Architectures and Algorithms for Cloud-Based Multimedia Conferencing

Abbas Soltanian

A Thesis

In

The Concordia Institute

For

Information and Systems Engineering

Presented in Partial Fulfilment of the Requirements

For the Degree of

Doctor of Philosophy (Information and Systems Engineering) at

Concordia University

Montreal, Quebec, Canada

October 2018

CONCORDIA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Abbas Soltanian

Entitled: Architectures and Algorithms for Cloud-Based Multimedia Conferencing

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Information and Systems Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. Luis Amador

_____ External Examiner
Dr. Ahmed Karmouch

_____ External to Program
Dr. Ferhat Khendek

_____ Examiner
Dr. Chadi Assi

_____ Examiner
Dr. Jamal Bentahar

_____ Thesis Supervisor
Dr. Roch Glitho

Approved by _____
Dr. Chadi Assi, Graduate Program Director

November 29, 2018 _____
Dr. Amir Asif, Dean
Gina Cody School of Engineering and Computer Science

ABSTRACT

Architectures and Algorithms for Cloud-Based Multimedia Conferencing

Abbas Soltanian, Ph.D.

Concordia University, 2018

Multimedia conferencing is the real-time exchange of multimedia content between multiple parties. It is the basis of several applications, such as distance learning, online meetings, and massively multiplayer online games. Cloud-based provisioning of multimedia conferencing has several benefits, like resource efficiency, elasticity, and scalability. However, it remains very challenging. A challenge, for instance, is the lack of holistic architectures which cover both the infrastructure and the platform layers of cloud-based multimedia conferencing applications. Another challenge is the lack of appropriate algorithms for resource allocation in the conferencing cloud to accommodate the fluctuating number of participants, while meeting the required quality of services (QoS). Yet another example is the lack of suitable algorithms for scaling the multimedia conferencing applications in the cloud while meeting both QoS requirements and cost efficiency objective. Unfortunately, the solutions proposed so far do not address these challenges.

This thesis focuses on the architectural and algorithmic challenges of cloud-based multimedia conferencing. It proposes architectural components and interfaces for multimedia conferencing application provisioning, covering both the Platform-as-a-Service (PaaS) and the Infrastructure-as-a-Service (IaaS) layers. The proposed interfaces simplify multimedia conference service provisioning for a wide range of application providers. On the algorithmic side, it proposes resource allocation mechanisms that support scalability in terms of the number of participants while meeting the QoS. These

mechanisms allocate the actual resources (e.g., CPU, RAM, and storage) in an optimal manner. Besides these mechanisms, it proposes the scalability approaches for cloud-based multimedia conferencing applications. To ensure cost efficiency, these proposed solutions enable fine-grained scalability of the applications with respect to the number of participants while considering the QoS requirements. All algorithmic problems in this thesis are formulated using the Integer Linear Programming (ILP) and heuristics have been designed and validated to solve them.

Acknowledgments

All praise is due to God who gave me the strength and determination and guided me throughout this work and beyond. Also, I am immensely and forever thankful to Imam Reza, who my Ph.D. application was accepted while I was working on a project for him. All I have achieved in my life is because of God, Imam Reza, and his holly family.

First, and foremost, I am grateful to my Ph.D. supervisor Dr. Roch Glitho for his guidance, support, patience, and encouragement. He is a dedicated and caring advisor. Thank you.

I gratefully acknowledge my Ph.D. committee members, Dr. Ferhat Khendek, Dr. Chadi Assi, and Dr. Jamal Bentahar for their time, effort, and constructive comments.

I am also thankful to Dr. Halima Elbiaze, Dr. Fatna Belqasmi, Dr. M. Ali Salahuddin, Dr. Sami Yangui, and Dr. Diala Naboulsi for all the enlightening discussions, comments, and collaborations. It was a great pleasure working with you.

Last but not least, I am forever indebted to my parents for their encouragement, continuous support, love and prayers. Without them and their support, not only this thesis, but also none of my achievements would have been possible. There are no words that can express my gratitude and love for you. Also, I am so thankful to my love, who has always been beside me in all good and bad times. I will love you Leila and honor you in all days of my life. I am also thankful to my parents in law, my close family and friends.

Contents

List of Figures

List of Tables

List of Abbreviations

ADS	Adaptive and Dynamic Scaling
AOS	Adaptive Online Scaling
CMCU	Cloud-Based Multipoint Control Unit
CMIP	Cloud Multiple Infrastructure Provider
CRAM	Cloud-based Resource Allocation for Multimedia conferencing
CSIP	Cloud Single Infrastructure Provider
IaaS	Infrastructure-as-a-Service
ILP	Integer Linear Programming
ITU	International Telecommunication Union
MCU	Multipoint Control Unit
MMOG	Massively Multiplayer Online Games
NCC	Non-Cloud Conferencing
NIST	National Institute of Standards and Technology
PaaS	Platform-as-a-Service
QoS	Quality of System
SaaS	Software-as-a-Service
SLA	Service Level Agreement

SOA	Service Oriented Architecture
SubaaS	Substrate-as-a-Service
VM	Virtual Machine
VMRA	Video Mixing Resource Allocation
WoW	World of Warcraft

Chapter 1

1. Introduction

This chapter first presents an overview of the challenges and contributions that are discussed in this thesis. Then, it discusses the required background information on cloud computing and multimedia conferencing. Finally, it presents the outline of the rest of this thesis.

1.1 Overview

Cloud computing is a paradigm in which resources (e.g., storage, network, and services) are provisioned rapidly on demand. It offers three main service models, Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) [1]. It provides several benefits, such as scalability and elasticity. Multimedia conferencing (or conferencing in short) is the real-time exchange of multimedia content (e.g., audio, video, and text) between different parties [2]. It has several applications, such as massively multiplayer online games (MMOG) and distance learning. In some conferencing applications like MMOGs, there might be thousands or hundreds of thousands of users (conference participants). This number of participants may have considerable fluctuations over a short period of time. For instance, in one study, the number of players in the World of Warcraft game (a famous MMOG) fluctuates between 1.5 and

2.5 million during 10 hours [3]. Therefore, such applications require scalability and elasticity that cloud-based implementations may provide.

Conferencing application provisioning refers to the entire life-cycle of the conferencing application, i.e., development, deployment, and management [4]. Cloud-based provisioning of the conferencing applications will bring several benefits including rapid provisioning, resource efficiency, scalability, and elasticity. However, it is quite challenging. A challenge, for instance, is the lack of holistic architectures which take all aspects of cloud-based conferencing applications (e.g., PaaS and IaaS) into account. The holistic architecture can ease provisioning of the conferencing applications. For instance, it can help the conferencing application providers to not master low-level details of conferencing technologies, protocols, and their dependencies. Therefore, provisioning of the conferencing applications can be easier especially for non-expert providers.

Another challenge is the lack of appropriate algorithms for resource allocation in the conferencing cloud to accommodate the fluctuating number of participants while meeting the required QoSs. As it was mentioned before, the fluctuation in terms of the number of participants is high in some conferencing applications. If the allocated resources are not enough, the participants cannot attend the conference. In consequence, it reduces the participants' satisfaction and may result in decreasing the QoS. On the other hand, if the allocated resources are more than demand, it increases the cost. Thus, the efficient resource allocation algorithms can help to avoid under-provisioning and over-provisioning of resources.

Yet another challenge is the lack of suitable algorithms for scaling conferencing applications in the cloud while meeting both QoS requirements and cost efficiency objective. Besides the actual resources (e.g., computational resources and storage), the conferencing applications also need to scale for accommodating the fluctuated number of participants. Thus, there is a need for having efficient scaling mechanisms for the conferencing applications.

Unfortunately, the solutions proposed so far do not address these challenges. This Ph.D. thesis addresses the architectural and the algorithmic challenges of cloud-based

multimedia conferencing. It consists of three main contributions which are presented as follows.

(i) Holistic Cloud-based Architecture for Multimedia Conferencing Applications [5], [6]

The first contribution is on the architectural components and the interfaces which covers both the infrastructure and the platform layers of cloud-based multimedia conferencing applications. This architecture simplifies the provisioning of the conferencing applications for expert and non-expert application providers. For this contribution, novel architectural components are proposed for the PaaS and the IaaS layers of multimedia conferencing. The proposed architecture provides novel application programming interfaces (APIs) to simplify the provisioning of the conferencing applications for a wide range of application providers (experts vs. non-experts). It allows the conferencing application providers to utilize the offered conferencing services (e.g., audio and video mixing) without having to deal with the complexities of conferences. The proof-of-concept prototypes are also implemented.

(ii) Resource Allocation Mechanisms for Multimedia Conferencing Applications [7], [8]

The second contribution is the cloud-based resource allocation algorithms for multimedia conferencing applications. In this contribution, we consider conferencing applications with video mixing and compressing. The proposed algorithms allocate the actual resources in an optimal manner while supporting scalability in terms of the number of participants, and guaranteeing the required QoS. Since these algorithms are designed to scale the actual resources (e.g., CPU, RAM, and Storage), they are suitable to be executed on the conferencing IaaS.

(iii) Scaling Mechanisms of Multimedia Conferencing Applications [9], [10]

Lastly, the third contribution is the fine-grained scaling algorithms for multimedia conferencing applications. These algorithms enable the conferencing applications to scale in an elastic manner with respect to the number of participants. The proposed algorithms

also guarantee to meet the QoS requirements while considering the future demands of the conferencing applications and cost efficiency objective. Instead of dealing with the actual resources, the proposed algorithms scale the conferencing applications in a higher level of abstraction which is the number of participants. In fact, these algorithms in collaboration with the resource allocation algorithms in the conferencing IaaS can scale the conferencing applications. Therefore, the proposed algorithms in this contribution are suitable to be executed on the conferencing PaaS.

In both algorithmic contributions, the problems are mathematically modeled as integer linear programming (ILP) problems. We solve the mathematical models to achieve optimality for the small-case scenarios using the optimization tools (e.g., LPSolve Java Library). We propose heuristics to solve the problems for the large-scale scenarios in an acceptable time. The heuristics are evaluated in different scenarios and with different parameters and settings.

More details and background information on cloud computing and multimedia conferencing are presented in the following two sections.

1.2 Cloud Computing

There are several definitions for cloud computing. This thesis adopts the definition of cloud computing provided by the National Institute of Standards and Technology (NIST) [11]:

"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."

Cloud computing has five essential characteristics [11]:

1. On-demand self-service: The cloud computing services (e.g., computing, networking, and storing) can be provisioned by consumers without human interactions with cloud service providers.

2. Broad network access: All of the services are available and accessible through the network.
3. Resource pooling: The cloud resources (e.g., compute, storage, and network) are pooled to provide services to multi-tenants according to the demand for each customer. In fact, physical and virtual resources are dynamically assigned and reassigned according to the consumers' demands.
4. Rapid elasticity: The cloud is capable of provisioning of services according to the consumer's workload requirements.
5. Measured service: The usage of cloud resources can be monitored and controlled using some metering capabilities. It provides transparency for both providers and consumers of the utilized services.

Cloud computing can be represented using a service-driven business model. In this model, hardware and applications are provided as on-demand services [12]. These services can be grouped into three layers:

(i) Infrastructure-as-a-Service (IaaS)

The IaaS is composed of physical and virtualized resources (e.g., network, storage, and servers) and provides scalable and cost-efficient resources as a service to the customers. IaaS relies on virtualization technology that enables the abstraction of hardware resources from the services. Virtualization allows consolidation of hardware resources into pools of virtual shared resources. The consumers of IaaS have limited access to the underlying infrastructure resources. However, the offered services can be tailored to the consumers' requirements [13]. Amazon EC2, OpenNebula, and IBM Blue Cloud are some examples of cloud IaaS.

(ii) Platform-as-a-Service (PaaS)

The PaaS provides the environment needed to facilitate the application provisioning lifecycle. The application provisioning includes development, testing, deployment, and execution. The PaaS allows developers to focus on creating applications and freeing them from the operations or platform maintenance. In consequence, it eases and accelerates the application provisioning. The PaaS consumers only have control on their deployed

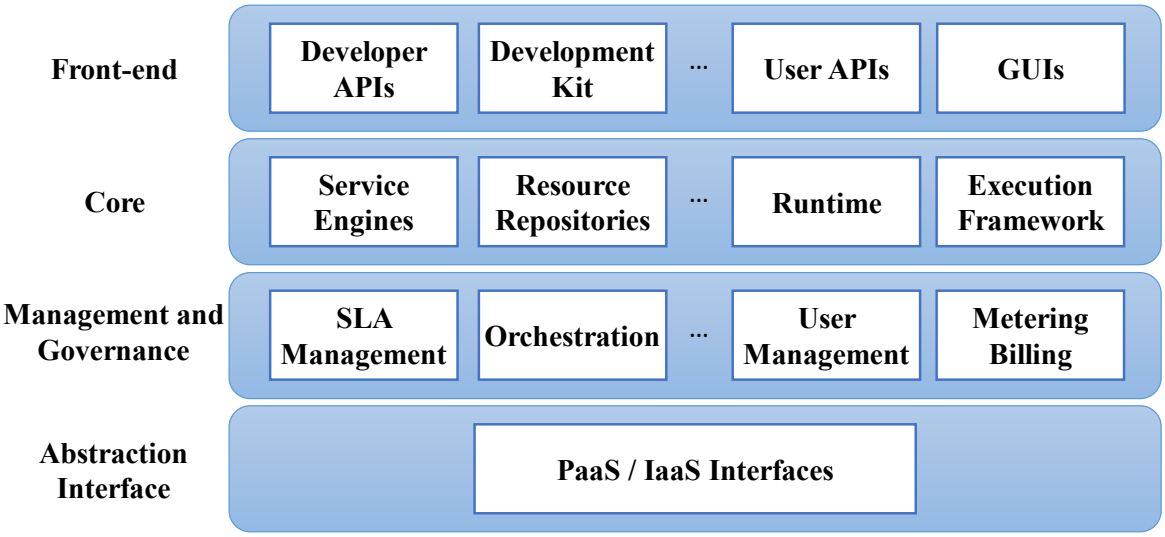


Fig. 1.1. IBM PaaS Reference Architecture

applications [13]. Google Cloud Platform, Aneka, and Cloud Foundry are some examples of this cloud service model.

(iii) Software-as-a-Service (SaaS)

The SaaS is a software delivery model in which applications are hosted by the service providers on cloud and delivered as a service to the users over the network. Here, the consumers do not have any control over the application and the underlying infrastructure layer [13]. SalesForce.com and Google Docs are examples of SaaS cloud service model.

1.2.1 PaaS Architectures

There are some reference models for PaaS architecture such as the one introduced by IBM [14] and Aneka [15]. As an example, the IBM PaaS layered architecture is depicted in Fig.1.1. It consists of four layers:

1. Front-end: It has a set of user and developer APIs and tools. Development APIs allow the developers for allocating and managing the PaaS resources. The user APIs and graphical user interfaces (GUIs) allow the users for invoking and executing the applications which are running in the PaaS.

2. Core: It has necessary frameworks (e.g., containers and storage services) required for application hosting and execution.

3. Management and Governance: Consists of entities for managing the PaaS and the hosted applications (e.g., monitoring and scaling). Moreover, it has the required entities to support the PaaS Business model (e.g., billing and membership).

4. Abstraction Interface: It has a set of APIs and operations that enable the interaction with the underlying IaaS.

1.2.2 IaaS Architectures

Similar to the PaaS, the IaaS also has some reference architecture model such as the one introduced in [16]. In this architecture model, the IaaS has three main layers:

1. Cloud Management: It has the responsibility of managing the overall IaaS. It also acts as an interface with IaaS consumers (e.g., PaaS and another IaaS).
2. Virtual Infrastructure Management: It provides a uniform and homogenous view of virtual resources. It provides primitives to schedule and manage VMs across multiple physical hosts.
3. Virtual Machine Management: It provides simple primitives (e.g., start, stop, suspend) to manage VMs on a single host.

1.3 Multimedia Conferencing

Multimedia conferencing has three main architectural components, namely signaling, media handling, and conference control [2]. Signaling is responsible for the establishment, modification, and teardown of multimedia sessions. Session establishment can be done in two different ways: dial-in or dial-out. In dial-in conferences, the participants should call the signaling server to join the conference while in dial-out conferences, the server calls all the participants.

Media handling is related to media functionalities such as audio and video mixing, transcoding, and compressing. Some researchers believe that the mixers are the core of the media handling systems [17]. Audio mixer and video mixer deal with several received media streams from multiple sources, combine them, and send the mixed stream to the participants. Some systems may only work with specific codecs. As an example, a device may have the ability to only play the “H.264” video codec. In order to support the heterogeneity of audio and video codecs, there is a need to have transcoding ability in the

media handling component. Transcoding is a functionality to convert one codec signal to another one. The media compressing is another functionality of media handling. It is used to reduce the size of media. Its input media type is the same as its output's. However, the output stream size is less than that of the input.

Conference control encompasses the management functions to define and control the conference policies and floor control. The conference policy functions include conference arrangement, admission control, participant management, and voting. Based on the RFC4582, the floor is: "A temporary permission to access or manipulate a specific shared resource or set of resources". Based on this definition, the floor control is a mechanism which enables the management of the joint or exclusive access to the shared resources (e.g., audio channel, video channel) among the participants inside a conference. There are three entities involved in the floor control mechanism: 1) Floor Participants – a conference participant who is requesting for the access to the shared resources in the conference; 2) Floor Chair – a conference participant who grants or denies the requests of floor participants; and 3) Floor Control Server – a logical entity between the floor chair and all floor participants which maintains the state of the floor (e.g., who is the chair, who has the floor) and transmit all requests, decisions, and notifications.

There are some conferencing classification schemes. One example is whether the conference has the sub-conferencing capability or not. This capability simulates a conference inside another conference. In other words, sub-conferencing simulates a conference with some different rooms. In each room, entitled as a sub-conference, the participants can hear or see each other while they cannot hear or see other participants in other sub-conferences.

Another classification scheme is whether the conference can be prearranged or ad-hoc. In prearranged, the conference starts at a predetermined time and the duration of the conference may also be predefined. However, in ad-hoc, the conference starts when the first two participants decide to create a session and it ends when the last two participants leave.

1.4 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 discusses the challenges, general motivation scenario, and the requirements, followed by the review of the state-of-the-art. Chapter 3 to 7 present the main contributions of this thesis. Chapter 3 discusses the proposed holistic cloud-based architecture for multimedia conferencing applications. Chapter 4 presents a proposed resource allocation algorithm for multimedia conferencing applications. The proposed algorithm in chapter 4 has some limitations which will be covered in another algorithm presented in chapter 5. Chapter 6 discusses a proposed scaling mechanism of multimedia conferencing applications. The proposed algorithm in this chapter also has some limitations which will be covered in another algorithm presented in chapter 7. Finally, chapter 8 concludes the thesis and provides future directions for this research.

Chapter 2

2. Challenges, Requirements, and Related Work

Despite all improvements in the conferencing technologies, the proposed multimedia conferencing solutions so far still face many challenges. Cloud computing, as an enabler, can help to solve some of these challenges. This chapter presents the motivations behind this research by discussing the challenges to be tackled in this thesis. Then, it derives the requirements of cloud-based multimedia conferencing. After that, the related works are reviewed in light of the derived requirements. Lastly, it concludes with the summary of the related work.

2.1. Challenges

The challenges are classified into three categories: general challenges, PaaS related challenges, and IaaS related challenges.

2.1.1. General Challenges

Nowadays, there are several existing multimedia conferencing applications such as MMOGs, distance learnings, and online meetings. In some of these applications like MMOGs, there might be thousands or hundreds of thousands of conference participants. This number of participants may have considerable fluctuations over a short period of time. For instance, in one study, the number of players in World of Warcraft (WoW- a famous

multiplayer online game), fluctuates between 1.5 and 2.5 million over 10 hours [3]. Therefore, scalability is essential in such applications. Non-cloud multimedia conferencing solutions do not scale well. For example, a media server (a device or a software that is responsible for media transmission, mixing, and transcoding) can offer media handling services to the limited number of media streams. Assuming each participant uses a camera or a microphone, increasing the number of participants leads to increasing the number of media streams. Thus, if the media server is overloaded, a new media server should add to offer media handling services to the new participants. However, in non-cloud multimedia conferencing solutions, increasing the required resources (i.e., media servers in this example) to cope with demands is a challenge. On one hand, it is time-consuming (e.g., it may take several hours to several days to add the required resources). On the other hand, changing the applications' configuration in the runtime (to work with the newly added resources) may not be possible or may cause application outage. Cloud-based multimedia conferencing solutions can tackle these challenges and enable scalability. The resources can dynamically increase or decrease on-demand and during the application's runtime. In addition, thanks to the virtualization technology that can be used in the cloud solutions, increasing or decreasing the resources can be hidden from the participants' perspective. Therefore, there is no need to change the applications' configuration in the runtime.

Besides the scalability problem, the efficient use of resources is another challenge. Non-cloud conferencing solutions usually suffer from over-provisioning or under-provisioning of resources. In such solutions, they may over-provision the resources in advance to ensure they can accommodate all possible participants in the near future. For instance, WoW uses more than ten-thousand servers while most of the servers' capacities remain idle most of the time [3]. On the other hand, under-provisioning of resources causes application outage for the incoming participants. Cloud-based solutions can enable allocating and de-allocating of required resources in an elastic manner and in a fine granularity. In consequence, it enables efficient use of resources.

2.1.2. PaaS Related Challenges

There are several conferencing concepts which conferencing application providers should consider. For instance, there are different conference models such as pre-arrange or

ad-hoc. As another example, there could be different conferencing technologies such as SIP, WebRTC or hybrid. In addition, each concept has its own technical details. For instance, the acceptable audio and video encodings in each considered conference technology should be defined. All these technical details require experienced conferencing application providers. However, their expertise may be different. A conferencing PaaS can enable hiding the technical details required for provisioning the conferencing applications. Therefore, it can simplify the provisioning of conferencing applications for a wide range of conferencing application providers (experts vs. non-experts). In addition, a conferencing PaaS can offer suitable algorithms for scaling the conferencing applications while meeting different criteria such as QoS and cost efficiency.

2.1.3. IaaS Related Challenges

The IaaS layer can enable the on-demand provisioning of the actual resources such as CPU, RAM, and storage. Consequently, it enables minimizing the associated capital costs of having individual IT infrastructures. However, there are some issues related to having a conferencing IaaS. For example, as it was mentioned in chapter one, the IaaS has an architectural layer entitled as Cloud Management. This layer is responsible for the overall IaaS management and also acts as an interface with IaaS consumers (e.g., PaaS or other IaaSs). Generally, the usual IaaS consumers expect IaaS services which can be Computing, Storing, and Networking. However, the consumers of a conferencing IaaS might expect other services such as Audio and Video Mixing, Transcoding, and Signaling. This difference in their expectations brings the need of having new interfaces and APIs. In addition, it brings the need of having new resource allocation algorithms in the conferencing IaaS. These algorithms can enable efficient resource allocation for the new expected services while guaranteeing different requirements such as QoS and cost.

Solving all mentioned challenges is the motivation of research in cloud-based architectures and algorithms for multimedia conferencing.

2.2. Requirements

According to the stated challenges, requirements are classified. Some of the mentioned challenges have architectural aspects and some others have algorithmic dimensions.

Therefore, the requirements are categorized into two categories: architecture-specific requirements and algorithm-specific requirements.

2.2.1. Architecture-Specific Requirements

These requirements should be considered in the conferencing architectural contribution.

1) **Scalability**: A multimedia conferencing application should function well with different workloads (e.g., having few or several participants). It needs to be scalable in terms of different conferencing concepts such as the number of conferences, sub-conferences, floors, and conference participants. The conferencing PaaS, in collaboration with the conferencing IaaS, should scale the conferencing applications in response to the new demand.
2) **Elasticity**: The conferencing PaaS and IaaS, should scale the conferencing applications in a fine-grained (elastic) manner in response to the new demand (e.g., the fluctuating number of participants, increasing or decreasing the number of conferences). This enables the cost efficiency and follows the pay-per-use principle of cloud.
3) **Meeting the Quality of Services**: As it was mentioned before, multimedia conferencing is the real-time exchange of media contents between different parties. To guarantee the real-time exchange of media, meeting the QoS requirements, such as latency, jitter, and throughput is critical in conferencing applications.
4) **Publish-and-Discovery Mechanism**: The cloud conferencing can simplify the provisioning of conferencing applications (e.g., distance learnings) by offering conferencing services (e.g., audio and video mixing) that may use by these applications. Therefore, the providers of the multimedia conferencing applications need to find the appropriate conferencing services which can fulfill their requirements. Publish and discovery mechanism allows the conference application providers to discover available conferencing services. It also enables the conferencing PaaS to discover a conferencing IaaS as well as a conferencing IaaS to discover other conferencing IaaSs for excess workloads.
5) **Composition**: This feature simplifies creating a complex conferencing service based on the basic conferencing services. For example, a dial-in audio conference service

might be composed of a dial-in service and an audio-mixing service. As another example, a video mixer with compression and transcoding capabilities can be composed of three video mixing, compressing, and transcoding conferencing services.

6) **High-level PaaS Northbound Interfaces**: The conferencing PaaS northbound interfaces should enable the conference application provisioning for a wide range of providers (experts vs. non-experts). Having a conferencing PaaS with high-level northbound interfaces helps to provision new applications without having to deal with the complexities of conferencing components and their interactions. The interfaces should also be flexible enough for creating complex and novel conferencing applications (e.g., a distance learning application with dial-in audio conference capability and five minutes of chat per hour).
7) **Conference-rooted IaaS Interfaces**: The conferencing IaaS interfaces should support communication with IaaS consumers in terms of the virtual conference or finer abstracted level such as virtual mixers or conference participants. Thus, the conferencing IaaS interface needs to be rooted in the conferencing concepts.

2.2.2. Algorithm-Specific Requirements

The following requirements are identified as algorithm-specific requirements.

1) **Scalability**: As it was mentioned before, multimedia conferencing applications need to scale in terms of different conferencing concepts to function well in different workloads. Thus, the resource allocation algorithms for these applications need to consider scalability in terms of conferencing concepts. These algorithms should be able to dynamically scale the required resources to cope with new demands.
2) **Efficient Use of Resources**: Scaling the conferencing applications and their required resources need to be done in a fine-grained manner. This enables the efficient use of resources and consequently, cost efficiency.
3) **Meeting the Quality of Services**: Meeting the QoS requirements, such as latency, jitter, and throughput is crucial in conferencing applications. Therefore, the responsible algorithms for scaling these applications and their required resources need to meet the QoS requirements. Considering the future demands of the application can also play an

important role in meeting the QoS. Therefore, the conferencing scaling algorithms need to take into account the future demand of the application as well.

2.3. Related Work

In this section, the state-of-the-art for cloud-based multimedia conferencing is presented. First, we discuss the works related to the architectural aspects of our work. After that, the related algorithmic works are reviewed.

2.3.1. Architectural Related Work

In this section, the existing architectures of cloud-based conferencing, PaaS, and IaaS are reviewed. In addition, service composition and discovery solutions are also discussed.

(i) Cloud-based Conferencing Architectures

The existing architectures can be categorized with a focus on the SaaS or IaaS layers. Examples of the first category are presented in [18] and [19]. The two solutions focus on developing conferencing services at the application layer, without addressing the challenges related to the PaaS and IaaS layers (e.g., scalability, QoS, publication, and discovery of conferencing services). Ref. [18] offers conferencing services as SaaS, while using a conventional PaaS for deployment and execution. Ref. [19] presents an approach for providing video conferencing as a web service and defines the interfaces to be used by the conferencing application providers. This work tries to transform the existing telecommunication services into a reusable resource for the third parties. However, it does not address how these services are provisioned.

Ref. [20] is an example of the relevant works with a focus on the IaaS layer. The proposed architecture relies on conferencing substrates (i.e., basic conferencing building blocks such as signaling, audio and video mixing) and enables scalability in an elastic manner. It also proposes PaaS/IaaS interfaces rooted in substrates and proposes a broker between IaaS and PaaS that allows finding suitable substrates. However, it does not consider the PaaS and SaaS layers and their relevant issues. Neither does it include high-level PaaS interfaces for application providers.

Other works in the relevant literature, such as [21], [22], and [23], address specific problems of cloud-based conferencing, such as inter-datacenter network utilization, media mixing, and transcoding. While they focus on how conferencing components can efficiently utilize the cloud, they do not address conferencing application provisioning. In addition, as these works only offer one service, they do not tackle the service publication, discovery, and composition.

(ii) Existing PaaS Solutions

Aneka [15] and Cloud Foundry [24], the two PaaS representatives, are evaluated. Aneka provides high-level PaaS interfaces and supports scalability in an elastic manner, specifically for distributed application provisioning. Nonetheless, it does not offer any conferencing APIs. Cloud Foundry provides no interfaces for conferencing application provisioning. It supports the scaling of application instances but does not address scaling in terms of conference concepts. Neither does it address composition and QoS.

(iii) Existing IaaS Solutions

Some relevant literature propose a conceptual architecture of open-source IaaSs. Ref. [25], for example, proposes the OpenStack architecture that consists of five layers: Compute (Nova), Storage (Swift), Image (Glance), Identity (Keystone), and Dashboard (Horizon). Nova is the computing fabric controller for OpenStack and it is all about access to the computing resources. Swift, as the storage infrastructure in OpenStack, offers APIs to store and retrieve lots of data. Glance builds a discovery and retrieval system for VM images. Keystone is responsible for authentication and authorization. Horizon provides a web-based user interface to all above OpenStack services. In [26], instead of having one layer for Storage, it is broken down into two layers: Block Storage and Object Storage. Block Storage offers storage volume for Compute layer while Object Storage stores the actual virtual disc files. Their architecture also has a Network layer to provide virtual networking for the Compute layer. All components in both architectures follow a shared-nothing policy, meaning each component can be installed on any server.

The OpenNebula architecture proposed in [25] and [27] has three layers: Drivers, Core, and Tools. Drivers do the communication with the underlying operating system. VM

creation, startup and shutting down are parts of this layer's functionality. The core is a centralized layer that manages the VM life cycle. To manage VMs, Tools offers different interfaces for communication with users. Authors in [16] keep the Core and Drivers layers and propose Scheduler to replace Tools. Scheduler decides about VM placement. This layer keeps track of all the incoming requests in order to send an appropriate deployment command to the Core layer, based on those requests. They also have an Interface layer to communicate with users.

All above IaaS solutions are VM-based, thus, their interfaces should change to support the communication rooted in conferencing concepts (e.g., start, stop and modify the conferencing substrates). Moreover, they support scalability in terms of computing resources, storage, and networking. However, as a conferencing IaaS, there is a need to scale resources in terms of conferencing concepts (e.g., the number of participants) to collaborate with the conferencing PaaS.

(iv) Service Composition and Discovery

Service composition is a well-researched topic as several solutions and alternatives have been proposed to cater to different situations [28], [29], [30]. Service composition can be done in a static or dynamic way [31]. In a static composition, the basic services as part of the composition are selected in advance and their aggregation takes place at the design time. In contrast, dynamic composition allows to select and replace the basic services during the runtime. The composition can also be done manually, semi-automated or automatically [31]. In manual composition, the service provider should define and create an abstract composite process and manually bind the services to the abstract process. Some web service standard languages such as BPEL [32] or OWL-S [33] can be used to create the abstract process. In automatic composition, the new composite service specification can be generated by selecting adequate services based on the specified requirements [31]. Semi-automatic composition leverages both manual and automatic approaches. Workflow-based and template-based compositions are other composition planning techniques [34]. In the workflow-based composition, the process is depicted as an acyclic directed graph with control and data flow. This technique requires the developers' extensive domain knowledge and is time-consuming. In the template-based composition, templates describe

the outline of activities required to solve the problem. Templates are parameterized and use variables that allow customization based on the users' needs and preferences. In fact, the templates lead to creating an executable workflow.

Ref. [35] proposes a cloud service broker to facilitate the deployment of cloud application topologies from multiple cloud providers. The authors also propose a multi-criteria optimization algorithm to select the basic services to be composed. The algorithm sets cost efficiency as the main objective. Authors in [36] consider a wide range of objectives to design their cloud broker selection mechanism, such as user constraints, financial, energetic, geographic or operator contractual preferences. Ref. [37] considers multimedia conferencing requirements for designing the service broker. The authors here propose an architecture for substrate service publication and discovery. Their service broker acts between the substrate providers and the conferencing IaaSs and offers some REST APIs as the interfaces between them.

2.3.2. Algorithmic Related Work

In this section, the related algorithmic works are reviewed. First, we review the resource allocation solutions proposed so far for cloud-based multimedia conferencing. This is followed by a discussion of the other cloud-based solutions. This discussion includes multimedia solutions which are not multimedia conferencing. After that, we will present the general cloud resource allocation solutions that are not bounded to multimedia and multimedia conferencing applications. The solutions which are based on Network Function Virtualization (NFV) [38] are also reviewed. NFV is a technology that enables dynamic provisioning of network services. However, these solutions are discussed because NFV is also considered as a candidate technology for provisioning other services such as multimedia services [39]. Finally, the traditional approaches for multimedia conferencing are reviewed.

(i) Resource Allocation for Multimedia Conferencing in the Cloud

There are some algorithmic and architectural works done in multimedia conferencing in the cloud. Negralo et al. [40] present algorithms for scaling resources based on the real-time demands by using load balancing and the addition or removal of virtual machines

(VMs). Reaching a predefined threshold for CPU or bandwidth usage triggers scaling. Gao et al. [41] also work on cost-efficient video transcoding in the cloud. They minimize the overall storage and computing cost by partially using offline and online transcoding. The main focus of these works is cost efficiency and they do not consider QoS.

Hajiesmaili et al. [42] model the video conferencing cost in multiparty cloud video conferencing architecture. The main focus in this work is minimizing the operational cost by finding the best assignment of users to VMs. Besides minimizing the cost, they aim to reduce the conferencing delay as well. However, this work does not consider the resource allocation problem in case of having fluctuations in the number of participants. Abdallah et al. in [43] survey other architectural works on delay-sensitive conferencing video services. They present some related applications such as Cloud Gaming, Virtual Reality (VR) and Augmented Reality (AR) and their requirements for conferencing services. They also review the architectural designs for the management of such services. In addition, they briefly talk about optimization techniques. None of the reviewed papers meet the requirement of scalability in terms of considering fluctuations in the number of participants.

