A Cloud Platform-as-a-Service for Multimedia Conferencing Service Provisioning

Ahmad Ferdous Bin Alam

A Thesis

in

The Department

of

Computer Science & Software Engineering

Presented in Partial Fulfillment of the Requirements for the

Degree of Master of Applied Science at

Concordia University

Montréal, Québec, Canada

July 2016

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: **Ahmad Ferdous Bin Alam**

Entitled: "**A Cloud Platform-as-a-Service for Multimedia Conferencing Service Provisioning**" and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science

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. S. Bergler

____ Examiner
Dr. T. Eavis

____ Examiner
Dr. E. Shihab

____ Supervisor
Dr. R. Glitho

Approved by: ____

Chair of Department or Graduate Program Director

___________2016 ____________________________
Dean of Faculty

ABSTRACT

A Cloud Platform-as-a-Service for Multimedia Conferencing Service Provisioning

Ahmad Ferdous Bin Alam

Multimedia Conferencing is the real-time exchange of media content (e.g. voice, video and text) between multiple participants. It is the basis of a wide range of conferencing applications such as massively multi-player online games and distance learning applications. For faster development as well as cost efficiency, developers of such conferencing applications can use conferencing services (e.g. dial-in audio conference) provided by third-parties. However, the third-party service providers face several challenges with respect to conferencing service provisioning (i.e. service development, deployment and management). One challenge is mastering complex low-level details of conferencing technologies, protocols and their interactions. Another challenge is resource elasticity. Number of conference participants varies during runtime. So resource utilization in an elastic manner is a critical factor to achieve cost efficiency.

Cloud Computing can help tackle these challenges. It is a paradigm for swiftly provisioning a shared pool of configurable resources (e.g. services, applications, network and storage) on demand. It has three main service models: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). Using a PaaS, service providers can provision conferencing services easily and offer them as SaaS. Nonetheless, cloud-based provisioning of conferencing services still remains a big challenge due to the shortcomings of existing PaaS.

In this thesis, a PaaS architecture for conferencing service provisioning is proposed. It is based on a business model from the state of the art. It relies on conferencing IaaSs that, instead of VMs, offer conferencing substrates (e.g. dial-in signaling, video mixer and audio mixer). The conferencing PaaS enables composition of new conferences from substrates on the fly. Moreover, it provides conferencing service providers, who are experienced in programming, with high-level interfaces to abstract the internal complexities of conferencing. In order for PaaS to scale ongoing conferences elastically, an algorithm is also presented in this thesis. The conferencing PaaS is prototyped and performance measurements are made. The proposed algorithm's performance is also evaluated.

Acknowledgments

It is good to have an end to journey toward; but it is the journey that matters, in the end. I would like to express my deepest gratitude to Dr. Roch Glitho, my thesis supervisor, for guiding me throughout the journey of this thesis and for believing in me when I was lost. I would also like to thank Dr. Sami Yangui and Dr. Mohammad Ali Salahuddin for their valuable time, help and advices on different parts of this thesis.

It was a great pleasure to work with my colleagues at Telecommunications Service Engineering (TSE) lab. I would like to thank Abbas Soltanian for his cooperation and encouragement.

I am grateful to Dr. T. Eavis and Dr. E. Shihab for serving as members of my thesis committee. I also thank Dr. S. Bergler for serving as the chair at my thesis defense.

I am grateful to Dr. Roch Glitho and Concordia University for their financial support and for giving me the opportunity to work in research.

Last but not the least, I would like to thank my loving and caring wife, Farah Sheherin, who sacrificed her job to accompany me in Canada. I appreciate her continuous support very much.

Table of Contents

1. Introduction 1
  1.1 Definitions 1
    1.1.1 Multi-party Multimedia Conferencing 1
    1.1.2 Conferencing Service Provisioning 2
    1.1.3 Cloud Computing 2
    1.1.4 Platform-as-a-Service 3
  1.2 Motivation and Problem Statement 3
  1.3 Thesis Contributions 4
  1.4 Thesis Organization 4
2. Background 6
  2.1 Multi-party Multimedia Conferencing 6
    2.1.1 A Brief Introduction to Conferencing 6
    2.1.2 Architectural Components of Conferencing 7
    2.1.3 Key Conferencing Technologies 10
      2.1.4.1 Traditional SIP-based Conferencing 10
      2.1.4.2 WebRTC-based Conferencing 12
  2.2 Cloud Computing 15
    2.2.1 Definition of Cloud Computing 15
    2.2.2 Benefits of Cloud Computing 16
    2.2.3 Service Models of Cloud Computing 17
      2.2.3.1 Infrastructure-as-a-Service (IaaS) 17
      2.2.3.2 Platform-as-a-Service (PaaS) 18
      2.2.3.3 Software-as-a-Service (SaaS) 18
  2.3 Platform-as-a-Service (PaaS) 18
    2.3.1 Definition of PaaS 18

2.3.2 Advantages Offered by PaaS 19
2.3.2.1 Rapid Application Development 20
2.3.2.2 Easy and Fast Application Deployment 20
2.3.2.3 Efficient Application Management 20
2.3.3 Example of Existing PaaS 21
2.4 Conferencing Substrates 22
2.5 Conferencing Service Provisioning 23
2.6 Chapter Summary 24
3. Scenarios, Requirements and State of the Art Evaluation 25
3.1 Scenarios 25
3.1.1 Actors 25
3.1.2 Conferencing Service Development and Deployment Scenario 27
3.1.3 Conferencing Service Execution Scenario 29
3.2 Requirements 31
3.2.1 Requirements on the Conferencing PaaS 31
3.2.2 Requirements on the Conference Scaling Algorithm 32
3.3 State of the Art Evaluation 34
3.3.1 Cloud-based Conferencing Architectures 34
3.3.1.1 Cloud-based Conferencing Architectures in Literature 34
3.3.1.2 Cloud-based Conferencing Products in the Market 38
3.3.2 PaaS Solutions 39
3.3.3 Conference Scaling Algorithms 42
3.4 Chapter Summary 46
4. Proposed Architecture 47
4.1 Overall Architecture 47
4.1.1 Architectural Principles 47
4.1.2 Architectural Components 48
4.2 Conferencing Service Development APIs 52
4.3 Illustrative Scenario 58

4.3.1 Conferencing Service Development and Deployment 59
4.3.3 Conferencing Service Execution 61
4.4 How the Proposed Architecture Meets the Requirements 63
4.5 Chapter Summary 64
5. Conference Scaling Algorithm 66
5.1 Problem Background and Motivation 66
5.2 Problem Statement 69
5.3 Problem Analysis 72
5.4 Proposed Dynamic Programming Algorithm 77
5.4.1 A Brief Overview of Dynamic Programming 77
5.4.2 High Level View 78
5.4.3 Detailed Algorithm Description 78
5.4.4 Time Complexity Analysis 81
5.5 An Alternative Greedy Algorithm 83
5.5.1 A Brief Overview of Greedy Algorithm 83
5.5.2 Motivation for the Greedy Heuristic Used 84
5.5.3 High Level View 86
5.6 Chapter Summary 87
6. Validation: Prototype and Evaluation 89
6.1 Overall Software Architecture 89
6.1.1 Conferencing PaaS GUIs and APIs 90
6.1.2 Management (Services & PaaS) 91
6.1.3 Service Hosting & Execution 91
6.1.4 Conference Orchestration and Management 92
6.1.5 Operational Procedures 92
6.2 Prototype 94
6.2.1 Implemented Scenario 94
6.2.2 Implementation Scope 95
6.2.3 Prototype Description 96

6.2.4 Software Tools 98

6.2.4.1 Cloud Foundry 98

6.2.4.2 Camunda 98

6.2.4.3 Node.js 99

6.2.4.4 OpenStack 99

6.2.4.5 SAVI testbed 100

6.2.4.5 Additional Software Tools 100

6.3 Prototype Setup and Performance Measurements 100

6.3.1 Prototype Setup 100

6.3.2 Performance Measurements 101

6.3.2.1 Comparison Scenarios 101

6.3.2.2 Performance metrics 102

6.3.2.3 Performance Results 103

6.4 Conference Scaling Algorithm and Performance Measurements 105

6.4.1 Algorithm Implementation and Test Sets 105

6.4.2 Performance Metrics 106

6.4.3 Performance Results 107

6.5 Chapter Summary 112

7. Conclusion and Future Work 113

7.1 Contribution Summary 113

7.2 Future Work 115

Appendix 116

1. Detailed Description of the Proposed Greedy Algorithm for Conference Scaling 116

2. Time Complexity Analysis of the Proposed Greedy Algorithm 123

Bibliography 127

List of Figures

Figure 2-1: Key Architectural Components of Conferencing..7
Figure 2-2: Centralized Architecture of SIP-based Conferencing [11] 11
Figure 2-3: Separate Media Handling and Signaling Components [11].......................................12
Figure 2-4: Service Models of Cloud Computing [29]...17
Figure 3-1: Conferencing Service Provisioning in the Cloud...26
Figure 3-2: Conferencing Service Development and Deployment Scenario 28
Figure 3-3: Conferencing Service Execution Scenario...30
Figure 3-4: SOA Layers of Cloud Conferencing [48] 35
Figure 3-5: REST Architecture of Nuve [49] 36
Figure 3-6: Architecture of conferencing IaaS [50]..37
Figure 4-1: Overall Architecture of Conferencing PaaS...49
Figure 4-2: Pseudo-code of dial-in audio conferencing service 60
Figure 4-3: Conferencing Service Deployment Scenario 61
Figure 4-4: Conferencing Service Execution Scenario...62
Figure 5-1: Overlapping subproblems 75
Figure 5-2: An example of predicted number of participants...84
Figure 5-3: (a) Scaling with naive approach (b) Scaling with extrema optimization.....................86
Figure 5-4: (a) Waste of resources is 115 (b) Waste of resources is 140 87
Figure 6-1: Software architecture of conferencing PaaS 90
Figure 6-2: Interactions of software components for conferencing service development and deployment...93
Figure 6-3: Interactions of software components for creating a conference 94
Figure 6-4: Prototype architecture of conferencing PaaS 97
Figure 6-5: Average conference start time...103
Figure 6-6: Average participant joining time...104
Figure 6-7: Resource Allocation Evaluation..105
Figure 6-8: Effects of scaling time lag (δ) on waste of resources in terms of number of participants..107
Figure 6-9: Difference in waste of resources between DP and greedy algorithms.......................108
Figure 6-10: Elastic conference size allocation by DP algorithm..109
Figure 6-11: Elastic conference size allocation by greedy algorithm..110
Figure 6-12: Comparison of elastic conference size allocation between DP and greedy algorithms 110
Figure 6-13: Comparison of running time between DP and greedy algorithms 111

