdar, S., Guo, Y., Han, R., Satzger, B., and Truong, H.-L. (2012). Programming directives for elastic computing. Internet Computing, IEEE, **16**(6), 72–77.

Edmonds, A. and Nyrén, R. (2011). Open cloud computing interface - core. Technical Report GFD-P-R.183, Open Cloud Computing Interface.

Elhabbash, A., Samreen, F., Hadley, J., and Elkhatib, Y. (2019). Cloud brokerage: A systematic survey. ACM Comput. Surv., **51**(6).

Erl, T., Puttini, R., and Mahmood, Z. (2013). Cloud Computing: Concepts, Technology & Architecture. The Prentice Hall service technology series from Thomas Erl. Prentice Hall.

Esposito, C., Ficco, M., Palmieri, F., and Castiglione, A. (2013). Interconnecting federated clouds by using publish-subscribe service. Cluster Computing, pages 1–17.

Ferrer, A. J., Hernández, F., Tordsson, J., Elmroth, E., Ali-Eldin, A., Zsigri, C., Sirvent, R., Guitart, J., Badia, R. M., Djemame, K., Ziegler, W., Dimitrakos, T., Nair, S. K., Kousiouris, G., Konstanteli, K., Varvarigou, T., Hudzia, B., Kipp, A., Wesner, S., Corrales, M., Forgó, N., Sharif, T., and Sheridan, C. (2012). Optimis: A holistic approach to cloud service provisioning. Future Generation Computer Systems, **28**(1), 66 – 77.

Ferrer, A. J., Marquès, J. M., and Jorba, J. (2019). Towards the decentralised cloud: Survey on approaches and challenges for mobile, ad hoc, and edge computing. ACM Comput. Surv., **51**(6).

Ferretti, S., Ghini, V., Panzieri, F., Pellegrini, M., and Turrini, E. (2010). Qos-aware clouds. In Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, pages 321 –328.

Ficco, M., Tasquier, L., and Martino, B. (2014). Interconnection of federated clouds. In F. Zavoral, J. J. Jung, and C. Badica, editors, Intelligent Distributed Computing VII, volume 511 of Studies in Computational Intelligence, pages 243–248. Springer International Publishing.

Foster, I., Zhao, Y., Raicu, I., and Lu, S. (2008). Cloud computing and grid computing 360-degree compared. In Grid Computing Environments Workshop, 2008. GCE '08, pages 1 –10.

Galante, G. and de Bona, L. (2012). A survey on cloud computing elasticity. In Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on, pages 263–270.

Galante, G. and Erpen De Bona, L. C. (2015). A Programming-level Approach for Elasticizing Parallel Scientific Applications. J. Syst. Softw., **110**(C), 239–252.

Gill, J. (2021). Private cloud is still alive and kicking. https://www.datacenterdynamics.com/en/opinions/private-cloud-is-still-alive-and-kicking. Accessed in 2021-09-16.

Gill, S. H., Razzaq, M. A., Ahmad, M., Almansour, F. M., Haq, I. U., Jhanjhi, N., Alam, M. Z., and Masud, M. (2022). Security and Privacy Aspects of Cloud Computing: A Smart Campus Case Study. Intelligent Automation & Soft Computing, **31**(1), 117–128.

Global Inter-Cloud Technology Forum. (2010). Use cases and functional requirements for inter-cloud computing. Technical report, Telecommunication Technology Comitee. URL http://www.ttc.or.jp/files/8614/1214/5480/GICTF_Whitepaper_20100809.pdf. Last access: 07-22-2015.

Grozev, N. and Buyya, R. (2014). Inter-cloud architectures and application brokering: taxonomy and survey. Software: Practice and Experience, **44**(3), 369–390.

Hamdaqa, M. and Tahvildari, L. (2012). Cloud Computing Uncovered: A Research Landscape. In A. Hurson and A. Memon, editors, Advances in Computers, volume 86 of Advances in Computers, pages 41 – 85. Elsevier.