(ii) Non-Conferencing Related Cloud Resource Allocation Solutions

Several researchers have proposed solutions for resource allocation to multimedia services in the cloud. However, they do not focus on multimedia conferencing. Xavier et al. in [44], [45] propose resource allocation algorithms for audio and video services in the content delivery network (CDN). The proposed solutions scale the resources at the VM-level while attempting to minimize the cost. They also consider meeting the users' quality of experience in their algorithms. Gao et al. in [46] present a resource allocation algorithm for transcoding as a cloud service. In this work, they try to maximize the service profit while achieving their performance requirements such as service processing delays. Although these works consider scalability, fine-grained resource allocation and efficient use of resources are not considered.

He et al. in [47] and Dong et al. in [48] consider fluctuation in the number of audio and video sources. In these two works, they consider numerous users as video broadcasters

which live stream their video content such as their mobile camera feed or online game scenes. The authors in [47] propose a generic cloud framework that considers the viewers' quality of experience (QoE) and cloud resource cost. They only consider transcoding as a media handling service in their study. The authors in [48] propose an algorithm that makes a tradeoff between QoE of users and the total cost for a media service provider. None of these two works consider having a video mixing service. It means that in these works, the videos are just streamed from a source to a destination and never mixed with other video sources.

Several cloud resource allocation solutions consider meeting the QoS requirements and cost efficiency. Considering the future demands of the application also can play an important role in meeting the QoS. Some of these solutions consider the future demands of the application while others only take real-time demands into account. Therefore, we categorized them into two group.

(iii) General Cloud Resource Allocation Solutions

There are several cloud-based resource allocation solutions with different objectives such as reducing the cost or meeting the QoS requirements. We categorize them into existing PaaS resource allocation solutions, and IaaS resource allocation solutions.

a) PaaS Resource Allocation Solutions

Anselmi et al. [49] model the resource allocation problem of PaaS as a Generalized Nash Equilibrium problem. Their scaling model relies on the number of VMs that host the applications which are offered as SaaS. Their proposed game-theoretic approach tries to manage the capacity of a PaaS provider among multiple competing SaaSs at runtime. Gomez et al. [50] introduce a PaaS framework that enables provisioning of cloud-based services and applications by using the blueprints. The blueprint in this work refers to the technical description of an application and all its dependencies (e.g., required resources and deployment geolocations). Their platform supports both horizontal and vertical scaling and relies on different IaaSs. Hu et al. [51] present an adaptive resource management algorithm. Their proposed PaaS dynamically allocates and de-allocates the application

instances based on the fluctuation of resource demands. Their algorithm monitors the performance statistics to tune the scaling decision.

Machado et al. [52] present a PaaS framework that supports the deployment of multi-tier and stateful applications while assuring their availability. In this work, they use different profiles to represent the requirements of applications such as response time and budget. Satoh [53] also proposes a resource allocation mechanism for applications in the cloud. This approach considers the runtime data and it tries to minimize the required resources by reducing the redundant functions and data of the applications. None of the aforementioned works take the future demand for the applications into account. Moreover, the scaling decisions in these solutions lead to adding a new instance of an application or a VM. Therefore, their scaling decision may not suitable for cost-efficiency objective.

Babaioff et al. [54] present a scheduling and pricing framework for cloud resources based on the predicted demands and completing a job within a deadline. The proposed solution updates the prediction with every new request. Also, their architecture provides some internal APIs which enable plugging the algorithmic modules such as demand prediction. Bunch et al. [55] present a pluggable auto-scaling mechanism for PaaS. Their solution considers different resource pricing model offered by IaaS. They use an exponential smoothing algorithm to forecast the future demands for a specific period. The algorithm runs periodically and predicts the future demands based on the requests over the last t seconds. Roy et al. [56] also developed a model-predictive algorithm for workload forecasting. They use Autoregressive-Moving-Average method for their workload prediction. While [55] considers the uncertainty in the prediction model, [56] has no consideration for misprediction. These solutions also can only support scalability at the VM-level granularity and do not consider real-time demands.

b) IaaS Resource Allocation Solutions

An online resource allocation solution is proposed by Mashayekhy et al. in [57]. Their solution runs as soon as a request by a user arrives or a resource is released. Their objective is to allocate resources in terms of VMs while minimizing the cost for both IaaS providers and users. Shen et al. in [58] and Han et al. in [59] also propose a resource allocation

mechanism with the objective of minimizing the wastage of resources by considering the real-time demands for resources. The scaling decisions of all these mechanisms result in the addition or removal of VMs. Moreover, the main focus of these works is cost efficiency and they do not consider the QoS requirements.

The future demand is taken into account in other IaaS resource allocation solutions. Xavier et al. in [60] consider the similarity of future demands with the historical VM allocation data. In this work, they proactively allocate required VMs hosting the required components such as an encoder, decoder, and transcoder. The resource allocation in this work is also in terms of VMs. Gong et al. [61] also propose an elastic resource scaling solution that considers future resource demands as well as real-time demands. The aim of this work is to minimize the cost of resources. They derive a pattern window from the historic resource usage and use that in their demand prediction of a window time-slot ahead. While [60] considers QoS, [61] does not consider this requirement.

(iv) NFV Resource Allocation Solutions

There are several works done in NFV resource allocation domain. Herrera and Botero in [62] present a comprehensive survey on NFV architecture and its resource allocation problems. They define different optimization strategies for NFV resource allocation, followed by emerging challenges. The reviewed works are focused on optimizing the Virtual Network Functions (VNFs) placement in the network and not focused on scaling based on the fluctuating demands.

Other researchers such as Fei et al. in [63] and Wang et al. in [64] focus on scaling the VNFs and considering the fluctuations in the demands of a service. In [63], they propose a proactive approach for provisioning VNFs by using traffic prediction. The goal of this work is to instantiate fewer VNFs to reduce cost. Also, they use online learning to intelligently scale VNFs to cope with traffic fluctuations. The authors in [64] propose an online deployment of VNF chains and dynamic scaling in response to changes in traffic. Similar to [63], the goal of [64] is to reduce the cost by deploying a minimum number of VNFs. However, they also consider VNF placement and minimizing network congestion. The scaling in these works is in terms of a VNF instance and they do not consider increasing or

decreasing the resources of existing VNFs. Dieye et al. in [39] introduce a cost-efficient proactive VNF placement for CDNs. In this work, the location of end-users as destinations are known in advance while the location of their surrogate servers (i.e., media sources) are not known. Similar to [63] and [64], they do not consider scaling in an elastic manner of resources in the existing VNFs.

(v) Traditional Resource Allocation for Conferencing

There are some resource allocation solutions for peer-to-peer (P2P) conferencing and centralized multimedia conferencing [65]. Yuen and Chan [66] reduce worst-case video transmission delay from different video sources to users. They propose an algorithm to select peers as mixers to achieve minimum overall delay. However, their algorithm does not account for media handling response time. Chen et al. [67] also propose a P2P multi-party video conferencing solution to achieve a low end-to-end delay. They optimize the streaming rates of all peers subject to network bandwidth constraints. Their study reduces the end-to-end delay without tackling the specifics of media handling services. Multipoint Control Unit (MCU) [68] is a media handling component that can include different media handling functionalities. Traditionally, all requests are handled by a single MCU, where resources are allocated in a static manner. Thus, this approach is not scalable and uses resources inefficiently.

2.4. Conclusion

As it was discussed in this chapter, there are some existing works done close to this research area. However, none of them satisfy all the requirements of multimedia conferencing applications. Table 2.1 and 2.2 summarize the evaluation of the related work with respect to the mentioned requirements. The check marks in these tables indicate that the requirement is met in the related work.

Table 2.1. Summary of the architectural related work

		Publish and discovery	Composition	High level PaaS northbound interface	Conference rooted IaaS interface	Elasticity, Efficient use of resource	Scalability	Meeting the QoS
Architectural Related Work	[18]	–	–	✓	–	–	–	–
	[19]	–	–	✓	–	–	–	–
	[20]	–	–	–	✓	✓	✓	–
	[21]	–	–	–	–	✓	✓	–
	[22]	–	–	–	–	✓	✓	–
	[23]	–	–	–	–	✓	✓	–
	[15]	–	–	✓	–	✓	✓	–
	[24]	–	–	–	–	✓	✓	–
	[25]	–	–	–	–	–	✓	–
	[26]	–	–	–	–	–	✓	–
	[27]	–	–	–	–	–	✓	–
	[16]	–	–	–	–	–	✓	–
	[35]	✓	✓	–	–	–	–	–
	[36]	✓	–	–	–	–	–	–
	[37]	✓	–	–	✓	–	–	–

Table 2.2. Summary of the algorithmic related work

		Elasticity, Efficient use of resource	Scalability	Meeting the QoS
Algorithmic Related Work	[40]	–	✓	✓
	[41]	✓	✓	–
	[42]	✓	–	–
	[30]	–	✓	✓
	[31]	–	✓	✓
	[46]	–	✓	✓
	[47]	✓	✓	–
	[48]	✓	✓	–
	[49]	–	–	✓
	[50]	✓	–	✓
	[51]	✓	–	✓
	[52]	–	✓	–
	[53]	–	✓	–
	[54]	–	✓	✓
	[55]	–	✓	✓
	[56]	–	✓	–
	[57]	–	✓	–
	[58]	✓	✓	–
	[59]	✓	✓	–
	[60]	–	✓	✓
	[61]	✓	✓	–

		Elasticity, Efficient use of resource	Scalability	Meeting the QoS
	[63]	✓	✓	–
	[64]	✓	✓	–
	[39]	✓	✓	–

Chapter 3

3. A Cloud-based Architecture for Multimedia Conferencing

3.1. Introduction

This chapter proposes a holistic conferencing cloud architecture that provides novel application programming interfaces (APIs) to simplify the provisioning of the conferencing applications for a wide range of application providers (experts vs. non-experts). It also describes the process of composing a complex conferencing service from the basic conferencing services (e.g., signaling, video mixing, and compressing). Service composition can be done in orchestration and choreography approaches. The choreography defines the sequences and conditions where different independent services exchange data while orchestration defines the sequences and conditions where one service invokes other services [69]. In this chapter, we entitled the basic conferencing services as Conferencing Substrate. The Proposed architecture allows the application providers to utilize the offered conferencing services without having to deal with the complexities of conferences.

This architecture is based on the business model in [2], which introduces six roles: connectivity provider, broker, conferencing substrate provider, conferencing infrastructure provider, conferencing platform provider and conferencing service provider. This work reuses and extends this business model by adding a new role, entitled as the conferencing application provider. It also assumes that the conferencing infrastructure provider plays the role of the substrate provider too. In this architecture, the infrastructure provider exposes the conferencing substrates as services (SubaaS) to the platform (i.e., PaaS) provider. The PaaS provider offers high-level APIs to create innovative conferencing services and it enables the on-the-fly composition of SubaaS into full-fledged conference services. The conferencing application providers reuse the conferencing services offered as SaaS in building new applications. They also use PaaS to update the running conferences in their applications at runtime (e.g., switching from audio conference to audio/video conference) without stopping the ongoing conferences.

The rest of this chapter is as follows. First, the motivating scenario for this work is described. Later, the architectural principles are presented followed by the proposed architecture. Then, the service composition will be discussed. Finally, we discuss the implementation architecture, followed by the measurements and conclusion of this chapter.

3.2. Motivating Scenario

Fig. 3.1 depicts the motivating scenario. There are conferencing application providers that use conferencing services offered as SaaS to develop their applications. Three conferencing applications are provisioned: (1) an online game that allows dial-in audio conferencing between the game players, (2) a distance learning application that enables dial-out audio conferencing between students and teachers, and (3) an online meeting application that offers dial-out video conferencing with floor control. The conferencing service providers in the scenario use the conferencing PaaS to provision the conferencing services these applications are based on. One service provider offers Conferencing Service "A" that supports both dial-in and dial-out audio conferences. The distance learning and the game applications utilize Service A. Another conferencing service provider offers a dial-out video conference service with floor control, i.e., Service B. This second service is used by the online meeting application.

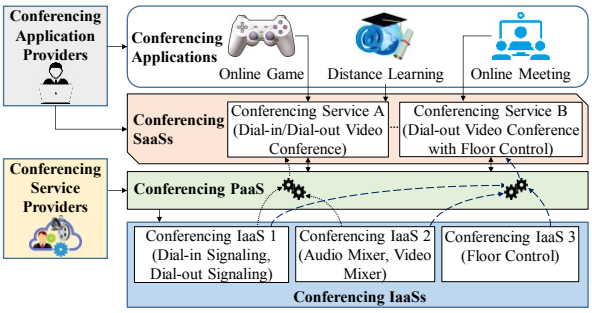


Fig. 3.1. Scenario for conferencing application provisioning in the cloud

The conferencing SaaSs create new conferences when they receive corresponding requests from the conferencing applications. For example, Service A creates a dial-in audio conference when it receives a request from the game application. To run the conference, the PaaS finds the appropriate SubaaSs (i.e., dial-in signaling and audio mixer in this example), composes them, and requests the relevant IaaS(s) to create and activate an adequate instance of each substrate (e.g., the audio mixer with the capability of supporting 500 users). The SubaaSs involved in a given composed conference application may belong to different substrate/IaaS providers. As the players join and leave a conference, PaaS scales the conference up and down in terms of the number of participants. Then, the conferencing IaaS should scale the corresponding instances up and down in terms of the virtualized hardware (e.g., CPU, RAM, and Storage) and software (e.g., the number of running instances of each substrate). Scaling in both layers is done in an elastic manner.

3.3. Proposed Conferencing Architecture

In this section, the architectural principles are presented. Then, the architectural components and service development APIs are discussed in detail, followed by an illustrative scenario.

3.3.1. Architecture Principles

The first principle is to adopt the orchestration approach for the SubaaS composition because it provides PaaS with a greater control on the substrates and their interactions. In fact, orchestration allows a central entity to control different services and their interactions. The second principle is to use high-level PaaS/IaaS interfaces rooted in the conferencing substrates. This principle enables PaaS to request IaaSs for scaling conferences in terms of conference concepts (e.g., the number of participants) rather than VM or the container resources. The third principle is to leverage the existing PaaSs and IaaSs. This allows reusing the existing solutions for the conferencing PaaS and IaaS implementation. The last principle is that the conferencing IaaSs expose substrates as RESTful web services.

3.3.2. General Architecture

The proposed cloud-based conferencing architecture, as shown in Fig. 3.2, includes two main layers (i.e., PaaS and IaaSs) and a broker. The figure also shows the conferencing service providers, the conferencing applications, and the conferencing application users referred to as the conference participants. Note that PaaS may need to communicate with multiple IaaSs to provision a given conferencing service.

(i) PaaS Components:

The PaaS layer consists of six components, which deal with two key facets: 1) conferencing service provisioning and utilizing, and 2) conference management.

a) Conferencing Services Provisioning and Usage

This facet covers conferencing SaaSs development, deployment, and management in addition to conferencing SaaSs utilizing. It includes four components. The Conferencing Service Provisioning APIs component offers high-level APIs to the conferencing service providers, for easy provisioning of new conferencing SaaSs. It also allows the SaaSs providers to make their services available to the application developers via publishing them into a PaaS local service repository.

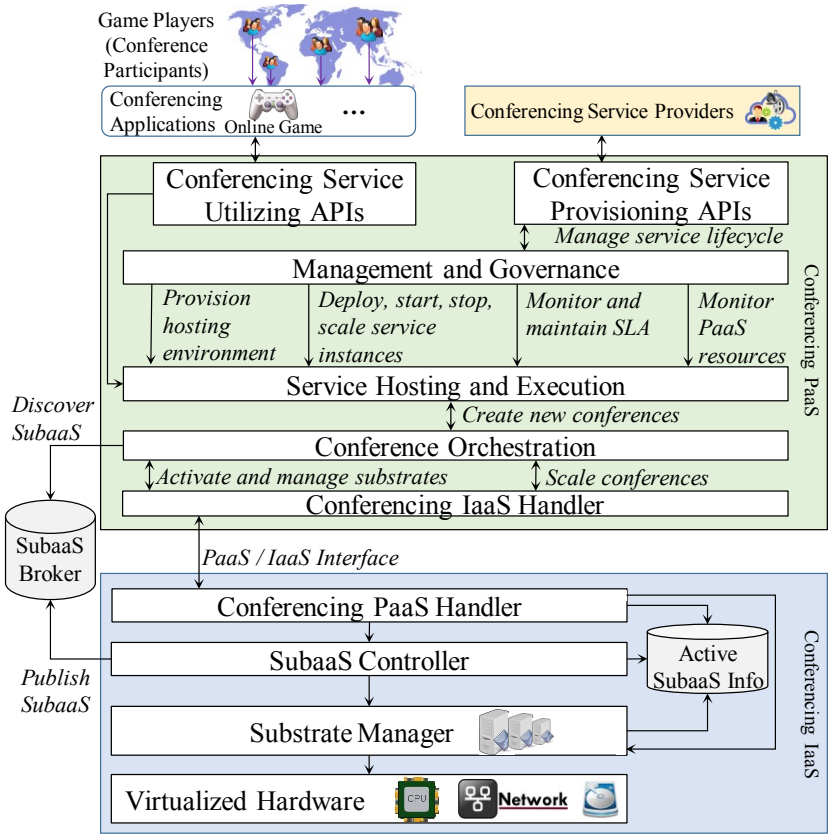


Fig. 3.2. Overall cloud-based conferencing architecture

The Conferencing Service Utilizing APIs provides high-level APIs for conferencing application providers, to discover (from the local service repository), reuse, and control the existing conferencing SaaSs.

The Management and Governance component manages the conferencing services and monitors their QoS and SLAs during service execution. It deploys and executes new services in the Service Hosting and Execution component, upon receiving the requests from the conferencing Service Provisioning APIs.

The *Service Hosting and Execution* component hosts the conferencing services. It allocates necessary PaaS resources (e.g., server runtime and database drivers) and prepares the execution environment before hosting.

Note that the *Conferencing Service Provisioning* and *Utilizing APIs* are the extensions of the application provisioning front-end available in regular PaaS architectures. The

Management and Governance, as well as the *Service Hosting and Execution* components are reused from the conventional PaaS architectures.

b) Conference Management

This facet concerns the management of the actual conferences (i.e., the virtual rooms where people can meet and communicate). It encompasses conference creation as well as the management of the created conferences (e.g., scaling the size of a conference to support more participants). The main component of this facet is *Conference Orchestration* with the following five tasks: First, it determines the necessary substrate types and their associated requirements by using, for instance, syntactic matching with the categorized API parameters. This task starts upon receiving the execution or modification request for a specific conferencing SaaS. Second, based on the determined types and requirements, it discovers the most suitable conferencing SubaaSs from the broker. The existing algorithms for cloud service selection, such as [70], can be reused in this context. Third, it orchestrates conferences from the selected SubaaSs and executes them. Note that conferences are executed in this component. In contrast, the conferencing SaaSs that create conferences are executed in the Service Hosting and Execution component. Fourth, it manages the composed conferences. For example, it can add the video mixing ability to a conference or remove it from it. Fifth, it monitors the running conferences to make decisions if any scaling is required. For instance, if the number of participants in a conference increases, it decides to scale the conference size. Thus, it requests the conferencing IaaSs to scale the corresponding substrates to cope with the new workloads.

Another component under this facet is the Conferencing IaaS Handler, which is in charge of communications between the conferencing PaaS and the conferencing IaaSs. For instance, a scaling request initiated by the Conference Orchestration component is sent to the corresponding conferencing IaaSs through the Conferencing IaaS Handler. Note that Conference Orc*hestration* is a novel component while *Conferencing IaaS Handler* is an extension of IaaS communication component in conventional PaaS architectures.

(ii) IaaS Components:

The IaaS layer consists of five components, dealing with two key facets: 1) resource management and 2) SubaaS management.

a) Resource Management

This facet is in charge of providing the required resources in order to run a substrate. The Virtualized Hardware is one of the components in this facet. It has a pool of typical virtualized IaaS resources such as CPU, Network, and Storage. The second component of this facet is Substrate Manager with three main tasks: First, it creates and hosts resources in order to run the substrates. These resources can be a VM or a container [71] that uses virtualized hardware to host a substrate. Each substrate can be hosted on one or many VMs or containers (e.g., two instances of the same substrate may be activated in two different machines). In addition, each VM or container may host more than one substrate. The second task is modifying the allocated resources upon receiving the scaling request for a substrate. For instance, to scale up a running substrate, it can add some virtualized hardware to the VM that hosts the target substrate. The third task is inserting and updating the information of all running substrates in a repository called Active SubaaS Info.

b) SubaaS Management

This facet includes the managing functionalities to offer substrates as services. The first component of this facet is the Active SubaaS Info. It is a repository that keeps information about all running SubaaSs. For instance, for each running SubaaS, it keeps the related conference ID, IP of the VM(s) or container(s) hosting that substrate, etc.

Another component of this facet is SubaaS Controller. This component has two main tasks. First, it decides how and when to scale a running substrate, based on the Service Level Agreements (SLAs) between the PaaS and IaaS (e.g., end-to-end delay should be less than 400 msec). Upon receiving the scaling request from the PaaS and its required QoS, it uses the stored information in the Active SubaaS Info repository to make the scaling decisions. The resource allocation algorithm and video mixing procedure that we will present in the next chapter (chapter 5) are used for this purpose. Second, it maintains a repository of all available substrates in the IaaS. It selects a suitable substrate from this

repository when it receives a request to create and start a substrate. It then instructs the Substrate Manager to create the actual resource. Moreover, it publishes the information of SubaaSs in the broker.

The third component of this facet is Conferencing PaaS Handler, which is in charge of all communications between the PaaS and IaaS layers. This component has two main tasks: First, it receives and dispatches the PaaS requests (e.g., to create a substrate and scale up a substrate) to the appropriate IaaS components and forwards the IaaS replies to the PaaS. Second, it handles the conference participants' requests (e.g., joining a conference). The participants' requests are sent from the conferencing applications to the PaaS, which forwards them to the conferencing IaaS. The Conferencing PaaS Handler, in collaboration with the Active SubaaS Info repository, identifies the appropriate substrates and forwards the requests to them. This feature increases the level of abstraction for the substrates working in a single conference. Moreover, there is no need to update the participants on any changes in the substrates' hosting resources.

(iii) Broker:

The Broker lists the SubaaSs offered by different IaaSs. The SubaaSs description is semantic-based to allow for rich descriptions and queries. It includes high-level information such as the type of service, QoS parameters, and cost. In this paper, we reuse the description model and the broker publication and discovery interfaces from [37].

3.3.3. Conferencing Service Development APIs

Three principles are followed to design the proposed APIs. The first principle is leveraging basic conferencing concepts (e.g., conference, participant, media, and floor) in the API design. This helps in achieving an abstraction level higher than conferencing components (e.g., signaling, media mixer, and media transcoder) and their complex interactions. The second principle is categorizing API parameters, which helps service providers to easily understand conference mandatory and optional aspects, required API parameters for each aspect and dependencies among parameters. The third principle is the

use of RESTful design. It is standard-based, lightweight and flexible for data representation, which allows describing the APIs in a generic way.

Table 3.1 delineates four API examples. It shows some of the REST resources along with an example operation for each. The request parameters and the response contents are also listed. Showing the categorization of API parameters, Table 3.2 highlights that a service provider has to specify one conference model, at least one media and the conferencing technology. It also shows the conditional dependencies of parameters. For example, for WebRTC-based conferencing [72], the signaling protocol must be specified. In this table, the parameters that the service providers can change during the runtime are italicized.

3.3.4. Service composition

As per our first design principle, the conferencing services are composed of SubaaSs using the orchestration approach. The *Conference Orchestration* component of the PaaS plays the role of the central entity that invokes and controls the composing SubaaSs.

In addition to the composition approach, two other composition aspects are considered: binding dynamicity and automation level [31]. Since the PaaS discovers, selects, and activates the composing SubaaSs on the fly, dynamic binding to IaaSs (i.e., SubaaS

Table 3.1. Examples of conferencing service development APIs

REST Resource	Operation	HTTP action and resource URI	Request body parameters	Most important info in response
List of Conferences	Create conference	POST: /conferences	Conference model, Media, Conference technology, floor control, conference size, QoS requirements	ID and URI of the created conference resource
List of participants	Add participant	POST: /conference/ {conferenceId}/ participants	Participant description: name, URI	ID and URI of the new participant resource
List of floors	Add floor	POST: /conferences /{conferenceId} /floors	Floor description: chair, floor participants	ID and URI of the newly created floor resource
Specific sub-conference	Remove subconference	DELETE: /conferences /{conferenceId} /subconferences /{subconferenceId}	None	Success or failure indication

Table 3.2. Categorization of API parameters

<table>
<tr><th></th><th>Categories of Parameters</th><th colspan="3">Example Values</th></tr>
<tr><td rowspan="11">Mandatory Aspects</td><td rowspan="3">Conference Model</td><td rowspan="2">Pre-arranged conference</td><td colspan="2">Dial-in conference</td></tr>
<tr><td colspan="2">Dial-out conference</td></tr>
<tr><td colspan="3">Ad-hoc conference</td></tr>
<tr><td>Media</td><td colspan="3">At least one of audio, video, and text</td></tr>
<tr><td rowspan="7">Conferencing Technology</td><td rowspan="3">SIP-based</td><td>Signaling protocol</td><td>Default protocol: SIP. No need to specify</td></tr>
<tr><td>Audio encodings</td><td>Default encoding: NULL. It should be specified</td></tr>
<tr><td>Video encodings</td><td>Default encoding: NULL. It should be specified</td></tr>
<tr><td rowspan="3">WebRTC-based</td><td>Signaling protocol</td><td>Default protocol: NULL. It should be specified</td></tr>
<tr><td>Audio encodings</td><td>Default encoding: G.711 and Opus. Can specify additional</td></tr>
<tr><td>Video encodings</td><td>Default encoding: H.264 and VP8. Can specify additional</td></tr>
<tr><td>Hybrid (SIP + WebRTC based)</td><td colspan="2">Mandatory protocols and encodings from both technologies apply. Can specify additional</td></tr>
<tr><td rowspan="2">Optional Aspects</td><td>Floor control</td><td colspan="3">At least one floor control policy, e.g., chair-moderated or round-robin</td></tr>
<tr><td>Subconference</td><td colspan="3">Enabled or not</td></tr>
</table>

providers) is required. As for the automation level, the semi-automated approach is adopted to take advantage of more mature and widely used techniques, such as workflow.

In this work, the conferencing PaaS provider develops a generic workflow template for the composite conference, considering the various substrate types that may be required. It uses a workflow automation tool (e.g., Activiti [73]) to ease and speed up the process. When the *Conference Orchestration* component selects the SubaaSs to be composed (i.e., at runtime), it creates an instance of the workflow template and then configures the instance with the selected and activated substrate instances. Thus, the conference is dynamically bound to its composing substrate services. This dynamic binding makes it possible and easy to change the substrates used by an ongoing conference at runtime if needed. Note that a

PaaS provider may define multiple workflow templates and then select the most suitable one based on the required substrate types and the rest of the users' requirements.

3.3.5. Illustrative Scenario

The illustrative scenario consists of (i) an online game application where players can talk for unlimited time but can have private text chat for only 5 minutes per hour, (ii) a service provider that offers dial-in audio conferencing as SaaS with text chat for a limited time and (iii) a conferencing PaaS that subscribes to three conferencing IaaSs: A, B and C, which offer dial-in signaling, audio mixing and instant messaging SubaaSs respectively. The scenario illustrates how the conferencing PaaS creates a conference when the game application sends a request to the conferencing SaaS and how the conferencing IaaSs allocate the resources.

Fig. 3.3 shows the interactions. For brevity, the game application is omitted in the figure. Using the Conferencing Service Utilizing APIs, the game application developer finds the offered conferencing services and requests for conferencing SaaS A. When conferencing SaaS A receives the game application request for creating a conference, it invokes the *create conference* API (step 1). The API handling is delegated to the *Conference Orchestration* component, which determines necessary substrate types (step 2) and finds appropriate

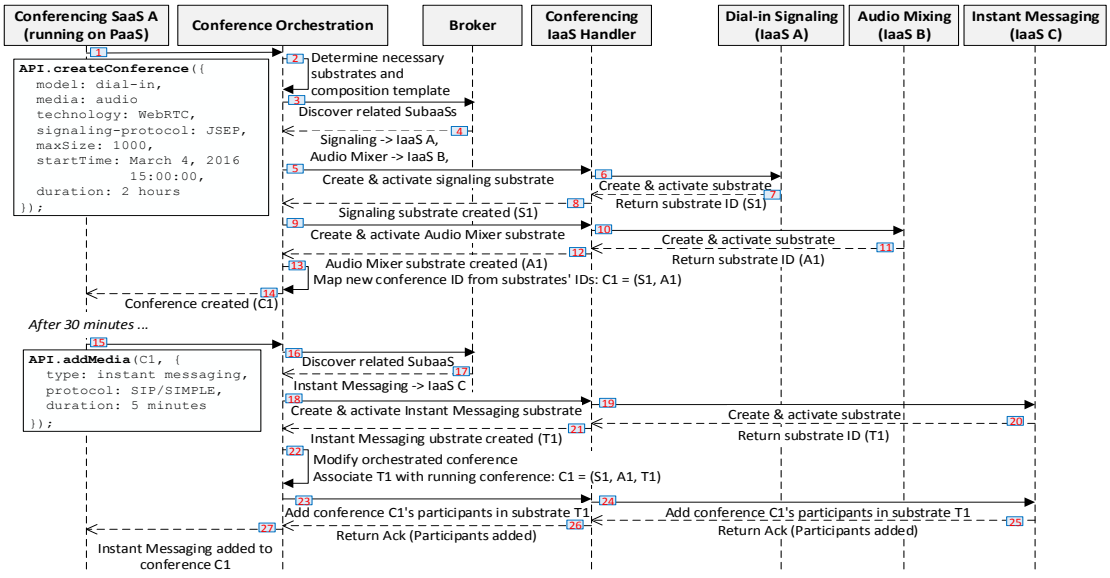


Fig. 3.3. Conference creation and modification steps

SubaaSs through the broker (step 3). In this scenario, the dial-in signaling and the audio mixing SubaaSs are selected from IaaSs A and B respectively (step 4).

Next, the PaaS requests the IaaSs, via the *Conferencing IaaS Handler*, to activate the substrates (steps 5 to 12). For activation, the *Conferencing PaaS Handler* component in the IaaS receives the request and forwards it to the *SubaaS Controller*. The latter selects the requested substrate's code from its repository and sends the required information to the *Substrate Resource Manager* to allocate the required resources (e.g., it selects the audio mixer code that can handle 200 participants and asks the *Substrate Resource Manager* to create and run a new VM to accommodate 200 participants, install the substrate code on the VM, and run the code to initialize and activate the audio mixer as a substrate).

After activating the substrates, the *Conference Orchestration* binds the SubaaSs in the composing template (selected in step 2) and then executes the new dial-in audio conference (step 13). The orchestrated conference represents a full-fledged conference. Finally, the ID of the full-fledged conference is returned to the game (step 14).

It is assumed that the conferencing service enables private text chat after 30 minutes. When the timer expires, the service invokes the *addMedia* API to add instant messaging to the conference for 5 minutes (step 15). Thus, the *Conference Orchestration* discovers the appropriate SubaaS from the broker (step 16). It selects IaaS C, activates the instant messaging substrate and modifies the conference workflow to add instant messaging (step 17 to 22). On the new substrate, an individual conference is created for 5 minutes and the existing participants are added to it (step 23 to 26). A notification is sent to the game application (step 27) and the participants can start exchanging text messages. For optimization purposes, the messaging SubaaS can be added to the conference when created and it can be enabled and disabled when needed. Meanwhile, the messaging SaaS can be discovered and added at runtime if, for instance, the original one is no more available. The scenario is showing the latter case.

3.4. Implementation and Measurements

An implementation architecture is first presented. Next, the developed prototype is described.

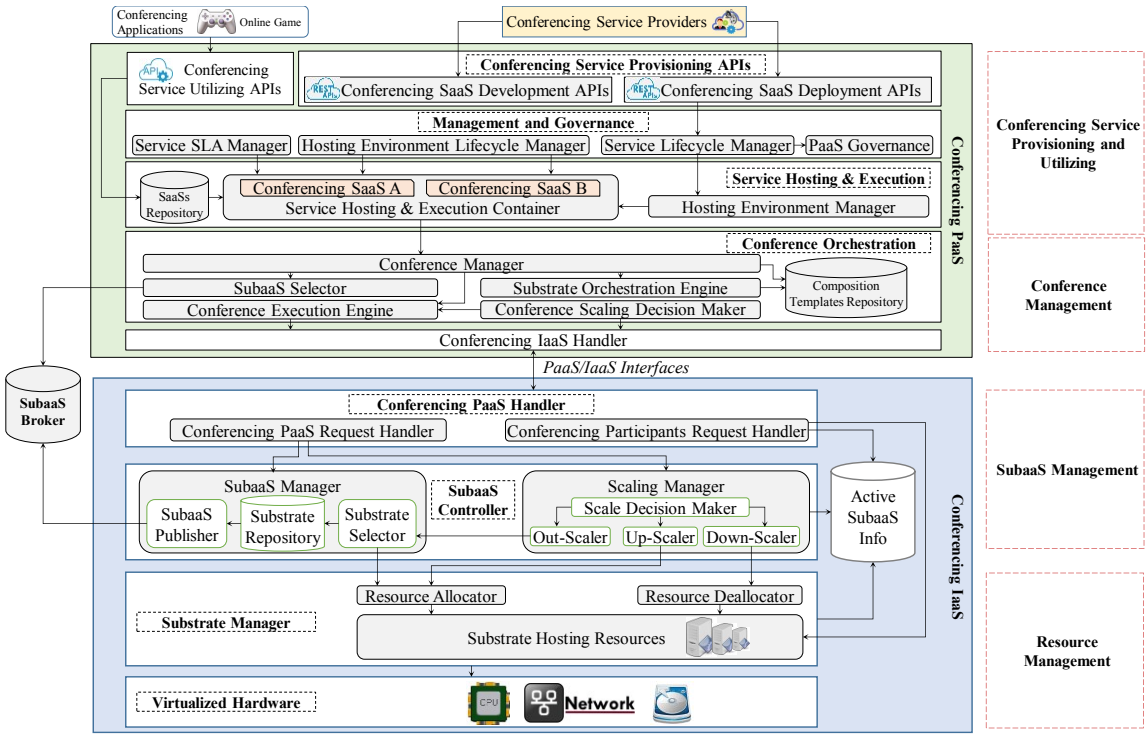


Fig. 3.4. Implementation architecture

3.4.1. Implementation Architecture

Fig. 3.4 shows the implementation architecture including Conferencing PaaS, Conferencing IaaS, and the SubaaS Broker.