List of Tables

Table 3-1: Summary of Evaluation of the Related Works for Conferencing PaaS 41
Table 3-2: Summary of Evaluation of Related Works for Conferencing Scaling Algorithm 46
Table 4-1: Conferencing Service Development APIs..56
Table 4-2: Categorization of API Parameters..57
Table 5-1: An example of predicted number of participants in a conference.................................69
Table 5-2: List of notations...71

Acronyms and abbreviations

API	Application Programming Interface
BFCP	Binary Floor Control Protocol
CF	Cloud Foundry
CLI	Command Line Interface
DP	Dynamic Programming
DTLS	Datagram Transport Layer Security
GUI	Graphical User Interface
IaaS	Infrastructure-as-a-Service
ICE	Interactive Connectivity Establishment
IETF	Internet Engineering Task Force
MGCP	Media Gateway Control Protocol
MSCML	Media Server Control Markup Language
NIST	National Institute of Standards and Technology
PaaS	Platform-as-a-Service
QoS	Quality of Service
REST	Representational State Transfer
RTP	Real-time Transport Protocol

RTCP	RTP Control Protocol
SaaS	Software-as-a-Service
SIP	Session Initiation Protocol
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SRTP	Secure Real-time Transport Protocol
VM	Virtual Machine
W3C	World Wide Web Consortium
WebRTC	Web Real-Time Communication

Chapter 1

1. Introduction

In this chapter we first provide an overview of the key concepts related to our research such as Multi-party Multimedia Conferencing, and Cloud Computing with a focus on Platform-as-a-Service. Then the motivation and problem statement are discussed. A summary of thesis contributions is also presented. The chapter concludes with an outline of how this thesis is organized.

1.1 Definitions

We provide definitions of four key concepts that are related to our research on conferencing PaaS.

1.1.1 Multi-party Multimedia Conferencing

Multi-party multimedia conferencing is the real-time exchange of media content (e.g. voice, video, text) between multiple participants [1]. It is an important component of many *conferencing applications* such as massively multiplayer online games, audio/video conference, distance learning applications, etc. There are several models to operate conferencing applications, for instances, dial-in, dial-out and ad-hoc. The first two are scheduled conferencing models. In a dial-in conference, participants join the conference themselves, whereas the conference server invites participants in a dial-out conference according to the planned time [2]. In ad-hoc conferencing model, a participant of a point-to-point call creates a new conference and then adds new participants [3].

1.1.2 Conferencing Service Provisioning

A *conferencing service* offers full-fledged conferencing functionality, for instances, dial-in audio conferencing, dial-out video conferencing with floor control, etc. Conferencing applications consume conferencing services, which are provisioned and then offered by third-party service providers. *Conferencing service provisioning* refers to the entire lifecycle of the conferencing service, i.e. development, deployment and management [4].

1.1.3 Cloud Computing

Cloud Computing is a paradigm for swiftly provisioning a shared pool of configurable resources (network, storage, application, services) on demand. It allows provisioning resources with minimal management effort and on a pay-per-use basis [5]. Since cloud computing allows us to easily access and use virtualized resources, we can adjust provisioned resources dynamically, meaning we can scale with ease which makes optimum resource utilization feasible [6].

It has three main service models: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). IaaS provides computing, storage and networking infrastructure through virtualized hardware resources. PaaS adds level of abstraction to the infrastructure. In addition to managing the infrastructure as needed under the hood, it provides the software environment to easily and rapidly develop, build, deploy and maintain applications or services. Once the services are deployed and run on PaaS, they can be offered to other applications or end-users as SaaS on a pay-per-use basis.

1.1.4 Platform-as-a-Service

PaaS is defined as an enabler for the service providers to develop and deploy their services onto the cloud without worrying about underlying infrastructure [5]. It provides the service development and hosting environment out of the box. It also acts as an abstraction level on top of virtualized infrastructure, provisioning resources on demand during execution of running services [6]. In short, PaaS facilitates service provisioning which consists of the whole lifecycle of a service, i.e. service development, deployment and management (start, stop, scaling). Notable examples of PaaS are Google App Engine, Microsoft Azure, Cloud Foundry etc.

1.2 Motivation and Problem Statement

Conferencing is an indispensable component of many conferencing applications such as massively multiplayer online games, distance learning and audio/video conference. For cost efficiency and faster development, developers of conferencing applications can use conferencing services. Third-party service providers can provision such services using a PaaS and then offer them as SaaS. However, service providers face several challenges with respect to conferencing service provisioning. One of them is that development of conferencing services requires a steep learning curve - gaining adequate knowledge of complex conferencing concepts, protocols and different technologies - that makes service development costly and time-consuming. Another challenge is to scale running conferences on demand. Number of participants changes during the conference. So resource elasticity is crucial to minimize cost. Conferencing service providers will be benefitted from a conferencing PaaS that deals with these challenges.

In order to realize such a conferencing PaaS, we need a sound architecture that enables easy conferencing service provisioning. Also, we need algorithms that makes large-scale conference possible by utilizing resources efficiently in an elastic manner. In particular, we need an algorithm that enables PaaS to provision resources on demand on IaaS.

1.3 Thesis Contributions

The thesis contributions are as follows:

- A set of requirements on the conferencing PaaS
- Analysis of the state of the art with an evaluation summary based on our set of requirements.
- A general architecture of a conferencing PaaS
- An algorithm for the conferencing PaaS to scale running conferences on demand.
- Implementation architecture, a proof of concept prototype, and performance evaluation.

1.4 Thesis Organization

The rest of the thesis is organized as follows:

Chapter 2 discusses the key concepts related to our research domain in more details.

Chapter 3 introduces the scenarios and the set of requirements on a conferencing PaaS derived from the scenarios. The state of the art is also evaluated against the requirements.

Chapter 4 presents the proposed architecture for a conferencing PaaS. Architectural components as well as the proposed interfaces are discussed.

Chapter 5 presents an algorithm for the conferencing PaaS to scale running conferences on demand.

Chapter 6 describes the implementation architecture and technologies used for the proof-of-concept prototype. Then performance measurements evaluating the architecture as well as simulation results evaluating the algorithm are discussed.

Chapter 7 concludes the thesis by giving a summary of the overall contributions and identifies future research directions.

Chapter 2

2. Background

This chapter presents the background concepts relevant to research domain of this thesis. The following concepts are explained: multi-party multimedia conferencing, cloud computing, platform-as-a-service (PaaS), conferencing substrates and conferencing service provisioning.

2.1 Multi-party Multimedia Conferencing

In this section we provide an overview of Multi-party Multimedia Conferencing. We first briefly introduce the concept of conferencing. A short description of its key technical components follows. We also discuss different types of conferencing at the end.

2.1.1 A Brief Introduction to Conferencing

Conferencing is the conversational exchange of media content (e.g. voice, video, text) between multiple participants [1]. Some examples of applications, where conferencing is an indispensable component, are audio/video conference, distance learning, massively multiplayer online games, etc. Conferencing is resource-intensive. Moreover, conferences can vary in size (number of participants), for instance, from several hundreds to thousands of participants.

Conferencing can be operated in one of several models, for instances, dial-in, dial-out and ad-hoc. In a dial-in conference, users join the conference themselves, while the conference server calls up participants in a dial-out conference according to the predefined time [2]. In ad-hoc conferencing model, one participant of an ongoing point-to-point call creates a new conference and then adds new participants [3]. Typically dial-in and dial-out conferences are pre-arranged, whereas ad-hoc conferences are not.

2.1.2 Architectural Components of Conferencing

A typical conference is comprised of the following key architectural components (depicted in Figure 2-1): Signaling, Media Handling and Conference Control.

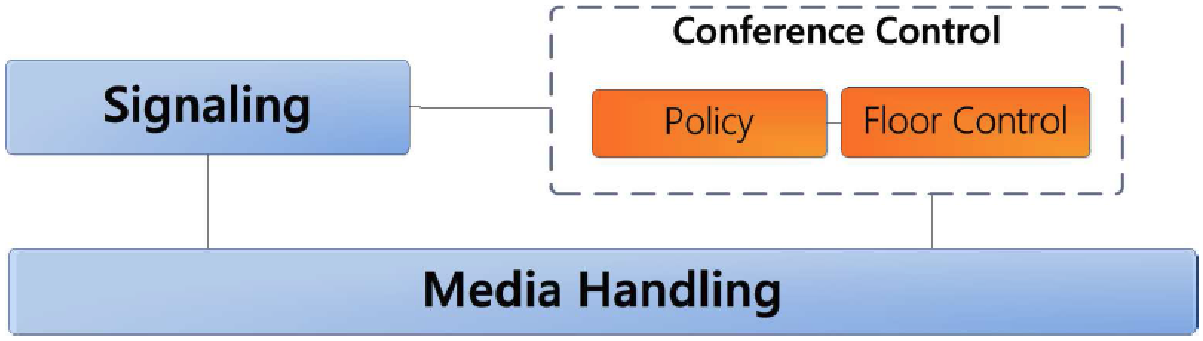


Figure 2-1: Key Architectural Components of Conferencing

- **Signaling:** In order for a conference participant to be able to communicate with other participants, there should be a mechanism through which the participants' locations or addresses can be known. Moreover, the exchange of media among multiple participants in a conference requires that the participants negotiate capabilities such as acceptable media format, bit rate, etc. In addition to that, a client device or software also needs to inform the address at which it expects media from other participants. *Signaling* entity addresses these aspects. This entity establishes session with each participant. It also takes care of capability negotiation, session modification and termination. Communication between conference end-points and signaling entity follows standardized protocols. Examples of signaling protocols are Session Initiation Protocol (SIP) [7], H.323 [8].