Han, R., Ghanem, M. M., Guo, L., Guo, Y., and Osmond, M. (2014). Enabling cost-aware and adaptive elasticity of multi-tier cloud applications. Future Generation Computer Systems, **32**, 82 – 98. Special Section: The Management of Cloud Systems, Special Section: Cyber-Physical Society and Special Section: Special Issue on Exploiting Semantic Technologies with Particularization on Linked Data over Grid and Cloud Architectures.

Herbst, N. R., Kounev, S., and Reussner, R. H. (2013). Elasticity in cloud computing: What it is, and what it is not. In J. O. Kephart, C. Pu, and X. Zhu, editors, ICAC, pages 23–27. USENIX Association.

Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R., Shenker, S., and Stoica, I. (2011). 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, pages 295–308, Berkeley, CA, USA. USENIX Association.

Hoffert, J., Schmidt, D. C., and Gokhale, A. (2010). Adapting distributed real-time and embedded pub/sub middleware for cloud computing environments. In Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware, Middleware '10, pages 21–41, Berlin, Heidelberg. Springer-Verlag.

Hohpe, G. and Bobby, W. (2004). Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. The Addison-Wesley Signature Series. Prentice Hall.

Hykes, S. (2013). Docker in Swarm Mode. .

Imai, S., Chestna, T., and Varela, C. (2012). Elastic scalable cloud computing using application-level migration. In Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on, pages 91–98.

Imai, S., Chestna, T., and Varela, C. A. (2013). Accurate Resource Prediction for Hybrid IaaS Clouds Using Workload-Tailored Elastic Compute Units. In Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing, UCC '13, pages 171–178, Washington, DC, USA. IEEE Computer Society.

Imai, S., Patel, P., and Varela, C. A. (2016). Developing Elastic Software for the Cloud, chapter 50, pages 609–627. John Wiley & Sons, Ltd.

Islam, S., Lee, K., Fekete, A., and Liu, A. (2012). How a consumer can measure elasticity for cloud platforms. In Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering, ICPE '12, pages 85–96, New York, NY, USA. ACM.

Jain, R. (1991). The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling. Wiley professional computing. Wiley.

Jayaram, K. R. (2013). Elastic remote methods. In D. Eyers and K. Schwan, editors, Middleware 2013, volume 8275 of Lecture Notes in Computer Science, pages 143–162. Springer Berlin Heidelberg.

Keahey, K., Tsugawa, M., Matsunaga, A., and Fortes, J. A. B. (2009). Sky computing. IEEE INTERNET COMPUTING, **13**(5), 43–51.

Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. Computer, **36**(1), 41–50.

Kertesz, A., Kecskemeti, G., Oriol, M., Kotcauer, P., Acs, S., Rodríguez, M., Mercè, O., Marosi, A., Marco, J., and Franch, X. (2013). Enhancing federated cloud management with an integrated service monitoring approach. Journal of Grid Computing, pages 1–22.

Kirthica, S. and Sridhar, R. (2017). CIT: A Cloud Inter-operation Toolkit to enhance elasticity and tolerate shut down of external clouds. Journal of Network and Computer Applications, **85**, 32 – 46. Intelligent Systems for Heterogeneous Networks.

Kirthica, S. and Sridhar, R. (2018). A residue-based approach for resource provisioning by horizontal scaling across heterogeneous clouds. International Journal of Approximate Reasoning, **101**, 88 – 106.

Kratzke, N. (2017). Smuggling multi-cloud support into cloud-native applications using elastic container platforms. In Proceedings of the 7th International Conference on Cloud Computing and Services Science. SCITEPRESS - Science and Technology Publications.

Leite, A. F., Alves, V., Rodrigues, G. N., Tadonki, C., Eisenbeis, C., and Magalhaes Alves de Melo, A. C. (2016). Autonomic provisioning, configuration, and management of inter-cloud environments based on a software product line engineering method. In Gupta, I and Diao, Y, editor, 2016 INTERNATIONAL CONFERENCE ON CLOUD AND AUTONOMIC COMPUTING (ICCAC), pages 72–83. IEEE International Conference on Cloud and Autonomic Computing (ICCAC), Augsburg, GERMANY, SEP 12-16, 2016.

Lenk, A., Klems, M., Nimis, J., Tai, S., and Sandholm, T. (2009). What's inside the cloud? an architectural map of the cloud landscape. In CLOUD '09: Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing, pages 23–31, Washington, DC, USA. IEEE Computer Society.