(i) Conferencing PaaS

In the Conferencing Service Provisioning APIs component, two sets of REST APIs are developed: Conferencing SaaS Development APIs and Conferencing SaaS Deployment APIs. These are used for service creation and deployment respectively. The Conferencing Service Utilizing APIs have been also implemented as REST APIs. Management and Governance and Service Hosting and Execution components are not discussed here as they are reused from the conventional PaaS architectures.

The Conference Orchestration component uses a repository to store the workflows of composing templates. The Conference Manager in this component receives the northbound requests for running conferences, selects an appropriate template from the repository, and determines the required substrates for the conference. It then sends that information to the SubaaS Selector and the Substrate Orchestration Engine.

The SubaaS Selector chooses the most suitable conferencing SubaaS from the SubaaS Broker, given the substrate requirements. The discovery mechanism and the interfaces between these two are reused from the existing work [37]. The Substrate Orchestration Engine uses the chosen template to compose the selected substrates and deploy it in the Conference Execution Engine that hosts the running conferences. The Conference Scaling Decision Maker monitors the running conferences and requests scaling when needed.

(ii) Conferencing IaaS

The Conferencing PaaS Handler includes two components: Conferencing PaaS Requests Handler and Conferencing Participants Request Handler. These are used to process the requests initiated by the PaaS and by the Conference Participants (i.e., the users of conferencing applications), respectively. These requests are of three types: (1) to create and activate a conference; (2) to scale a specific conference (e.g., change the conference size); and (3) to join and leave a conference. The first two are initiated by the PaaS while the third is used by the participants. Both handlers are implemented using REST APIs.

The conference creation and activation requests are sent to the SubaaS Manager in the SubaaS Controller component. The SubaaS Manager uses the Substrate Selector to choose the appropriate substrates for creating the new conference. Also, it uses the SubaaS Publisher to publish the existing SubaaSs to the Broker.

The conference scaling requests are forwarded to the Scaling Manager in the SubaaS Controller component. It relies on a Scale Decision Maker to decide how to scale the conference. The decision maker first fetches the information about the conference-related SubaaSs from the Active SubaaS Info (e.g., the IP of the hosting VM(s)/container(s) and the information of the server(s) hosting those substrates, such as available RAM, CPU, etc). Then, based on this information and the new scaling requirements, the decision maker determines which substrate(s) should be changed and how (i.e., scale up/out/down). It then instructs the appropriate component to do it (i.e., Up-Scaler, Out-Scaler, and Down-Scaler). For instance, if the requirement is to update an audio conference with 50 users to support 100 users and the current server hosting the audio mixing substrate does not have enough

resources, the decision is to scale out the audio mixing substrate on another server. Thus, a new VM or container will create on another server to host the audio mixer substrate.

For the Substrate Manager, we use the OpenStack Compute (Nova) layer. It creates and updates VMs/Containers to host the running substrates. It allocates or deallocates resources based on the incoming requests from the SubaaS Manager and the Scaling Manager. It also keeps the Active SubaaS Info up-to-date after each operation.

3.4.2. Prototype

The prototype scenario includes a service provider offering dial-in audio conferencing service and a game application utilizing that service. It also includes the conferencing PaaS and two conferencing IaaSs – both providing dial-in signaling and audio mixer substrates.

In this prototype, the Cloud Foundry PaaS is used to provide the implementation of typical PaaS components. We also extend it to implement our novel component (i.e., Conference Orchestration). For Substrate Orchestration Engine and Conference Execution Engine, we use Activiti [74], a light-weight workflow and Business Process Management (BPM) platform. Conference Manager and Conferencing IaaS Handler are implemented using Express.js framework [75]. Advanced REST Client [76] is also used to simulate SaaS APIs invocation by the game.

For the conferencing IaaS, OpenStack [77] is used. Conferencing PaaS Handler is implemented as a Java application with REST-based APIs to communicate with the PaaS. The open source framework Asterisk [78] is used for signaling, media handling, and floor control substrates. To publish a SubaaS information, we implement a subset of the model proposed in [37]. Our published SubaaS information is shown in table 3.3. For the Scaling Manager, we use the proposed resource allocation mechanism in chapter 4.

3.4.3. Validations and Measurements

To validate our architecture, we run the implementation according to the steps presented in Fig. 3.3. Fig. 3.5 shows the Activiti orchestration process to create a dial-in audio conference. The workflow execution corresponds to steps 5 to 14 in Fig. 3.3. Implementing

Table 3.3. Published information of a SubaaS into the broker in our implementation

Categories of Parameters	**Example Values**		
Type of Service	Signaling	Signaling Protocol	Acceptable signaling protocols (e.g., SIP)
		Conference Model	Acceptable conference models (e.g., Dial-in, Dial-out, Ad-hoc)
	Media Handling	Audio Mixer	Acceptable audio encodings (e.g., G.711)
		Video Mixer	Acceptable video encodings (e.g., H.264)
	Conference Control	Floor Control Policy	Supported control policies (e.g., round-robin)
		Floor Chairs	Maximum number of acceptable chairs (e.g., 5)
Service Endpoint	URI	The domain or IP of the service, along with the port number (e.g., http://audiomixer.com:263)	
Service Limit	Maximum number of accepting participants	Zero means there is no limit on maximum number of accepting participants	
Service Integration Templates	Workflows	The bpmn20.xml files that describes how this service can integrate with some other known services	

the dial-in audio conference service using Activiti proved the simplicity of service creation, which is very useful for non-expert developers. Indeed, while expert conferencing service providers can use offered APIs to create their provisioned services, non-expert providers can use an orchestration tool to provision their services. Fig. 3.6 shows the parameters sent back to the game application after the workflow execution.

Three experimental environments are considered for performance measurements: 1) A Non-Cloud Conferencing (NCC) environment, where resources are allocated beforehand. 2) A Monolithic IaaS Provider (MIP) environment, where an IaaS offers multiple substrates in a single SubaaS (i.e., the SubaaS is composed of multiple coupled substrates). Thus, the IaaS hosts all substrate instances on the same VM. This is the same if several SubaaSs from the same IaaS run on the same VM. 3) A Non-Monolithic IaaS Provider (NMIP) environment, where IaaS offers every single substrate as a separate service. In NMIP, the IaaS hosts substrate instances on separate VMs.

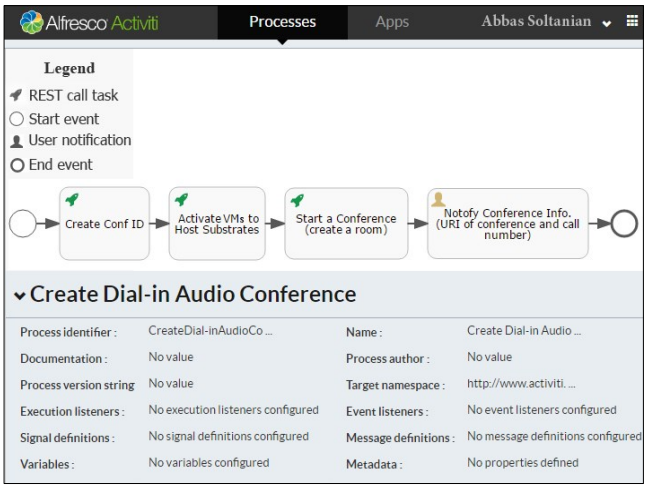


Fig. 3.5. Dial-in audio conference creation and activation workflow

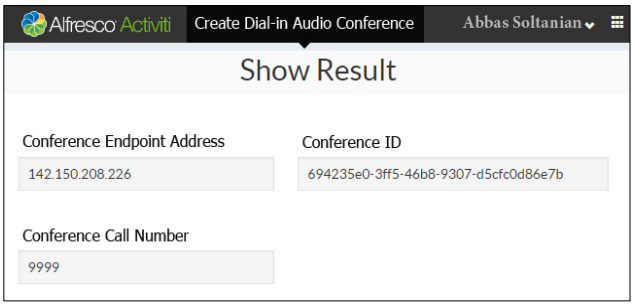


Fig. 3.6. Conference information which passed to the game application

The following four metrics are used: (1) Resource Allocation – the total amount of allocated resources, such as memory and CPU, to accommodate all participants, (2) Scale Time – the time to add resources to scale the conference, (3) Conference Start Time – the time to get a conference ready upon the receipt of a request and (4) Participant Joining Time – the time to add a participant to a running conference.

To analyze the allocated resources, we consider a conferencing application with considerable fluctuation. A good example of such application is a massively multiplayer online game (MMOG) which offers the audio/video conferencing. This kind of application may include thousands or even millions of players who share their audio and video in the logic of the game. For example, the study in [3] reported that the number of users in World of Warcraft (a famous online game) fluctuates between 1.5 and 2.5 million over 10 hours. Fig. 3.7 shows the allocated amount of memory (i.e., RAM) for a conference when the number of participants fluctuates between 1 and 3000. To simulate this fluctuation, we

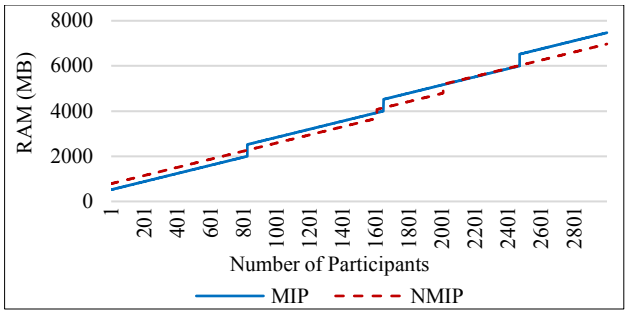


Fig. 3.7. Resource Allocation Evaluation

increase the conference size by 200 participants every 10 minutes. The results are based on the observed resource usage per participant. The Scale Decision Maker in IaaSs scales the VMs up and out while maintaining the QoS requirements. Two QoS requirements are considered: 1) the end-to-end delay which includes the audio and video mixing time should not take more than 400 msec and 2) the amount of allocated resources should be minimized. The VMRA resource allocation algorithm presented in chapter 4 is used for this prototype.

In NCC, there are always some idle and non-utilized resources because of upfront resource provisioning. Hence, we do not show the NCC allocated resources in Fig. 3.7. As it is depicted in this figure, MIP scales better than NMIP (i.e., it allocates fewer resources) for smaller conferences whereas NMIP wins for bigger conferences. In NMIP, the substrates are hosted on separate VMs. Thus, for smaller conferences, it leads to more VMs and more non-utilizable resources (e.g., the resources consumed by the operating system) than in MIP. The bigger the size of a conference, the more resources the substrates required to perform well. However, they do not require the same thing; e.g., a signaling substrate may need less extra resources than the mixer because it is only used in the first phase of the conference. In MIP, because of having monolithic SubaaS, the rate of adding resources is the same for all substrates. This results in more scaling out decisions and therefore more VMs. Indeed, by applying the VMRA allocation algorithm, the resources exceed its maximum extra amount for scaling up, which makes scaling out a better decision. By contrast, in NMIP, the resources are allocated to each substrate based on their need, resulting in less scaling out decisions. This makes NMIP achieve better scalability because of the fewer number of VMs and better resource utilization than in MIP.

Regarding CPU usage, we used a 2.6 GHz single core CPU for each VM. The CPU utilization per VM fluctuated between 20% and 80% for each VM in both scenarios. This fluctuation is based on the number of users that are connected to the VM. This shows that VMs' resources are not fully used, in both MIP and NMIP. Therefore, CPU utilization for small conferences is better in MIP, since it has a fewer number of VMs to accommodate users in comparison with NMIP. Similarly, when the size of the conference is big, NMIP has better results because of its fewer VMs usage.

For the Scale Time metric, we observe the scaling performance of the system under two conditions. The first condition demonstrates the behavior of the system when the conference starts with the minimum required amount of resources, i.e., the least possible substrate instances (Fig. 3.8). The second condition demonstrates the behavior of the system under resource over-provisioning situation, i.e., the conference starts with more substrate instances than required (Fig. 3.9). Note that the second experiment is exclusively aimed for demonstrating the impact of the number of substrates on the scaling time. Therefore, in that experiment, the SLA violations are not taken into account; i.e., the amount of allocated resources is not minimized. The provided set of experiments helps the conference service providers to evaluate the tradeoff between over-provisioning and (sub)optimal substrate allocation.

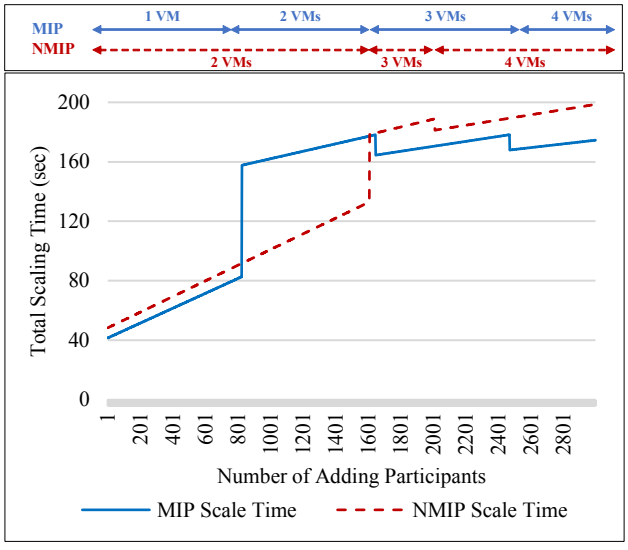


Fig. 3.8. Total Time for Scaling the Size of a Conference with Single Participant to a Conference with 2 up to 3000 Participants

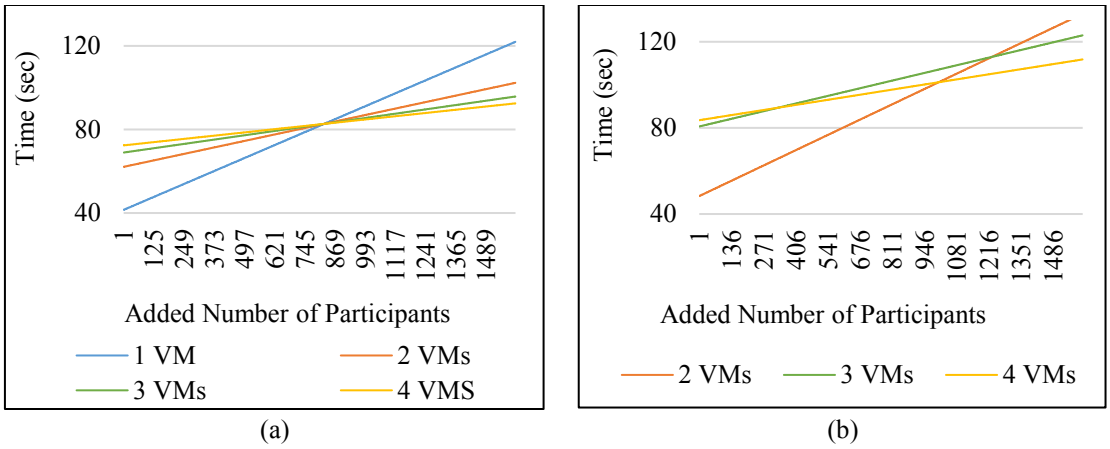


Fig. 3.9. Conference Scaling Time by Having Different Number of VMs for (a) MIP and (b) NMIP

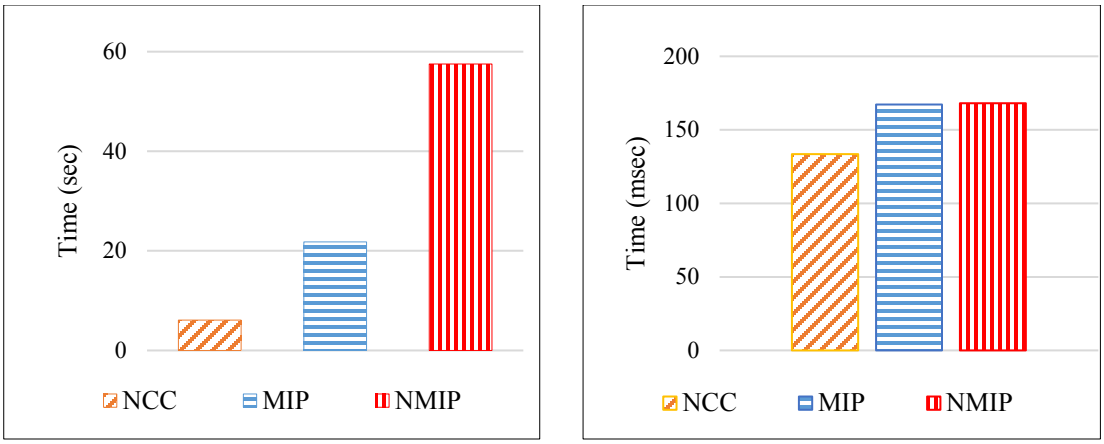
The scaling time under the first condition for both MIP and NMIP scenarios are depicted in Fig. 3.8. In this experiment, we first run a conference with one participant. This conference starts with the minimum required resources (i.e., one VM in MIP and two VMs in NMIP). By increasing the size of the conference, the required resources are added to the existing VMs (i.e., those hosting the substrates). If the required QoSs cannot be satisfied by adding resources to the existing VMs, (e.g., the end-to-end delay is more than 400 msec), the Scaling Manager in the IaaS starts new VMs for hosting another instance of required substrates. We scale the size of the conference between 1 and 3000 participants in this experiment. The scaling time accounts for several parameters, including the time for creating a new VM, the time for adding resources (i.e., RAM in this experiment) to the existing VMs and reconfiguring the system (e.g., updating the list of available audio mixers), and the time for adding all the participants. Basically, the scaling time from 10 to 100 participants, for instance, is the total time for moving from a running conference with 10 participants to a running conference with 100 participants.

As shown in Fig. 3.8, for adding a large number of participants, the scaling time in MIP is lower than that of NMIP. The main reason is, as discussed earlier, MIP creates more instances/VMs than NMIP when the number of participants is large. This makes MIP able to add participants in parallel to several substrate instances/VMs. Besides, although reconfiguring the system with more instances has some overhead, the gain from load balancing makes the scaling time in MIP lower than in NMIP.

Also, MIP gives better results when a limited number of participants is added (scaling between 1 through ~825 in this graph). Adding resources to the existing VMs in both scenarios takes the same time, i.e., it is the exact same process. However, the overhead time of reconfiguring the system with more VMs in NMIP leads to having a longer scaling time. Once MIP starts to create new VMs, the time for creating these VM leads to an increase in the scaling time in MIP (in the middle part of the figure). Based on the results, no matter how many VMs are created, it does not noticeably affect the scaling time in MIP and NMIP because several VMs can be created in parallel.

The case of over-provisioning (Fig. 3.9) shows that, in both MIP and NMIP, the scaling time for a limited number of participants (up to ~800 participants) is less when the number of VMs is less. In contrast, when the number of participants to be added is large, having a conference that is hosted on more VMs results in less scaling time because the participants can join multiple instances/VMs in parallel. Therefore, we can conclude that having more resources does not always lead to having less scaling time in the conferencing domain. In fact, in conferencing, the collaboration between different substrate instances hosted on different VMs causes some overhead. Although increasing the number of substrate instances and balancing the loads between them leads to some saving in scaling time, the overhead of reconfiguring the system might be more than the gain.

Fig. 3.10(a) compares the conference start time in the three studied environments (i.e., NCC, MIP, and NMIP). It shows that NCC takes the least time to start a new conference, which is obvious due to the absence of virtualization overhead. And, since in NMIP the substrate instances are hosted on separate VMs and they need to connect to each other over the network, it takes more time than it does in MIP. However, since starting a conference happens just once, this time is endurable in the Cloud scenarios. Participant joining time is also the least in the NCC as shown in Fig. 3.10(b). Cloud-based scenarios take more time because of the notification overhead between IaaSs, PaaS and the game server. However, this time length remains acceptable (can be seen as the waiting time to join the conference) and is not noticeable by end users. In addition, the participant joining time of the two cloud-based scenarios are close as IaaSs can notify PaaS in parallel.



(a) (b)

Fig. 3.10. Average (a) Conference Start Time (b) Participant Joining Time

3.5. Conclusion

This chapter presents a novel holistic architecture for multimedia conferencing applications. This architecture covers both PaaS and IaaS layers of cloud. The proposed architecture simplifies the provisioning of the conferencing applications for expert and non-expert application providers by providing novel APIs. It also supports scaling the conferencing applications in an elastic manner. The conferencing API examples and the categorization of their parameters are presented in this chapter. The implemented prototype and the experiments show the feasibility and validation of the proposed architecture. Although in cloud-based scenarios the conference start time and the participant joining time are more than those in NCC, the cloud-based conferencing architecture helps to scale the system easily and avoids the over-provisioning or under-provisioning of resources. The results of scaling duration and allocated resources help the conferencing service providers with better provisioning of their services. For instance, MIP is a better choice for provisioning small conferencing services (e.g., to support 300 users) as it results in less resource usage and less scaling time than it does in NMIP. However, for a conferencing service with 1200 users, NMIP gives better scaling time and lower resource consumption. In the case of big scenarios (e.g., 3000 users or more), there is a tradeoff between using fewer resources (i.e., NMIP) and having less scaling time (i.e., MIP).

Chapter 4

4. A Resource Allocation Mechanism for Multimedia Conferencing Applications with Video Mixing

4.1. Introduction

In the previous chapter, we proposed a holistic cloud-based architecture for multimedia conferencing applications. In the conferencing IaaS of our proposed architecture, we assumed that there is an efficient resource allocation mechanism that can optimize the allocation of actual resources (e.g., CPU, RAM, and Storage). This chapter proposes a cloud-based resource allocation mechanism for conferencing applications with video mixing. The proposed solution optimizes resource allocation and scales resources in terms of the number of participants while guaranteeing QoS. Fig. 4.1 depicts the assumed business model. It has four main roles: conferencing application providers, conferencing service providers, media handling service providers, and conferencing IaaS providers. In this model, conferencing applications rely on a conferencing service that is offered as a SaaS. Media handling services are also offered to conferencing service providers as SaaSs. The actual resources for media handling services are provided by geographically

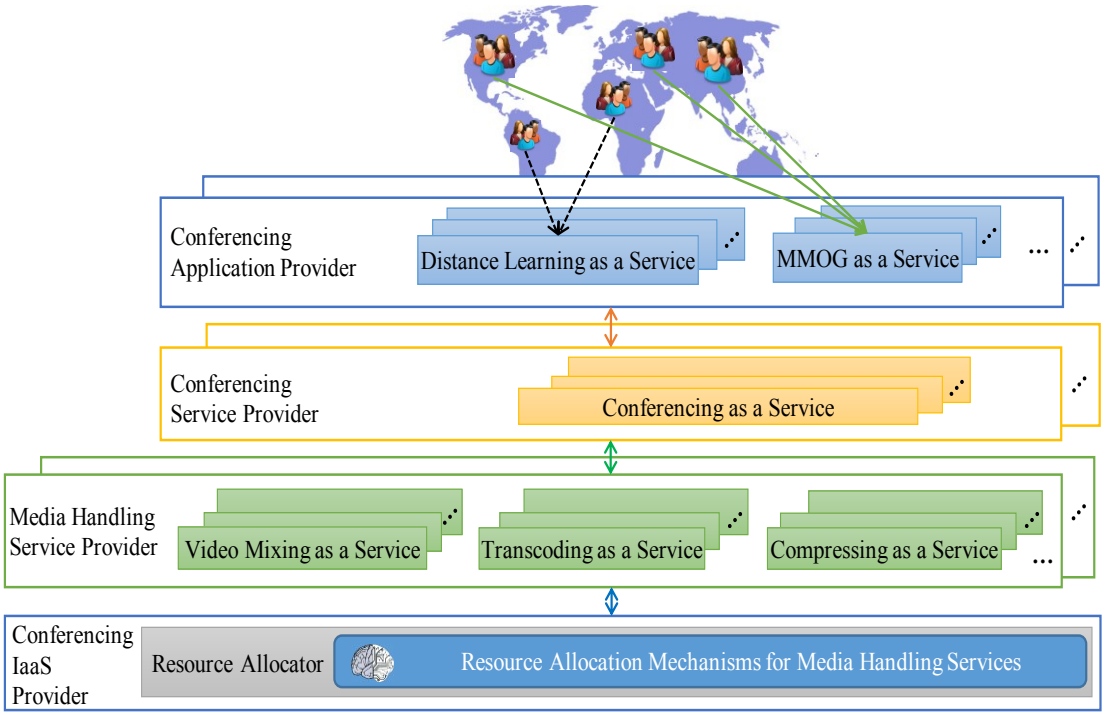


Fig. 4.1. Cloud-based conferencing business model

distributed IaaSs. As it is shown in this figure, the proposed resource allocation mechanism runs in the IaaS.

We entitled the proposed algorithm in this chapter as VMRA (Video Mixing Resource Allocator). In designing the VMRA, we consider the conferencing applications with video mixing. It allocates or deallocates resources for conferencing applications based on the fluctuation in the number of participants. Besides efficient resource utilization, it caters to the QoS, with respect to the video mixing response time. It performs a fine-grained scaling of resources to improve efficiency in resource utilization. We analyze VMRA theoretically by modeling it as an optimization problem. Then, we design the heuristic that can reach the sub-optimal solution for the large-scale scenarios in an acceptable time.

The rest of this chapter is as follows. First, it presents the VMRA by discussing its system model. Then, it discusses the designed heuristic. After that, it presents the simulation parameters and settings of VMRA followed by the validation results. We conclude this chapter at the end.

4.2. VMRA System Model

The system model of VMRA includes cooperation, video mixing, and mathematical models. In our mathematical model, we define VMRA as an Integer Linear Programming (ILP) problem.

4.2.1. Cooperation Model

We consider a large-scale geographically distributed cloud infrastructure to support conferencing applications and video mixing as a service, consisting of users, separate zones and an IaaS in each zone z, as depicted in Fig. 4.2. We illustrate users scattered across a large geographical area, wanting to join a conferencing application, such as MMOG. We assume that in each zone z, there is a data center providing IaaS, where each data center consists of a number of servers (N_z), hosting VMs. Furthermore, we assume that zones are interconnected in a full mesh manner. The same assumption applies to VMs in a data center, as shown in Fig. 4.2.

Users in each zone will connect to their local data center to join a conferencing application. Each user is considered as a video source, sending video and requesting video mixing service. The challenge lies in allocating the resources for video mixing to achieve optimal resource utilization while guaranteeing QoS requirements.

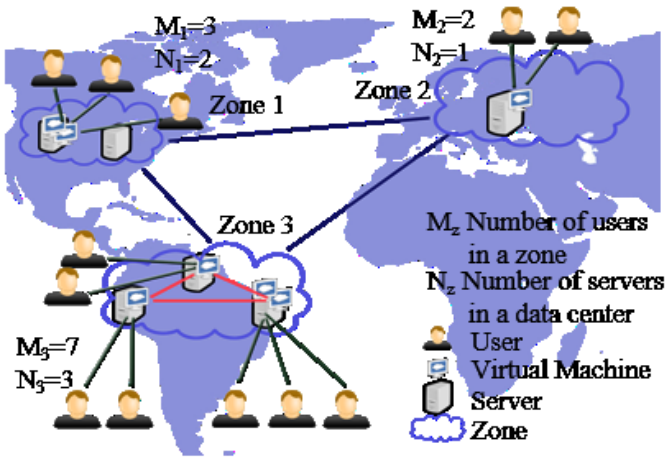


Fig. 4.2. Communication model

4.2.2. Video Mixing Model

VMRA decides to add resources to existing VMs or create a new VM when a video source is added to a data center. Adding resources is done in fine granularity. This implies that VMRA will add minimal required resources in an elastic manner. It will also balance the load between all the VMs in a data center. After provisioning appropriate resources, a video source will join a VM, that is, a video mixer and video mixing will start. The video mixing process is illustrated in Fig. 4.3.

Our video mixing model follows the Fork/Join parallelism technique [79]. All video mixing requests in a data center fork off to several other mixing processes, which are concurrently executed in each VM, until they finally join into a single mixed video. VMs mix their video sources in parallel. Therefore, the required time for this step depends on the maximum number of video sources connected to any VM (V_z) in zone z.

Each VM will send the result to other VMs in the same data center. This intra-zone video exchange time is in T_{int}. Next, each VM mixes the incoming videos from other VMs with the result of its own mixed video source. The time for this step depends on the total number of VMs in the data center.

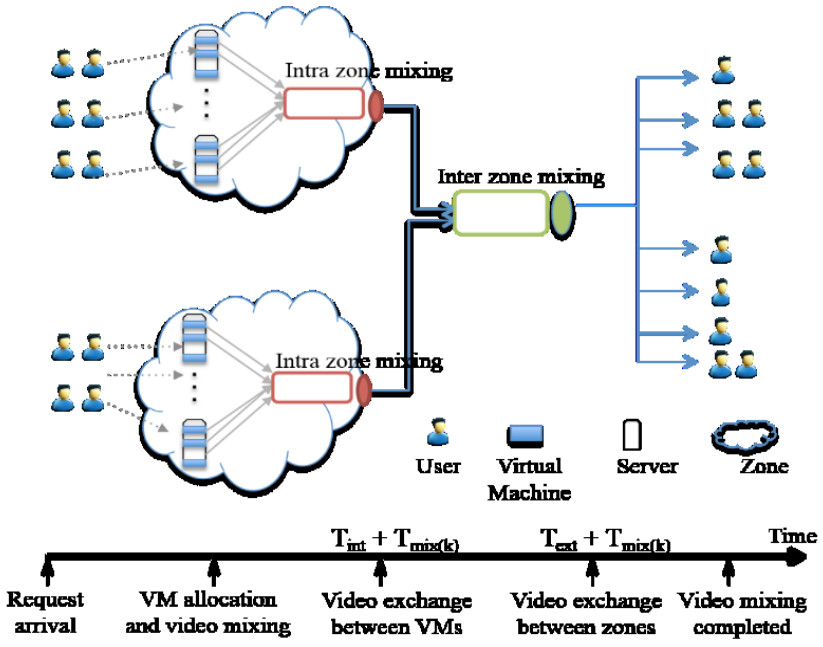


Fig. 4.3. An example of our video mixing model

Here, mixed video of a data center is ready and sent to all other data centers. This inter-zone video exchange time is in T_{ext}. Then, VMs will start mixing the incoming videos from other zones with the one of their own zone. Here, the required time depends on the total number of zones and the mixed video across all zones is ready to be sent back to the users.

4.2.3. Mathematical Model

This subsection presents our VMRA problem formulation, which is modeled as an ILP problem. It presents the problem statement followed by the objectives and constraints.

(i) Problem Statement

Given a data center with N_z servers and M_z users (video sources), let $T_{mix(k)}$ and $R_{mix(k)}$ represent the time and the resource required to mix k video sources, respectively. Also, let T_{int} and T_{ext} denote the time to exchange a video across VMs and zones, respectively. R_O are the resources dedicated to VM operation, hence, they cannot be utilized for video mixing. There are thresholds T_ε on QoS, pertaining to the maximum acceptable video mixing response time, and R_ε on server resource capacity, respectively. Find the minimum number of VMs, while efficiently using resources and respecting QoS.

We model this as an ILP problem, where we assume a video mixer to be analogous to a VM. Tables 4.1 and 4.2 delineate the inputs and variables of our problem, respectively.