- **Media Handling:** This entity deals with media transmission, mixing and transcoding. A participant in a multi-party conference receives media data from the other participants. It is not efficient if media data from other participants is received individually. A more efficient approach is to receive a single media stream that is a combination of media data streams from all other participants. The process of combining multiple incoming media streams of the same type into a single output stream is called *mixing*. The mixer usually generates multiple output streams – one for each participant – having incoming streams of all participants except the target participant.

  Conference end-points (devices, softwares) differ in their media capabilities in terms of media format, resolution, bit rate, frame rate, etc. Conference participants may use different kinds of end-points such as mobile devices, desktop software, dedicated conferencing devices whose media capabilities vary greatly. For example, mobile end-points need a lower video resolution than desktop clients. So media handling entity should take this into consideration. It should transform the incoming media stream into a stream appropriate for the target client. The process of converting media content between different media formats is called *transcoding*. Before transmitting media to participants, transcoder converts the media stream to a format compatible with the target conferencing client device or software.

  The two most widely used protocols for media transmission are Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP). RTP is used to deliver the media data and RTCP is used to get feedback from clients and to monitor quality of media transmission [9]. Secure Real-time Transport Protocol (SRTP) is used when confidentiality, message authentication, and replay protection to RTP and RTCP traffic are needed [10].

- **Conference Control:** Conference control provides advanced capabilities such as floor control and policy control.

  - **Policy Control:** *Conference Policy* is the complete set of rules governing a particular conference. That means each conference is always associated with a conference policy. The rules in conference policy can be as simple as a list of allowed participants in a conference. They can also be complex, for instances, time-of-day-based rules on participation, and conditional rules on the presence of other participants. There is no restriction on the type of rules that can be included in a conference policy.

    Policy Control entity stores and manipulates the conference policy. When a new participant requests to join a conference, this entity determines if that participant is allowed or not based on policy. Similarly, a participant may be removed from the conference if conference policy is changed not to allow that participant in the conference.

    Mechanism to manipulate conference policy is not standardized. It can be through web applications or voice applications, non-SIP-specific protocols or proprietary protocols [11]. Conference Policy Control Protocol [12] remained as an IETF draft and eventually expired.

  - **Floor Control:** A conference usually has shared resources, for instances, right to talk and input access to a video channel. It is often necessary to control who can provide input to or has access to the shared resources. A floor is an *individual temporary access or manipulation permission for a specific shared resource (or* *group of resources)* [13]. Floor control is a mechanism to manage joint or exclusive access to shared resources in a conference.

Floor policy can be moderator-controlled or autonomous. In the former case, a user (not necessarily a participant in the conference), called floor chair, manages a floor. In the latter case, the decisions (grant or revoke floor) are made automatically based on predefined policy. An example of protocol for floor control is Binary Floor Control Protocol [14].

For any conferencing applications, Signaling and Media Handling are essential parts. Conference Control component can be added to provide more capabilities and control.

2.1.3 Key Conferencing Technologies

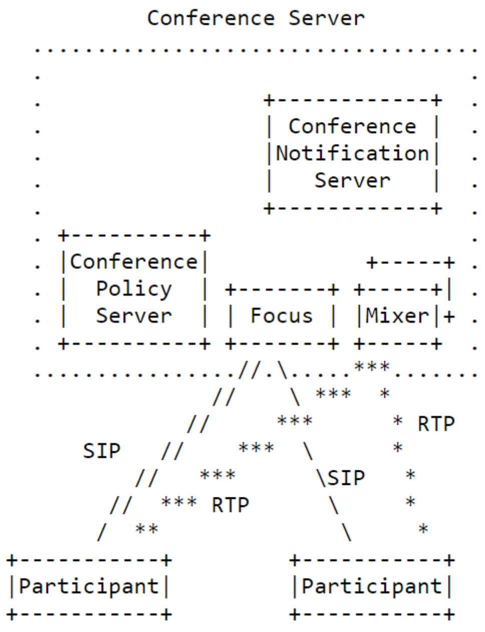
In this section we present two most common and popular conferencing technologies – one is the conventional SIP-based conferencing and the other is emerging WebRTC-based conferencing.

2.1.4.1 Traditional SIP-based Conferencing

Session Initiation Protocol (SIP)-based conferencing is the most widespread solution for conferencing. SIP, an application layer protocol developed by IETF, is used as signaling protocol for multi-media communication sessions. It has been reused in other IETF standards to provide signaling and control functionalities for a large range of multimedia communications including voice, data, images, messaging, presence, file transfers etc. [15].

SIP-based conferencing technology relies on a suite of IETF protocols used together to realize conferencing. In traditional SIP-based conferencing system, besides signaling protocol SIP, RTP/RTCP protocols are used for media transmission and BFCP for floor control. When media handling entity is separate from signaling entity, media control protocols are needed for communication between these two entities. Examples of media control protocols developed by IETF are Media Gateway Control Protocol (MGCP) [16], Media Control Channel Framework [17], Media Server Control Markup Language (MSCML) Protocol [18].

IETF specification on SIP-based conferencing [11] describes several possible architectures. One of them is centralized architecture as depicted in figure 2-2. In the figure, focus is the signaling entity as we have discussed in section 2.1.2. Mixer belongs to Media Handling entity.



```
          Conference Server
  ....................................
  .                                  .
  .                  +------------+  .
  .                  | Conference |  .
  .                  |Notification|  .
  .                  |   Server   |  .
  .                  +------------+  .
  . +----------+                     .
  . |Conference|            +-----+  .
  . |  Policy  | +-------+ +-----+|  .
  . |  Server  | | Focus | |Mixer|+  .
  . +----------+ +-------+ +-----+   .
  ................//.\.....***.......
                 //   \ ***   *
               //     ***      * RTP
      SIP    //     ***  \     *
            //   ***      \SIP   *
          //  *** RTP      \     *
         /  **              \     *
+-----------+         +-----------+
|Participant|         |Participant|
+-----------+         +-----------+
```

Figure 2-2: Centralized Architecture of SIP-based Conferencing [11]

Another architecture separates media handling entity from signaling entity as shown in the following figure 2-3. In this figure, signaling entity belongs to the Application Server on the left and media handling entity is the “Conf. Cmpnt.” component on the right.

```
 +------------+            +------------+
 | App  Server|     SIP    |Conf. Cmpnt.|
 |            |------------|            |
 |   Focus    |   non-SIP  |   Focus    |
 |   C.Pol    |------------|   C.Pol    |
 |            |            |   Mixers   |
 |Notification|            |            |
 |            |            |            |
 +------------+            +------------+
     |      \                   .. .
     |       \\            RTP...   .
     |         \\           ..      .
     |     SIP   \\       ...       .
 SIP |             \\ ...           .RTP
     |              ..\             .
     |           ...   \\           .
     |        ...        \\         .
     |      ..             \\       .
     |   ...                 \\     .
     | ..                      \    .
+-----------+             +-----------+
|Participant|             |Participant|
+-----------+             +-----------+
```

Figure 2-3: Separate Media Handling and Signaling Components [11]

2.1.4.2 WebRTC-based Conferencing

There has been an increasing interest in adding real-time voice and video communication capabilities to browsers because there are numerous use-cases where a web application user may need real-time multimedia communication. For example, a collaboration web application where team members visiting the same internal project web page could auto-join a video conferencing application embedded in that web page. Another example is an enterprise website where a visitor can start voice conversations with the enterprise' customer service agent from that website. The development of such web applications having multimedia communication capabilities was difficult due to browsers' lack of support for real-time multimedia communication capabilities.

The W3C WebRTC and the IETF RTCWEB working groups are jointly working to define both the application programming interfaces (APIs) and the underlying communication protocols for the setup and management of a reliable communication path between next-generation web browsers.

The technology resulting from efforts of these two standardization bodies is known as Web Real-Time Communication (WebRTC) [15]. The IETF and W3C are working on different but complementary aspects. The IETF is working for identification and definition of network related aspects, including control protocols, connection establishment and management, and selection of the most suitable media codecs. On the other hand, the W3C focuses on the definition of JavaScript APIs, mechanisms in order for browsers to have secure access to input devices, and the network protocols chosen for communication [15].

The realization of integrating real-time multimedia communication into web browsers has proved revolutionary in the world of telecommunications and renders traditional SIP-based conferencing as 'legacy' because the latter did not envisage web browser among the set of supported end-points [15]. With the advent of WebRTC, web developers can now easily embed real-time multimedia communication in their websites using fairly simple high-level APIs, enabling the users to have voice and video conversation without installing any plug-in in the browser.

WebRTC focuses on peer-to-peer communication between browsers and does not provide any particular mechanism to realize multiparty conferencing. So in a WebRTC-based conference with no intermediate entity, each browser has to receive and handle the media streams generated by the other browsers, as well as deliver its own generated media streams to the other browsers. The application-level topology is a mesh network in this case. Although the mechanism is simple, it is not efficient in terms of network bandwidth and the use-case is not suitable for low-bandwidth mobile devices.

WebRTC-based conferencing usually relies upon a star topology where each peer connects to a dedicated server responsible for negotiating parameters with every other peer in the network, mixing the media streams, distributing the proper (mix of) streams to each and every peer participating in the conference [19]. However, this essentially introduces a centralized infrastructure in the WebRTC peer-to-peer communication model. Several approaches and models for WebRTC-based multiparty conferencing have been studied in [20], [21], [22].

From the network communication point of view, WebRTC defines the protocol stack only for media plane. To avoid redundancy and to maximize compatibility with established technologies, it does not specify any particular signaling protocol and leaves it to the application developer [23]. It also does not address the conference control aspects (policy control, floor control). Regarding media plane protocols, WebRTC requires Secure Real-time Transport Protocol (SRTP) for encrypting and delivering the media streams and Datagram Transport Layer Security (DTLS) protocol for secure exchange of encryption keys (key management) [24]. Audio codecs G.711 and Opus, video codecs VP8 and H.264 are mandatory for the WebRTC end-points to implement. In order to establish peer-to-peer media path, WebRTC-enabled end-points must be aware of Interactive Connectivity Establishment (ICE) protocol. WebRTC also requires both RTP and RTCP streams be multiplexed on the same port.