Lewis, G. A., Ionita, A., and Litoiu, M. (2013). Report of the 2012 ieee 6th international workshop on the maintenance and evolution of service-oriented and cloud-based systems (mesoca 2012). SIGSOFT Softw. Eng. Notes, **38**(2), 29–32.

Liu, F., Tong, J., Mao, J., Bohn, R. B., Messina, J. V., Badger, M. L., and Leaf, D. M. (2011). Nist cloud computing reference architecture - recommendations of the national institute of standards and technology. Technical Report 500-292, National Institute of Standards nd Technology, Gaithersburg, EUA.

Lorido-Botrán, T., Miguel-Alonso, J., and Lozano, J. A. (2014). A review of auto-scaling techniques for elastic applications in cloud environments. Journal of Grid Computing, **12**(4), 559–592.

Maximilien, E. M., Ranabahu, A., Engehausen, R., and Anderson, L. (2009a). Ibm altocumulus: a cross-cloud middleware and platform. In OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications, pages 805–806, New York, NY, USA. ACM.

Maximilien, E. M., Ranabahu, A., Engehausen, R., and Anderson, L. C. (2009b). Toward cloud-agnostic middlewares. In OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications, pages 619–626, New York, NY, USA. ACM.

Mazidi, A., Mahdavi, M., and Roshanfar, F. (2021). An autonomic decision tree-based and deadline-constraint resource provisioning in cloud applications. Concurrency and Computation: Practice and Experience, **33**(10), e6196.

McGillicuddy, Shamus (2021). The State of Data Center Networking: 2021 Annual Report. resreport, EMA – IT & Data Management Research, Industry Analisys & Consulting.

Mell, P. and Grance, T. (2011). The nist definition of cloud computing. Technical Report 800-145, National Institute of Standards and Technology, http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

Merle, P., Rouvoy, R., and Seinturier, L. (2011). A reflective platform for highly adaptive multi-cloud systems. In Adaptive and Reflective Middleware on Proceedings of the International Workshop, ARM '11, pages 14–21, New York, NY, USA. ACM.

Mondéjar, R., García-Lṕez, P., Pairot, C., and Pamies-Juarez, L. (2013). CloudSNAP: A transparent infrastructure for decentralized web deployment using distributed interception. Future Generation Computer Systems, **29**(1), 370 – 380. Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures.

Moreno-Vozmediano, R., Montero, R., and Llorente, I. (2012). Iaas cloud architecture: From virtualized datacenters to federated cloud infrastructures. Computer, **45**(12), 65–72.

Muñoz-Escoí, F. D. and Bernabéu-Aubán, J. M. (2016). A survey on elasticity management in paas systems. Computing, pages 1–40.

Naskos, A., Gounaris, A., and Sioutas, S. (2016). Cloud elasticity: A survey. In Revised Selected Papers of the First International Workshop on Algorithmic Aspects of Cloud Computing - Volume 9511, ALGOCLOUD 2015, pages 151–167, New York, NY, USA. Springer-Verlag New York, Inc.

Nelson, M. R. (2009). Building an open cloud. Science, **324**(5935), 1656–1657.

Ngu, A. H., Gutierrez, M., Metsis, V., Nepal, S., and Sheng, Q. Z. (2017). IoT Middleware: A Survey on Issues and Enabling Technologies. IEEE Internet of Things Journal, **4**(1), 1–20.

Nogueira, E., Moreira, A., Lucrédio, D., Garcia, V., and Fortes, R. (2016). Issues on developing interoperable cloud applications: definitions, concepts, approaches, requirements, characteristics and evaluation models. Journal of Software Engineering Research and Development, **4**(1), 7.

Ömer Köksal and Tekinerdogan, B. (2017). Obstacles in data distribution service middleware: A systematic review. Future Generation Computer Systems, **68**, 191 – 210.

Opara-Martins, J., Sahandi, R., and Tian, F. (2016). Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective. Journal of Cloud Computing, **5**.

Pahl, C. (2015). Containerization and the paas cloud. IEEE Cloud Computing, **2**(3), 24–31.