(ii) Objectives

We assume the operational cost of a VM, in terms of non-utilizable resources, supersedes the cost of resources required for handling the video mixing request of a participant, as in (1). Furthermore, we assume homogeneous costs of video mixing resources across servers. Therefore, the operational cost R_O, associated with a VM, inhibits the introduction of a new VM, in the event of a new participant arrival. That is, a new VM is only instantiated if an incoming request cannot be handled by increasing the resource of an existing VM.

$$R_O \gg (R_{mix\,(k+1)} - R_{mix(k)}) \quad (1)$$

Table 4.1. Problem inputs

Input	Definition
Z	number of zones
N_z	number of servers in zone z
M_z	number of users, i.e., video sources in zone z
T_{int}	time to send a video between VMs in a zone
T_{ext}	time to send a video between zones, $Z = 1 \Rightarrow T_{ext} = 0$
$T_{mix(k)}$	time to mix k video sources, $T_{mix(1)} = 0$
T_{ε}	QoS threshold (acceptable mixing response time)
$R_{mix(k)}$	required resources for mixing k video sources in a VM
R_O	non-utilizable VM operating resources
R_{ε}	threshold on the maximum amount of resources on a server
β	large enough constant

Table 4.2. Problem variables

Variable	Definition	
X	$N_z \times M_z$ binary matrix, where	$x_{i,j} = \begin{cases} 1, & if\ server\ i\ hosts\ VM\ j \\ 0, & otherwise \end{cases}$
Y	$M_z \times M_z$ binary matrix, where	$y_{j,k} = \begin{cases} 1, & if\ user\ k\ is\ connected\ to\ VM\ j \\ 0, & otherwise \end{cases}$
V_z	Maximum number of users that are connected to a VM in zone z	
U	A vector where u_j is the number of users connected to VM j	
C	$N_z \times M_z$ matrix, where	$c_{i,j} = \begin{cases} u_j, & if\ server\ i\ hosts\ VM\ j \\ 0, & otherwise \end{cases}$

Equation (2) depicts our multiple objectives. Primarily, we minimize the allocated resources across all zones, by minimizing the number of VMs. On the other hand, the time to mix videos in zone z depends on the maximum number of users connected to a VM (V_z). We balance the load between VMs to decrease the overall video mixing time. Note that these are competing objectives. Therefore, we prioritize minimizing the number of VMs by normalizing V_z with the maximum number of users in zone z.

$$minimize \left\{ \sum_{i=1}^{N_z} \sum_{j=1}^{M_z} x_{i,j} + \frac{V_z}{M_z} \right\} \tag{2}$$

(iii) Constraints

VMs and users cannot be split across multiple servers and VMs, respectively. Equation (3) ensures that a VM exists on a single server. Similarly, (4) allows a user to connect to a single VM. Furthermore, if there are users connected to a VM, that VM should exist on one server, as depicted in (5) and (6).

$$\sum_{i=1}^{N_z} x_{i,j} \leq 1 \qquad \forall 1 \leq j \leq M_z \tag{3}$$

$$\sum_{j=1}^{M_z} y_{j,k} = 1 \qquad \forall 1 \leq k \leq M_z \tag{4}$$

$$\sum_{k=1}^{M_z} y_{j,k} \leq \beta \cdot \left(\sum_{i=1}^{N_z} x_{i,j} \right) \qquad \forall 1 \leq j \leq M_z \tag{5}$$

$$\sum_{k=1}^{M_z} y_{j,k} \geq \sum_{i=1}^{N_z} x_{i,j} \qquad \forall 1 \leq j \leq M_z \tag{6}$$

Video mixing required resources, that is, the VMs operating resources and their connected number of users is bounded by the server resource capacity R_{ε}, in (7).

$$R_O \cdot \left(\sum_{j=1}^{M_z} x_{i,j} \right) + R_{mix\left(\sum_{j=1}^{M_z} (x_{i,j} \cdot \sum_{k=1}^{M_z} y_{j,k})\right)} \leq R_{\varepsilon} \qquad \forall 1 \leq i \leq N_z \tag{7}$$

Note that the product $\sum_{j=1}^{M_z} (x_{i,j} \cdot \sum_{k=1}^{M_z} y_{j,k})$ in (7) is non-linear. Therefore, we linearize (7) by replacing it with constraints (8)-(13).

$$\sum_{k=1}^{M_z} y_{j,k} = u_j \qquad \forall 1 \leq j \leq M_z \tag{8}$$

$$c_{i,j} \leq M_z \cdot x_{i,j} \qquad \forall 1 \leq i \leq N_z \,, \forall 1 \leq j \leq M_z \tag{9}$$

$$c_{i,j} \leq u_j \quad \forall 1 \leq i \leq N_z \ , \forall 1 \leq j \leq M_z \tag{10}$$

$$c_{i,j} \geq u_j - M_z\left(1 - x_{i,j}\right) \quad \forall 1 \leq i \leq N_z \ , \forall 1 \leq j \leq M_z \tag{11}$$

$$c_{i,j} \geq 0 \quad \forall 1 \leq i \leq N_z \ , \forall 1 \leq j \leq M_z \tag{12}$$

$$R_O \cdot \left(\sum_{j=1}^{M_z} x_{i,j}\right) + \ R_{mix\left(\sum_{j=1}^{M_z} c_{i,j}\right)} \leq R_\varepsilon \qquad \forall 1 \leq i \leq N_z \tag{13}$$

The maximum number of users, V_z, in a zone z influences the video mixing time. Equation (14) finds V_z, for each zone.

$$\sum_{k=1}^{M_z} y_{j,k} \leq V_z \qquad \forall 1 \leq j \leq M_z \tag{14}$$

Video mixing response time for a zone z, depends on the maximum number of users connected to a single VM in that zone ($T_{mix(V_z)}$). Note that VMs should mix the output of video mixing from other VMs too, therefore, the video mixing response time will also be influenced by the total amount of VMs across all servers in z. This time is given by $T_{mix(\sum_{i=1}^{N_z}\sum_{j=1}^{M_z} x_{i,j})}$, with an inter-zone exchange time of T_{int}. Furthermore, VMs should mix the incoming videos from all other zones, time for which is represented by $T_{mix(Z)}$, with an intra-zone exchange time of T_{ext}. Equation (15) ensures that this total video mixing response time for each zone z, abides by the QoS threshold T_ε.

$$T_{mix(V_z)} + \ T_{int} + \ T_{mix(\sum_{i=1}^{N_z}\sum_{j=1}^{M_z} x_{i,j})} + T_{ext} + T_{mix(Z)} \leq T_\varepsilon \qquad \forall 1 \leq z \leq Z \tag{15}$$

VMRA executes in each zone separately. However, because video mixing as a service relies on multiple IaaSs, the total number of zones will influence VMRA's decision. Based on (15), different response times across zones are attributed to the different values of $T_{mix(V_z)}$ and $T_{mix(\sum_{i=1}^{N_z}\sum_{j=1}^{M_z} x_{i,j})}$. Zone z will send its mixed video to other zones and wait to receive from them. Waiting time in (16) will add to the video mixing response time of

zones that perform video mixing faster than the other zones. Thus, the video mixing response time will be equal to the maximum response time across all zones.

$$\begin{cases} MAX \begin{cases} \left(T_{mix(V_p)} + T_{mix\left(\sum_{i=1}^{N_p}\sum_{j=1}^{M_p} x_{i,j}\right)}\right) - \\ \left(T_{mix(V_z)} + T_{mix\left(\sum_{i=1}^{N_z}\sum_{j=1}^{M_z} x_{i,j}\right)}\right) \end{cases} \\ 0, if\ MAX \leq 0 \end{cases} \qquad \forall 1 \leq p \leq Z \qquad (16)$$

4.3. VMRA Heuristic

Based on (1), VMRA always processes a new mixing request by adding required resources to the existing VMs unless it cannot satisfy the QoS requirement or there are not enough free resources on the server. In this case, VMRA instantiates a new VM and balances the load between VMs in the data center. Load balancing helps to minimize the maximum number of connected users to each VM. We achieve this by employing MinMax our objective, that is, the minimization of the maximum number of users on VMs and consequently, based on (15), it decreases the total response time.

VMRA checks the available resources when it decides to instantiate a new VM. Moreover, it checks the possibility of satisfying QoS requirement, by adding a new VM. Our heuristic is as described in Algorithm 4.1. We consider the constants and variables shown in Table 4.1 and Table 4.2 as the input to this algorithm.

Algorithm 4.1. Video mixing resource allocation (VMRA)

Input:
$Max\_M = M$; // Max number of users that can be served in DC
$\alpha = 0$; // number of VMs
$\beta = 1$; // number of used servers
$R_\beta = R_\varepsilon$; // available resources on server β
Remain_User = 0; // auxiliary variable to scatter users between VMs
Output: α, U, Max_M

1. **For each** m ∈ M **do**

Phase 1: Test if there is a VM with lower users than V_z

2. **If** ($R_\beta \geq R_{mix(1)}$) **Then**
3. **For** j =1 → α **do**
4. **If** ($u_j < V_z$) **Then**
5. $u_j \leftarrow u_j + 1$
6. Break, serve next **m**
7. **end for**
8. **end if**

Phase 2: Create first VM in DC

If (α==0) **Then**
    $\alpha \leftarrow 1$
    $u_1 \leftarrow 1$
    $V_z \leftarrow 1$
end if

Phase 3: Test response time by increasing V_z without adding VM

Else if ($R_\beta \geq R_{mix(1)}$ **AND** Response time($V_z \leftarrow V_z + 1, \alpha$)$\leq T\varepsilon$)**Then**
    $u_1 \leftarrow u_1 + 1$
    $V_z \leftarrow V_z + 1$
end else if

Phase 4: Test response time by adding a new VM on the same server

Else if ($R_\beta \geq R_{mix(1)} + R_O$) **Then**
    If (Response time($V_z \leftarrow \left\lceil \frac{m}{\alpha+1} \right\rceil, \alpha \leftarrow a+1$)$\leq T\varepsilon$) **Then**
        $\alpha \leftarrow a+1$
        Remain_User ← m
        For j = $\alpha \rightarrow 1$ **do**
            $u_j \leftarrow$ Remain_User / j
            Remain_User ← Remain_User $- u_j$
        end for
        $V_z \leftarrow \left\lceil \frac{m}{\alpha} \right\rceil$
    end if
    Else
        $Max_M \leftarrow m - 1$
        Break, DC cannot serve **m** users
    end else
end else if

Phase 5: Test response time by adding new VM on the other server

Else If (($N_z - \beta > 0$) **AND** ($R_\beta \geq R_{mix(1)} + R_O$)) **Then**
    If (Response time($V_z \leftarrow \left\lceil \frac{m}{\alpha+1} \right\rceil, \alpha \leftarrow a+1$)$\leq T\varepsilon$) **Then**
        $\beta \leftarrow \beta + 1$
        $\alpha \leftarrow a+1$
        Remain_User← m
        For j = $\alpha \rightarrow 1$ **do**
            $u_j \leftarrow$ Remain_User / j
            Remain_User ← Remain_User $- u_j$
        end for
        $V_z \leftarrow \left\lceil \frac{m}{\alpha} \right\rceil$
    end if
    Else
        $Max_M \leftarrow m - 1$
        Break, DC cannot serve **m** users
    end else
end else if
Else
    $Max_M \leftarrow m - 1$
    Break, DC cannot serve **m** users
end for each

Return α, U, Max_M

In phase 1, VMRA tries to find a VM with lowest number of connected users. If VMRA finds such a VM, it will add required resources to that VM and assigns the new user to it. In phase 2, the first user wants to join. So, VMRA will create the first VM and assign that user to it. VMRA will reach phase 3 if all the VMs have the same number of users. Here, VMRA checks the available resources and the feasibility of satisfying QoS requirements, if it assigns a new user to one of the existing VMs. This assignment is crucial as it increases V_z, thus, impacting the video mixing time.

If increasing V_z causes sacrificing QoS, VMRA decides to instantiate a new VM on the same server or on other servers based on available resources, in phase 4 and 5, respectively. If there are available resources, but VMRA cannot find any feasible solution to satisfy QoS, it will stop accepting new users in both phases 4 and 5.

This algorithm has a nested loop and its time complexity is based on the number of iterations of each loop. Therefore, the time complexity of our VMRA algorithm is $O(M_z \,.\, \alpha)$.

4.4. Validations and Measurements

In this section, we present the simulation results of VMRA resource allocation algorithm. First, the comparisons baselines are presented. Then, it describes the simulation environment and settings followed by the results and a conclusion for this section.

4.4.1. Comparison Baselines

We compare VMRA with (i) popular traditional MCU [68], for video mixing, (ii) Nan et al. [80], cost minimization queuing model in cloud, for a single class service, and (iii) cloud-based MCU (CMCU), which avoids upfront resource costs. Since these models do not support multi-zone video mixing, we assume that each model is implemented in a zone and exchange mixed video amongst each other until all sources are mixed.

4.4.2. Environment and Settings

We assume a MMOG, where player's video is shared in the logic of the game and developed a custom simulator in JAVA. We simulate multiple data centers and game

Table 4.3. Simulation parameters

Parameter	Value	Parameter	Value	Parameter	Value
Z	1-6	$T_{mix(k)}$	7 msec	$R_{mix(k)}$	20 MB (RAM)
N_z	3	T_ε	300 msec	R_O	400 MB (RAM)
M_z	1-500	T_{int}	10 msec	R_ε	10240 MB (RAM)
β	M+1	T_{ext}	15 msec		

players as conferencing participants. VMRA heuristic runs on each data center part in our simulator. Players (conference participants) send their video mixing requests to the local data center and receive the result from it. The total number of game players across all zones fluctuates since they can join or leave the game whenever they want to. For our simulation, we assume a snap-shot of the number of players in each zone. Our simulation parameters are depicted in Table 4.3.

4.4.3. Validations and Measurements

We simulate VMRA heuristic to check supported number of participants, resource utilization and video mixing response time.

(i) Number of Users

It is evident from Fig. 4.4, that VMRA can serve more users in a single zone in comparison to other baselines. This is because VMRA has the leverage to increase resources whenever it reaches the QoS threshold in contrast to the queuing model, where the number of computation nodes is fixed. VMRA also performs better than MCU and CMCU. Due to their centralized nature, both MCU and CMCU models leverage a single server entity and consequently are not equipped to handle a large number of users.

When we increase the number of zones, we have to account for the inter-zone communication time of mixing videos. As a result, to satisfy video mixing response time threshold, video mixing as a service can serve a lower number of users in each zone, while the number of zones increases. Although there is a tradeoff between the number of zones and the number of users that can be served in each zone, total number of users that can be served across all zones will increase, as depicted in Fig. 4.5. In addition, VMRA shows a better growth rate, thus it shows better scalability, in terms of the number of users, in comparison to the other models.

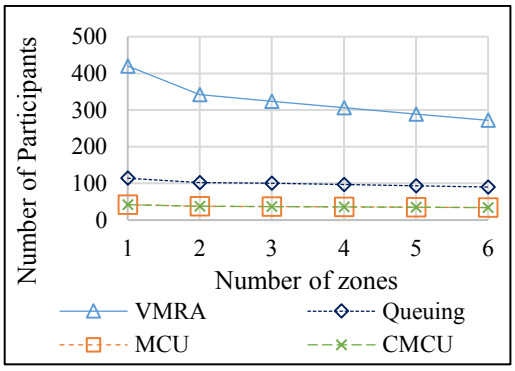


Fig. 4.4. Maximum participants that can be served in a zone

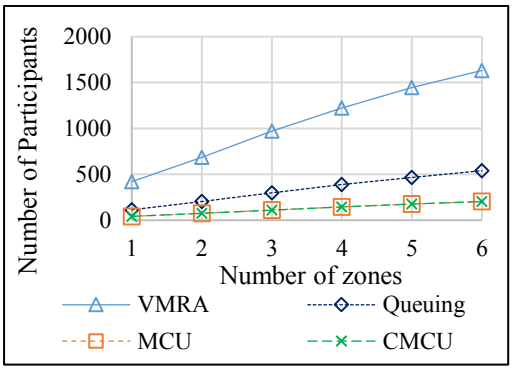


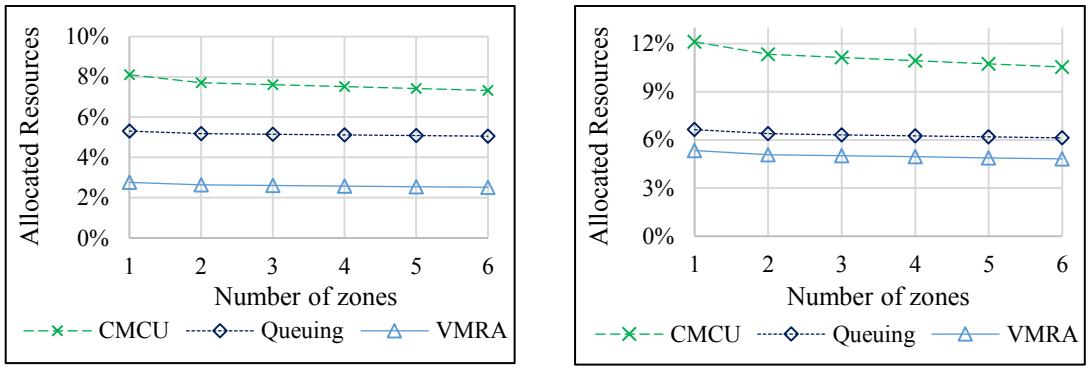
Fig. 4.5. Total number of participants that can be served across all zones

(ii) Resource Utilization and Video Mixing Response Time

Required resources for video mixing depends on the maximum number of served users. Accordingly, we study two different scenarios, each with a different number of video mixing requests: (i) Meet-By-All - In this scenario, we assume that there exists a maximum number of users, which can be served by all the resource allocation models in a zone while respecting QoS. (ii) Meet-By-Some - In this scenario, we assume for all models, the number of users to be the maximum supported by VMRA while respecting QoS. In this scenario, we relax the QoS constraint for the other models, giving them the leverage to support a higher number of users.

a) Resource Utilization: Meet-By-All Scenario

Fig. 4.6(a) and 4.6(b), depict the average and the maximum allocated resources over the total available resources in a data center, respectively. In MCU, because of the upfront resource over-provisioning, there are always some idle resources, which remain unutilized. However, because the allocated resources in MCU are always at 100%, we do not show it in the resource allocation figures. Other baselines allocate resources as needed. VMRA has better results compared to the other baselines, in both average and maximum cases in this scenario. This is because the maximum number of users in this scenario is equal to the number of users that MCU can support and just one computation entity is enough to serve them. However, the queuing model, based on our simulation settings, always uses 3 servers to accommodate users. Whereas, VMRA uses 2 VMs to accommodate the same number of users, which leads to the allocation of fewer resources, compared to the queuing model and



(a) (b)

Fig. 4.6. (a) Average, (b) Maximum allocated resources in a datacenter in Meet-By-All scenario

more resources, compared to MCU and CMCU. However, because the total available resources in VMRA are more than those of MCU and CMCU, the allocated resource percent is lower in comparison to both.

b) Video Mixing Response Time: Meet-By-All Scenario

The average video mixing response time for the Meet-By-All scenario is shown in Fig. 4.7. As it can be seen, the queuing model shows better video mixing response time than VMRA. This is because the objective of our model is maximizing resource utilization while respecting QoS. Intuitively, for lower response time, we should allocate more resources; however, this is in contradiction to our objective. So, in VMRA, as long as video mixing response time is lower than QoS threshold, it does not reduce video mixing response time. On the other hand, MCU and CMCU models have a higher video mixing response time, in comparison to VMRA. This is directly attributed to the centralized architecture of these models. Interestingly, the video mixing response time for MCU and CMCU are the same.

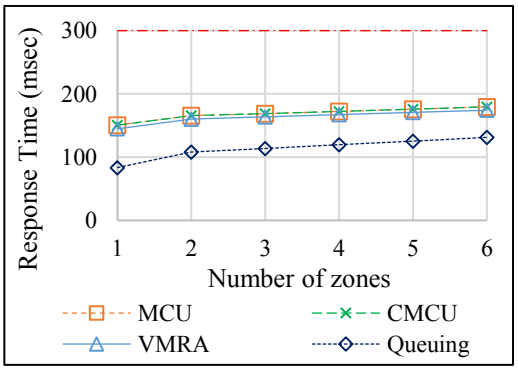


Fig. 4.7. Average video mixing response time in Meet-By-All scenario

It shows cloud has an effect only on the amount of allocated resources in CMCU and not on the video mixing response time.

c) Resource Utilization: Meet-By-Some Scenario

Recall, our model can serve the maximum number of users, as shown in Fig. 4.5. Hence, in this scenario, we have as many users as VMRA can serve. As depicted in Fig. 4.8(a) and 4.8(b), the resource allocation of the queuing model performs better compared to VMRA. This is because, VMRA will add more resources to accommodate as many users as possible, within the QoS threshold, while queuing model serves requests by leveraging a fixed number of servers.

d) Video Mixing Response Time: Meet-By-Some Scenario

Previous results show queuing model allocates a lower amount of resources in Meet-By-Some scenario compared to VMRA. However, this model is not suitable for video mixing as a service after comparing the corresponding video mixing response time. This is because the queuing model sacrifices QoS to serve the same number of users, compared to VMRA. As shown in Fig. 4.9, if we choose resource allocation based on the queuing model for video mixing as a service in cloud we have a high violation in terms of QoS. Based on our simulation results, if we serve as many users as VMRA can support using the queuing resource allocation model, QoS will be sacrificed between 66% and 72%. The same holds true when comparing with CMCU. In fact, VMRA allocates more resources, compared to queuing model and CMCU, to satisfy QoS for more users.

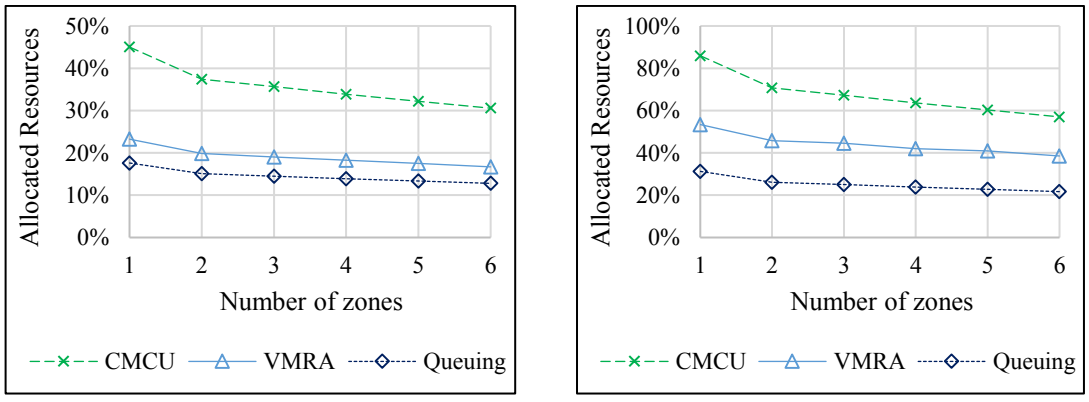


Fig. 4.8. (a) Average, (b) Maximum allocated resources in a datacenter in Meet-By-Some scenario

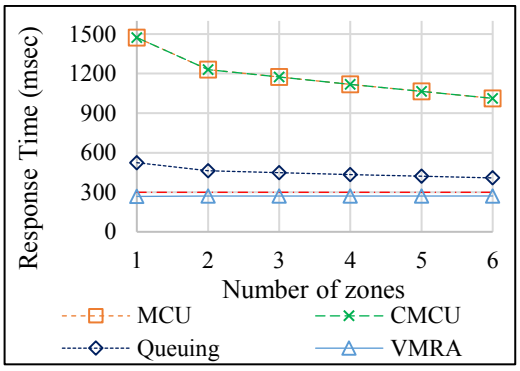


Fig. 4.9. Average video mixing response time in Meet-By-Some scenario

It is important to note that Fig. 4.6 and Fig. 4.8 reveal that to accommodate a larger number of users for video mixing, it is desirable to have more data centers with fewer resources. Furthermore, as evident from the results, our novel VMRA addresses the specific needs of video mixing as a service, which cannot be handled by generic cloud-based resource allocation models.

4.5. Conclusion

This chapter presents VMRA, a novel cloud-based resource allocation algorithm for multimedia conferencing applications with video mixing. VMRA scales resources in an elastic manner while meeting the QoS requirements and considering the fluctuation in the number of participants. We mathematically formulated the problem and also proposed the heuristic to solve the large-scale scenarios in an acceptable time. Simulation results show that VMRA outperforms other resource allocation techniques for video mixing because it considered both resource efficiency and video mixing QoS requirements. VMRA is suitable for conferencing applications with video mixing service. However, other media handling services such as compressing may be used in a conferencing application. Since each media handling service has different requirements, the VMRA may not be suitable for such conferencing applications. In addition, VMRA just considers reducing the servers' resource cost and does not consider reducing the network cost. These limitations of VMRA will be solved in chapter 5.

Chapter 5

5. A Resource Allocation Mechanism for Multimedia Conferencing Applications with Video Mixing and Compressing

5.1. Introduction

The previous chapter proposed VMRA, a novel resource allocation mechanism to optimize resource allocation in terms of the number of participants while guaranteeing QoS for conferencing applications with video mixing. As it was mentioned before, VMRA has some limitations. For instance, it does not consider the requirements of other media handling services such as compressing while allocating the resources. In addition, it does not consider reducing the network cost. This chapter proposes another cloud-based resource allocation mechanism for conferencing applications to solve the limitations of VMRA. We entitled this algorithm as CRAM (Cloud-based Resource Allocation for Multimedia conferencing). In designing the CRAM, we consider the conferencing applications with video mixing and compressing together. Similar to VMRA, the CRAM algorithm allocates or deallocates resources for conferencing applications based on the fluctuation in the number of participants. Besides efficient servers' resource utilization, CRAM considers reducing the network cost as well. It also caters to the QoS, with respect to both media handling response times and network latency. To reduce the network cost and latency, CRAM algorithm selects adequate locations for allocating resources. We

analyze CRAM theoretically by modeling it as an optimization problem. Then, we design the heuristic that can reach the sub-optimal solution for the large-scale scenarios in an acceptable time.

The rest of this chapter is as follows. First, it presents the CRAM algorithm by discussing its system model. Then, it discusses the designed heuristic, followed by a conclusion of this chapter at the end.

5.2. CRAM System Model

In CRAM, besides video mixing, we consider using the compressing service. In addition, we consider reducing both network and servers' resources costs. CRAM system model includes the general assumptions that we made in this work and the mathematical model. In our mathematical model, we define CRAM as an ILP problem.

5.2.1. General Assumptions

There are some assumptions that are considered to model the problem. We categorize them into two sections.

(i) Assumptions on Conferencing Applications

We assume that conferencing applications run in a large scale geographically distributed cloud. Also, we consider multiple conferencing participants who want to join a conferencing application and share their videos with each other. Moreover, participants are simultaneously considered as video sources and destinations. It is assumed that the conferencing application requires the video streams from all participants to be mixed and sent to each of them.

(ii) Assumptions on Media Handling Services

Media handling services can be placed in any data center, as long as the participants' required QoS (such as latency) is satisfied. It is assumed that each media handling service is hosted on a VM. To connect media handling services, we consider different cost and latency for each network link.

Similar to VMRA (presented in the previous chapter), our video mixing model follows the Fork/Join parallelism technique [79]. Therefore, the video mixing process for all participants depends on all video mixer instances. In this work, we assume the video mixing time in a video mixer depends on the number of input streams of that mixer. Note that all video mixers across different servers need the results from each other to complete the mixing process. Thus, the total mixing time depends on the number of video mixers and network latency.

5.2.2. Mathematical Model

This section presents the CRAM problem formulation, which is modeled as an ILP problem.

(i) Problem Statement

Given S and U as sets of servers and participants (i.e., video sources and destinations) respectively, let $T_{m(k)}$ and $R_{m(k)}$ represent the time and the resource required to mix or compress k video sources, respectively. Note that we assume $T_{m(k)}$ and $R_{m(k)}$ are linear functions of k. Also, let $T_{a,b}$ and $P_{a,b}$ denote the time and cost to exchange a video from location a to b, respectively. Each compressor instance can reduce the size of video by $\%\gamma$. The $T_{a,b}$ and $P_{a,b}$ are reduced by $\%\gamma$ if there is a compressor at location a. In addition, R_O are the resources which cannot be utilized for video mixing or compressing (e.g., OS required resources). There are thresholds T_ε on QoS, pertaining to the maximum acceptable end-to-end delay, and R_ε^s on resource capacity of server s. The problem is finding the minimum number of VMs and minimum network cost, while respecting QoS. Also, finding the optimal order of using media handling services to efficiently use resources is part of the problem.

We model this as an ILP problem, where we assume a media handling service to be analogous to a VM. Tables 5.1 and 5.2 delineate the inputs and variables of our problem, respectively.

Table 5.1. Problem inputs

Input	Definition
S	set of servers
U	set of users, i.e., video sources and destinations
M	set of video mixer instances
C	set of compressor instances
V	set of all VMs, where $V = \{C \cup M\}$
$T_{m(k)}$	time to mix or compress k video sources
$R_{m(k)}$	required resources to mix or compress k video sources in a VM
R_O	non-utilizable VM operating resources
$T_{a,b}$	time to send a video between location a and b
$P_{a,b}$	cost to send a video between location a and b
P_s	cost of provisioning a VM on server s, $s \in S$
γ	compress rate, $0 < \gamma < 100$
R_ε^s	threshold on the maximum amount of resources in server s
T_ε	QoS threshold (acceptable mixing response time)
β	large enough constant

Table 5.2. Problem variables

Variable	Definition	
D	$(4\|U\| - 2) \times (4\|U\| - 2)$ binary matrix, where	$d_{a,b} = \begin{cases} 1, if\ 'a' is\ directly\ connected\ to\ destination\ 'b' \\ 0, otherwise \end{cases}$
E	$\|U\| \times (3\|U\| - 2)$ binary matrix, where	$e_{a,b} = \begin{cases} 1, if\ user\ 'a'\ directly\ or\ indirectly\ is\ connected\ to\ VM\ 'b' \\ 0, otherwise \end{cases}$
X	$\|S\| \times (3\|U\| - 2)$ binary matrix, where	$x_{s,v} = \begin{cases} 1, if\ server\ 's'\ hosts\ VM\ 'v' \\ 0, otherwise \end{cases}$
Y	$\|U\| \times (3\|U\| - 2)$ matrix where, $y_{a,b}$ is the required time to transmit a video stream from user $'a'$ to VM $'b'$ and the total required time for media handling services to reach location $'b'$	
Z	$\|S\| \times (3\|U\| - 2)$ matrix, where	$z_{s,v} = \begin{cases} g_v, if\ server\ 's'\ hosts\ VM\ 'v' \\ 0, otherwise \end{cases}$
G	A vector where g_v is the number of users connected to the VM v	
F	$\|U\| \times (3\|U\| - 2) \times (3\|U\| - 2)$ binary matrix, where	$f_{i,v}^u = \begin{cases} 1, if\ user\ 'u' is\ indirectly\ connected\ to\ VM'v'through\ VM\ 'i' \\ 0, otherwise \end{cases}$

(ii) Objectives

We assume the operational cost of a VM, in terms of non-utilizable resources, supersedes the cost of resources required for media handling services request of a participant, as in (1).

Furthermore, we assume homogeneous costs of video mixing and compressing resources on each server. Therefore, the operational cost R_O, associated with a VM, inhibits the introduction of a new VM, in the event of a new participant's arrival. That is, a new VM is only instantiated if an incoming request cannot be handled by increasing the resources of an existing VM.

$$R_O \gg \left(R_{m\,(k+1)} - R_{m(k)}\right) \tag{1}$$

Equation (2) depicts our multiple objectives which are aiming at minimizing the overall cost. We aim to minimize the cost of allocated resources by minimizing the number of VMs. Moreover, we want to minimize the network cost. We use $x_{s,v}$ to represent a VM v which is hosting on server s. Also, $d_{a,b}$ represents a video stream connection from source a to the location b. The network cost between location a and b is shown by $P_{a,b}$.

$$min\left\{\sum_{s\in S}\sum_{v\in V} x_{s,v} \times P_s + \sum_{a\in U\cup V}\sum_{b\in U\cup V} d_{a,b} \times P_{a,b}\right\} \tag{2}$$

In this work, we assume the cost of sending a video from one location to another location in both directions are the same (i.e., $P_{a,b} = P_{b,a}$). Note that we know the locations of participants and servers. Therefore, to find the cost of sending a video from a participant to a VM, or from a VM to another VM, we use equations (3) and (4).

$$P_{u,v} = P_{v,u} = \sum_{s\in S}(x_{s,v} \times P_{s,u}) \qquad \begin{matrix}\forall\, v \in V \\ \forall\, u \in U\end{matrix} \tag{3}$$

$$P_{v_1,v_2} = \sum_{s_1\in S}\sum_{s_2\in S}(x_{s_1,v_1} \times x_{s_2,v_2} \times P_{s_1,s_2}) \qquad \forall\, v_1, v_2 \in V \tag{4}$$

Since equation (4) is not linear, we linearize it through equations (4-1) and (4-4). We use a binary auxiliary variable j_{s_1,s_2} for linearizing this equation.

$$j_{s_1,s_2} \le x_{s_1,v_1} \qquad \begin{matrix}\forall\, v_1 \in V \\ \forall\, s_1, s_2 \in S\end{matrix} \tag{4-1}$$

$$j_{s_1,s_2} \le x_{s_2,v_2} \qquad \begin{matrix}\forall\, v_2 \in V \\ \forall\, s_1, s_2 \in S\end{matrix} \tag{4-2}$$

$$j_{s_1,s_2} \geq x_{s_1,v_1} + x_{s_2,v_2} - 1 \qquad \begin{matrix} \forall\, v_1, v_2 \in V \\ \forall\, s_1, s_2 \in S \end{matrix} \qquad (4\text{-}3)$$

$$P_{v_1,v_2} = \sum_{s_1 \in S} \sum_{s_2 \in S} (j_{s_1,s_2} \times P_{s_1,s_2}) \qquad \forall\, v_1, v_2 \in V \qquad (4\text{-}4)$$

(iii) Constraints

Based on the set U, we can define two sets for video mixers (M) and compressors (C). We know that each video mixer has at least two video streams as input. Therefore, set M can be defined such that $|M| = |U| - 1$ and $M = \{m_1, m_2, \dots, m_{|U|-1}\}$. Also, we assume we can have compressors between participants and mixers as well as between mixers. Therefore, set C can be defined such that $|C| = |2U| - 1$ and $C = \{c_1, c_2, \dots, c_{|2U|-1}\}$. Since each VM hosts just one media handling service, we define a set for all possible virtual machines as V where $V = \{C \cup M\}$. These sets are used in the following equations.