The openness of WebRTC on signaling plane as well as security layer and multiplexing requirement added to media plane pose challenges for interworking between conventional SIP-based and the emerging WebRTC-based conferencing systems. By introducing signaling gateway and media gateway, the interworking between SIP-based and Web-RTC based end-points can be made possible [15] [24].

2.2 Cloud Computing

In this section we present a general overview of Cloud Computing. We start with its definition followed by the key benefits that Cloud Computing offers. We also discuss its different service models.

2.2.1 Definition of Cloud Computing

Cloud computing has been defined in several ways. NIST (US National Institute of Standards and Technology) defines [5] it as "*model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction*". Vaquero et al. [6] has provided an integrative definition of cloud computing based on the 20 previous definitions available at that time. The authors define cloud computing as a "*large pool of easily usable and accessible virtualized resources that can be dynamically reconfigured to adjust to a variable load (scale), allowing for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the infrastructure provider by means of customized SLAs*". This definition covers three main characteristics - resource pooling, rapid elasticity and measured services. However, it does not mention two other key characteristics of cloud computing - on-demand self-service (computing resources can always be used without human interaction with infrastructure service provider) and broad network access (access to computing resources over network). NIST definition covers all essential characteristics of Cloud Computing.

2.2.2 Benefits of Cloud Computing

Cloud Computing offers several important benefits. They are:

- **Scalability:** Virtually unlimited scalability is possible because of the massive capacity offered by the cloud providers [25]. Services hosted on the cloud can be easily scaled which is very useful in the event of rapid service demand change.
- **Elasticity:** It refers to a system's capability of adapting to variable workload by provisioning and de-provisioning resources in an autonomic manner [26].
- **Reliability:** Services running on the cloud should meet several desired requirements such as Quality of Service (QoS), availability, performance, fault tolerance, etc. These requirements are regulated under the framework of Service Level Agreement (SLA) between cloud service providers and customers. SLAs contain the details of the service as well as penalty for violations [25].
- **Multi-tenancy:** Cloud providers can serve multiple customers by assigning and reassigning the virtualized and physical resources dynamically according to demand. It facilitates resource sharing resulting in optimum resource utilization and cost.
- **On-demand self-service:** Customers can provision cloud resources any time without human interaction with the cloud service providers [27].
- **Pay-per-use Model:** Customers are charged only for the amount of resources they consumed. This measurement parameter can vary based on the services offered. For

instance, usage of a virtual machine (of a particular configuration) per hour, number of users consuming a service, etc. [28].

- **Easy access:** Customers can easily access provisioned resources over network through various types of devices.

2.2.3 Service Models of Cloud Computing

Cloud Computing has three main service models [6] – Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). Figure 2-4 illustrates the service models:

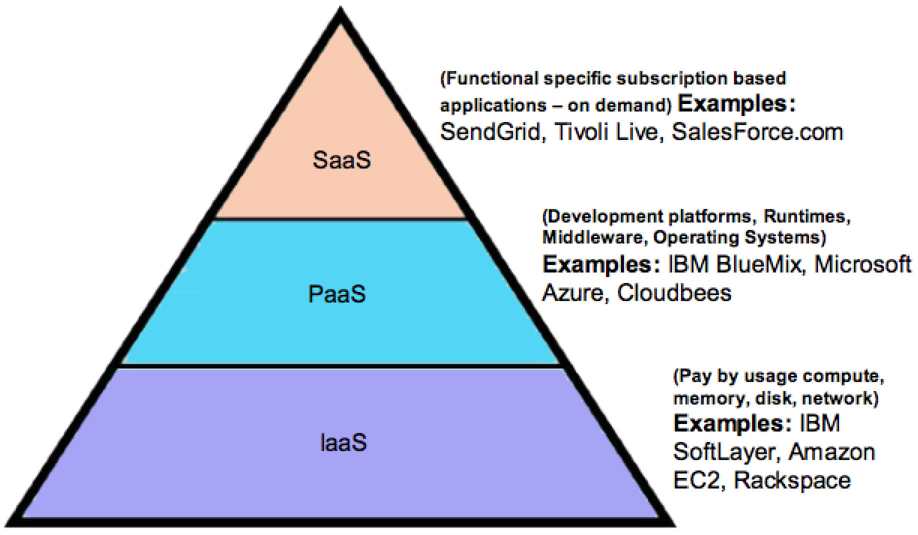

SaaS
(Functional specific subscription based applications – on demand) Examples: SendGrid, Tivoli Live, SalesForce.com
PaaS
(Development platforms, Runtimes, Middleware, Operating Systems) Examples: IBM BlueMix, Microsoft Azure, Cloudbees
IaaS
(Pay by usage compute, memory, disk, network) Examples: IBM SoftLayer, Amazon EC2, Rackspace

Figure 2-4: Service Models of Cloud Computing [29]

2.2.3.1 Infrastructure-as-a-Service (IaaS)

Among the three service models, IaaS provides resources with the lowest level of abstraction. Examples of typical IaaS services are computing, storage and network [25]. Users access the underlying infrastructures through the provisioned virtual machines (VMs). Heterogeneous resources can co-exist on the same hardware system by using virtualization technology. IaaS provides resource and cost efficiency through shared resources and multi-tenancy. Examples of well-known IaaS providers are Amazon EC2, Google Compute Engine, Rackspace etc.

2.2.3.2 Platform-as-a-Service (PaaS)

Platform-as-a-Service is discussed in detail in the next section 2.3.

2.2.3.3 Software-as-a-Service (SaaS)

This model provides services with highest level of abstraction. SaaS services can be consumed in two ways - by end-users directly and by third-party applications through APIs. From end-users' perspective, SaaS is a convenient alternative to applications that need to be run locally on a PC because they can easily access the service through web browser. From third-party application providers' point of view, they can start using SaaS services immediately without having to spend for capital expenditure. SaaS users have no control over the underlying infrastructure, application or services. Examples of SaaS services include Salesforce.com, Google docs etc.

2.3 Platform-as-a-Service (PaaS)

In this section we first present the available definitions of Platform-as-a-Service. Then we describe its various advantages.

2.3.1 Definition of PaaS

NIST defines Platform-as-a-Service as a service model providing capability to the consumer "*to deploy onto the cloud infrastructure consumer-created or acquired applications created using* *programming languages, libraries, services, and tools supported by the provider*" [5]. Boniface et al. [30] defines PaaS as "*the provision of a development platform and environment providing services and storage, hosted in the cloud*". The consumer cannot manage or control the underlying cloud infrastructure such as network, servers, operating systems, or storage. However, the applications deployed and possibly their hosting environment configurations can be controlled by the consumer [5].

PaaS provides platform resources on top of infrastructure in order to provision applications. Application provisioning encompasses the whole life cycle of applications i.e. application development, deployment and management. For development phase, PaaS provides developers with different programming platforms, languages, frameworks and even for different application domains (e.g. big data analytics in IBM Bluemix PaaS). For deployment phase, PaaS instantiates runtime environment for hosting the application. Application execution is part of application management. For application management phase, PaaS provides automated operations to start, stop and scale applications, to monitor applications' status as well as associated QoS parameters. Examples of open-source PaaS are Cloud Foundry, Apache Stratos, Red Hat's OpenShift Online. Notable proprietary PaaS examples include Google App Engine, Amazon's AWS Elastic Beanstalk, Heroku, Salesforce.com's App Cloud, Oracle Cloud Platform, Red Hat's OpenShift Enterprise, Pivotal CF, IBM Bluemix.

2.3.2 Advantages Offered by PaaS

PaaS facilitates the entire application life cycle by offering the underlying services for application development, deployment and management [31].

2.3.2.1 Rapid Application Development

PaaS can facilitate application development for both professional and non-professional developers [31], meaning it can help developers develop applications easily and rapidly, regardless of their programming experience. To this end, PaaS can provide user interfaces as well as application programming interfaces (API). Two examples of user interfaces are IDE plug-in such as Eclipse plug-in and graphical user interfaces where application features can be selected by the developer. The former is for developers with programming experience whereas the latter for developers without programming experience. The PaaS APIs enable application developers to take advantage of different services offered by PaaS such as database, analytics etc.

2.3.2.2 Easy and Fast Application Deployment

Developers can deploy applications on PaaS without having to worry about the complexity of purchasing and managing the underlying hardware and software layers [30]. Moreover, it eliminates the burden of maintaining three different environments (a development environment, a test environment and a production environment) as in the on-premises software development model [31]. PaaS offers the same hosting environment for all stages and thus reduces the deployment time.

2.3.2.3 Efficient Application Management

PaaS provides easy user interfaces for application start, stop and scaling. In additional to that, PaaS can also provide auto-scalability, reliability and security, built-in integration with web services and databases, and support for deep instrumentation of application (such as resource usage) and of user activity [31]. These features make application management a lot easier and faster. However, not all features are provided by all PaaS providers.

2.3.3 Example of Existing PaaS

In this section, we look at a notable example of existing PaaS – Cloud Foundry. It is an open-source Platform-as-a-Service. Since its inception, maintaining openness and extensibility has been a key design goal. It provides mechanisms to extend support for new programming languages and frameworks, application services, underlying infrastructures [32].

By default, Cloud Foundry supports various programming languages (e.g. Java, JavaScript, Ruby, Go, PHP, Python) as well as popular platforms and frameworks of those languages (e.g. Spring, Node.js) [33]. It provides a mechanism called buildpack through which support for new programming languages, frameworks can be added to the platform. Buildpacks for the supported languages are provided out of the box. Custom buildpacks can be developed to add new programming language support [34]. With regard to application development, Cloud Foundry is minimalistic. However, IBM Bluemix – a PaaS based on Cloud Foundry – extends application development support by providing boilerplate projects and codes for different types of applications (e.g. web, mobile backend, IoT) [35], which developers can use to get started quickly.

With regard to application deployment, Cloud Foundry not only builds and prepares environment for hosting applications, but also provides some common application services such as databases, messaging service, application metrics, and application logging. In order to make provisioning of these application services easy, Cloud Foundry supports a marketplace from where users can choose and provision services for their applications [36]. This feature is also extensible. User provided services can be integrated into the platform through service broker API [37].