Pahl, C., Jamshidi, P., and Zimmermann, O. (2018). Architectural principles for cloud software. ACM Trans. Internet Technol., **18**(2), 17:1–17:23.

Palma, D. and Spatzier, T. (2013). Topology and orchestration specification for cloud applications - tosca. Technical report, Advancing Open Standards for the Information Society - OASIS.

Papazoglou, M. and van den Heuvel, W. (2011). Blueprinting the cloud. Internet Computing, IEEE, **15**(6), 74 –79.

Paraiso, F., Merle, P., and Seinturier, L. (2016). socloud: a service-oriented component-based paas for managing portability, provisioning, elasticity, and high availability across multiple clouds. Computing, **98**(5), 539–565.

Pariseau, B. and Jones, T. (2014). Cloud outage audit 2014 reveals aws on top, azure down. `http://searchcloudcomputing.techtarget.com/news/2240237323/`

Cloud-outage-audit-2014-reveals-AWS-on-top-Azure-down. Last access 07-19-2015.

Parkhill, D. F. (1966). The challenge of the computer utility. Addison-Wesley Pub. Co. Reading, Mass.,.

Parlavantzas, N., Pham, L. M., Sinha, A., and Morin, C. (2018). Cost-effective reconfiguration for multi-cloud applications. In 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), pages 521–528.

Petcu, D. (2014). Consuming resources and services from multiple clouds. Journal of Grid Computing, **12**(2), 321–345.

Petcu, D., Craciun, C., Neagul, M., Lazcanotegui, I., and Rak, M. (2011). Building an interoperability api for sky computing. In High Performance Computing and Simulation (HPCS), 2011 International Conference on, pages 405–411.

Petcu, D., Martino, B., Venticinque, S., Rak, M., Máhr, T., Lopez, G., Brito, F., Cossu, R., Stopar, M., Šperka, S., and Stankovski, V. (2013). Experiences in building a mosaic of clouds. Journal of Cloud Computing, **2**(1).

Pokahr, A. and Braubach, L. (2015). Towards elastic component-based cloud applications. In D. Camacho, L. Braubach, S. Venticinque, and C. Badica, editors, Intelligent Distributed Computing VIII, volume 570 of Studies in Computational Intelligence, pages 161–171. Springer International Publishing.

Pokahr, A., Braubach, L., and Jander, K. (2010). Unifying agent and component concepts: Jadex active components. In Proceedings of the 8th German conference on Multiagent system technologies, MATES'10, pages 100–112, Berlin, Heidelberg. Springer-Verlag.

Puder, A., Römer, K., and Pilhofer, F. (2006). Distributed Systems Architecture. A Middleware Approach. Morgan Kaufmann.

Qu, C. (2016). Auto-scaling and Deployment of Web Applications in Distributed Computing Clouds. Ph.D. thesis, Department of Computing and Information Systems – University of Melbourne.

RackSpace (2010). Openstack. http://www.openstack.org/. Acessed: 2021-07-30.

Ravi, N. and Thangarathinam, M. (2019). Emergence of middleware to mitigate the challenges of multi-cloud solutions onto mobile devices. International Journal of Cooperative Information Systems, **28**(04), 1950012.

Rodriguez, M. A. and Buyya, R. (2019). Container-based cluster orchestration systems: A taxonomy and future directions. Software: Practice and Experience, **49**(5), 698–719.

Roumani, Y. and Nwankpa, J. K. (2019). An empirical study on predicting cloud incidents. International Journal of Information Management, **47**, 131–139.

Saatkamp, K., Breitenbücher, U., Kopp, O., and Leymann, F. (2020). Method, formalization, and algorithms to split topology models for distributed cloud application deployments. Computing, **102**(2), 343–363.

Schantz, R. E. and Schmidt, D. C. (2002). Research advances in middleware for distributed systems: State of the art. In Proceedings of the IFIP 17th World Computer Congress - TC6 Stream on Communication Systems: The State of the Art, pages 1–36, Deventer, The Netherlands, The Netherlands. Kluwer, B.V.