We consider each participant has only one directed connection for sending the video stream and receiving the mixed video. Equations (5) and (6) ensure that there is only one directed connection from participants to VMs, and from VMs to participants, respectively.

$$\sum_{v \in V} d_{u,v} = 1 \qquad \forall\, u \in U \qquad (5)$$

$$\sum_{v \in V} d_{v,u} = 1 \qquad \forall\, u \in U \qquad (6)$$

Note that $d_{a,b}$ is a directed connection where a and b are the head and tail, respectively. Moreover, participants need the mixed video from all others in the conference. Therefore, there is no direct connection between participants. Equation (7) ensures this constraint.

$$\sum_{i \in U} \sum_{j \in U} d_{i,j} = 0 \qquad (7)$$

To complete the video mixing process, there should be at least one VM, which is the tail of a direct or indirect connection to all original sources of video streams (i.e., participants). After finishing the whole video mixing process, the final mixed video stream should be sent

to the participants from the mixers or compressors that have the whole mixing result. Equations (8) and (9) find the direct and indirect connection between all participants and all VMs. Equation (10) ensures that there is no indirect connection to any VM which has no direct connection. In addition, equations (11) and (12) consider all possible indirect connections from a participant$u$ to the VM v through all other VMs. Based on these connections, equation (13) ensures that the final video streams comes from the VMs which are directly or indirectly connected to all participants. Note that $e_{a,b}$ is an indirect connection where a and b are the head and tail, respectively.

$$e_{u,v} \geq d_{i,v} + e_{u,i} - 1 \qquad \forall\, i, v \in V,\ \forall\, u \in U \tag{8}$$

$$e_{u,v} \geq d_{u,v} \qquad \forall\, v \in V,\ \forall\, u \in U \tag{9}$$

$$e_{u,v} \leq \sum_{k \in U \cup V} d_{k,v} \qquad \forall\, v \in V,\ \forall\, u \in U \tag{10}$$

$$e_{u,v} \leq \sum_{i \in V} f_{i,v}^{u} + d_{u,v} \qquad \forall\, v \in V,\ \forall\, u \in U \tag{11}$$

$$f_{i,v}^{u} \leq \frac{d_{i,v} + e_{u,i}}{2} \qquad \forall\, i, v \in V,\ \forall\, u \in U \tag{12}$$

$$d_{v,u} \leq \frac{\sum_{p \in U} e_{p,v}}{|U|} \qquad \forall\, v \in V,\ \forall\, u \in U \tag{13}$$

As described before, each media handling service has its own functionality. The compressors can just reduce the video size. Therefore, the total number of input and output streams are the same. This constraint is considered in equation (14). In addition, compressors can help to reduce the size of video and in consequence, reduce the network cost and transmission time. In this work, we assume there is no need to have two consecutive compressors. Thus, there is no direct connection between two compressor instances. Equation (15) ensures this constraint.

$$\sum_{k \in U \cup V} d_{k,c} = \sum_{k \in U \cup V} d_{c,k} \qquad \forall\, c \in C \tag{14}$$

$$\sum_{i \in C} \sum_{j \in C} d_{i,j} = 0 \tag{15}$$

On the other hand, mixers are responsible to mix video streams. Therefore, at least one video mixer should be directly or indirectly connected to all participants as the tail. This constraint is ensured in equation (16).

$$\sum_{m \in M} \left\lfloor \frac{\sum_{u \in U} e_{u,m}}{|U|} \right\rfloor \geq 1 \tag{16}$$

Since equation (16) is not linear, we linearize it through equations (16-1) and (16-2) by using h_m as an auxiliary variable.

$$\sum_{m \in M} h_m \geq 1 \tag{16-1}$$

$$h_m \leq \frac{\sum_{u \in U} e_{u,m}}{|U|} \qquad \forall\, m \in M \tag{16.2}$$

A VM, that is hosting a media handling service, cannot be split across multiple servers. Equation (17) ensures that a VM exists on a single server. Furthermore, if there are any input streams connected to a VM, that VM should exist on one server, as depicted in (18) and (19). Also, if there are any output streams from a VM, that VM needs to exist on a server as shown in (20) and (21). Note that β is a big enough constant used for linearization purpose.

$$\sum_{s \in S} x_{s,v} \leq 1 \qquad \forall\, v \in V \tag{17}$$

$$\sum_{k \in U \cup V} d_{k,v} \leq \beta \times \left(\sum_{s \in S} x_{s,v} \right) \qquad \forall\, v \in V \tag{18}$$

$$\sum_{k \in U \cup V} d_{k,v} \geq \sum_{s \in S} x_{s,v} \qquad \forall\, v \in V \tag{19}$$

$$\sum_{k\in U\cup V} d_{v,k} \le \beta \times \left(\sum_{s\in S} x_{s,v}\right) \qquad \forall\, v \in V \qquad (20)$$

$$\sum_{k\in U\cup V} d_{v,k} \ge \sum_{s\in S} x_{s,v} \qquad \forall\, v \in V \qquad (21)$$

The number of VMs and their resources are bounded by the servers' capacities. Equation (22) ensures that the required resources for media handling services and operating system in VMs are bounded by the server resource capacity.

$$R_O \times \left(\sum_{v\in V} x_{s,v}\right) + R_{m\left(\sum_{v\in V}\left(x_{s,v}\times\sum_{k\in U\cup V} d_{k,v}\right)\right)} \le R_{\varepsilon}^{s} \qquad \forall\, s \in S \qquad (22)$$

Note that the product $\sum_{v\in V}(x_{s,v} \times \sum_{k\in \mathrm{U\cup V}} d_{k,v})$ in (22) is non-linear. Therefore, we linearize (22) by replacing it with constraints (22-1)-(22-6).

$$\sum_{k\in U\cup V} d_{k,v} = g_v \qquad \forall\, v \in V \qquad (22\text{-}1)$$

$$z_{s,v} \le |U| \times x_{s,v} \qquad \forall\, s \in S\,, \forall\, v \in V \qquad (22\text{-}2)$$

$$z_{s,v} \le g_v \qquad \forall\, s \in S\,, \forall\, v \in V \qquad (22\text{-}3)$$

$$z_{s,v} \ge g_v - |U| \times \left(1 - x_{s,v}\right) \qquad \forall\, s \in S, \forall\, v \in V \qquad (22\text{-}4)$$

$$z_{s,v} \ge 0 \qquad \forall\, s \in S, \forall\, v \in V \qquad (22\text{-}5)$$

$$R_O \times \left(\sum_{v\in V} x_{s,v}\right) + R_{m\left(\sum_{v\in V} z_{s,v}\right)} \le R_{\varepsilon} \qquad \forall\, s \in S \qquad (22\text{-}6)$$

The whole mixing procedure time, depends on the video mixing, compressing, and the time required for video transmission over the network. To satisfy the QoS requirement, the mixing procedure time for all participants should be less than or equal to T_{ε}. Equations (23) to (25) ensure that this end-to-end time for all participants, abides by the QoS threshold T_{ε}.

$$y_{u,v} \geq d_{i,v} \times T_{i,v} + y_{u,i} + T_{m(\sum_{k \in U \cup V} d_{k,v})} \qquad \begin{matrix} \forall\, i, v \in V \\ \forall\, u \in U \end{matrix} \qquad (23)$$

$$y_{u,v} \geq d_{u,v} \times T_{u,v} \qquad \begin{matrix} \forall\, v \in V \\ \forall\, u \in U \end{matrix} \qquad (24)$$

$$y_{u,v} + d_{v,u} \times T_{v,u} \leq T_{\varepsilon} \qquad \begin{matrix} \forall\, v \in V \\ \forall\, u \in U \end{matrix} \qquad (25)$$

We assume that the required time and cost for sending a video from one location to another location in both directions are the same (i.e., $T_{a,b} = T_{b,a}$ and $P_{a,b} = P_{b,a}$). To find the cost of sending a video from a participant to a VM, or from a VM to another VM, we use equations (26) and (27).

$$P_{u,v} = P_{v,u} = \sum_{s \in S} (x_{s,v} \times P_{s,u}) \qquad \begin{matrix} \forall\, v \in V \\ \forall\, u \in U \end{matrix} \qquad (26)$$

$$P_{v_1,v_2} = \sum_{s_1 \in S} \sum_{s_2 \in S} (x_{s_1,v_1} \times x_{s_2,v_2} \times P_{s_1,s_2}) \qquad \forall\, v_1, v_2 \in V \qquad (27)$$

Since equation (27) is not linear, we linearize it through equations (27-1) and (27-2). We use a binary auxiliary variable k_{s_1,s_2} for linearizing this equation.

$$k_{s_1,s_2} \leq x_{s_1,v_1} \qquad \begin{matrix} \forall\, v_1 \in V \\ \forall\, s_1, s_2 \in S \end{matrix} \qquad (27\text{-}1)$$

$$k_{s_1,s_2} \leq x_{s_2,v_2} \qquad \begin{matrix} \forall\, v_2 \in V \\ \forall\, s_1, s_2 \in S \end{matrix} \qquad (27\text{-}2)$$

$$k_{s_1,s_2} \geq x_{s_1,v_1} + x_{s_2,v_2} - 1 \qquad \begin{matrix} \forall\, v_1, v_2 \in V \\ \forall\, s_1, s_2 \in S \end{matrix} \qquad (27\text{-}3)$$

$$P_{v_1,v_2} = \sum_{s_1 \in S} \sum_{s_2 \in S} (k_{s_1,s_2} \times P_{s_1,s_2}) \qquad \forall\, v_1, v_2 \in V \qquad (27\text{-}4)$$

5.3. CRAM Heuristic

CRAM allows determining the number of VMs for mixers and compressors needed in order to serve a set of media handling requests. In addition, it identifies the servers that will

host these VMs, together with the resulting service composition. These aspects are covered with the objective of minimizing the overall costs while meeting QoS thresholds for multimedia conferencing applications. Note that to reach the lower media handling processing time, CRAM always assigns video streams to the VMs which have fewer connected streams on each server. Also, to respect the QoS threshold, CRAM may decide for using compressors. Note that using compressors leads to lower video resolution. However, in a dense network or when participants are very far from each other, it may help to abide the latency threshold.

Finding the best possible servers to host VMs can be mapped to the NP-hard facility location problem [81]. Besides finding the best servers to host VMs, our problem determines the best composition of media handling services. Solving our resource allocation problem for large-scale scenarios using exact algorithms is time-consuming. Thus, we introduce a heuristic to solve the problem efficiently and in a reasonable time. In this section, we propose the CRAM heuristic. It handles the composition of media handling services, together with the placement of the corresponding VMs.

The CRAM heuristic first calculates the minimum required number of VMs for mixing all streams, regardless of participants' locations. Then, it finds the possible servers with the minimum distance from all participants to host the mixers. Using these servers results in minimizing network latency and network cost. The CRAM heuristic also ensures that the available resources on these servers are enough to instantiate new VMs. Then, it checks the possibility of satisfying QoS requirements by having this minimum number of VMs hosting the mixers. If the QoS is not satisfied, the heuristic tries to increase the number of mixers (to reduce the mixing time) or add compressors (to reduce the transmission time). In these processes, our CRAM heuristic considers minimizing the cost as the main objective as well. Our solution is divided into four parts as described in Algorithms 5.1 to 5.4. We consider the constants and variables shown in Table 5.1 and Table 5.2 as the input to these algorithms. Also, to simplify the code, we assume the same resource capacity for all servers (i.e., R_{ε}).

Algorithm 5.1. Media Handling Resource Allocation

Input:
U, S; // the sets of participants' and servers' locations, respectively
P_s // cost of resources on a server
$R_{m(k)}$, $T_{m(k)}$, R_O;

R_ε; // the maximum capacity for all servers
$R \leftarrow R_\varepsilon$; // the set of available resources on each server
T_ε; // the maximum acceptable end-to-end delay
Output: M, C, D;//list of Mixers (M), Compressors (C) and the connections between participants/mixers/compressors (D)
$total\_delay$; // maximum end-to-end delay

Phase 1: Find the minimum number of mixers

1. $Min\_mixer \leftarrow 0$;
2. $handling\_time \leftarrow \infty$;
3. **do**
4. $Min\_mixer \leftarrow Min\_mixer + 1$;
5. $Max\_user = \boldsymbol{ceil}\left[\frac{|U|}{Min\_mixer}\right]$;
6. **If** $(handling\_time < T_{M(Max\_user)} + T_{M(Min\_mixer)})$ **Then**
7. **return null;** //there is no possible solution for the given |U|
8. **end if**
9. $handling\_time \leftarrow T_{M(Max\_user)} + T_{M(Min\_mixer)}$
10. **while** $((handling\_time \geq T_\varepsilon)\ OR\ (R_O + R_{m(Max\_user)} > R_\varepsilon))$

Phase 2: Select the best servers for hosting mixers

11. $vm \leftarrow 0$;
12. $i \leftarrow 0$;
13. $S \leftarrow \boldsymbol{DSort}(S, U)$;// sort servers based on minimum distance to the group of participants
14. **do**
15. $i \leftarrow i + 1$;
16. **while** $(R[S[i]] \geq R_O + R_{m(Max_{user})}$ **AND** $vm < Min\_mixer)$ **do**
17. $M[S[i]] \leftarrow M[S[i]] + 1$;// number of mixers hosted on server i
18. $vm++$;
19. $R[S[i]] \leftarrow R[S[i]] - (R_O + R_{m(Max\_user)})$;
20. **end while**
21. **If** $(i == |S|$ **AND** $vm < Min\_mixer)$ **Then**
22. **return null;** //not enough resources to support |U|
23. **end if**
24. **while** $(vm < Min\_mixer)$

Phase 3: Check the need of compressor between mixers

25. $used\_servers \leftarrow i$;
26. **For** j =1 → $used\_servers$ **do**
27. $mix\_time[S[j]] \leftarrow 0$; // maximum mixing time for each server
28. **For** n =1 → $used\_servers$ **do**
29. $total\_time \leftarrow T_{M(Max\_user)} + T_{M(M[S[j]])} + T_{M(used\_servers)} + T[S[j]][S[n]]$;
30. **if** $(total\_time \geq T_\varepsilon)$ **Then**
31. $t \leftarrow total\_time - T_\varepsilon$; // required time to compress
//Create/assign a compressor between servers j and n
32. $compress\_results \leftarrow \boldsymbol{Compress}(j, S[n], t, "server")$;
33. **if** $(compress\_results == Null)$ **Then**
34. **return null;** // there is no possible solution
35. **end if**
36. $total\_time \leftarrow T_\varepsilon$;
37. **end if**
38. **If** $(total\_time > mix\_time[S[j]])$ **Then**
39. $mix\_time[S[j]] \leftarrow total\_time$; // keep track of mixing time and network transmission time between all mixers
40. **end if**
41. **end for**
42. **end for**

Phase 4: Assign participants to mixers AND check the need of compressors

43. $max\_delay \leftarrow 0$;
44. **For** u =1 → $|U|$ **do**
45. $total\_time \leftarrow 0$;
    //find the closest server with a mixer that can accept a participant
46. $s \leftarrow \boldsymbol{ACS}(u, M)$; //acceptable closest server to the participant u
47. $total\_time \leftarrow mix\_time[S[s]] + 2 \times T[U[u]][S[s]]$;
48. **If** $(total\_time \leq T_\varepsilon)$ **Then**
    //Assign the participant u to a mixer on server $s, (s \in S)$
49. $D[u][S[s]] \leftarrow 1$; //connection from participant to server
50. $D[S[s]][u] \leftarrow 1$; //connection from server to participant
51. **end if**
52. **Else**
    //Create/assign a compressor between participant u and server s
53. $t \leftarrow total_{time} - T_\varepsilon$; // required time to compress
54. $compress\_result \leftarrow \boldsymbol{Compress}(u, S[s], t, "user")$;
55. **if (**$compress\_results == Null$**) Then**
56. **return null;** // there is no possible solution
57. **end if**
58. $total\_time \leftarrow T_\varepsilon$;
59. **end else**
60. **If** $(total\_time > max\_delay)$ **Then**
61. $max\_delay \leftarrow total\_time$;//maximum end-to-end delay
62. **end if**
63. **end for**

Return M, C, D, max_delay

Algorithm 5.1 is the main body of the CRAM heuristic. It takes as main inputs: (i) the list of participants and their locations, (ii) the list of servers and their locations, and (iii) the network transmission time and cost between different locations. This algorithm in collaboration with algorithms 5.2 to 5.4, finds the list of mixers, compressors, network connections, and the maximum end-to-end delay. This algorithm runs at the starting point of the conferencing application. In addition, it re-runs periodically to scale the system based on the fluctuations in the number of participants.

Algorithm 5.1 has four main phases. In the first phase, it finds the minimum possible number of mixers that can mix the total number of video streams from all participants. To find this minimum number, it considers both the QoS threshold and the available resources on the servers.

After finding the minimum number of mixers, in phase two, it places these mixers on the servers which are closer to the majority of participants. Also, it makes sure that the selected

server has enough resources to host VMs. To find the servers based on the minimum distances to the majority of participants, it uses Algorithm 5.2 (i.e., DSort) in phase two.

After placing the mixers on the chosen servers, in phase three it checks the need of having compressors between mixers. Note that we consider full mesh topology between mixers on a server and also between servers which host mixers. If the total time of the mixing process and the network transmission time between two servers cannot abide the QoS threshold, a compressor will be added between these servers. To assign or create a compressor between two servers, Algorithm 5.3 (i.e., Compress) is used in this phase. At the end of phase three, all mixers and required compressors between them are placed. Moreover, the mixing time for each specific server will be known.

In the last phase, participants are assigned to the closest mixer which can accept a new participant. The acceptable closest server is retrieved by using Algorithm 5.4 (i.e., ACS). Moreover, if the end-to-end delay is greater than the QoS threshold, it uses Algorithm 5.3 to assign a compressor between the participant and the mixer.

Algorithm 5.2. (DSort): Sort servers based on minimum distance to a group of participants

Input:
S; // the sets of servers' locations
U; // the sets of participants' locations
Output:$Server$ // sorted list of servers

1. $delay[]$; // an array to keep track of distance for each server
2. **For** n =1 → $|S|$ **do**
3. **For** u =1 → $|U|$ **do**
4. $delay[n] \leftarrow delay[n] + T[User[u]][S[s]]$;
5. **end for**
6. **end for**
7. $delay2[] \leftarrow sort(delay[])$;// keep sorted distances in another array
8. **For** $i = 1 \rightarrow |S|$ **do**
9. **For** $j = 1 \rightarrow |S|$ **do**
10. **if** ($delay2[i] == delay[j]$) **Then**
11. $Server[i] \leftarrow j$; // keep track of server n's location
12. $delay[j] \leftarrow -1$; //change to a negative value to make sure not using the same server more than once
13. **break;**
14. **end if**
15. **end for**
16. **end for**

Return $Server$

Algorithm 5.2 sorts the servers based on their minimum distances to a group of participants. It takes the list of servers and participants and returns a list of sorted servers. This algorithm calculates the total distance from each server to all participants and uses a simple sort function (e.g., binary sort).

Algorithm 5.3. (Compress): Create or assign a compressor

Input: $sender$// video sender (i.e., a participant or a server)
b // the location of destination server
t // minimum time that needs to be reduced by compression
$string$ // to find video sender is a participant or another server
P // the matrix of video transmission costs over the network
T // the matrix of video transmission times over the network
$Rate_{max}$ // the maximum acceptable compression rate (0 to 1)
Output: C, D // list of compressors and their connections
$Rate$ // compression rate for the requested compress

1. **if (**$string == "server"$**) Then**
2. $a \leftarrow S[sender]$; // keep location of the server in a
3. **else**
4. $a \leftarrow Users[sender]$; keep location of the participant in a
5. **end if/else**
6. $Max\_distance \leftarrow T[a][b] - t - T_{m(1)}$;
7. $possible\_servers[]$; list of possible servers that can host compressors between locations a and b
8. $flag[] \leftarrow 0$; // to keep the demand for adding a new compressor
9. $j \leftarrow 0$;

Phase 1: Find possible servers to host compressors between a and b

10. **For** i =1 → $|S|$ **do**
11. **if (**$T[a][S[i]] < Max\_distance$ **AND** $R[S[i]] > R_{m(1)}$**) Then**
12. $j \leftarrow j + 1$;
13. $possible\_servers[j] \leftarrow i$; keep server i as a possible server
14. **end if**
15. **end for**
16. **if (**$|possible\_servers| == 0$**) Then**
17. **return null;** // there is no possible server to host compressors
18. **end if**

Phase 2: Find the corresponding cost for hosting or using compressors on each possible server found

19. $C\_R \leftarrow 1 - Rate_{max}$; //
20. **For** i =1 → $|possible\_servers|$ **do**
21. $s \leftarrow possible\_servers[i]$
22. **if (**$C[S[s]] == 0$**) Then** //no existing compressor on server s
23. **if (**$R[S[s]] < R_O + R_{m(1)}$**) Then** //not enough resources
24. Cost[$S[s]$] ← ∞;
25. continue;
26. **end if**
27. Cost[$S[s]$] ← $P[a][S[s]]+(R_{m(1)} + R_O) \times P_s$;
28. $flag[S[s]] \leftarrow 1$;
29. **end if**
30. **Else**
31. $min\_stream \leftarrow \infty$;
32. **For** c =1 → $C[S[s]]$ **do**
33. $m \leftarrow comp\_connections[S[s]][c]$;//connected number of streams to the compressor c on server s
34. **if (**$m < min\_stream$**) Then**

35. $min\_stream \leftarrow m$;
36. **end if**
37. **end for**
38. **if** ($T_{m(min\_stream+1)} + T[a][S[s]] + T[S[s]][b] \times C\_R \leq T[a][b] - t$) **Then**
39. Cost$[S[s]] \leftarrow P[a][S[s]]+(R_{m(1)}) \times P_s$;
40. **end if**
41. **Else**
42. **if** ($R[S[s]] < R_O + R_{m(1)}$) **Then** //not enough resources
43. Cost$[S[s]] \leftarrow \infty$;
44. continue;
45. **end if**
46. Cost$[S[s]] \leftarrow P[a][S[s]]+(R_{m(1)} + R_O) \times P_s$;
47. $flag[S[s]] \leftarrow 1$;
48. **end else**
49. **end else**
50. **end for**

Phase 3: Assign a compressor between locations a and b based on cost

51. $Cost2[] \leftarrow sort(Cost[])$;// keep sorted cost in another array
52. **For** $j = 1 \rightarrow |Cost|$ **do**
53. **if** ($Cost2[1] == Cost[S[j]]$) **Then**
54. $chose \leftarrow possible\_server[j]$; // chosen server to host the compressor between a and b
55. **break;**
56. **end if**
57. **end for**
58. **if**($string == user$)**Then**
59. $D[sender][S[chose]] \leftarrow 1$; connection from sender to server
60. $D[S[chose]][sender] \leftarrow 1$; connection from server to sender
61. **else**
62. $D[S[sender]][S[chose]] \leftarrow D[S[sender]][S[chose]] + 1$;
63. **end if/else**
64. $D[S[chose]][b] \leftarrow D[S[chose]][b] + 1$;
65. $C[S[chose]] \leftarrow C[S[chose]] + flag[S[chose]]$;
66. $min\_stream \leftarrow \infty$;
67. $used\_compressor \leftarrow 0$;
68. **For** c =1 $\rightarrow C[chose]$ **do**
69. $m \leftarrow comp\_connections[S[chose]][c]$; // number of streams
70. **if** ($m < min\_stream$) **Then**
71. $min\_stream \leftarrow m$;
72. $used\_compressor \leftarrow c$;
73. **end if**
74. **end for**
75. $comp\_connections[S[chose]][used\_compressor] \leftarrow comp\_connections[S[chose]][used\_compressor] + 1$;

Phase 4: Find the required compression rate for this stream

76. $New\_t_{s,b} \leftarrow T[a][b] - t - T[a][S[chose]] - T_{m(min\_stream+1)}$;
77. $Real\_Rate \leftarrow (T[S[chose]][b] - New\_t_{s,b})/T[S[chose]][b]$;

Return $C, D, Real\_Rate$

The CRAM heuristic considers video mixing and compressing as two main media handling services. The compressing process is described in Algorithm 5.3. It has three main inputs: (i) two locations that need a compressor in between, (ii) the minimum time that needs

to be reduced by compression, and (iii) the video mixing transmission times and costs between different locations. Note that our proposed compression algorithm does not have a fixed compression rate. It tries to compress as less as possible to have less impact on the video resolution. We also consider a maximum acceptable compression rate (i.e., $Rate_{max}$) as the input for this algorithm.

The compression algorithm has four main phases. In phase one, it finds the servers that are close enough to the video sender and have resources to compress a video stream. According to the servers found, in phase two, it calculates the corresponding cost for assigning the compressing request for each server. The cost is calculated based on the server's resource cost and the network transmissions cost. If the chosen server has no compressor on it, this phase considers the cost of creating a new compressor on the server in the total cost. However, if there is an existing compressor on the server, this phase checks if the compressor can accept another stream. It ensures by checking the satisfaction of the minimum time that needs to be reduced by compression. In case of satisfaction, there is no extra cost for creating a new VM and the server cost is calculated based on the required resources to compress one more stream. On the other hand, if it cannot satisfy, then another compressor needs to be created on this server and the cost of a new VM will be considered.

According to the calculated cost to host a compressor for each server, phase three selects the server with the minimum cost and allocates the required resources for the compressor. Also, it creates a link from the sender to the compressor and from the compressor to the destination. If there is more than one compressor on the chosen server, it always assigns the video stream to a compressor with minimum connected streams. It helps to minimize the overall media handling time. At the end of this algorithm, in phase four it calculates the exact reduced time by compression and also finds the compression rate.

Algorithm 5.4. (ACS): Find the acceptable closest server

Input: M // list of Mixers
u// a participant
S // the sets of servers' locations
Output: s //proposed server with mixer to host u

Phase 1: Find acceptable servers

1. $j \leftarrow 0$;
2. **For** i =1 → $|S|$ **do**
3. **if** ($M[S[i]] > 0$) **Then**
4. **For** m =1 → $M[S[i]]$ **do**

```
5.        if (mixer_connections[S[i]][m] < max_user) Then
6.          j ← j + 1;
7.          possible_servers[j] ← i; keep server i as a possible server
8.          break;
9.        end if
10.     end for
11.   end if
12. end for
Phase 2: Find the closest server from the acceptable servers
13. min_distance ← ∞;
14. s ← 0;
15. For i =1 → |possible_servers| do
16.     if (min_distance > T[U[u]][S[possible_servers[i]]]) Then
17.       s ← possible_servers[i]; //chosen server to assign the participant to a mixer
18.       min_distance ← T[U[u]][S[possible_servers[i]]];
19.     end if
20. end for
21. min_stream ← ∞;
22. For m =1 → M[S[s]]  do
23.     if (mixer_connections[S[s]][m] < min_stream) Then
24.       mixer ← m; // chosen mixer to support participant
25.       min_stream  ← mixer_connections[S[s]][m];
26. end for
27. mixer_connections[S[s]][mixer] ← mixer_connections[S[s]][mixer] + 1;
Return s
```

Algorithm 5.4 is responsible to find the closest server which is hosting a video mixer to a participant. It has two main phases. In the first phase, it finds the servers with at least one video mixer whose total connected streams is less than a maximum possible connection calculated in the phase one of Algorithm 5.1. In phase two, it selects the one which is closest to the participant. Also, it selects the video mixer on this server with the minimum connected streams to be responsible for this mixing request. In addition, it increases the number of connected video streams for the selected video mixer.

5.4. Validations and Measurements

This section describes our evaluation scenarios and the simulation settings followed by the obtained results.

5.4.1. Evaluation Scenarios and Simulation Settings

We consider two different conferencing applications as our evaluation scenarios. (i) Massively Multiplayer Online Game (MMOG) and (ii) Online Distance Learning (ODL). In these scenarios, the conference participants are sharing their videos in the logic of the

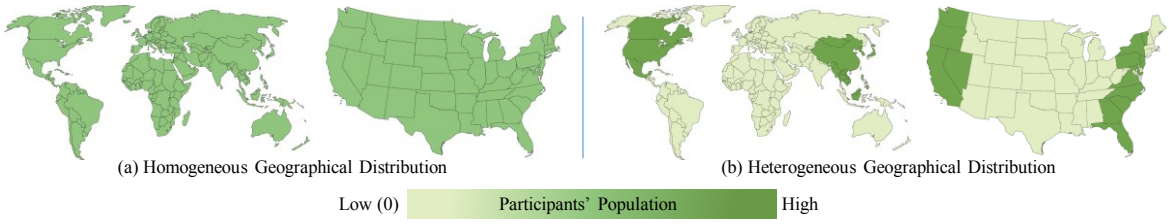


Fig. 5.1. Geographical distribution of participants in conferencing applications

application. The aim is to allow each participant to have a mixed video from all other participants. In MMOG, participants are from different geographical locations in the world. Thus, the end-to-end delay may be high. In contrast, in ODL, the number of participants is limited and they are distributed in a smaller area such as one country. For our simulation, we consider two different geographical distributions for participants as depicted in Fig. 5.1. (a) *Homogeneous* – participants are distributed over the whole area (i.e., world or country) with similar density. (b) *Heterogeneous* – the majority of the participants are geographically distributed in the east and the west side of the area. These distributions can help to understand the behavior of the proposed solution when the participants are close or far from each other.

For our simulations, we consider having servers in twenty cities over the world for MMOG and nine cities over the USA for ODL. For the network transmission time between servers, we use the information available at [82]. Fig. 5.2 shows the locations of considered servers. Also, we consider different number of participants for both scenarios. We assume a snapshot of the number of participants in this work. To study the impact of servers' resources and network costs, we consider various settings with different simulation parameters. We assume that the network transmission cost between two locations is a linear function of the transmission time between them. In fact, the farther two locations are, the higher is the network cost between them. Also, for the media handling time and required

Fig. 5.2. Geographical distribution of the servers

Table 5.3. Simulation parameters and settings

	MMOG	ODL
Number of servers	20	9
Servers geographical distribution	Over the world	Over the USA
Number of participants	100, 2000, 3000	100, 200, 500
Participants' geographical distribution	Homogeneous: Equally distributed in each server's location	
	Heterogeneous: Half of users are in the western city and half are in the eastern one	
$T_{m(k)}$	6 msec per video source	
$R_{m(k)}$	20 MB (RAM) per video source	
R_o	400 MB (RAM)	
R_ε^s	10240 MB (RAM) per each server	
T_ε	400 msec	
P_s	\$0.1 per MB	
$T_{a,b}$, $P_{a,b}$	As in [1]	
Maximum acceptable compression rate	0.95	

resources, we consider our prototype experience in chapter 4. The simulation parameters and settings are depicted in table 5.3. In our evaluation, we account for the server resource in terms of used memory. However, the mathematical model and our heuristic are general enough to accommodate other types of resources as well.

5.4.2. Results

We solve our mathematical model to achieve optimality for the small-case scenario using LPSolve Java Library (http://lpsolve.sourceforge.net). For the medium-scale and large-scale scenarios (i.e., scenarios with a higher number of participants) deriving the optimal solution with the exact algorithms used by the solver is very time-consuming. Therefore, we only present the results of our heuristic that can support the number of participants in our simulation settings. However, the results in the small-case scenario allow us to validate our mathematical model. In addition, they show that our mathematical model enables the orchestration of media handling services and the possibility of composing these services on the fly. As an example of the result of the mathematical model for a small-case scenario, we ran our model while having 6 participants in Seattle and 2 participants in Toronto. The result shows a composition of one video mixer and one compressor. It allocates required resources for the video mixer in Seattle and for the compressor in Toronto.

In ODL, we assume all participants are from the USA with homogeneous or heterogeneous geographical distributions. We run the CRAM heuristic for 100, 200, and

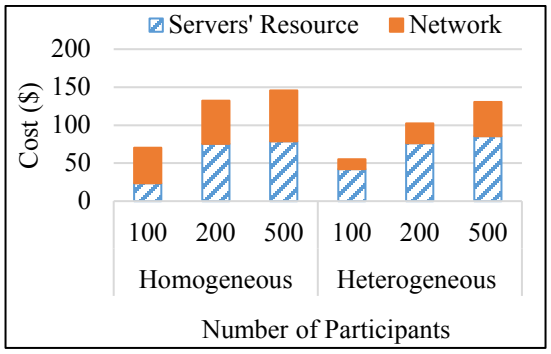


Fig. 5.3. CRAM heuristic total cost in ODL

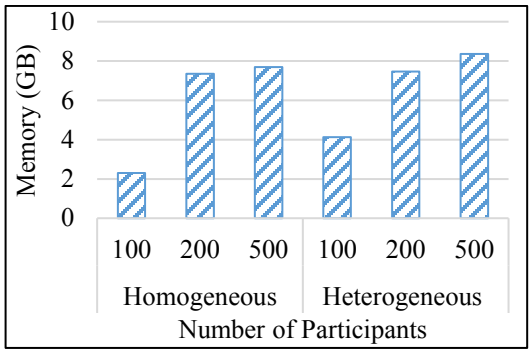


Fig. 5.4. CRAM heuristic total memory allocation in ODL

500 participants. Fig. 5.3 shows the total cost by considering both servers' resources and network costs. By increasing the number of participants, the need for media handling services increases. This leads to allocating more resources and implies a higher communication traffic as well. Thus, as depicted in fig. 5.3, the total cost increases as a higher number of participants is considered. However, considering the same number of participants, the total cost in homogeneous geographical distribution is greater than that of the heterogeneous geographical distribution. The reason is that the heterogeneous geographical distribution favors the execution of some media handling services locally. By that, it leads to transmit a lower number of streams over the network and implies a lower total cost.

Fig. 5.4 depicts the servers' resources (i.e., RAM) that is allocated for media handling services. By increasing the number of participants, our heuristic allocates more resources to media handling services to cope with the requests. The amount of memory allocation for the same number of participants is greater in the case of heterogeneous geographical distribution. In fact, in the homogeneous geographical distribution of ODL, most of the participants can reach the mixers without the need of passing through the compressors. It leads to using fewer compressors in homogeneous and less memory allocation compared to heterogeneous.

Fig. 5.5 shows the network cost. By increasing the number of participants, the traffic grows, implying a higher network cost. Unlike servers' resources, the network cost is less in heterogeneous geographical distribution in comparison with homogeneous for the same number of participants. In fact, the aggregation of participants helps to decrease the network

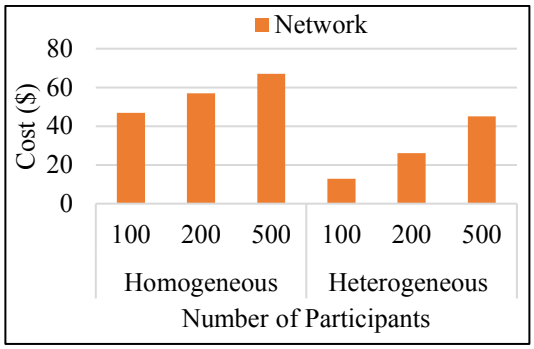


Fig. 5.5. CRAM heuristic network cost in ODL

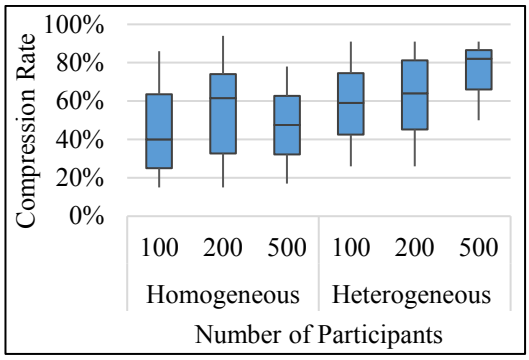


Fig. 5.6. CRAM heuristic video compression rate in ODL

communications and reduces the network cost. However, as it is depicted in Fig. 5.6, it causes more compression rate in heterogeneous in comparison with homogeneous geographical distribution for the same number of participants. In fact, the compressors should serve a higher number of participants in heterogeneous geographical distribution. Thus, it increases the compression rate to cope with the QoS threshold and reduces the network transmission time. The lines in the boxes indicate the median for the compression rate.