With regard to application management, Cloud Foundry provides Command Line Interface called CF CLI. Developers use it to build, test, deploy, scale and manage the applications. This tool is used to maintain the entire life-cycle of applications. Plug-ins can be developed and added to default CF CLI to support additional custom commands [38]. Cloud Foundry also provides a plug-in for Eclipse IDE as an alternative to CF CLI [39]. With this Eclipse plug-in, developers can provision applications (develop, deploy and manage) without leaving their development environment.

In order to be independent of underlying infrastructure, Cloud Foundry provides an interface called Cloud Provider Interface (CPI). It is a set of APIs that deployment tools of Cloud Foundry need to implement for the targeted infrastructure [40]. By default, Cloud Foundry can be deployed on OpenStack, AWS, vSphere/vCloud using a deployment tool named BOSH [41]. To deploy Cloud Foundry on a new IaaS, a developer only needs to extend BOSH tool for that IaaS by implementing Cloud Provider Interface.

2.4 Conferencing Substrates

This thesis follows a business model [1] from the state of the art, which proposes six roles: Connectivity provider, broker, conferencing substrate providers, conferencing infrastructure providers, conferencing platform providers and conferencing service providers. An important concept in this business model is *conferencing substrates* which are defined as fine-grained building blocks of conferencing. They can be virtualized and shared by multiple conferences for resource efficiency purposes. Substrates can be atomic or composite. Examples of atomic substrates are dial-out signaling, video mixer, floor control etc. Examples of composite substrates include dial-out video conferencing and dial-in audio conferencing. Different kinds of conferences can be created by composing different conferencing substrates on the fly. For example, a dial-in signaling substrate and a video mixer substrate can be composed to create a dial-in video conference.

Among the six roles of the business model followed, this thesis focuses on conferencing service providers, conferencing platform providers and conferencing infrastructure providers. It is assumed that the substrate provider plays the role of the conferencing infrastructure provider as well. It is also assumed that conferencing infrastructure providers provide atomic conferencing substrates.

2.5 Conferencing Service Provisioning

A *conferencing service* offers conferencing that comprises of the mandatory conferencing components (signaling and media) and may include optional components (e.g. floor control). Examples of conferencing services include dial-in audio conference and dial-out video conference with floor control. Contrary to this, an application where conferencing is an important part is referred to as *conferencing application*. Examples of conferencing applications are massively multi-player online games, distance learning applications etc. Third-party service providers can provision and offer conferencing services which conferencing application developers can use in their applications. *Conferencing service provisioning* entails the whole lifecycle of the service, which consists of service development, deployment and management. Using a conferencing PaaS, the service providers provision conferencing services and offer them as SaaS. Third-party conferencing applications (e.g. game and distance learning) consume conferencing services offered as SaaS.

In contrast, by using conventional PaaS, application developers can provision applications (e.g. mobile back-end applications and web applications) and offer them as SaaS, which third-party applications (e.g. game, web applications, mobile applications) can use. Thus, conferencing service provisioning using a conferencing PaaS is analogous to application provisioning using a conventional PaaS.

2.6 Chapter Summary

In this chapter we discussed the background concepts which are related to this thesis. First we introduced the concept of Multi-party Multimedia conferencing, its key architectural components, different types of conferencing and two key conferencing technologies. It was followed by a discussion of Cloud Computing, its different definitions, benefits and three main service models. Finally we discussed Platform-as-a-Service (PaaS), its capabilities and advantages and also two existing PaaS Cloud Foundry and Google App Engine.

Chapter 3

3. Scenarios, Requirements and State of the Art Evaluation

This chapter includes four sections. In the first section, we discuss two motivating scenarios for a conferencing PaaS that service providers use to provision conferencing services. In the second section, we derive requirements on conferencing PaaS from these scenarios. Conference scaling is an important feature of this PaaS. Therefore, we also derive a set of specific requirements on the conference scaling algorithm. In the third section, we review and evaluate the state of the art based on our set of requirements. Finally, we summarize the chapter.

3.1 Scenarios

The motivating scenarios that we present in this section cover all phases of conferencing service provisioning. The phases are service development, deployment and management. Service execution is part of the management. The first motivating scenario covers conferencing service development and deployment. The second scenario relates to conferencing service execution and shows how running services handle conference life-cycle, i.e., conference creating, starting, scaling and stopping. Before describing the scenarios, we first present the involved actors.

3.1.1 Actors

Figure 3-1 illustrates the motivating scenarios. The actors in the scenarios are:

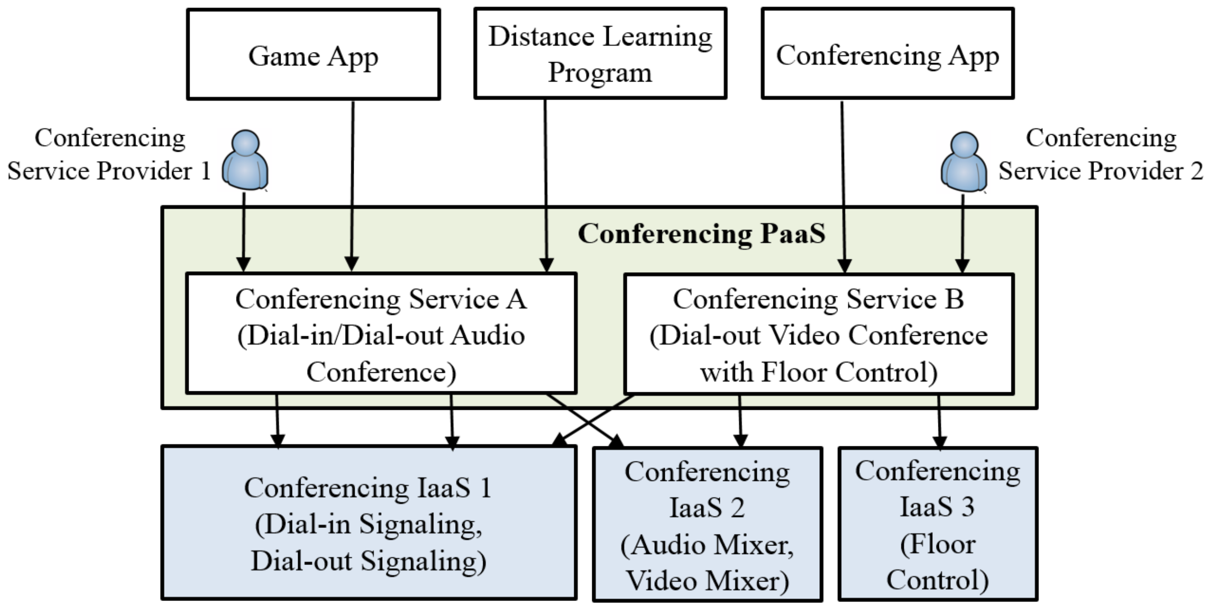

Game App
Distance Learning Program
Conferencing App
Conferencing Service Provider 1
Conferencing Service Provider 2
Conferencing PaaS
Conferencing Service A (Dial-in/Dial-out Audio Conference)
Conferencing Service B (Dial-out Video Conference with Floor Control)
Conferencing IaaS 1 (Dial-in Signaling, Dial-out Signaling)
Conferencing IaaS 2 (Audio Mixer, Video Mixer)
Conferencing IaaS 3 (Floor Control)

Figure 3-1: Conferencing Service Provisioning in the Cloud

1) **Conferencing application developers:** We consider developers of three conferencing applications – (i) a game using dial-in audio conferencing, (ii) a distance learning program using dial-out audio conferencing, and (iii) a plain conferencing application offering dial-out video conference with floor control.

2) **Conferencing service providers:** They provision conferencing services and offer them as SaaS to conferencing application developers. Service providers are assumed to have programming expertise. They are also knowledgeable about the high-level aspects of conferencing such as different kinds of conference models (e.g. dial-in, dial-out, and ad-hoc), media (e.g. audio, video, and text), conferencing technologies (e.g. SIP-based and WebRTC) and conference control (e.g. floor control and policy control). However, the service provider may not be an expert in complex details of conferencing protocols.

We consider two service providers. One provider offers conferencing service A that supports both dial-in and dial-out audio conferences. The distance learning and the game applications consume service A. The other provider offers dial-out video conference with floor control (service B), used by the plain conferencing application.

3) **Conferencing PaaS providers:** They offer conferencing PaaS to service providers for easy conferencing service provisioning. In the motivating scenarios, only one conferencing PaaS, offered by a PaaS provider, is considered.

4) **Conferencing IaaS providers:** We have reused a business model for cloud-based conferencing from the state of the art [1]. It relies on *Conferencing IaaS* that, instead of virtual machines, provides fine-grained, sharable and virtualized conferencing building blocks named substrates (e.g. dial-in signaling, audio mixer, floor control). The conferencing PaaS can provision these conferencing substrates to create new conferences. In this scenario, we assume that the conferencing PaaS has prior knowledge of the existing conferencing IaaSs and their offered substrates.

   We consider three conferencing IaaS providers. The first IaaS provides dial-in signaling and dial-out signaling substrates. The second IaaS offers audio mixer and video mixer substrates. The last IaaS provides floor control substrate.

3.1.2 Conferencing Service Development and Deployment Scenario

In this scenario, we consider the development and deployment of conferencing service A that supports both dial-in audio and dial-out audio conferencing. We assume that service A targets only WebRTC-based end-points.

The conferencing PaaS facilitates conferencing service provisioning by providing high-level interfaces, including a set of abstract conferencing service development APIs that service providers can use while writing code. An API may require one or more parameters. For example, *create conference* API requires several parameters such as conference model, media, conferencing technology that the target end-points support and whether the application needs conference control.

Figure 3-2 depicts the scenario of conferencing service development and deployment. In this scenario, while developing the conferencing service A, the service provider uses high-level *create conference* API in the code to handle requests from the service consumers to create new conferences. Necessary API parameters including media *audio*, and conferencing technology WebRTC are also passed. Regarding the parameter conference model, dial-in and dial-out are passed to the API when the service receives requests from the game and the distance learning applications, respectively. The API invocation with appropriate values of parameters happen during execution, which will be discussed in the next scenario.