Serrano, M., Davy, S., Johnsson, M., Donnelly, W., and Galis, A. (2011). Review and designs of federated management in futur in internet architectures. In J. Domingue, A. Galis, A. Gavras, T. Zahariadis, D. Lambert, F. Cleary, P. Daras, S. Krco, H. Müller, M.-S. Li, H. Schaffers, V. Lotz, F. Alvarez, B. Stiller, S. Karnouskos, S. Avessta, and M. Nilsson, editors, The Future Internet, volume 6656 of Lecture Notes in Computer Science, pages 51–66. Springer Berlin Heidelberg.

Singhal, M., Chandrasekhar, S., Ge, T., Sandhu, R., Krishnan, R., Ahn, G.-J., and Bertino, E. (2013). Collaboration in multicloud computing environments: Framework and security issues. Computer, **46**(2), 76 –84.

Sotiriadis, S. and Bessis, N. (2016). An inter-cloud bridge system for heterogeneous cloud platforms. Future Generation Computer Systems, **54**, 180 – 194.

Sotomayor, B., Montero, R., Llorente, I., and Foster, I. (2009). Virtual infrastructure management in private and hybrid clouds. Internet Computing, IEEE, **13**(5), 14 –22.

Spring, J. (2011). Monitoring cloud computing by layer, part 2. IEEE Security Privacy, **9**(3), 52–55.

Srirama, S. N. and Ostovar, A. (2014). Optimal resource provisioning for scaling enterprise applications on the cloud. In 2014 IEEE 6th International Conference on Cloud Computing Technology and Science, pages 262–271.

Takamura, T., Sakurai, H., Yamamoto, D., Murakami, M., Yamasaki, K., and Unno, Y. (2019). Realizing Next-Generation Hybrid Networks for Multi-Cloud Era. FUJITSU SCIENTIFIC & TECHNICAL JOURNAL, **55**(3), 61–68.

Tanenbaum, A. S. and Steen, M. V. (2007). Distributed Systems: Principles and Paradigms. Prentice Hall.

Toosi, A. N., Calheiros, R. N., and Buyya, R. (2014). Interconnected cloud computing environments: Challenges, taxonomy, and survey. ACM Comput. Surv., **47**(1), 7:1–7:47.

Truyen, E., Kratzke, N., Van Landuyt, D., Lagaisse, B., and Joosen, W. (2020). Managing Feature Compatibility in Kubernetes: Vendor Comparison and Analysis. IEEE Access, **8**, 228420–228439.

Vaquero, L. M., Rodero-Merino, L., Caceres, J., and Lindner, M. (2009). A break in the clouds: Towards a cloud definition. Computer Communication Review, **39**(1), 50–55.

Varghese, B. and Buyya, R. (2018). Next generation cloud computing: New trends and research directions. Future Generation Computer Systems, **79**(Part 3), 849 – 861.

Villegas, D., Bobroff, N., Rodero, I., Delgado, J., Liu, Y., Devarakonda, A., Fong, L., Sadjadi, S. M., and Parashar, M. (2012). Cloud federation in a layered service model. Journal of Computer and System Sciences, **78**(5), 1330 – 1344. JCSS Special Issue: Cloud Computing 2011.

Vinoski, S. (2002). Where is middleware? IEEE Internet Computing, **6**(2), 83–85.

Völter, M., Kircher, M., and Zdun, U. (2004). Remoting patterns: design reuse of distributed object middleware solutions. Internet Computing, IEEE, **8**(6), 60 – 68.

Völter, M., Kircher, M., and Zdun, U. (2005). Foundations of Enterprise, Internet and Realtime Distributed Object Middleware. John Wiley & Sons Ltd.

Yangui, S., Mohamed, M., Tata, S., and Moalla, S. (2011). Scalable service containers. In Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, pages 348 –356.

Zabolotnyi, R., Leitner, P., Hummer, W., and Dustdar, S. (2015). Jcloudscale: Closing the gap between iaas and paas. ACM Trans. Internet Technol., **15**(3), 10:1–10:20.

Zhang, S., Wang, W., Wu, H., Vasilakos, A. V., and Liu, P. (2013). Towards transparent and distributed workload management for large scale web servers. Future Generation Computer Systems, **29**(4), 913 – 925. Special Section: Utility and Cloud Computing.