On the other hand, in MMOG, we assume all participants are from different locations in the world. In this scenario, CRAM heuristic runs for 100, 2000, and 3000 number of participants. As depicted in Fig. 5.7, similar to the ODL, by increasing the number of participants, the total cost will increase as well. Also, the total cost for the same number of participants in heterogeneous geographical distribution is less than that of the homogeneous geographical distribution. Based on that, both evaluation scenarios show that regardless of the area size, the aggregation of participants can help reduce the total cost.

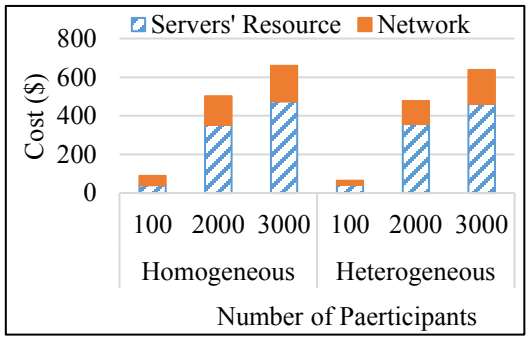


Fig. 5.7. CRAM heuristic total cost in MMOG

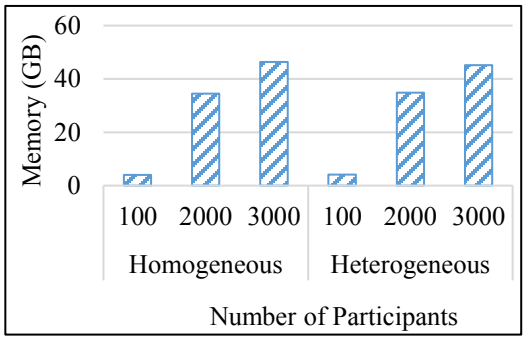


Fig. 5.8. CRAM heuristic total memory allocation in MMOG

The memory allocations for different numbers of participants in MMOG is depicted in Fig. 5.8. Unlike the results of ODL, the memory allocation for MMOG in both homogeneous and heterogeneous geographical distributions are almost the same. The reason is that in MMOG, even in the homogeneous geographical distribution, the participants are far from each other. This leads to using several compressors. In fact, the aggregation of the participants into two locations does not help to reduce the required resources for compressing service. However, as it is depicted in Fig. 5.9, the aggregation can help to reduce the network cost in heterogeneous geographical distribution. Although the network cost is decreased by the aggregation, it leads to more compression rate as it is shown in Fig. 5.10. In other words, more participants end up with lower video resolution in comparison with homogeneous geographical distribution.

For the composition, the CRAM heuristic orchestrates the required instances of media handling services for participants. Note that each participant may follow a specific media handling composition which differs from others. Fig. 5.11. shows an example of the created compositions for two different participants in different locations. As shown in the figure, CRAM may assign the participant from Seattle to a mixer which is hosted by a server in Seattle. Thus, this participant will receive the final mixed stream from that mixer as well. However, if CRAM allocates resources to the mixers in Seattle and a participant from Toronto wants to use the mixers, to respect the maximum latency, CRAM allocates a compressor in a location which reduces the total cost and assigns the participant from Toronto to it. Then, the result of compression is sent to the mixer in Seattle. For this specific

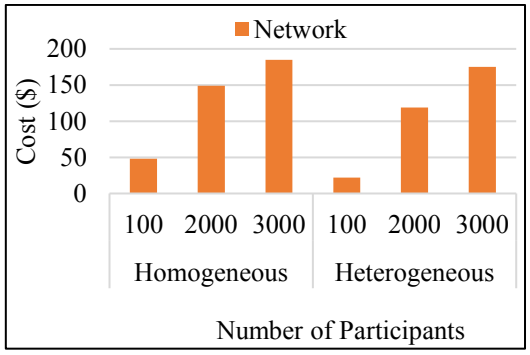


Fig. 5.9. CRAM heuristic network cost in MMOG

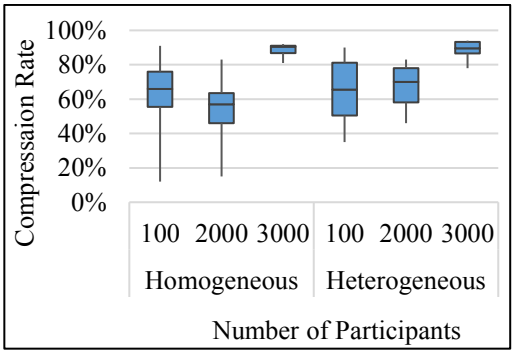


Fig. 5.10. CRAM heuristic video compression rate in MMOG

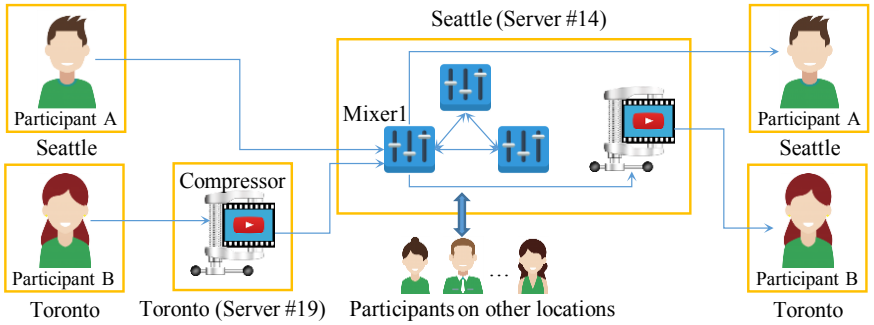


Fig. 5.11. Two different media handling compositions for users in Seattle and Toronto

example, CRAM allocates a compressor on the Seattle server as well. Therefore, the final results are compressed one more time and then it sends to the participant in Toronto.

5.5. Conclusion

This chapter presents another novel cloud-based resource allocation algorithm for multimedia conferencing applications. We consider the conferencing applications in this chapter with video mixing and compressing services. We proposed CRAM to allocate resources in an efficient manner for these applications. CRAM considers scaling the resources in an elastic manner while meeting the QoS requirements and considering the fluctuation in the number of participants. The proposed algorithm in this chapter considers reducing both servers' resource cost and network cost. Also, it takes into account the end-to-end delay as QoS requirements, considering both media handling service response time and network latency. We mathematically formulated the problem and also proposed the heuristics to solve the large-scale scenarios in an acceptable time. Our simulation results show that the number of participants and their geographical distribution have a significant impact on the servers' resource cost, network cost, and the required compression rate for video streams.

Chapter 6

6. An Offline Scaling Mechanism for Multimedia Conferencing Applications

6.1. Introduction

As it was described before, the conferencing PaaS in collaboration with the conferencing IaaS are responsible to scale the conferencing applications. In chapters 4 and 5, we proposed novel algorithms to allocate resources for conferencing applications in the IaaS layer. However, we still need to know when and for how many participants the conferencing applications should scale to meet the cost-efficiency objective and QoS requirements. This chapter presents a novel adaptive scaling algorithm for multimedia conferencing applications in the PaaS layer. The proposed scaling algorithm in the PaaS layer is responsible to find the best time for scaling these applications. In addition, it decides for how many participants the conferencing applications should scale to meet the cost-efficiency objective and QoS requirements. The proposed algorithm in this chapter enables the conferencing applications to scale in an elastic manner with respect to the number of participants. Also, it meets the QoS requirements while considering the future demands of the conferencing applications (i.e., future number of participants).

We entitled the proposed algorithm in this chapter as ADS (Adaptive and Dynamic Scaling). The main focus of ADS algorithm is on reducing the resource cost while considering the QoS requirements. ADS works in an offline manner and uses a prediction model to forecast the future number of participants. The dynamicity of ADS facilitates the on-demand scaling up or down of the conferencing applications. In addition, the scaling policies can change adaptively and in accordance with the fluctuating number of conference participants to ensure elasticity.

The rest of this chapter is as follows. First, it presents the ADS by discussing its system model. Then, it discusses the designed heuristic for it. After that, it presents the simulation parameters and settings of ADS followed by the validation results. We conclude this chapter at the end.

6.2. ADS System Model

The system model of ADS includes cooperation and mathematical models. In our mathematical model, we define ADS as an ILP problem.

6.2.1. Cooperation Model

We consider a large-scale cloud environment to support the scaling of the conferencing applications. It consists of users as conference participants, a conferencing PaaS and, multiple conferencing IaaSs. The conference participants across a large geographical area want to join a conferencing application, such as MMOG. We assume there is a service level agreement (SLA) between the conferencing application provider and the PaaS, where the QoS requirements are defined. One such requirement is the maximum acceptable delay for a participant to join the conference (θ). Moreover, we assume there is a SLA between the conferencing PaaS and the conferencing IaaSs, where another set of QoS requirements are defined. One such QoS requirement is the time to provision resources in the IaaSs (δ).

When a conference participant wants to join the conference, the required resources should be provisioned within θ time slots. In addition, when the scaling request is sent to the IaaSs, it takes δ time slots for resources to be provisioned. The challenge lies in finding

the best time to send the scaling request. Moreover, this entails finding the required amount of resources to achieve the optimal resource cost while guaranteeing QoS requirements.

6.2.2. Mathematical Model

This section presents our ADS problem formulation, which is modeled as an ILP problem. It presents the problem statement followed by the objective and constraints.

(iv) Problem Statement

Given n time slots of equal durations, let A and D represent the sets of expected arrivals and departures of conference participants, respectively. Such that, there will be a maximum of $a_i \in A$ and $d_i \in D$ participants, joining and leaving the conference during time slot i, respectively. It is assumed that A and D are available before the conference is started. Also, there is a threshold θ pertaining to the maximum acceptable delay before a participant can join the conference. We assume that θ is a multiple of time slots. Upon sending of the scaling request from the PaaS to the conferencing IaaSs, it is assumed that the required resources will be allocated within the time lag δ. We assume that δ is a multiple of time slots and the δs for scaling up and scaling down are equal. Moreover, we assume the IaaS does not accept parallel scaling requests for the same conferencing service. Therefore, we assume there is at least δ time slots between two consecutive scaling requests. To simplify the problem, we consider the same δ for all IaaSs. In addition, we assume $\delta < \theta$. The goal is to find the optimal scaling schedule, such that the total amount of allocated resources in terms of the number of participants is minimized over the conference duration.

We model this as an ILP problem where we assume that each conference participant needs the same amount of resources to join the conference. Tables 6.1 and 6.2 delineate the inputs and variables of our problem, respectively.

Table 6.1. Problem Inputs

Input	Definition
n	Total number of time slots in the entire conference duration
A	A set of expected arrivals of conference participants, such that during time slot i, a maximum $a_i \in A$ participants join the conference, $1 \leq i \leq n$
D	A set of expected departures of conference participants, such that during time slot i, a maximum $d_i \in D$ participants leave the conference, $1 \leq i \leq n$
L	A set of number of conference participants, such that during time slot i, a maximum of $l_i \in L$ participants are in the conference for more than θ time slots, $1 \leq i \leq n$
δ	The time lag, stipulated in the conferencing IaaS SLA for the response to the resource provisioning request. $\delta > 1$ time slot, otherwise the problem is trivial.
θ	Maximum acceptable delay for preparing the conference service
M	A big enough constant

Table 6.2. Problem Variables

Variable	Definition
X	$n \times n$ matrix, where $x_{i,j}$ is the actual number of participants allocated to the service at time slot i whose corresponding request is sent from PaaS to the IaaS at time slot j
Y	$n \times n$ matrix, where $y_{i,j}$ is the actual number of participants de-allocated from the service at time slot i whose corresponding request is sent from PaaS to the IaaS at time slot j
R	A vector of binary variables, where $r_j = \begin{cases} 1, if\ PaaS\ sends\ scaling\ request\ to\ IaaS\ at\ time\ slot\ j \\ 0,\ otherwise \end{cases}$

(v) Objectives

We assume that the cost of using resources at each time slot depends on the total number of participants in the conference at that time slot. Our objective is to minimize the cost while considering other QoS requirements. We consider the provisioned resources in terms of the number of participants and the remaining time of the conference after provisioning the resources. The resource allocation and de-allocation for time slot i, for which the request is sent to IaaSs at time slot j are represented as $x_{i,j}$ and $y_{i,j}$, respectively. Since the result of the scaling request will be ready after δ time slots, the remaining time of the

conference after sending the scaling request at time slot j will be $n - (j + \delta)$. Equation (1) depicts our objective.

$$minimize\left\{\sum_{i=1}^{n}\sum_{j=1}^{n-\delta}(x_{i,j} - y_{i,j}) \times (n - (j + \delta))\right\} \quad (1)$$

(vi) Constraints

To respect the maximum acceptable delay (i.e., threshold θ), the allocated resources, in terms of conference participants, between time slot i and $i + \theta$ should be greater than or equal to the expected number of participants arriving at time slot i. In other words, in the SLA between PaaS and the application providers, the conferencing PaaS guarantees that there will be no user waiting for more than θ time slots to be served before the conference ends. Equations (2) and (3) enforce this constraint. Note that the resources can be reserved before or after arrivals of users. It means that the scaling request time (i.e., j in these equations) can be from the moment that conference was started until the end of the conference.

$$\sum_{j=1}^{i+\theta-\delta} x_{i,j} \geq a_i \qquad \forall\, 1 \leq i \leq (n - \theta) \quad (2)$$

$$\sum_{j=1}^{n-\delta} x_{i,j} \geq a_i \qquad \forall\, (n - \theta) < i \leq n \quad (3)$$

If there are some participants in the conference and PaaS provides them their required service, the conference size cannot be scaled down more than the number of participants who are remaining in the conference. In fact, the conference size cannot shrink before participants leave the conference, as in equations (4), (5) and (6).

$$\sum_{j=1}^{n-\delta} y_{i,j} \leq d_i \qquad \forall\, 1 \leq i \leq \delta \quad (4)$$

$$\sum_{j=i-\delta}^{n-\delta} y_{i,j} \leq d_i \qquad \forall\, \delta + 1 \leq i \leq n \quad (5)$$

$$\sum_{j=1}^{i-\delta-1} y_{i,j} = 0 \qquad \forall\, \delta + 1 < i \leq n \quad (6)$$

The maximum amount of scaling down requests at each time slot cannot be more than the maximum of total allocated resources before that time slot. This is guaranteed in equation (7).

$$\sum_{i=1}^{n}\sum_{t=1}^{j} x_{i,t} \geq \sum_{i=1}^{n}\sum_{t=1}^{j} y_{i,t} \qquad \forall\, 1 \leq j \leq n \qquad (7)$$

Based on A and D, the set L can be defined, such that there will be a maximum of $l_i \in L$ participants in time slot i, who can be in the conference for more than θ time slots. Therefore, at each time slot, the prepared conference size should at least have the required resources for the participants who have been in the conference for more than θ time slots. Equation (8) represents this constraint.

$$\sum_{i=1}^{n}\sum_{t=1}^{j-\delta} x_{i,t} - \sum_{i=1}^{n}\sum_{t=1}^{j-\delta} y_{i,t} \geq l_j \qquad \forall\, \delta < j \leq n \qquad (8)$$

The conferencing IaaSs can accept the new scaling request from the PaaS after the previous request has been processed completely. Therefore, two consecutive scaling requests from the conferencing PaaS must be separated by δ, as depicted in (9).

$$\sum_{j=i}^{i+\delta-1} r_j \leq 1 \qquad \forall\, 1 \leq i \leq n-\delta \qquad (9)$$

Moreover, any changes in the conference size made at time slot j, should be mapped to their scaling request at the same time slot as shown in equations (10) and (11). We assume M is a big enough constant in these equations.

$$M \times r_j \geq x_{i,j} \qquad \forall\, 1 \leq i,j \leq n \qquad (10)$$

$$M \times r_j \geq y_{i,j} \qquad \forall\, 1 \leq i,j \leq n \qquad (11)$$

To avoid unnecessary resource allocation or de-allocation, there should be no scaling requests over the last δ time slots of the conference. In fact, such a request, if made, will take effect after the end of the conference. Through equation (12), we ensure that such requests are not sent.

$$r_j = 0 \qquad \forall\, n-\delta < j \leq n \qquad (12)$$

6.3. ADS Heuristic

Based on the proposed mathematical model, reaching the optimal solution for the large-scale scenarios is very time-consuming. Therefore, we propose an ADS heuristic as well to reach a sub-optimal solution in a reasonable time. The ADS heuristic tries to find the best schedule for scaling requests while respecting the SLAs. Algorithm 6.1 delineates the ADS heuristic. It iterates over the set of time slots throughout the conference. We consider the constants shown in Table 6.1 as the inputs of this algorithm. Also, the output of the ADS algorithm is an integer array S with n elements. Each $s_i \in S$ represents the required scaling amount at time slot i. ADS heuristic has two main phases. In the first phase, it tries to find the minimum possible conference size and the best time for scaling the conference. In the second phase, it makes sure that all scaling requests are separated by at least δ time slots.

Since the cost depends on the amount of the provisioned resources and their usage over time, ADS heuristic is designed with the objective of reserving the least resources, as late as possible. The latest time should respect δ and θ. Also, the minimum amount should respect the number of participants who are in the conference. Therefore, in phase 1, ADS tries to find the minimum size of the conference and the best time to send the scaling request. Based on the inputs, conference scaling takes δ time slots. Therefore, at each time slot i, ADS should consider the total conference size of δ time slots ahead. Also, new participants can wait up to θ time slots to join the conference. Thus, ADS can consider it as well and checks the total conference size up to θ time slots ahead. In consequence, since the objective is to find the minimum cost, ADS considers the minimum conference size between time slots $i + \delta$ and $i + \theta$.

In phase 2, ADS heuristic ensures that the consecutive scaling requests are separated by more than δ time slots. Moreover, it keeps track of the previous scaling request and its corresponding conference size. ADS compares the previous conference size with the result of phase 1 to decide about the scaling amount as the output of the algorithm. A positive value in the output means the request is to scale up, while a negative one means to scale down.

Algorithm 6.1. ADS Heuristic

Input:
$n, \delta, \theta, A, D;$ // same as the inputs of Table 6.1
Output: $\boldsymbol{S}$; // an schedule set of scaling decisions

64. old_size ← 0 // previously provisioned size of the conference
65. new_size ← 0 // conference size that should be provided for the future
66. **For each** $i \in n$ **do**
67. min_size ← ∞
68. best_t ← 0

Phase 1: Find the best possible time for sending the scaling request

69. **For** t = $i + \delta \rightarrow i + \theta$ **do**
70. total_size← 0
71. **For** p=1→ t **do**
72. total_size ← total_size + $a_p - d_p$
73. **end for**
74. **If** (min_size ≥ total_size) **Then**
75. min_size ← total_size
76. best_t ← t − δ
77. **end if**
78. **end for**

Phase 2: Set the amount of scaling request for the best found time and move i to the next available time for sending a request to the IaaSs

79. new_size ← min_size
80. $S[best\_t]$ ← new_size − old_size
81. old_size ← new_size
82. i ← best_t + δ − 1; // -1 because it is in the loop and i for next cycle will be (best_t− δ)
83. **end for each**

Return S

6.4. Validations and Measurements

In this section, we will describe our evaluation scenarios and the simulation settings, followed by comparison results.

6.4.1. Evaluation Scenarios and Simulation Settings

As the evaluation scenarios, we consider two different conferencing applications. (i) Massively Multiplayer Online Game (MMOG) and, (ii) Online Political Party Discussion (OPPD). In both scenarios, the users as the conference participants, are sharing their videos and audios in the logic of the application. In MMOG, users join and leave the game from all over the world. Thus, there is a significant fluctuation in the number of participants. In contrast, in OPPD, since the participants are limited, the fluctuation of the conference size is small.

Table 6.3. Simulation Parameters and Settings

General Parameters	Value	MMOG Settings		OPPD Settings	
n	100	A and D Fluctuation	0-1500	A and D Fluctuation	0-300
δ	3				
θ	4				
M	1000000				

For our simulation, we randomly generate the number of participants joining and leaving the conference at each time slot. To cover all possibilities, we keep the same conference size over a part of this time. This means that either no one joins or leaves the conference, or the number of users joining the conference is equal to the number of users leaving at each time slot, over that part. In our simulation, we divide the conference duration to 100 time slots. Also, we assume the resource provisioning time and the acceptable delay are 3 and 4 time slots, respectively. In addition, we set the fluctuation of the number of users to up to 1500 and 300 in MMOG and OPPD, respectively. Simulation parameters and settings are depicted in Table 6.3.

6.4.2. Results

We implement the ADS algorithm in JAVA. Also, we use the LPSolve engine [83] to find the ADS optimal solution for our mathematical model. We compare the results of our algorithm with that of the optimal solution and the expected conference size. Also, we use a greedy algorithm as the baseline of our comparison. Since there is no similar heuristic in the literature that meets all of our requirements, this allows us to assess how our heuristic performs with respect to a simple greedy approach. The greedy algorithm operates on a periodic basis with a period equal to δ. At time slot t (with $t \ mod \ \delta = 0$), it derives the maximum number of participants between time slots $t + \delta$ and $t + 2\delta$. It then scales the conference accordingly. By that, the greedy approach is capable of satisfying the threshold of user's acceptable delay. Fig. 6.1 and 6.2, depict the created conference size for MMOG and OPPD applications, respectively. As these figures show, both our optimal and heuristic solutions can scale the conference size up and down. The scaling is elastic and it respects the SLAs.

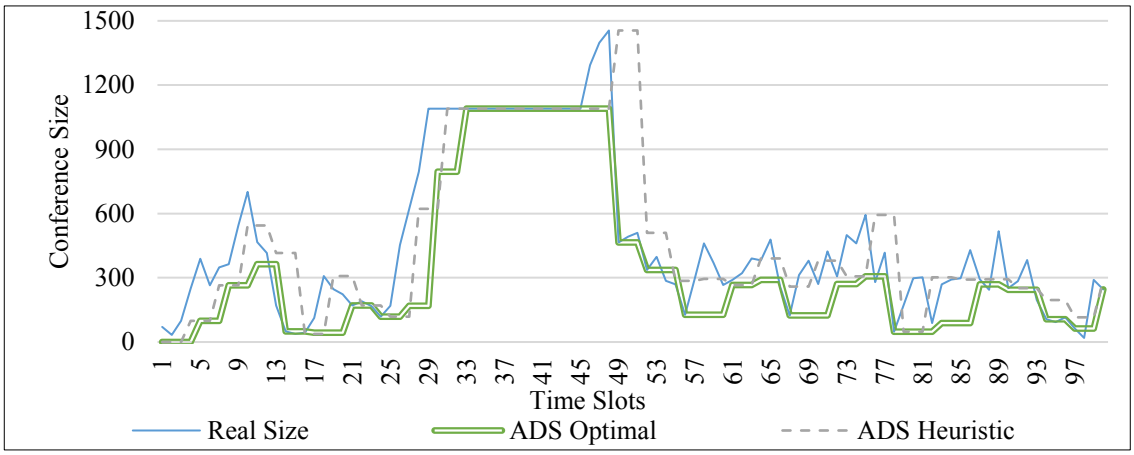


Fig. 6.1. Conference Size Comparison in MMOG

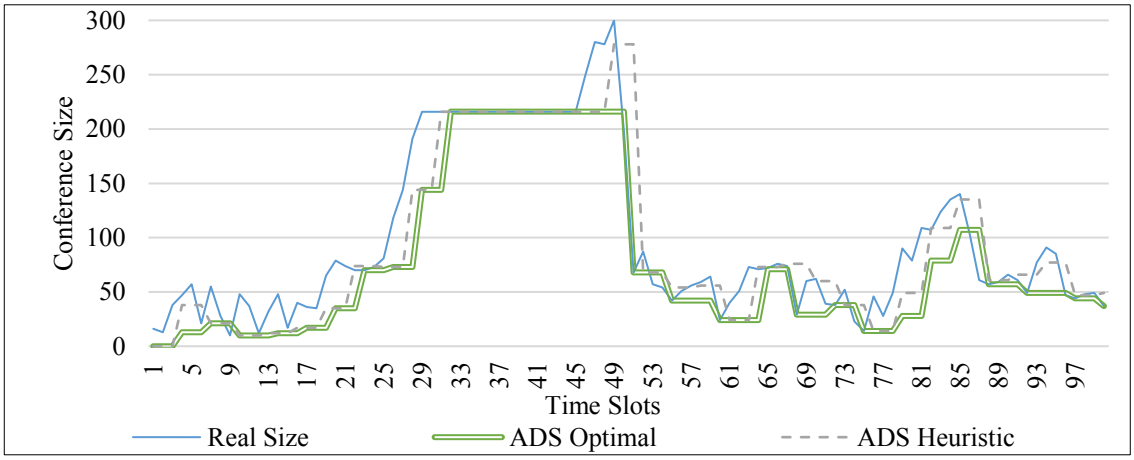


Fig. 6.2. Conference Size Comparison in OPPD

Although in our scenarios, users can wait up to θ time slots to join the conference, there could be a cost for the delay as QoS violation. Fig. 6.3 and 6.4 show the total resource allocation and QoS violation costs of our scaling mechanism for MMOG and OPPD, respectively. As shown in these figures, the ADS heuristic outperforms the greedy algorithm from a resource-efficiency perspective. It leads to a solution that is closer to optimality with respect to the solution of the greedy algorithm, implying lower resource cost. However, this comes at the cost of a higher QoS violation. By comparing the solutions obtained from different algorithms, we notice that the greedy approach implies the least cost of QoS violation. It is followed by our ADS heuristic, while the ADS optimal solution leads to the highest QoS violation cost. These results highlight the trade-off that exists between the resource efficiency and QoS.

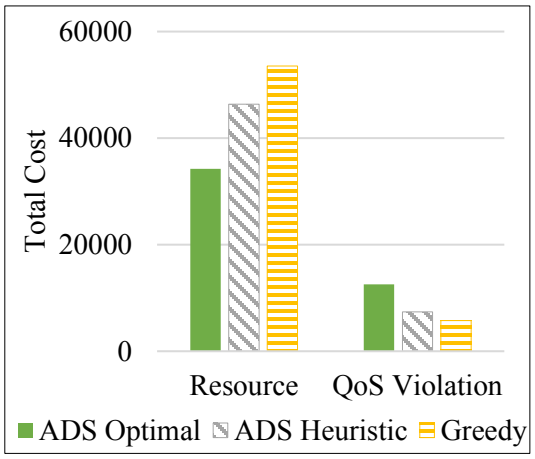


Fig. 6.3. Costs of Resources and QoS Violation in MMOG

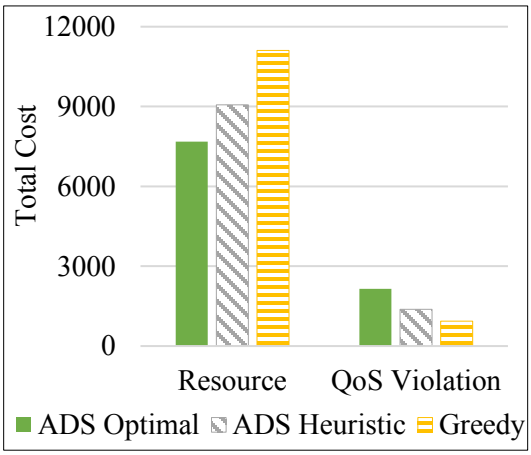


Fig. 6.4. Costs of Resources and QoS Violation in OPPD

Fig. 6.3 and 6.4 also show that the cost of the ADS heuristic for provisioning resources in OPPD and MMOG has an 18% and a 35% gap from the optimal solution, respectively. It means that the ADS heuristic can perform better when scaling conferences with lower fluctuations.

6.5. Conclusion

This chapter presents ADS, a novel scaling algorithm for cloud-based multimedia conferencing applications. The ADS produces the cost-efficient scaling schedule while considering the QoS requirements and the future demands of the conferencing applications. The main objective of ADS is minimizing the resource cost and it performs in an offline manner. We mathematically formulated the problem and also proposed the heuristic to solve the large-scale scenarios in an acceptable time. Simulation results show the elasticity of ADS mechanism for conferencing services. Moreover, we show that the proposed ADS heuristic outperforms a simple greedy algorithm from a resource-efficiency perspective. Although ADS considers the future demands of the conferencing applications, it does not consider the uncertainty in the prediction. In addition, it only minimizes the resource cost while meeting the QoS requirements. But it does not consider reducing the QoS violation cost. These limitations of ADS will be solved in chapter 7.

Chapter 7

7. An Online Scaling Mechanism for Multimedia Conferencing Applications

7.1. Introduction

In the previous chapter, we proposed ADS, an adaptive and dynamic scaling algorithm for conferencing applications. As it was mentioned before, the ADS works in an offline manner. Moreover, although it considers the future demand of the conferencing application, it does not take into account the uncertainty in the prediction model. In addition, the main focus of ADS is on reducing the resource cost and does not consider the QoS violation cost. This chapter proposes another novel scaling algorithm for multimedia conferencing applications in the PaaS layer to solve the limitations of the ADS. We entitled the proposed algorithm as AOS (Adaptive and Online Scaling). AOS performs in an online manner and finds the best time for scaling the conferencing applications. In addition, it decides for how many participants the conferencing applications should scale to meet the cost-efficiency objective and QoS requirements. Besides reducing the resource cost, AOS considers reducing the QoS violation cost as well. In addition, it takes into account the uncertainty of the prediction model. Similar to ADS, the AOS also enables scaling the

conferencing applications in an elastic manner and in terms of the number of conference participants.

The rest of this chapter is as follows. First, it presents the AOS by discussing its system model. Then, it describes the designed AOS heuristic. After that, it presents the simulation parameters and settings of AOS followed by the validation results. We will conclude this chapter at the end.

7.2. AOS System Model

In AOS, besides reducing the resource cost, reducing the QoS violation cost is part of the objective. In addition, it takes into account the uncertainty in the prediction model. Despite ADS that works in an offline manner, AOS is designed to work online. Similar to ADS, the AOS system model includes the cooperation and mathematical models. In the mathematical model, we define AOS as an ILP problem.

7.2.1. Cooperation Model

We consider a large-scale cloud environment to support the scaling of the conferencing services. The conference participants across a large geographical area request to join a conferencing application, such as MMOG. We assume there is an SLA between the conferencing application provider and the PaaS, where the QoS requirements are defined. One such requirement is the maximum acceptable delay for a participant to join the conference (θ). Moreover, we assume there is an SLA between the conferencing PaaS and conferencing IaaSs, where another set of QoS requirements are defined. One such QoS requirement is the time to provision resources in the IaaS (δ).

According to the defined SLAs, when a conference participant requests to join the conference, the required resources should be ready. Otherwise, PaaS should pay the QoS violation cost for waiting time of each participant. Moreover, the maximum acceptable waiting time is θ. In addition, when the scaling request is sent to the IaaSs, it takes δ time for the resources to be provisioned and a new scaling request will have to wait for the realization of the previous request. Therefore, choosing the time to send the scaling request

can be challenging as it affects QoS violation and resource costs. On one hand, if resources are not ready when a participant joins the conference, the participant will have to wait which implies QoS violation costs for the PaaS. On the other hand, if resources are allocated prior to the arrival of the participant, additional unnecessary resource costs may be incurred.

7.2.2. Mathematical Model

This subsection presents the AOS problem formulation, which is modeled as an ILP problem.

(iv) Problem Statement

Given n time slots of equal duration as the total conference time, let F represent a time frame in the conference that spans over $|F|$ time slots, where $|F|$ is less than n. Also, let A_F and D_F represent the sets of real-time arrivals and departures of conference participants during a time frame F, respectively. Such that, there will be $a_i \in A_F$ and $d_i \in D_F$ participants joining and leaving the conference during time slot i, respectively. Since A_F and D_F are in real-time, the corresponding values are not known in advance. It is assumed that there is a prediction model which can predict the arrivals and departures of the conference participants for one time frame ahead, with the accuracy of $\varepsilon\%$ and the prediction intervals of $\pm\gamma\%$. That is, $\varepsilon\%$ of predictions concur with the real-time number of participants while mispredictions are within $\gamma\%$. Let A'_F and D'_F represent the sets of predicted arrivals and departures of conference participants during a time frame F, respectively. Such that, there will be a prediction of $a'_i \in A'_F$ and $d'_i \in D'_F$ participants joining and leaving the conference during time slot i in F, respectively. It is assumed that the values of A_F, D_F, A'_F, and D'_F for all previous time frames are saved in A^P, D^P, A'^P, and D'^P sets, respectively. A'_F and D'_F are generated during the conference and use the values of A^P and D^P to tune the prediction.

We assume that the scaling process has no effect on the conferencing services during runtime. That is, adding or releasing of resources during runtime is supported. We also assume that δ and θ are multiples of time slots and δs for adding resources (i.e., scaling

up/out) and releasing resources (i.e., scaling down/in) are equal. We consider this to simplify the problem.

In a cloud environment, IaaSs do not accept parallel scale up and scale down requests for a specific resource (i.e., a virtual machine or a container) [25]. Therefore, we assume the IaaS does not accept parallel scaling requests for the same conferencing service. Consequently, there are at least δ time slots between two consecutive scaling requests. In our problem, we consider the same δ for all IaaSs. Hence, we do not choose the best offered time between different available IaaSs. In addition, we assume $\delta < \theta$ to simplify the problem. Moreover, since we consider the prediction knowledge for a time frame, we assume δ and θ are less than $|F|$. These assumptions ensure the feasibility of using the predicted information to make the scaling decisions over one time frame.

The goal is to find an optimal online scaling schedule, such that, the total cost of allocated resources and QoS violations are minimized over the conference duration. We model the problem as an ILP problem, where we assume that each conference participant needs the same amount of resources to join the conference. Tables 7.1 and 7.2 delineate the inputs and variables of our problem, respectively.

(v) Objectives

We assume that the cost of using resources at each time slot depends on the total number of participants in the conference at that time slot. Also, we assume that the cost of QoS violation at each time slot depends on the total number of participants waiting to join the conference. Our objective is to minimize the total resource allocation and QoS violation costs while considering QoS requirements. We aim to reach this objective by minimizing these costs over each individual prediction time frame while accounting for the decisions made in the previous time frames.