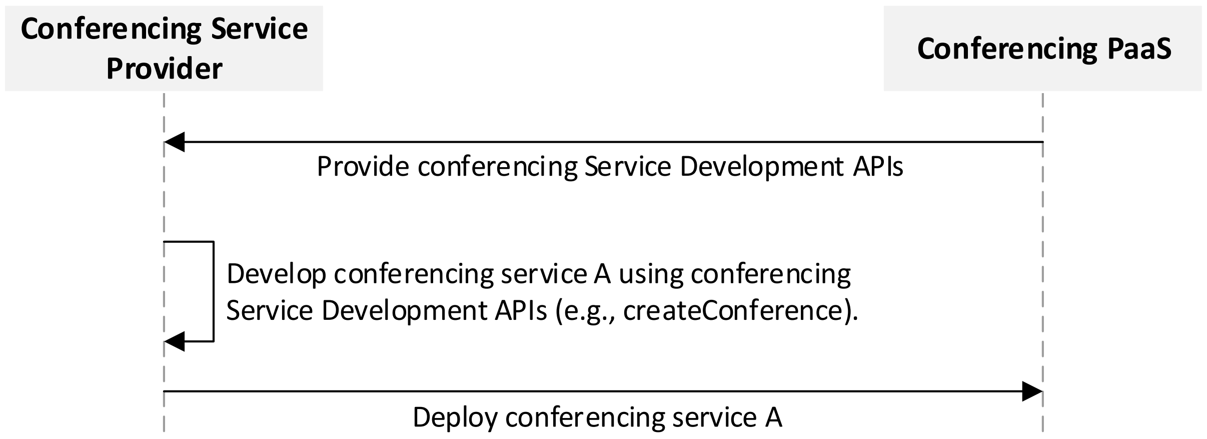
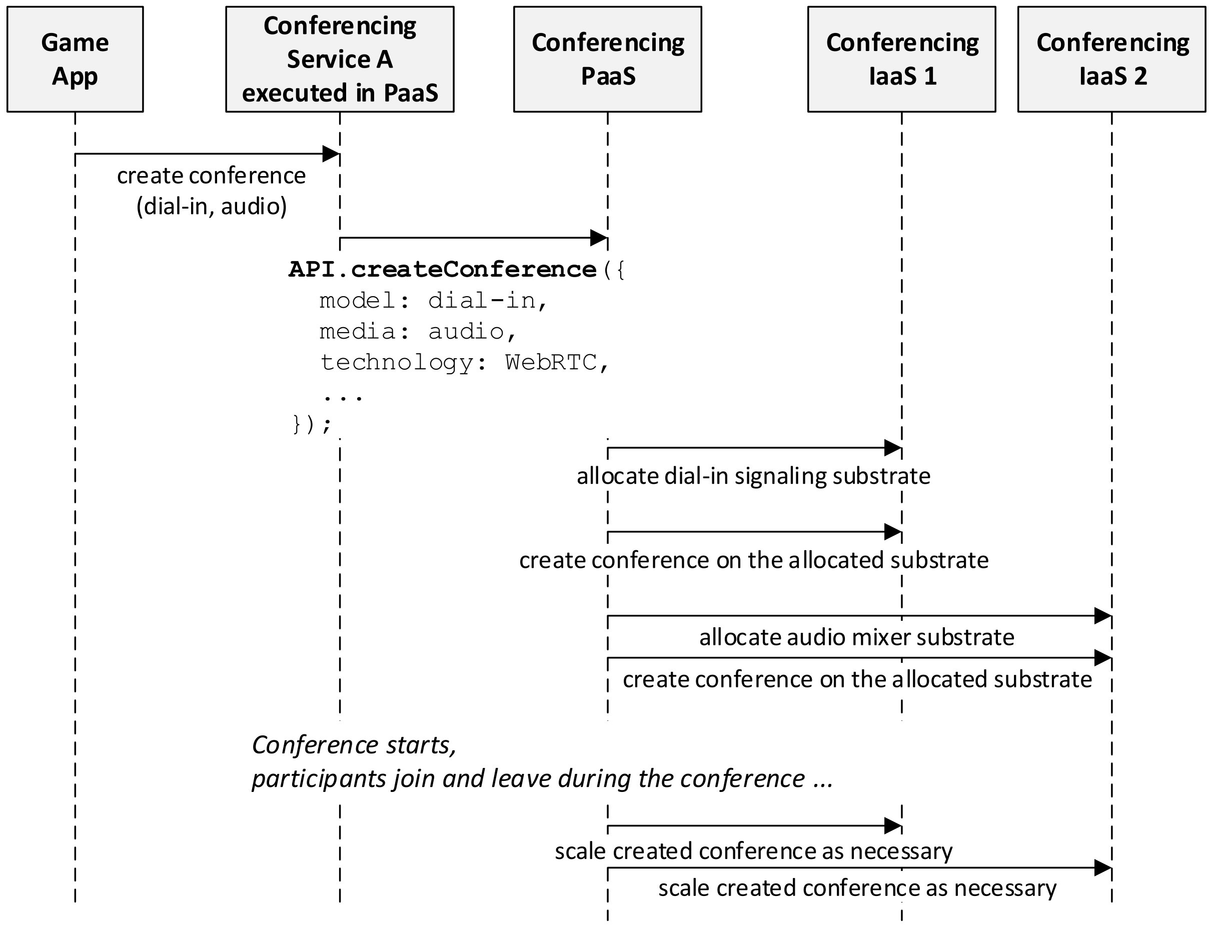

Conferencing Service Provider
Conferencing PaaS
Provide conferencing Service Development APIs
Develop conferencing service A using conferencing Service Development APIs (e.g., createConference).
Deploy conferencing service A

Figure 3-2: Conferencing Service Development and Deployment Scenario

After the service provider finishes developing the conferencing service, another high-level interface, such as a GUI provided by the conferencing PaaS, is used to deploy and to start the service in the PaaS.

3.1.3 Conferencing Service Execution Scenario

Figure 3-3 depicts the scenario of conferencing service execution. In this scenario, we assume that the conferencing service A is running in the conferencing PaaS. Consider that the conferencing service receives from the game application a request to create a new conference. In order to handle such requests, the service provider has already used in the code the *create conference* API provided by the conferencing PaaS. Therefore, that API is invoked during runtime. Conferencing PaaS, in response to this API invocation, creates a new dial-in audio conference by using necessary substrates offered by conferencing IaaSs. In this scenario, it provisions dial-in substrate from the first conferencing IaaS and audio mixer substrate from the second IaaS.



Game App
Conferencing Service A executed in PaaS
Conferencing PaaS
Conferencing IaaS 1
Conferencing IaaS 2
create conference (dial-in, audio)
allocate dial-in signaling substrate
create conference on the allocated substrate
allocate audio mixer substrate
create conference on the allocated substrate
Conference starts, participants join and leave during the conference ...
scale created conference as necessary
scale created conference as necessary

```
API.createConference({
  model: dial-in,
  media: audio,
  technology: WebRTC,
  ...
});
```

Figure 3-3: Conferencing Service Execution Scenario

The conference gets started at the scheduled time. During the conference, as more participants (game players) join the conference (the game), PaaS needs to scale up the conference to accommodate more participants. In a similar manner, as participants leave the conference, PaaS scales down the conference to minimize cost. This helps maintain the pay-per-use principle of Cloud Computing. However, elastic conference scaling requires an algorithm that conferencing PaaS performs to scale the conference up or down during runtime. Conferencing PaaS uses the algorithm to determine when to scale and the new conference size.

3.2 Requirements

In this section, we first present the general requirements on the conferencing PaaS. Then we discuss the specific requirements on conference scaling algorithm.

3.2.1 Requirements on the Conferencing PaaS

The following four requirements on the conferencing PaaS are derived from the motivating scenarios described in section 3.1. The first requirement is derived from the conferencing service development and deployment scenario. The last three are derived from the conferencing service execution scenario.

1) *High-level Interfaces for Service Providers:* The conferencing PaaS interfaces should enable the service providers to provision new services 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 services (e.g. a dial-in video conference with five minutes of chat per hour). This requirement is discussed in [42] for conferencing application providers.

2) *Composition of Conferences from Substrates:* When a conferencing service receives a request to create a new conference, the PaaS should determine necessary substrates, select appropriate conferencing IaaSs providing those substrates and then compose the requested conference from the selected substrates. This requirement is discussed in [43] for application layer instead of platform layer.

3) *Elastic Scalability:* The conferencing PaaS, in collaboration with the conferencing IaaSs, should scale the ongoing conferences in response to the fluctuating number of participants. This

allows the PaaS to gain cost efficiency and to follow the pay-per-use principle. This requirement is discussed in [43] and [44]. The conferencing PaaS should be able to scale as much as required by the most demanding conferencing application. For example, massively multi-player online games are one of the most demanding applications with hundreds of thousands participants [45].

4) *Quality of Service:* Meeting Quality of Service (QoS) requirements, such as latency, jitter and throughput, is critical as conferencing services are real-time. This thesis focuses on the latency of operations performed during a conference (e.g. participant joining and setting floor chair). This requirement is discussed in [46], [47] and [48].

3.2.2 Requirements on the Conference Scaling Algorithm

In the business model [1] that we reuse in this thesis, the abstraction level of services offered by conferencing IaaS is substrate – which is an abstraction level higher than virtual machine. So instead of CPU or memory, the conferencing PaaS uses conference size (maximum number of participants) as the parameter for scaling conferences. For example, to scale up a conference, the PaaS increases the conference size unlike conventional PaaS which increases CPU and/or memory. When the conferencing PaaS scales an ongoing conference, it specifies the new conference size. Now an important factor in scaling is whether participants are likely to join or leave or stay in conference, in other words, the number of participants in near future. In this thesis, we assume that a prediction model is given as input to the conference scaling algorithm. This model predicts the number of participants in conference for future time instances. Depending on the quantity of predicted number of participants' increase or decrease in near future, the most appropriate time instances to scale need to be determined.

The first requirement is that the conference scaling algorithm should be able to determine the future time instances of scaling as well as the corresponding conference sizes. The corresponding conference sizes depend on predicted number of participants. Conferencing PaaS executes the algorithm and uses the obtained information to scale the conferences.

Conferencing IaaSs, upon receiving scaling requests from the PaaS, scale the substrates. This may require horizontal or vertical scaling of its internal resources which takes some time. So there is a delay between receipt of scaling request and its actual realization. The second requirement is that conference scaling algorithm should take into account this constraint of delay.

The third requirement is that the conference scaling algorithm should maximize resource efficiency by minimizing over-provisioning in terms of number of participants. This means the algorithm should minimize the difference between number of participants that the running conference can support and the number of participants that we may have in the conference. This resource efficiency results into cost optimization.

The fourth requirement is that the conference scaling algorithm should have a response time acceptable by the application which is using the conferencing service. This means the algorithm should be efficient in order for the conferencing PaaS to scale the ongoing conferences in a timely manner. The runtime complexity of the algorithm directly corresponds to processing time needed by PaaS to perform the algorithm. Since conferencing applications are real-time, the processing time should be in milliseconds, i.e. less than a second.

3.3 State of the Art Evaluation

In this section, we review and evaluate the state of the art for a conferencing PaaS. We divide the state of the art into three sub-sections. The first sub-section reviews the existing architectures related to cloud-based conferencing. The second sub-section discusses existing PaaS solutions. The third sub-section reviews the existing algorithms to scale conference in a cloud environment

3.3.1 Cloud-based Conferencing Architectures

In this section we first review systems or architectures for cloud-based conferencing proposed in the literature. Then we review some representative commercial cloud-based conferencing solutions.

3.3.1.1 Cloud-based Conferencing Architectures in Literature

Reference [43] discusses feasibility of cloud-based conferencing and proposes a high-level framework for that. It divides the proposed cloud-based conferencing system into four layers: physics, virtualization, platform and application layers. Platform layer consumes virtualized resources (computing, storage and networking) from IaaS only to host services. The framework presented in the paper follows principles of Service Oriented Architecture (SOA). The SOA layers span across PaaS and SaaS which is depicted figure 3-4:

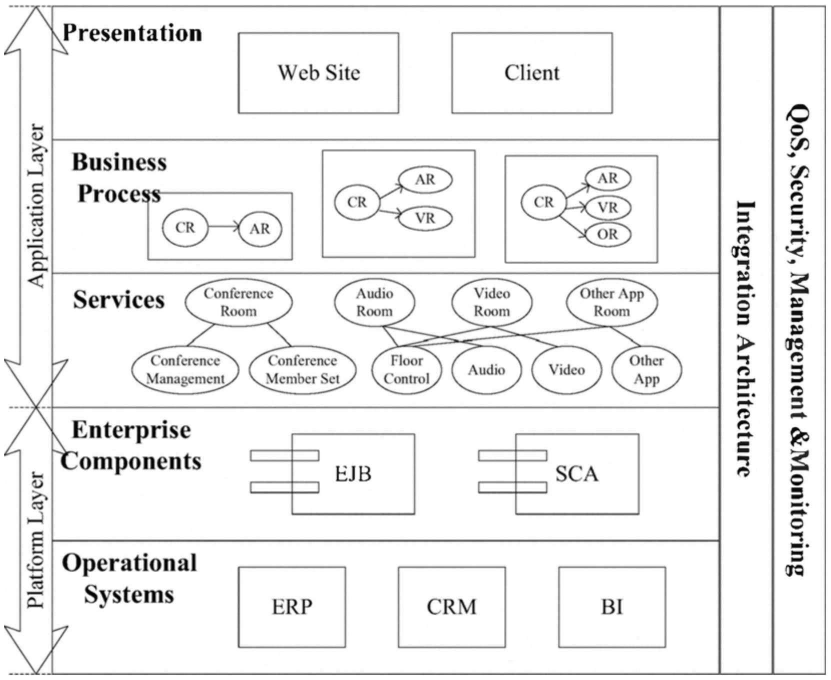


Figure 3-4: SOA Layers of Cloud Conferencing [43]

In figure 3-4, conferencing services and their composition process are handled in the application layer. However, when we evaluate it against our requirements on conferencing PaaS, high-level interfaces for the conferencing service providers are not provided. It claims benefit of scalability, but does not discuss how this can be achieved for conferencing services. It also does not provide any experimental data on QoS for cloud-based conferencing system.

Reference [42] presents an approach for providing video conference as web service and names their implemented system Nuve. It proposes a set of a high-level SaaS interfaces (REST APIs) offered by conferencing service providers. Third-party conferencing applications can consume these SaaS in order for their users to join virtual conference rooms and collaborate with audio, video, etc. It also proposes a REST architecture which is depicted in figure 3-5.

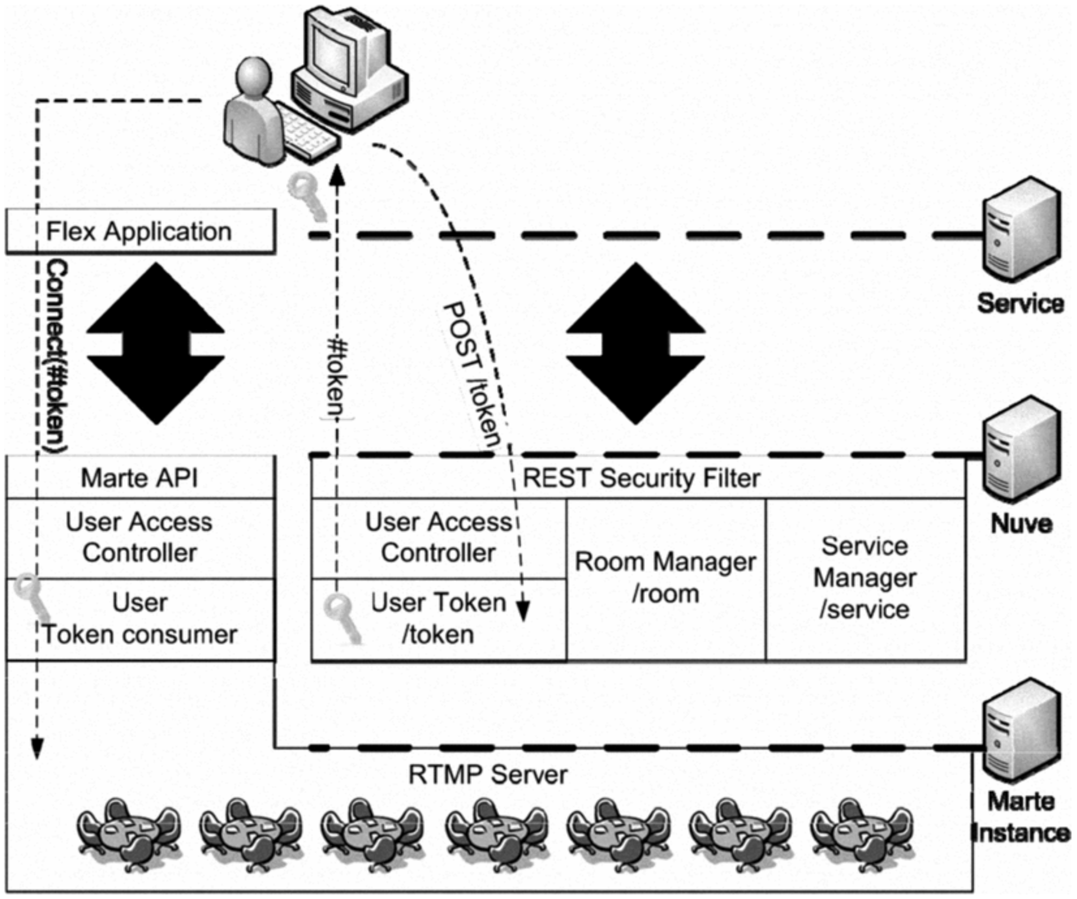


Figure 3-5: REST Architecture of Nuve [42]

However, it does not address how conferencing service providers could provision these SaaS. In other words, it does not provide high-level interfaces for conferencing service providers. Neither does it discuss conference composition, elastic scalability and QoS.

Reference [44] presents an architecture of virtualized infrastructure for cloud-based conferencing. Figure 3-6 depicts the infrastructure architecture proposed in [44]. It follows the same business model [1] that we reuse in this thesis. The infrastructure depends on fine-grained sharable virtualized conferencing substrates (e.g. dial-in signaling, video mixer), which can scale elastically. It also proposes PaaS/IaaS interfaces rooted in substrates. These characteristics make the infrastructure suitable for use by a conferencing PaaS. However, the PaaS-level issues including the interfaces for service providers and the composition of conference substrates are not taken into account. Neither do they provide QoS measurements for conference runtime operations.

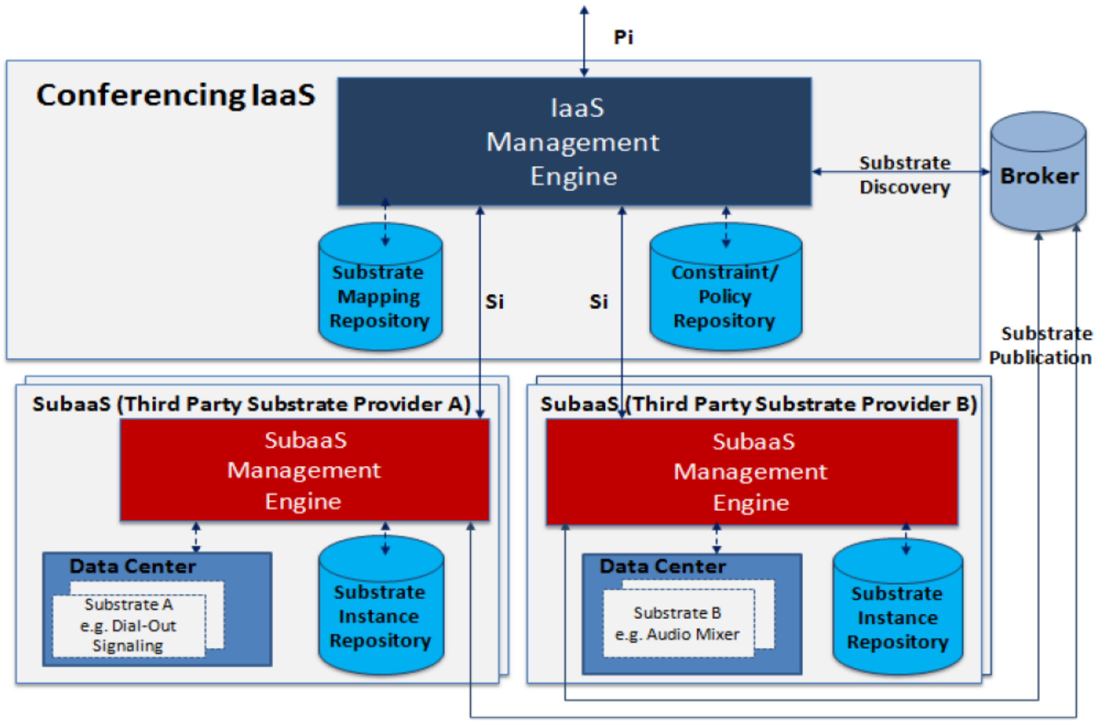

Pi
Conferencing IaaS
IaaS Management Engine
Substrate Discovery
Broker
Substrate Mapping Repository
Si
Si
Constraint/ Policy Repository
Substrate Publication
SubaaS (Third Party Substrate Provider A)
SubaaS Management Engine
Data Center
Substrate A e.g. Dial-Out Signaling
Substrate Instance Repository
SubaaS (Third Party Substrate Provider B)
SubaaS Management Engine
Data Center
Substrate B e.g. Audio Mixer
Substrate Instance Repository