We consider that the resources and QoS violation costs are evaluated in terms of the number of participants. The resource allocation cost is calculated based on the remaining time of the conference after provisioning the resources. We use $x_{i,j}$ and $y_{i,j}$ to represent a request sent at time slot j to allocate and de-allocate resources for time slot i, respectively.

Table 7.1. Problem Inputs

Input	Definition
n	Total number of time slots in the entire conference duration
$\|F\|$	The duration of a time frame
A_F	A set of real-time arrivals of conference participants in time frame F, such that in time slot i, $a_i \in A_F$ participants join the conference
D_F	A set of real-time departures of conference participants in time frame F, such that in time slot i, $d_i \in D_F$ participants leave the conference
A^P	A set of actual arrivals of conference participants during past time frames, such that in time slot i, $a_i^P \in A^P$ participants joined the conference
D^P	A set of actual departures of conference participants during past time frames, such that in time slot i, $d_i^P \in D^P$ participants left the conference
A'_F	A set of predicted arrivals of conference participants in time frame F, such that during time slot i, $a'_i \in A'_F$ participants are predicted to join the conference
D'_F	A set of predicted departures of conference participants in time frame F, such that during time slot i, $d'_i \in D'_F$ participants are predicted to leave the conference
A'^P	A set of predicted arrivals of conference participants during past time frames, such that in time slot i, $a'^P_i \in A'^P$ participants were expected to join the conference
D'^P	A set of predicted departures of conference participants during past time frames, such that in time slot i, $d'^P_i \in D'^P$ participants were expected to leave the conference
L_F	A set of the number of conference participants, such that in time slot i, maximum $l_i \in L_F$ participants had been in the conference for more than θ time slots
X^P	A set of allocated resources in the past time frames, such that $x_{i,j}^P \in X^P$ represents the allocated resource for time slot i whose request was sent at time slot j
Y^P	A set of de-allocated resources in the past time frames, such that $y_{i,j}^P \in Y^P$ represents the de-allocated resource for time slot i whose request was sent at time slot j
P_{t-1}	The gap between existing number of participants and allocated resources before a time frame starts at time slot t
δ	The time lag, stipulated in the conferencing IaaS SLA for meeting the resource provisioning request. $\delta > 1$ time slot, otherwise the problem is trivial
θ	The acceptable delay for preparing the conference service
ε	The accuracy rate of the prediction model
γ	The prediction interval of the prediction model
β	The weighting coefficient between resource cost and QoS violation cost in the objective
M	Large enough constant

Since the result of the scaling request will be ready after δ time slots, the remaining time of the conference after the request takes effect will be $n - (j + \delta)$. Let C_F^R represent the total resource allocation cost over a prediction time frame. Equations (1) and (2) depict C_F^R for a

Table 7.2. Problem Variables

Variable	Definition
X	$\|F\| \times \|F\|$ matrix, where $x_{i,j}$ is the allocated resources to the service, in terms of number of participants, at time slot i whose request is sent from PaaS to the IaaS at time slot j
Y	$\|F\| \times \|F\|$ matrix, where $y_{i,j}$ is the de-allocated resource from the service, in terms of number of participants, at time slot i whose request is sent from PaaS to the IaaS at time slot j
R	A vector of binary variables, where $r_j = \begin{cases} 1, if\ PaaS\ sends\ scaling\ request\ to\ IaaS\ at\ time\ slot\ j \\ 0,\ otherwise \end{cases}$
C_F^R	Total resource allocation cost in a time frame
C_F^Q	Total QoS violation cost in a time frame

time frame which starts at time slot t. Since each resource allocation or de-allocation takes δ time slots, we do not consider requests in the last δ time slots of the conference duration (i.e., in the last time frame) in these equations.

$$C_F^R = \sum_{i=t}^{t+|F|} \sum_{j=t}^{t+|F|} [x_{i,j} - y_{i,j}] \times [n - (j + \delta)] \tag{1}$$

$$\forall\, i,j \mid\ 1 \le t \le i,j \le t + |F| < n - \delta - |F|$$

$$C_F^R = \sum_{i=t}^{n} \sum_{j=t}^{n-\delta} [x_{i,j} - y_{i,j}] \times [n - (j + \delta)] \tag{2}$$

$$\forall\, i,j \mid\ \ n - \delta - |F| \le t \le i,j \le n - \delta$$

The resource de-allocation can exceed the resource allocation in one time-frame. This might happen when there are some allocated resources in previous time frames and some participants are leaving in the current time frame. Therefore, the value of resource allocation cost can be negative or positive in a time frame. The positive value indicates that more resources are allocated in a time frame while the negative value shows that more resources are released.

The QoS violation cost at each time slot is calculated based on the difference between the total required resources and total provisioned resources up to that time slot. Let c_k^Q represent the QoS violation cost at time slot k. Equation (3) depicts c_k^Q for a time frame that

starts at time slot t. Each time frame may start while there are some provisioned resources in the previous time frames. Let P_{t-1} in (3) denote the gap between real conference size and provisioned resources from the previous time frames. This gap shows the number of participants who are waiting to join the conference. Note that resource scaling will take effect δ time slots after sending the request. Thus, in the first δ time slots of each time frame, there is no change in the amount of existing resources and QoS violation cost only depends on the previous allocated resources and the expected demand.

$$c_k^Q = \begin{cases} P_{t-1} + \sum_{i=\mathrm{t}}^{k} [a_i' - d_i'] - \sum_{i=t}^{t+|F|} \sum_{j=t}^{k-\delta} [x_{i,j} - y_{i,j}], & if\ strictly\ positive \\ 0, otherwise \end{cases} \tag{3}$$

$$\forall\, k \mid 1 \le t + \delta \le k \le t + |F|$$

The value of P_{t-1} is calculated based on equation (4). We assume all allocation and de-allocation requests in previous time frames are saved as X^P and Y^P sets, respectively. Such that, $x_{i,j}^P \in X^P$ and $y_{i,j}^P \in Y^P$ represent the amount of allocated and de-allocated resources for time slot i with the request sent at time slot j in previous time frames, respectively.

$$P_{t-1} = \begin{cases} \sum_{i=1}^{t-1} [a_i^P - d_i^P] - \sum_{i=1}^{t-1} \sum_{j=1}^{t-1} [x_{i,j}^P - y_{i,j}^P] & \forall\, t > 1 \\ 0, \quad t = 1 \end{cases} \tag{4}$$

The QoS violation cost cannot have negative values. In fact, if the resources are under-provisioned, the PaaS should pay the QoS violation. However, in the case of the resource over-provisioning, there is no violation and the cost is 0.

Our objective is depicted in equation (5). We use C_F^Q to represent the total QoS violation cost over a prediction time frame. Also, we use a coefficient β as a weighting factor between resource cost and QoS violation cost in our objective.

$$minimize \left\{ \beta \times C_F^R + (1 - \beta) \times C_F^Q \right\} \tag{5}$$

(vi) Constraints

Since QoS violation cost in (3) is non-linear, we add two constraints to linearize it. These constraints are depicted in equations (6) and (7).

$$c_k^Q \geq 0 \quad \forall\, k \mid 1 \leq t \leq k \leq t + |F| \leq n \tag{6}$$

$$c_k^Q \geq P_{t-1} + \sum_{i=t}^{k} [a'_i - d'_i] - \sum_{i=t}^{t+|F|} \sum_{j=t}^{k-\delta} [x_{i,j} - y_{i,j}] \tag{7}$$

$$\forall\, k \mid 1 \leq t + \delta \leq k \leq t + |F| \leq n$$

To respect the acceptable delay (i.e., threshold θ), the allocated resources, in terms of conference participants, between time slot i and $i + \theta$ should be greater than or equal to the expected number of participants arriving at time slot i. In other words, in the SLA between PaaS and the application providers, the conferencing PaaS guarantees that there is a maximum $[(1 - \varepsilon)\% \times \gamma\%]$ of participants waiting for more than θ time slots to be served before the conference ends. Equation (8) enforces this constraint. Note that resource allocation takes δ time slots, an aspect that needs to be taken into account in this equation. Moreover, resources can be reserved before or after participants' arrivals. Thus, the scaling request time (i.e., j) can be anytime between the start of the time frame (i.e., time slot t) and its end.

$$\sum_{j=t}^{i+\theta-\delta} x_{i,j} \geq a'_i \tag{8}$$

$$\forall\, i \mid 1 \leq t \leq i \leq t + |F| - \theta + \delta \leq n - \delta$$

If there are some participants in the conference and PaaS provides them their required service, the conference size cannot be scaled down more than the number of participants that are expected to remain in the conference. In fact, the conference size cannot shrink before participants leave the conference, as shown in equations (9), (10), and (11). Note, in $(100 - \varepsilon)\%$ of times, the prediction is not correct. Also, there is a prediction interval

of $\pm\gamma\%$. Therefore, we consider mispredictions with $+\gamma\%$ interval to ensure this constraint is satisfied.

$$\sum_{j=t}^{t+|F|} y_{i,j} \leq d'_i - \left\lceil \frac{\gamma \times (1-\varepsilon)}{1+\gamma} \times d'_i \right\rceil \tag{9}$$

$$\forall\, i \mid 1 \leq t \leq i < t+\delta \leq n-\delta$$

$$\sum_{j=i-\delta}^{t+|F|} y_{i,j} \leq d'_i - \left\lceil \frac{\gamma \times (1-\varepsilon)}{1+\gamma} \times d'_i \right\rceil \tag{10}$$

$$\forall\, i \mid 1 \leq t \leq t+\delta \leq i \leq t+|F| \leq n-\delta$$

$$\sum_{j=t}^{i-\delta-1} y_{i,j} = 0 \tag{11}$$

$$\forall\, i \mid\ 1 \leq t \leq t+\delta \leq i \leq t+|F| \leq n-\delta$$

The maximum amount of scaling down requests at each time slot cannot be more than the maximum of total allocated resources before that time slot. This is guaranteed in equation (12). The allocated and de-allocated resources in previous time frames also need to be considered in this equation.

$$\sum_{w=1}^{t-1}\sum_{z=1}^{t-1} x^P_{w,z} + \sum_{i=t}^{t+|F|}\sum_{k=t}^{j} x_{i,k} \geq \sum_{w=1}^{t-1}\sum_{z=1}^{t-1} y^P_{w,z} + \sum_{i=t}^{t+|F|}\sum_{k=t}^{j} y_{i,k} \tag{12}$$

$$\forall\, j \mid t \leq j \leq t+|F|$$

Based on A^P, D^P, A'_F, and D'_F, the set L_F can be defined such that there will be a maximum of $l_i \in L_F$ participants in time slot i, that had been in the conference for more than θ time slots. Therefore, at each time slot, the prepared conference size should at least have the required resources for the participants that have been in the conference for more than θ time slots. Equations (13) and (14) represent this constraint.

$$\sum_{w=1}^{t-1}\sum_{z=1}^{t-1}(x_{w,z}^{P}-y_{w,z}^{P})+\sum_{i=t}^{t+|F|}\sum_{k=t}^{j-\delta}(x_{i,k}-y_{i,k})\geq l_j \tag{13}$$

$$\forall\, j \mid t+\delta \leq j \leq t+|F| \leq n$$

$$\sum_{w=1}^{t-1}\sum_{z=1}^{t-1}(x_{w,z}^{P}-y_{w,z}^{P})+\sum_{i=t}^{t+|F|}\sum_{k=t}^{j}(x_{i,k}-y_{i,k})\geq l_j \tag{14}$$

$$\forall\, j \mid |F|-\delta < j \leq t+|F| \leq n$$

The conferencing IaaSs can accept the new scaling request from the PaaS when the previous request is processed completely. Therefore, two consecutive scaling requests from the conferencing PaaS must be separated by δ, as depicted in (15).

$$\sum_{j=i}^{i+\delta-1} r_j \leq 1 \quad \forall\, i \mid 1 \leq t \leq i \leq t+|F|-\delta+1 \leq n-\delta \tag{15}$$

Moreover, any changes in the conference size made at time slot j, should be mapped to their scaling request at the same time slot, as shown in equations (16) and (17). We assume M is a large enough constant in these equations.

$$M \times r_j \geq x_{i,j} \qquad \forall\, i,j \mid 1 \leq t \leq i,j \leq t+|F| \leq n \tag{16}$$

$$M \times r_j \geq y_{i,j} \qquad \forall\, i,j \mid 1 \leq t \leq i,j \leq t+|F| \leq n \tag{17}$$

To avoid unnecessary resource allocation or de-allocation, there should be no scaling request over the last δ time slots of the conference. In fact, such a request, if made, will take effect after the end of the conference. We ensure not to send such requests through equation (18). Note that this equation only affects the last time frame of the conference.

$$r_j = 0 \qquad \forall\, n-\delta < j \leq n \tag{18}$$

To solve the problem, we operate over time frames dynamically throughout the conference period. The first time frame starts at time slot $t = 1$. Throughout the conference,

for each time slot i, if solving the problem implies a scaling request, the next time frame starts at $i + \delta$. Otherwise, the next time frame starts from time slot $i + 1$.

7.3. AOS Heuristic

Based on the proposed mathematical model, reaching the optimal solution for the large-scale scenarios is very time-consuming. Since the end to end delay is one of the main factors in the conferencing applications, reaching the scaling decision in terms of minutes and seconds are not acceptable [84]. Therefore, we propose an AOS heuristic to solve the problem in a reasonable time. The AOS heuristic operates over each individual time frame. It takes as main inputs: (i) the actual number of participants from previous time frames, (ii) the output of the heuristic over the last time frame, and (iii) the predicted number of participants for the current time frame. It finds a scaling schedule, together with the amount of resources for each scaling request while respecting SLAs. Algorithm 7.1 delineates the AOS heuristic. We consider some of the constants shown in Table 7.1 as the input of this algorithm. AOS algorithm has two outputs. The first one is an integer array S_F with $|F|$ elements. Each $s_i \in S_F$ represents the required scaling amount at time slot i. The second output is an integer value U_F, which represents the total amount of existing resources. Algorithm 7.1 iterates over each time frame throughout the conference while considering the total existing resources. Therefore, the second output of this algorithm (i.e., U_F) is used as an input for running the AOS heuristic over the next time frame.

AOS heuristic has three main phases, as depicted in Fig. 7.1. In the first phase (Fig. 7.1(a)), AOS heuristic targets to tune the misallocations caused by mispredictions in the previous time frame. Thus, it calculates the actual amount of resources needed before starting the current time frame. In the second phase (Fig. 7.1(b)), it identifies the minimum possible conference size and the best time for scaling the conference in the current time frame. This phase takes both resource and QoS violation costs into account to make a decision. In phase three (Fig. 7.1(c)), it ensures that all scaling requests are separated by at least δ time slots. In addition, this phase controls running the heuristic in the current time frame if enough time remains for having another scaling request. Otherwise, it stops the

Algorithm 7.1. AOS Heuristic

Input:

$|F|, \delta, \theta, A'_F, D'_F, A^P, D^P, L_F, \beta$; // same as the inputs of Table 7.1

$U_{(F-1)}$; // the U_F output of previous time frame. If it is the first run of the AOS heuristic, the $U_{(F-1)}$ = 0

Output:

S;//a schedule set of scaling decisions

U_F; // total existing resources

1. $t \leftarrow timestamp$ // first time slot of the time frame. It is 1 for the first time frame
2. $U_F \leftarrow U_{(F-1)}$

Phase 1: Find previous actual required resources

3. actual_size ← 0
4. **If** ($t > 1$) **Then**
5. **For** k = 1 → $t - 1$ **do**
6. actual_size ← actual_size + $a_k^P - d_k^P$
7. **end for**
8. **end if** //end of phase 1

Phase 2: Find the best possible time for sending the scaling request

9. **For each** $i \in F$ **do**
10. best_t ← t
11. min_size ← ∞ // minimum required changes in the conference size
12. min_cost← ∞
13. **For** k = $i + \delta \rightarrow i + \theta$ **do**
14. **For** p= $t \rightarrow$ k **do**
15. actual_size ← actual_size + $a'_p - d'_p$
16. **end for**
17. $l_{MAX} \leftarrow 0$
18. **For** z= $k \rightarrow$ k+δ **do**
19. **If** ($l_z > l_{MAX}$) **Then**
20. $l_{MAX} \leftarrow l_z$
21. **end if**
22. **end for**
23. result ← $Algorithm2(actual\_size, l_{MAX}, U_F, \beta)$
24. **If** (min_cost> result[total_cost]) **Then**
25. min_cost ← result[total_cost]// total cost at time slot k
26. min_size← result[size]// resources at time slot k
27. best_t ← k − δ
28. **end if**
29. **end for** // end of phase 2

Phase 3: Set the amount of scaling request for the best found time and move i to the next available time for sending request to the IaaSs

30. $S[best\_t]$ ← min_size
31. $U_F \leftarrow U_F$ + min_size
32. i ← best_t + δ − 1; // -1 because it is in the loop and i for next cycle will be (best_t+ δ)
33. **If** ($i + \theta + \delta > |F|$) **Then**
34. Break // it ends running the heuristic for current time frame since the predicted information is not enough
35. **end if**
36. **end for each**

Return S_F, U_F, QoS_{cost}

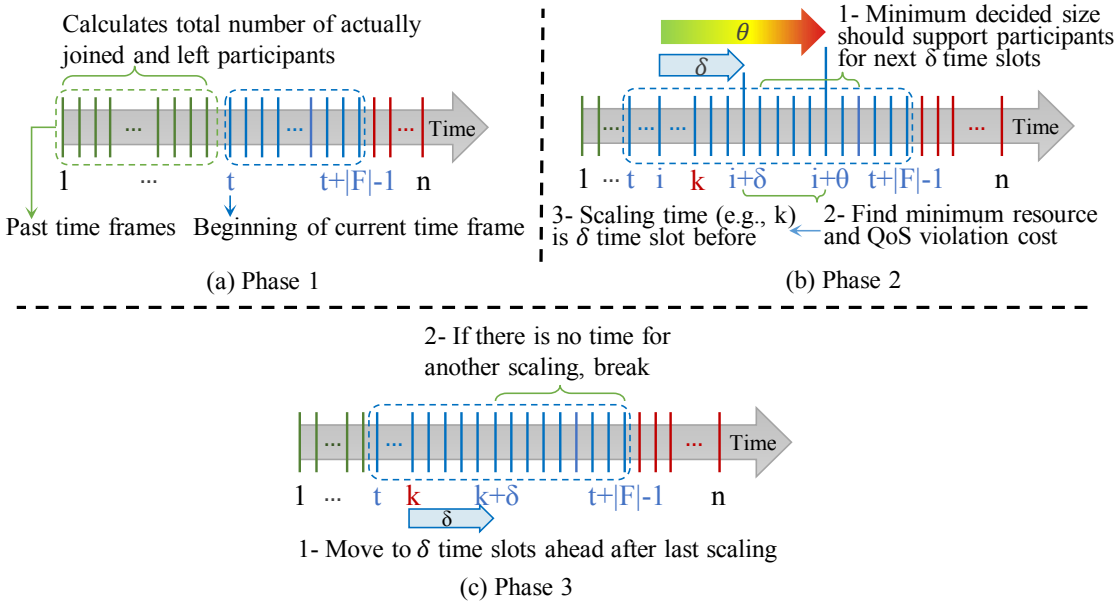


Fig. 7.1. AOS Heuristic Phase

AOS heuristic in the current time frame and consequently, the next time frame will start sooner.

The AOS heuristic runs over the whole conference duration and tries to minimize the total resource and QoS violation cost. The resource cost depends on the amount of provisioned resources and their usage over time. Thus, to reduce the resource cost, AOS heuristic should reserve the least of resources, as late as possible. This reservation respects δ, θ, and the number of participants who were in the conference. On the other hand, minimizing the QoS violation cost leads to allocating resources as much as demanded and as soon as possible. The AOS heuristic aims to solve this challenge and find the best possible time and amount for resource allocation to minimize the total cost.

AOS should consider the misallocations caused by mispredictions in the previous time frames. Therefore, in phase 1, it checks the actual required resources from the beginning of the conference until the start of the current time frame. This result is based on the real number of participants who had joined or left the conference in the past time frames. AOS will use this information to tune the possible previous misallocations in the new scaling requests at phase 2.

In phase 2, AOS tries to derive the scaling solution that leads to the minimum total cost of resources and QoS violations. Also, in this phase, AOS finds the best time to send the scaling request and the best amount of resources to fulfill the requirements. This phase considers the total required resources and allocated resources in the previous time slots to tune possible misallocations. Based on the inputs, conference scaling takes δ time slots. Therefore at each time slot i, AOS should consider the total conference size of δ time slots ahead. Also, new participants can wait up to θ time slots to join the conference. Although waiting time increases the QoS violation cost, AOS can consider it as well and checks the total conference size up to θ time slots ahead. In consequence, AOS finds the minimum total cost between time slot $i + \delta$ and $i + \theta$. Moreover, to ensure that the minimum required size of the conference (i.e., L_F) is respected, for any time slot z between $i + \delta$ and $i + \theta$, the minimum existing resources should at least support the participants in time slot $z + \delta$.

In phase 3, the AOS heuristic ensures that consecutive scaling requests are separated by more than δ time slots. Also, it keeps track of the best time for sending the scaling request and the scaling amount as the first output of the algorithm (i.e., S_F). The positive value in this array means the scaling up/out request while the negative one represents the scaling down/in request. In addition, this phase keeps track of the total amount of added and released resources during the current time frame in the second output of this heuristic (i.e., U_F) to be used for the next scaling requests. At the end of this phase, AOS checks the remaining time to the end of the current time frame. If the remaining time is less than $(\theta + \delta)$ time slots, it means that there are not enough predicted information to be used for having another scaling request in the current time frame. In this case, the AOS heuristic will stop in the current time frame and a new one will start.

The total cost for a single time slot is calculated in Algorithm 7.2. This algorithm calculates the minimum total cost with respect to the minimum required size of the conference and the weighting factor (i.e., β) between resource cost and QoS violation cost. The output of this algorithm is a set with two elements. The first element indicates the minimum possible cost. The second one has the required changes to the amount of resources. Algorithm 7.1 in phase 2 uses the output of this algorithm for deciding the best time and amount of resources to send the scaling request.

Algorithm 7.2. Find Minimum Cost and Required Changes

Input:

$total\_size$; // expected conference size

l_{MAX}; // maximum participants from time slot k to $k+\delta$, who can be in the conference for more than θ time slots.

U_F, β; // use U_F to keep track of all existing resources

Output:

$result$; // an array with three elements to keep total cost, QoS cost, and required conference size

1. $result[total_{cost}] \leftarrow \infty, result[size] \leftarrow 0;$
2. $required\_size \leftarrow l_{MAX}$
3. **For** $k = required\_size \rightarrow total\_size$ **do**
4. resource_cost $\leftarrow k * \beta$
5. $\text{QoS}_{\text{cost}} \leftarrow (total\_size - k) * (1 - \beta)$
6. **If** $(result[total_{cost}] > \text{resource}_{\text{cost}} + \text{QoS}_{\text{cost}})$ **Then**
7. $result[total_{cost}] \leftarrow \text{resource}_{\text{cost}} + \text{QoS}_{\text{cost}}$
8. $result[size] \leftarrow k - U_F$
9. **end if**
10. **end for**

Return $result$

7.4. Validations and Measurements

In this section, we will describe our evaluation scenarios and the simulation settings followed by comparison results.

7.4.1. Evaluation Scenarios and Simulation Settings

We consider three different conferencing applications as the evaluation scenarios. (i) Massively Multiplayer Online Game (MMOG), (ii) Online Distance Learning (ODL), and (iii) Online Political Party Discussion (OPPD). In all scenarios, the users, as the conference participants, are sharing their video and audio in the logic of the application. In MMOG, users join and leave the game from different geographical locations. Thus, there is a huge fluctuation in the number of participants. In contrast, in ODL and OPPD, since the participants are limited, there is less fluctuation in the conference size. Moreover, in MMOG, participants' waiting time is more tolerated than it is in OPPD and ODL. This means that in MMOG, minimizing the resource cost is more important than minimizing the QoS violation cost. In contrast, in OPPD, reducing the QoS violation cost and minimizing the participants' waiting time is much more important than the resource cost. In ODL, the resource cost and QoS violation cost are equally important.

For our simulations, we randomly generate the number of participants joining and leaving the conference at each time slot as the real-time number of participants. This dataset contains increased, fixed, and decreased number of participants over the conference duration. The fixed number of participants over some time slots means the total number of joining and leaving users are the same. We generate the predicted number of participants by applying the ε and γ on the real-time dataset for each time frame. In fact, we generate the results of assumed prediction model described in section 7.2.

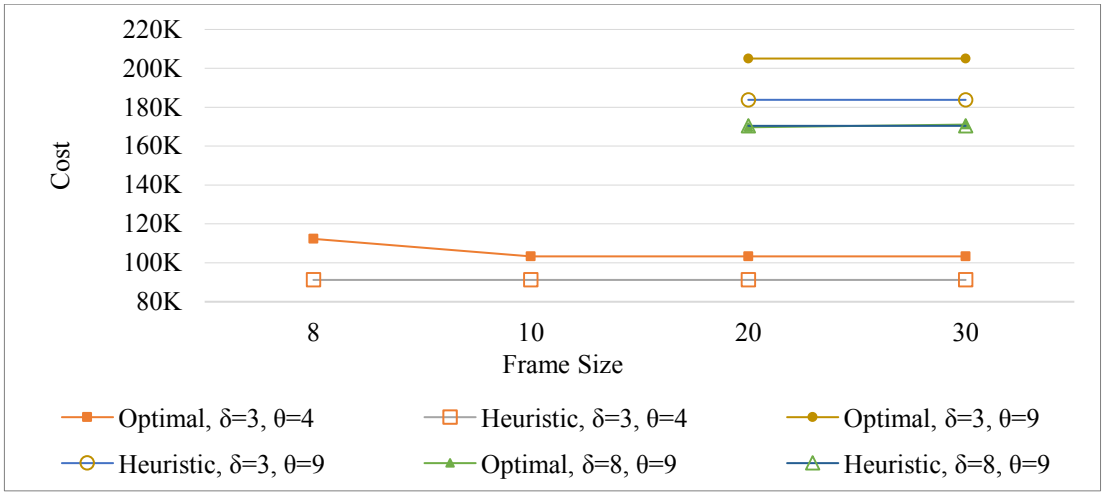
In our simulations, we consider dividing the conference duration into 100 time slots. To study the impact of time frame duration, resource provisioning time, and acceptable delay on all scenarios, we consider different settings of simulation parameters. In addition, we set the user fluctuation up to 12000 users in MMOG and 2000 users in ODL and OPPD. Moreover, we assume β for MMOG is 0.8 to stress more on reducing the resource allocation cost rather than QoS violation cost. Also, we assume β for ODL is 0.5 and for OPPD is 0.2. Simulation parameters and settings are depicted in Table 7.3.

7.4.2. Results

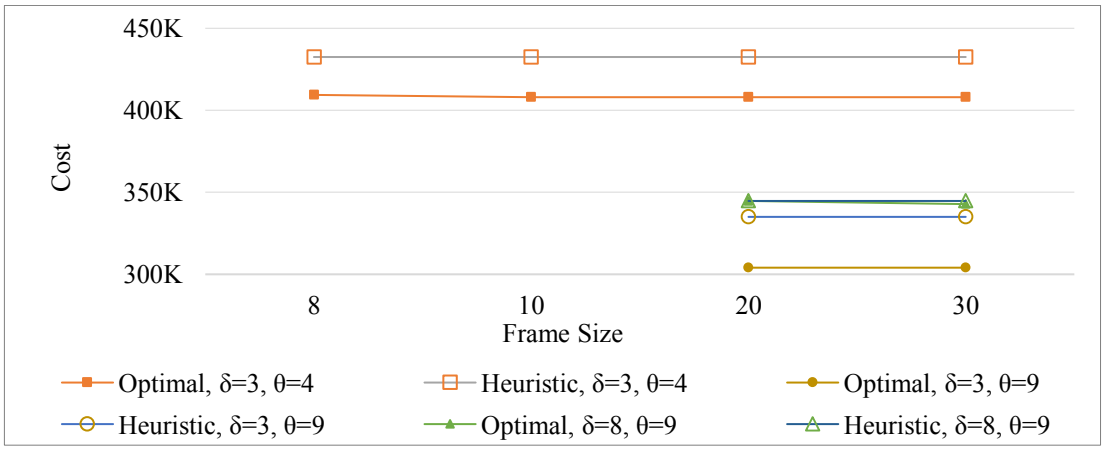
We implement the AOS algorithm in JAVA. Also, we use the LPSolve engine [83] to find the AOS optimal solution for our mathematical model. We study the impact of different settings on the results of the optimal solution and our heuristic. Also, we compare the results of our algorithm with those of the optimal solution. Fig. 7.2, 7.3, and 7.4 depict the corresponding costs for MMOG, ODL, and OPPD scenarios, respectively. These costs are normalized and cumulative. In MMOG, since resource cost is much more important, AOS

Table 7.3. Simulation Parameters and Settings

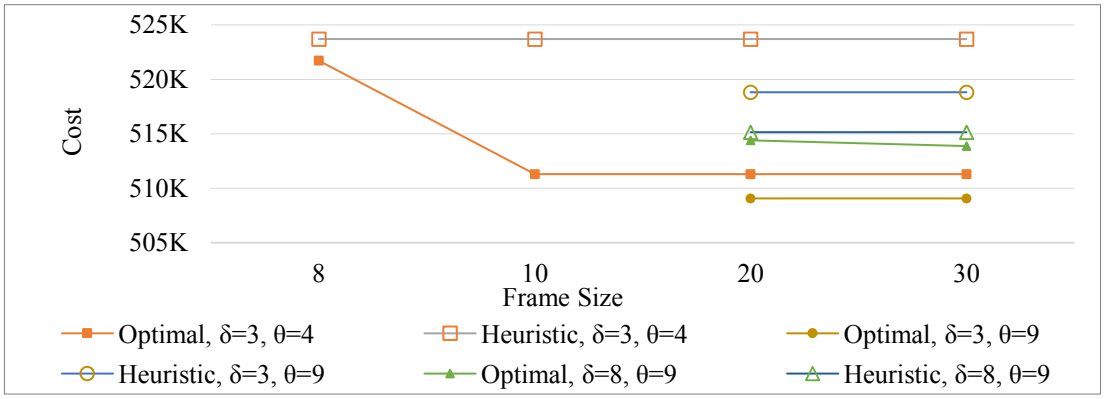
<table>
<tr><td></td><td colspan="2">MMOG</td><td colspan="2">ODL</td><td>OPPD</td></tr>
<tr><td>Fluctuation</td><td colspan="2">0-12000</td><td colspan="2">0-2000</td><td>0-2000</td></tr>
<tr><td>β</td><td colspan="2">0.8</td><td colspan="2">0.5</td><td>0.2</td></tr>
<tr><td rowspan="3">Variable Parameters</td><td colspan="2">$\delta = 3, \theta = 4$</td><td colspan="3">$F = \{8, 10, 20, 30\}$</td></tr>
<tr><td colspan="2">$\delta = 3, \theta = 9$</td><td colspan="3">$F = \{20, 30\}$</td></tr>
<tr><td colspan="2">$\delta = 8, \theta = 9$</td><td colspan="3">$F = \{20, 30\}$</td></tr>
<tr><td rowspan="2">Fixed Parameters</td><td>n</td><td>ε</td><td>γ</td><td colspan="2">M</td></tr>
<tr><td>100</td><td>80%</td><td>$\pm$10%</td><td colspan="2">1000000</td></tr>
</table>



(a) QoS Violation Cost



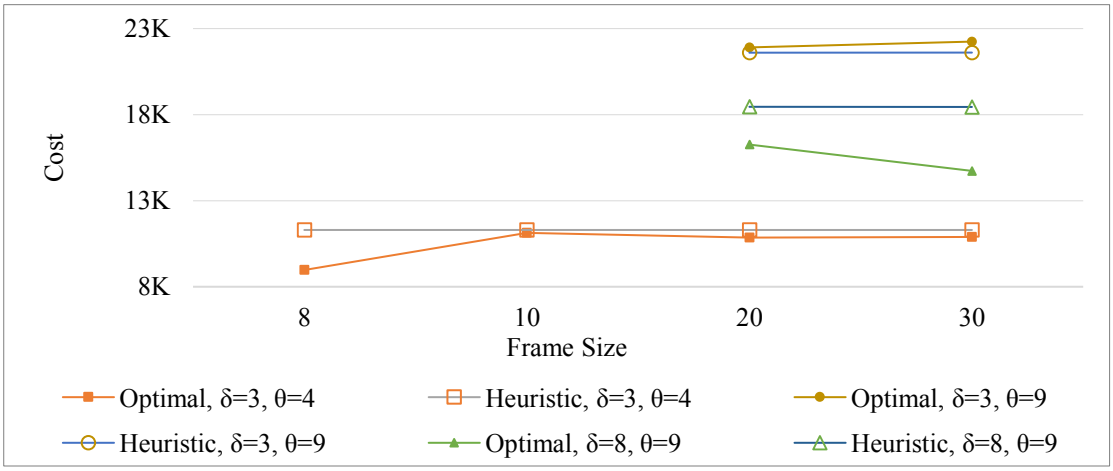
(b) Resource Cost – Noteworthy Cost



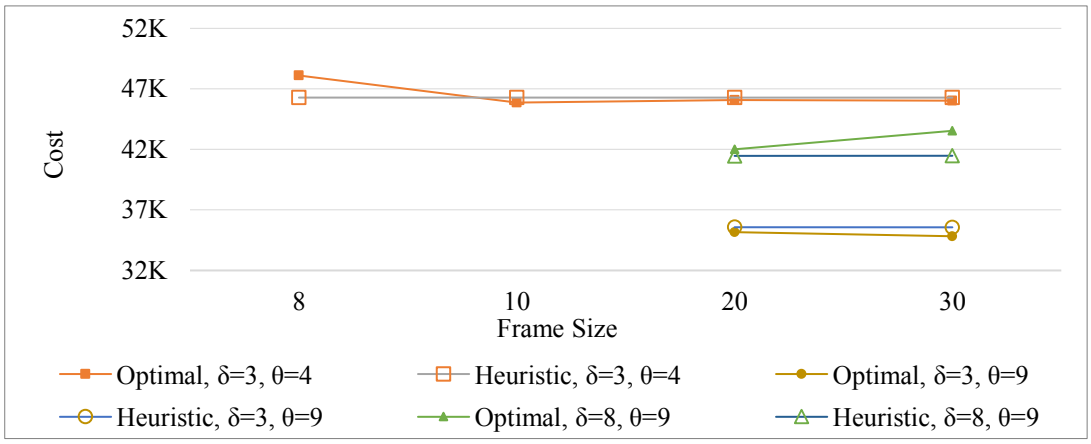
(c) Total Cost

Fig. 7.2. MMOG Cumulative Normalized Costs – Value of β = 0.8

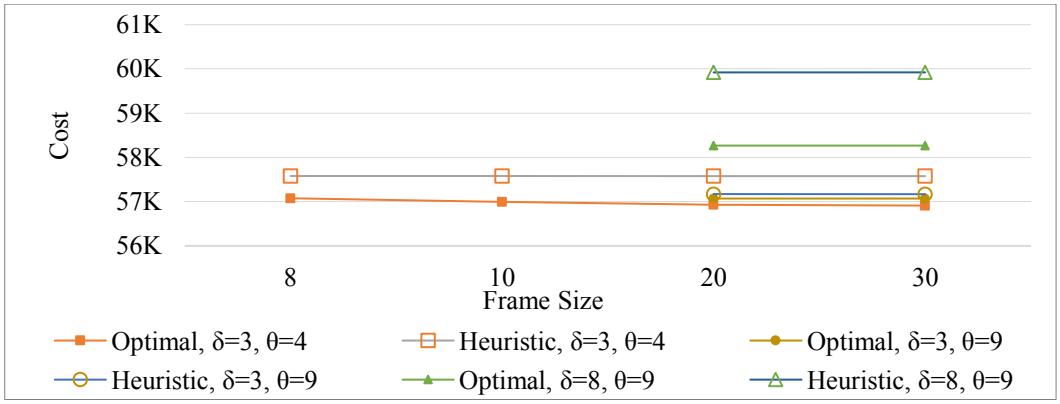
aims to reduce the resource cost, while considering the QoS requirements and the total cost. In ODL, the resource cost and QoS violation cost are equally important. So, AOS aims to



(a) QoS Violation Cost



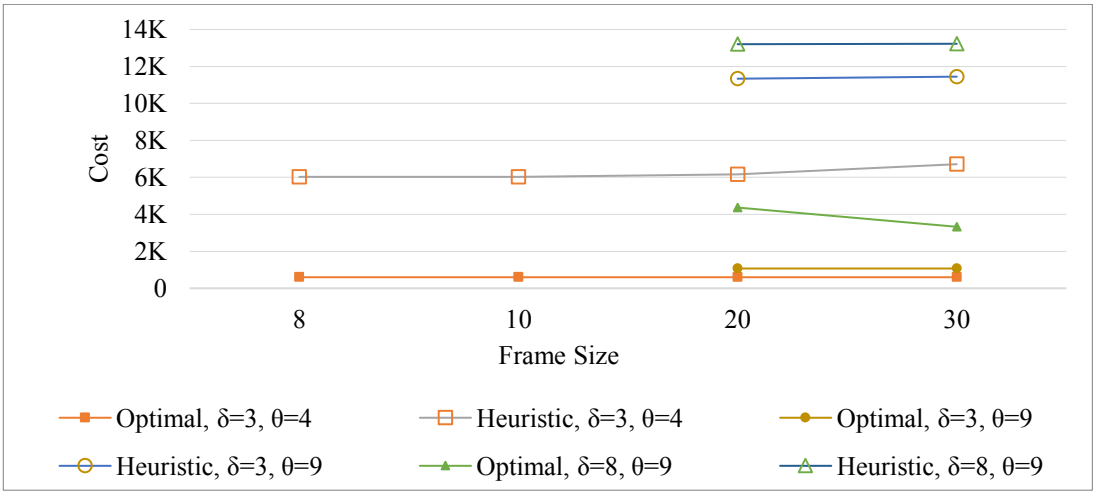
(b) Resource Cost



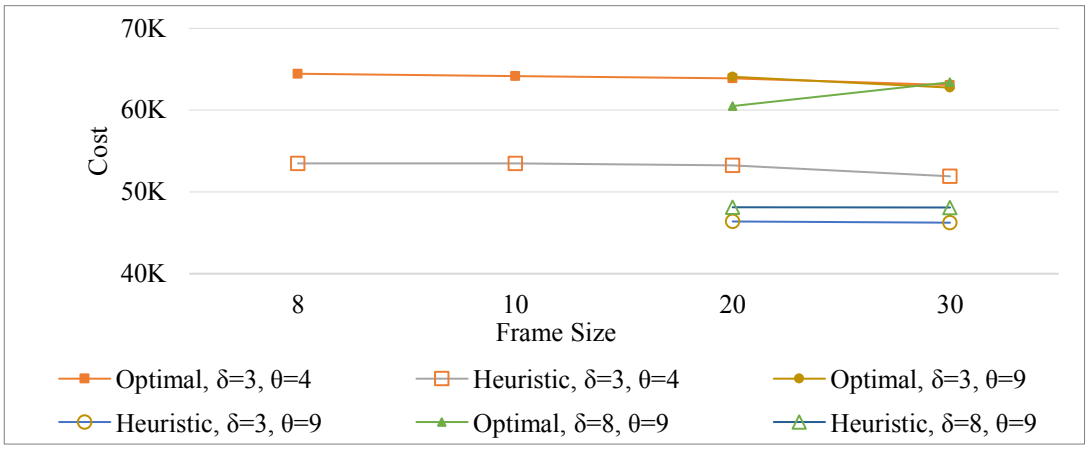
(c) Total Cost – Noteworthy Cost

Fig. 7.3. ODL Cumulative Normalized Costs – Value of β = 0.5

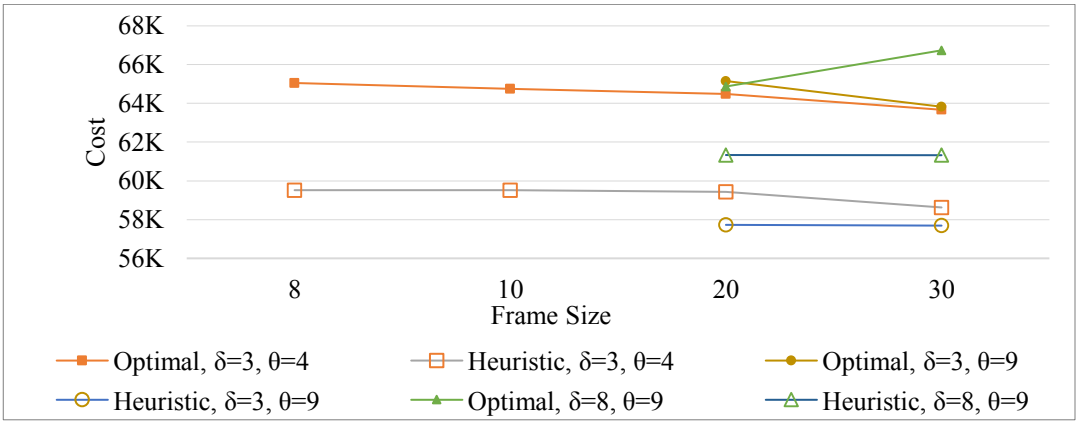
minimize the total cost by equally reducing the resource and QoS violation costs. In OPPD, the noteworthy cost is QoS violation and AOS tries to minimize it while considering another



(a) QoS Violation Cost – Noteworthy Cost



(b) Resource Cost



(c) Total Cost

Fig. 7.4. OPPD Cumulative Normalized Costs – Value of β = 0.2

objective, which is minimizing the total cost.

(i) Impact of the Prediction Time Frame

The results in Fig. 7.2, 7.3, and 7.4 show that by increasing the time frame, the AOS optimal solutions have more information about the future and it helps them to make better scaling decisions. However, as it is observed in Fig. 7.2(b), 7.3(c), and 7.4(a), increasing the future knowledge after some threshold is not helpful and the decisions are the same. This means that the scaling decisions towards minimizing the noteworthy cost for each scenario in these time frames are the same. In contrast, the results of AOS heuristic show that the bigger prediction time frame size has small negative impacts towards minimizing the noteworthy cost. This result is explicitly visible in Fig. 7.4(a) when $\delta = 3$ and $\theta = 4$. The main reason is at each time slot i, the AOS heuristic focuses on the information between $i + \delta$ and $i + \theta + \delta$. Since each time frame should have this information, there should be no impact by the time frame size. However, the AOS heuristic reruns as long as the time frame is not over. Therefore, lower time frame size leads to starting the heuristic sooner and updating the prediction information for previous time frames. This allows to tune the allocations better and leads to lower the noteworthy cost.

(ii) Impact of the Acceptable Waiting Time

The results of MMOG and ODL in Fig. 7.2(b) and 7.3(b) show that with the same value of δ, the higher value of θ can lead to a lower resource cost. Consequently, as it is shown in Fig. 7.2(c) and 7.3(c), their total costs are lower in this setting for both optimal and heuristic solutions. In fact, in this setting, AOS can allocate resources later. However, in OPPD, since waiting time is not much tolerated, the higher value of θ has no impact on the resource cost of optimal solutions and their results in Fig. 7.4(b) are almost the same.

(iii) Impact of the Resource Provisioning Time

The MMOG and ODL results in Fig. 7.2(b) and 7.3(b) show that with the same θ, a higher value of δ leads to an increase in the resource cost and consequently an increase in the total cost, as it is shown in Fig. 7.2(c) and 7.3(c). The reason is that AOS should ensure the QoS requirements. Since provisioning the resources takes longer when δ is higher, AOS should allocate sooner and it causes an increase in the resource cost in these scenarios. However, it does not affect the resource cost of OPPD. In OPPD, as it is shown in Fig.

7.4(a), with the same θ, a higher value of δ leads to an increase in the QoS violation cost. In fact, in OPPD, AOS aims to allocate resources sooner. Thus, a higher provisioning time leads to higher waiting times for participants and an increase in the QoS violation cost.

(iv) AOS Heuristic Performance

The results in Fig. 7.2(b) show that the AOS heuristic in MMOG can perform between 90% and 99% close to the results of the optimal solutions. Also, its performance in ODL is between 97% and 99% based on the results shown in Fig. 7.3(c). However, the heuristic result in OPPD shown in Fig. 7.4(a) is far from the optimal solution in minimizing the QoS violation cost. The main reason is that AOS heuristic relies on the acceptable waiting time for the users and for each time slot i, it finds the solution that leads to the minimum cost in a period of time between $i + \delta$ and $i + \theta$. Meaning that it finds the minimum QoS violation cost in a period of time while the optimal solution finds the solution that leads to the minimum QoS violation cost per each time slot. Thus, the heuristic leads to a greater waiting time compared to the optimal solution in OPPD and in consequence, a higher QoS violation cost.

Note that each scenario has a noteworthy cost which is the main objective of AOS to minimize. Therefore, the heuristic might have lower values at the other costs compared to those of the optimal solution. The main reason is that the existing trade-off between the resource cost and the QoS violation cost. For instance, the aim of AOS in MMOG is to minimize the resource cost as its noteworthy cost. Thus, as it is depicted in Fig. 7.2(b), the resource costs of AOS optimal solutions are lower than those of the heuristic solutions, while their QoS violation costs in Fig. 7.2(a) are higher. Similarly, the results of OPPD in Fig. 7.4(a) show that the optimal solutions have lower values in QoS violation costs compared to those of the heuristic solutions, while their resource costs are higher in Fig. 7.4(b).

The processing time of the AOS heuristic to reach the scaling solution is significantly lower than that of the AOS optimal solution. The results of the average processing time for AOS heuristic and optimal solutions are summarized in Table 7.4. The results show that

Table 7.4. AOS Heuristic and Optimal Solutions' Running Time

Scenario	Parameter Settings	Frame = 8		Frame = 10		Frame = 20		Frame = 30	
		Optimal	Heuristic	Optimal	Heuristic	Optimal	Heuristic	Optimal	Heuristic
MMOG $\beta = 0.8$	$\delta = 3, \theta = 4$	19 sec	13.0 ms	45 sec	12.97 ms	3 min	13.20 ms	13 min	12.58 ms
	$\delta = 3, \theta = 9$	---				3.2 min	15.60 ms	86 min	15.61 ms
	$\delta = 8, \theta = 9$					58 sec	11.91 ms	4 min	11.46 ms
ODL $\beta = 0.5$	$\delta = 3, \theta = 4$	14 sec	10.37 ms	26 sec	10.31 ms	78 sec	10.13 ms	2.17 min	10.44 ms
	$\delta = 3, \theta = 9$	---				125 sec	11.8 ms	73 min	12.1 ms
	$\delta = 8, \theta = 9$					11 sec	5.1 ms	82 sec	5.2 ms
OPPD $\beta = 0.2$	$\delta = 3, \theta = 4$	15 sec	10.90 ms	31 sec	11.05 ms	51 sec	10.71 ms	7 min	10.91 ms
	$\delta = 3, \theta = 9$	---				71 sec	12.35 ms	34.5 min	11.88 ms
	$\delta = 8, \theta = 9$					15 sec	4.38 ms	125 sec	4.34 ms

although increasing the prediction time frame can help the AOS optimal solution to get better results, it significantly increases the processing time.

7.5. Conclusion

This chapter presents AOS as a novel scaling algorithm for cloud-based multimedia conferencing applications. The AOS produces a cost-efficient scaling schedule while considering the QoS requirements and the future demands of the conferencing services. The AOS algorithm minimizes the resource cost and QoS violation cost as multiple objectives. It performs in an online manner and it takes into account the uncertainty in the prediction model. We model AOS as an optimization problem and design a heuristic to solve it in large-scale scenarios. We solve the problem and evaluate the performance of the AOS heuristic on different multimedia conferencing applications. We also study the impact of resource provisioning time, acceptable delay, and the prediction time frame on the resource cost and QoS violation cost. The evaluation shows that the AOS heuristic derives results that are more than 90% close to the results of the optimal solutions while the main objective is reducing the resource cost as well as the total cost.

Chapter 8

8. Conclusion and Future Work

Cloud-based provisioning of multimedia conferencing applications will bring several benefits, including rapid provisioning, resource efficiency, scalability, and elasticity. However, it is quite challenging. This thesis addressed architectural and algorithmic challenges associated with cloud-based provisioning of multimedia conferencing applications. It presented three main contributions. As the architectural contribution, in chapter 3, it presented a holistic cloud-based architecture for multimedia conferencing applications. It discussed the architectural components and the interfaces which cover both the infrastructure and the platform layers of cloud. This contribution simplifies the provisioning of the conferencing applications for expert and non-expert application providers by proposing novel APIs and GUIs. It also allows the conferencing application providers to utilize the offered conferencing services (e.g., audio and video mixing) without having to deal with the complexities of conferences.

To scale the actual resources (e.g., compute, storage, and network) of conferencing applications based on demand, we proposed a scaling manager component in the conferencing IaaS layer. This component is equipped with the resource allocation algorithms that can allocate and deallocate resources to cope with demands. The VMRA and CRAM, two novel resource allocation algorithms for multimedia conferencing applications that run in this component are proposed in chapter 4 and 5. These algorithms scale the actual resources required for multimedia conferencing applications in an optimal

manner while guaranteeing the required QoS. Different multimedia conferencing applications with video mixing and compressing services are also considered in designing these algorithms.

The VMRA and CRAM can scale the actual resources in the IaaS layer. However, finding the best time for scaling the conferencing applications and deciding the amount of resources to be scaled for meeting both cost-efficiency objective and QoS requirements are still challenging. Therefore, in our proposed architecture, we presented a scaling decision maker component in the PaaS layer to get this decision. This component is equipped with the algorithms for scaling the multimedia conferencing applications. The ADS and AOS, two novel scaling algorithms for multimedia conferencing applications that run in this component are presented in chapter 6 and 7. These algorithms enable the conferencing applications to scale in an elastic manner with respect to the number of participants. The proposed algorithms also guarantee to meet the QoS requirements while considering the future demands of the conferencing applications and cost-efficiency objective. We discussed the impact of uncertainty of the prediction model on the result of scaling the multimedia conferencing applications as well.

In both algorithmic contributions, the problems are mathematically modeled as ILP problems. We solve the mathematical models to achieve optimality for the small-case scenarios using the optimization tools. In addition, to solve the problems for the large-scale scenarios in an acceptable time, the heuristics were proposed.

8.1 Future Work

This thesis presented significant contributions in the cloud-based provisioning of multimedia conferencing applications. Yet, there exist several research directions for the future. To tackle all algorithmic challenges mentioned in this thesis, we assumed having a prediction model that can forecast the future number of participants. As the future work, prediction algorithms to predict participants' arrivals and departures can be introduced.

In the CRAM heuristic, to allocate actual resources, we first found the minimum number of video mixers and then allocated their required resources. After that, based on allocated resources for video mixers, we found the minimum number of compressors and

then we allocated resources for compressors if needed. In fact, we solve the problem in a local optimum manner. As future work, fining the global optimum solution can be considered.

In the ADS and AOS algorithms, we assume all IaaS can offer the required resources and all with the same price and QoS. However, as the future work, all these assumptions can be relaxed. Therefore, selecting the best IaaS that complies with the objectives can be considered in solving those problems.

Despite possible future works in the algorithmic dimensions, there are some research directions in the conferencing architecture as well. As an example, our designed APIs and GUIs are required a minimum knowledge of conferencing to create a conference. As the future work, we can enhance the GUIs for non-expert providers to suggest the best possible workflow to create a conference.

Bibliography

[1] P. Mell and T. Grance, “The NIST definition of cloud computing,” 2011.

[2] R. H. Glitho, “Cloud-based multimedia conferencing: Business model, research agenda, state-of-the-art,” in *Commerce and Enterprise Computing (CEC), 2011 IEEE 13th Conference on*, 2011, pp. 226–230.

[3] V. Nae, R. Prodan, and T. Fahringer, “Cost-efficient hosting and load balancing of massively multiplayer online games,” in *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, 2010, pp. 9–16.

[4] M. Jacobs and P. Leydekkers, “Specification of synchronization in multimedia conferencing services using the TINA lifecycle model,” *Distrib. Syst. Eng.*, vol. 3, no. 3, p. 185, 1996.

[5] A. F. Alam, A. Soltanian, S. Yangui, M. A. Salahuddin, R. Glitho, and H. Elbiaze, “A Cloud Platform-as-a-Service for multimedia conferencing service provisioning,” in *Computers and Communication (ISCC), 2016 IEEE Symposium on*, 2016, pp. 289–294.

[6] A. Soltanian, F. Belqasmi, S. Yangui, M. A. Salahuddin, R. Glitho, and H. Elbiaze, “A Cloud-based Architecture for Multimedia Conferencing Service Provisioning,” *IEEE Access*, vol. 6, no. 1, pp. 9792–9806, 2018.

[7] A. Soltanian, M. A. Salahuddin, H. Elbiaze, and R. Glitho, “A resource allocation mechanism for video mixing as a cloud computing service in multimedia conferencing applications,” in *Network and Service Management (CNSM), 2015 11th International Conference on*, 2015, pp. 43–49.

[8] A. Soltanian, D. Naboulsi, R. Glitho, and H. Elbiaze, “Resource Allocation Mechanism for Media Handling Services in Cloud Multimedia Conferencing,” *IEEE J. Sel. Areas Commun.*, no. (submitted), 2018.

[9] A. Soltanian, D. Naboulsi, M. A. Salahuddin, R. Glitho, H. Elbiaze, and C. Wette, “ADS: Adaptive and Dynamic Scaling Mechanism for Multimedia Conferencing Services in the Cloud,” in *Consumer Communications & Networking Conference (CCNC), 2018 15th IEEE Annual*, 2018, pp. 1–6.

[10] A. Soltanian, D. Naboulsi, M. A. Salahuddin, R. Glitho, and H. Elbiaze, “AOS: Adaptive and Online Scaling for Multimedia Conferencing Services in the Cloud,” *IEEE Trans. Cloud Comput.*, no. (submitted), 2018.

[11] P. Mell and T. Grance, “The NIST definition of cloud computing. National Institute of Standards and Technology,” *Inf. Technol. Lab. Version*, vol. 15, no. 10.07, p. 2009, 2009.

[12] Q. Zhang, L. Cheng, and R. Boutaba, “Cloud computing: state-of-the-art and research challenges,” *J. Internet Serv. Appl.*, vol. 1, no. 1, pp. 7–18, 2010.

[13] National Institute of Standards and Technology, “NIST Cloud Computing Standards Roadmap,” NIST Special Publication 500 - 291, 2013.

[14] L. Coyne, T. Hajas, M. Hallback, M. Lindström, C. Vollmar, and others, *IBM Private, Public, and Hybrid Cloud Storage Solutions*. IBM Redbooks, 2016.

[15] C. Vecchiola, X. Chu, and R. Buyya, “Aneka: a software platform for .NET-based cloud computing,” *High Speed Large Scale Sci. Comput.*, vol. 18, pp. 267–295, 2009.

[16] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, “Virtual infrastructure management in private and hybrid clouds,” *IEEE Internet Comput.*, vol. 13, no. 5, pp. 14–22, 2009.

[17] D. B. Khedher, R. H. Glitho, and R. Dssouli, “Media handling aspects of multimedia conferencing in broadband wireless ad hoc networks,” *IEEE Netw.*, vol. 20, no. 2, pp. 42–49, 2006.

[18] J. Li, R. Guo, and X. Zhang, “Study on service-oriented Cloud conferencing,” in *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, 2010, vol. 6, pp. 21–25.

[19] P. Rodríguez, D. Gallego, J. Cerviño, F. Escribano, J. Quemada, and J. Salvachúa, "Vaas: Videoconference as a service," in *Collaborative Computing: Networking, Applications and Worksharing, 2009. CollaborateCom 2009. 5th International Conference on*, 2009, pp. 1–11.

[20] F. Taheri, J. George, F. Belqasmi, N. Kara, and R. Glitho, "A cloud infrastructure for scalable and elastic multimedia conferencing applications," in *Network and Service Management (CNSM), 2014 10th International Conference on*, 2014, pp. 292–295.

[21] Y. Feng, B. Li, and B. Li, "Airlift: Video conferencing as a cloud service using inter-datacenter networks," in *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, 2012, pp. 1–11.

[22] R. Cheng, W. Wu, Y. Lou, and Y. Chen, "A cloud-based transcoding framework for real-time mobile video conferencing system," in *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on*, 2014, pp. 236–245.

[23] J. Liao, C. Yuan, W. Zhu, P. Chou, and others, "Virtual mixer: Real-time audio mixing across clients and the cloud for multiparty conferencing," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, 2012, pp. 2321–2324.

[24] "Cloud Foundry Overview." [Online]. Available: http://docs.cloudfoundry.org/concepts/overview.html. [Accessed: 04-Nov-2015].

[25] X. Wen, G. Gu, Q. Li, Y. Gao, and X. Zhang, "Comparison of open-source cloud management platforms: OpenStack and OpenNebula," in *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*, 2012, pp. 2457–2461.

[26] O. Litvinski and A. Gherbi, "Openstack scheduler evaluation using design of experiment approach," in *16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2013)*, 2013, pp. 1–7.

[27] S. U. Malik, S. U. Khan, and S. K. Srinivasan, "Modeling and Analysis of State-of-the-art VM-based Cloud Management Platforms," *IEEE Trans. Cloud Comput.*, vol. 1, no. 1, pp. 1–1, 2013.

[28] N. Milanovic and M. Malek, “Current solutions for web service composition,” *IEEE Internet Comput.*, vol. 8, no. 6, p. 51, 2004.

[29] A. Urbieta, G. Barrutieta, J. Parra, and A. Uribarren, “A survey of dynamic service composition approaches for ambient systems,” in *Proceedings of the 2008 Ambi-Sys workshop on Software Organisation and MonIToring of ambient systems*, 2008, p. 1.

[30] C. Peltz, “Web Services Orchestration and Choreography,” *Computer*, vol. 36, no. 10, pp. 46–52, 2003.

[31] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, “Web services composition: A decade’s overview,” *Inf. Sci.*, vol. 280, pp. 218–238, 2014.

[32] W. Chareonsuk and W. Vatanawood, “Formal Verification of Cloud Orchestration Design with TOSCA and BPEL.”

[33] D. Martin *et al.*, “OWL-S: Semantic markup for web services,” *W3C Memb. Submiss.*, vol. 22, pp. 2007–04, 2004.

[34] A. Kim, M. Kang, C. Meadows, E. Ioup, and J. Sample, “A framework for automatic web service composition,” DTIC Document, 2009.

[35] P. Pawluk, B. Simmons, M. Smit, M. Litoiu, and S. Mankovski, “Introducing STRATOS: A Cloud Broker Service.,” *IEEE CLOUD*, vol. 12, pp. 891–898, 2012.

[36] S. Yangui, I.-J. Marshall, J.-P. Laisne, and S. Tata, “CompatibleOne: The open source cloud broker,” *J. Grid Comput.*, vol. 12, no. 1, pp. 93–109, 2014.

[37] J. George, F. Belqasmi, R. H. Glitho, and N. Kara, “A Substrate Description Framework and Semantic Repository for Publication and Discovery in Cloud Based Conferencing.,” in *CSWS*, 2013, pp. 41–44.

[38] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, “Network function virtualization: State-of-the-art and research challenges,” *IEEE Commun. Surv. Tutor.*, vol. 18, no. 1, pp. 236–262, 2016.

[39] M. Dieye *et al.*, “CPVNF: Cost-efficient Proactive VNF Placement and Chaining for Value-Added Services in Content Delivery Networks,” *IEEE Trans. Netw. Serv. Manag.*, 2018.

[40] A. P. Negralo, M. Adaixo, L. Veiga, and P. Ferreira, “On-Demand Resource Allocation Middleware for Massively Multiplayer Online Games,” in *Network*

Computing and Applications (NCA), 2014 IEEE 13th International Symposium on, 2014, pp. 71–74.

[41] G. Gao, W. Zhang, Y. Wen, Z. Wang, and W. Zhu, "Towards cost-efficient video transcoding in media cloud: Insights learned from user viewing patterns," *IEEE Trans. Multimed.*, vol. 17, no. 8, pp. 1286–1296, 2015.

[42] M. H. Hajiesmaili, L. T. Mak, Z. Wang, C. Wu, M. Chen, and A. Khonsari, "Cost-Effective Low-Delay Design for Multiparty Cloud Video Conferencing," *IEEE Trans. Multimed.*, vol. 19, no. 12, pp. 2760–2774, 2017.

[43] M. Abdallah, C. Griwodz, K.-T. Chen, G. Simon, P.-C. Wang, and C.-H. Hsu, "Delay-Sensitive Video Computing in the Cloud: A Survey," *ACM Trans. Multimed. Comput. Commun. Appl. TOMM*, vol. 14, no. 3s, p. 54, 2018.

[44] R. Xavier, H. Moens, B. Volckaert, and F. De Turck, "Design and evaluation of elastic media resource allocation algorithms using CloudSim extensions," in *Network and Service Management (CNSM), 2015 11th International Conference on*, 2015, pp. 318–326.

[45] R. Xavier *et al.*, "Cloud resource allocation algorithms for elastic media collaboration flows," in *Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference on*, 2016, pp. 440–447.

[46] G. Gao, H. Hu, Y. Wen, and C. Westphal, "Resource provisioning and profit maximization for transcoding in clouds: a two-timescale approach," *IEEE Trans. Multimed.*, vol. 19, no. 4, pp. 836–848, 2017.

[47] Q. He, J. Liu, C. Wang, and B. Li, "Coping with heterogeneous video contributors and viewers in crowdsourced live streaming: A cloud-based approach," *IEEE Trans. Multimed.*, vol. 18, no. 5, pp. 916–928, 2016.

[48] C. Dong, Y. Jia, H. Peng, X. Yang, and W. Wen, "A Novel Distribution Service Policy for Crowdsourced Live Streaming in Cloud Platform," *IEEE Trans. Netw. Serv. Manag.*, 2018.

[49] J. Anselmi, D. Ardagna, and M. Passacantando, "Generalized nash equilibria for saas/paas clouds," *Eur. J. Oper. Res.*, vol. 236, no. 1, pp. 326–339, 2014.

[50] S. García-Gómez *et al.*, "4CaaSt: Comprehensive management of Cloud services through a PaaS," in *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, 2012, pp. 494–499.

[51] R. Hu, Y. Li, and Y. Zhang, "Adaptive resource management in PaaS platform using feedback control LRU algorithm," in *Cloud and Service Computing (CSC), 2011 International Conference on*, 2011, pp. 11–18.

[52] M. Machado *et al.*, "Prototyping a high availability PaaS: Performance analysis and lessons learned," in *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on*, 2017, pp. 805–808.

[53] I. Satoh, "Self-adaptive resource allocation in cloud applications," in *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, 2013, pp. 179–186.

[54] M. Babaioff *et al.*, "ERA: A Framework for Economic Resource Allocation for the Cloud," in *Proceedings of the 26th International Conference on World Wide Web Companion*, 2017, pp. 635–642.

[55] C. Bunch, V. Arora, N. Chohan, C. Krintz, S. Hegde, and A. Srivastava, "A pluggable autoscaling service for open cloud PaaS systems," in *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, 2012, pp. 191–194.

[56] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, 2011, pp. 500–507.

[57] L. Mashayekhy, M. M. Nejad, D. Grosu, and A. V. Vasilakos, "An online mechanism for resource allocation and pricing in clouds," *IEEE Trans. Comput.*, vol. 65, no. 4, pp. 1172–1184, 2016.

[58] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: elastic resource scaling for multi-tenant cloud systems," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, 2011, p. 5.

[59] R. Han, L. Guo, M. M. Ghanem, and Y. Guo, "Lightweight resource scaling for cloud applications," in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, 2012, pp. 644–651.

[60] R. Xavier, H. Moens, B. Volckaert, and F. De Turck, “Resource Allocation Algorithms for Multicast Streaming in Elastic Cloud-based Media Collaboration Services,” in *Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on*, 2016, pp. 947–950.

[61] Z. Gong, X. Gu, and J. Wilkes, “Press: Predictive elastic resource scaling for cloud systems,” in *Network and Service Management (CNSM), 2010 International Conference on*, 2010, pp. 9–16.

[62] J. G. Herrera and J. F. Botero, “Resource allocation in NFV: A comprehensive survey,” *IEEE Trans. Netw. Serv. Manag.*, vol. 13, no. 3, pp. 518–532, 2016.

[63] X. Fei, F. Liu, H. Xu, and H. Jin, “Adaptive vnf scaling and flow routing with proactive demand prediction,”,” in *IEEE INFOCOM*, 2018.

[64] X. Wang, C. Wu, F. Le, and F. C. Lau, “Online Learning-Assisted VNF Service Chain Scaling with Network Uncertainties,” in *Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on*, 2017, pp. 205–213.

[65] S. Zhang, D. Niu, Y. Hu, and F. Liu, “Server selection and topology control for multi-party video conferences,” in *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*, 2014, p. 43.

[66] P. Yuen and G. Chan, “MixNStream: multi-source video distribution with stream mixers,” in *Proceedings of the 2010 ACM workshop on Advanced video streaming techniques for peer-to-peer networks and social networking*, 2010, pp. 77–82.

[67] X. Chen, M. Chen, B. Li, Y. Zhao, Y. Wu, and J. Li, “Celerity: a low-delay multi-party conferencing solution,” in *Proceedings of the 19th ACM international conference on Multimedia*, 2011, pp. 493–502.

[68] M. H. Willebeek-LeMair, D. D. Kandlur, and Z.-Y. Shae, “On multipoint control units for videoconferencing,” in *Local Computer Networks, 1994. Proceedings., 19th Conference on*, 1994, pp. 356–364.

[69] “Web service glossary. Technical report.” W3C, 2004.

[70] T. Yu and K.-J. Lin, “Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints,” in *Service-Oriented Computing - ICSOC 2005*, B. Benatallah, F. Casati, and P. Traverso, Eds. Springer Berlin Heidelberg, 2005, pp. 130–143.

[71] S. Fu, J. Liu, X. Chu, and Y. Hu, "Toward a Standard Interface for Cloud Providers: The Container as the Narrow Waist," *IEEE Internet Comput.*, vol. 20, no. 2, pp. 66–71, 2016.

[72] A. B. Johnston and D. C. Burnett, *WebRTC: APIs and RTCWEB protocols of the HTML5 real-time web*. Digital Codex LLC, 2012.

[73] "Activiti." [Online]. Available: http://activiti.org/. [Accessed: 07-Nov-2016].

[74] M. Geiger, S. Harrer, J. Lenhard, M. Casar, A. Vorndran, and G. Wirtz, "BPMN conformance in open source engines," in *Service-Oriented System Engineering (SOSE), 2015 IEEE Symposium on*, 2015, pp. 21–30.

[75] "Express - Node.js web application framework." [Online]. Available: https://expressjs.com/. [Accessed: 25-Mar-2017].

[76] "Advanced REST client." [Online]. Available: https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo. [Accessed: 27-Feb-2016].

[77] "OpenStack Open Source Cloud Computing Software." [Online]. Available: https://www.openstack.org/. [Accessed: 26-Feb-2016].

[78] "Asterisk.org." [Online]. Available: http://www.asterisk.org/. [Accessed: 11-Sep-2015].

[79] J. Diaz, C. Munoz-Caro, and A. Nino, "A survey of parallel programming models and tools in the multi and many-core era," *Parallel Distrib. Syst. IEEE Trans. On*, vol. 23, no. 8, pp. 1369–1386, 2012.

[80] X. Nan, Y. He, and L. Guan, "Optimal resource allocation for multimedia cloud based on queuing model," in *Multimedia Signal Processing (MMSP), 2011 IEEE 13th International Workshop on*, 2011, pp. 1–6.

[81] R. Z. Farahani and M. Hekmatfar, *Facility location: concepts, models, algorithms and case studies*. Springer, 2009.

[82] "Global Ping Statistics," *WonderNetwork*. [Online]. Available: https://wondernetwork.com/pings. [Accessed: 26-Aug-2018].

[83] M. Berkelaar, K. Eikland, P. Notebaert, and others, "lpsolve: Open source (mixed-integer) linear programming system," *Eindh. U Technol.*, 2004.

[84] O. T. Time, "ITU-T Recommendation G. 114," *ITU-T May*, 2000.