Figure 3-6: Architecture of conferencing IaaS [44]

Reference [46] advocates that video conference be delivered as a cloud service. In order for the video conference cloud service providers to utilize the inter-datacenter network in an efficient way, it proposes a new application layer protocol named Airlift. Its objective is maximizing total throughput in the inter-datacenter network across all conferences while meeting end-to-end delay constraints. The problem it deals with is related to a particular QoS parameter, namely throughput. However, the other important requirements of conferencing service provisioning including high-level interfaces for service providers, conference composition and elastic scalability are not addressed.

Reference [47] proposes a cloud-based transcoding framework to achieve scalable and efficient video adaptation for mobile devices. It introduces a prediction-based scheduling algorithm to optimize both latency requirement and cloud utility cost for mobile clients. It focuses on scalability and QoS issues for a particular conferencing substrate, namely media transcoder. However, it does not address high-level interfaces for conferencing service providers and conference composition.

Reference [48] presents a cloud-based media mixer which performs mixing in a distributed way over the network compared to single node Multipoint Control Unit (MCU) used by traditional multi-party conferencing. It proposes heuristic algorithms for optimizing the virtual mixer topology that is adapted to the particular set of clients and servers available in the cloud. Similar to [46], its focus is related to a particular QoS parameter, namely delay between end-points. Nonetheless, conferencing service provisioning issues including high-level interfaces for service providers, conference composition and elastic scalability are not discussed.

3.3.1.2 Cloud-based Conferencing Products in the Market

Vidyo provides a software platform and development environment named VidyoWorks™ [49], which conferencing service providers can use. For service development, it provides development SDK as well as APIs. However, for deployment and management, it does not provide any high-level interface. Rather the service providers themselves have to deploy and manage the conferencing services. Vidyo does not set any limit on the number of participants in a conference. But elastic scalability is not included in its list of capabilities [50]. Measurements for QoS are not publicly available, either.

Cisco WebEx Meetings and other conferencing products [51] are offered as Software-as-a-Service (SaaS). These conferencing services benefit from Cisco's proprietary infrastructure named Cisco

WebEx Cloud. However, information is not available as to how the offered conferencing services are provisioned. WebEx family of products are claimed to be scalable but they support a maximum of 1000 participants [52] in a conference.

Blue Jeans [53] is a video conferencing service offered as SaaS. Information is not publicly available as to how Blue Jeans, the service provider, provisions the video conferencing service. QoS measurements for its offered service is also unavailable. In spite of its claim to be cloud-based, it supports a maximum of only 100 conference participants [54] and does not meet the requirement of elastic scalability.

3.3.2 PaaS Solutions

In this section, we review some widely used open-source as well as commercial Platform-as-a-Service solutions. Review of each PaaS solution is followed by evaluation against our set of requirements on conferencing PaaS.

Aneka [55] is a Platform-as-a-Service for provisioning scalable distributed applications that are developed using .NET framework. It can provision Windows based machines from both private and public infrastructure providers. But it is portable over different platforms and operating systems. For application development, Aneka provides Software Development Kit (SDK) and a rich set of APIs for expressing the business logic of distributed applications using the preferred programming abstractions (e.g. task, thread, MapReduce job). For application deployment and management, it provides necessary tools. However, the interfaces are not suitable for conferencing service provisioning. Aneka can scale distributed applications in an elastic manner. Therefore, conferencing services cannot be scaled elastically using Aneka. Its architecture does not address conference composition from substrates and conferencing QoS.

Cloud Foundry [32] is one of the most popular open-source Platform-as-a-Service. It has gathered support from many big companies such as EMC, HP, IBM, Intel, Pivotal, SAP, VMware as well as 40 other organizations. Its architecture is open and extensible, making it possible to integrate support for new programming frameworks, application services and underlying IaaS. It does not provide high-level interfaces for provisioning conferencing services. Neither does it address conference composition and QoS. It supports scaling of application instances but does not address elastic scaling of conferences.

Notable examples of commercial PaaS solutions are Google App Engine [56], Heroku [57], AWS Elastic Beanstalk [58], Salesforce.com's App Cloud [59], Red Hat's OpenShift Enterprise [60], Pivotal CF [61], IBM Bluemix [62], etc. None of these PaaS supports conferencing service provisioning. Therefore, they also do not address conference composition and conferencing QoS requirements. They provide scaling of only application instances and do not support conference scaling.

Table 3-1 summarizes the evaluation of existing works related to conferencing PaaS. The column value "Not addressed' means the related work does not deal with the requirement whereas "No" means the related work deals with the requirement but does not meet it. "Yes" means the requirement is fully satisfied by the related work. "Partially satisfied" indicates that the related work deals with and satisfies only parts of the whole requirement.

Related work	Requirements	**High-level interfaces**	**Composition of conferences from substrates**	**Elastic scalability of conferences**	**Quality of Service**
Cloud-based Conferencing Architectures	[43]	No	No	Not addressed	Not addressed
	[42]	No	Not addressed	Not addressed	Not addressed
	[44]	Not addressed	Not addressed	Yes	Not addressed
	[46]	Not addressed	Not addressed	Not addressed	Partially satisfied
	[47]	Not addressed	Not addressed	Partially satisfied	Partially satisfied
	[48]	Not addressed	Not addressed	Not addressed	Partially satisfied
	[49]	Partially satisfied	Not addressed	No	Not addressed
	[51]	Not addressed	Not addressed	No	Not addressed
	[53]	Not addressed	Not addressed	No	Not addressed
PaaS Solutions	[55]	No	Not addressed	No	Not addressed
	[32]	No	Not addressed	No	Not addressed

Table 3-1: Summary of Evaluation of the Related Works for Conferencing PaaS

3.3.3 Conference Scaling Algorithms

In this section, we review the existing works in literature that tackle the problem of elastic resource provisioning for scalable multi-party multimedia conferencing. The existing works vary in granularity of elasticity. Some of the existing works have coarse-grained elasticity (e.g. virtual machine instance, application instance) while the others have fine-grained elasticity (e.g. CPU instance, memory, storage).

Conferencing is an important part of Massively Multiplayer Online Games (MMOGs). Reference [63] presents a cloud-based dynamic resource provisioning middleware named CloudDReAM (Dynamic Resource Allocation Middleware) targeting MMOGs. The game developer first defines the most important load metrics (e.g. CPU or bandwidth usage) as well as their underloaded and overloaded threshold values. CloudDReAM continuously monitors those load metrics. Resource scaling is triggered when a threshold value is reached. The authors also propose two algorithms for resource scaling - each corresponding to a load event (underloaded or overloaded). Based on the detected load event, CloudDReAM performs one of the two algorithms which initiates load balancing and subsequently virtual machines are added to or removed from the system. When evaluated against the requirements on conference scaling algorithm, the granularity of scaling in CloudDReAM (virtual machine) is different from our target granularity (number of players that a game can support which is analogous to conference size). The two algorithms proposed do not take VM instantiation time into account. So it does not satisfy our first and second requirements on conference scaling algorithm. The remaining requirements also are not addressed.

Reference [64] proposes an elastic resource scaling scheme called PRedictive Elastic reSource Scaling (PRESS). The objective is to minimize waste of resources in order to optimize resource provisioning costs without violating service level objectives (SLOs). It does not assume advanced application profiling, model calibration or deep understanding of the application; rather it monitors usage of resources (CPU, memory, bandwidth). Based on the observed resource usage, it predicts future resource demand and scale accordingly. Two complementary resource demand prediction techniques are also proposed. When compared with the requirements on conference scaling algorithm, the granularity for resource scaling (e.g CPU, memory) does not match with our desired granularity (conference size). Resource provisioning delay also is not considered. It therefore does not satisfy the first and second requirements. It maximizes resource efficiency by minimizing waste of resources but the granularity considered (CPU, memory) does not match with ours (conference size). So the third constraint is only partially satisfied. The authors show good response time for their proposed approach of elastic resource scaling.

Reference [65] presents a system called CloudScale that can automate fine-grained elastic resource scaling for multi-tenant cloud infrastructures. It addresses two key problems in prediction-driven dynamic resource scaling. The first problem is under- and over-estimation errors. It provides two complementary under-estimation error handling schemes, namely online adaptive padding and reactive error correction. The second problem is that scaling up can lead to conflicts among resource demand of applications that are co-located on the same host. Because the sum of resources required by co-located applications can exceed the maximum that a host can provide. In order to resolve the second problem, it uses a conflict prediction model to estimate when the conflict will happen, how serious the conflict will be (conflict degree) and how long the conflict will last (conflict duration). It proposes two techniques – one is handling the conflict locally by mitigating SLO violations and the other is predictive VM migration. The first technique is used when conflict degree is small and duration is short. Otherwise expensive VM migration is performed to avoid the conflict.

When it is evaluated against the requirements on conference scaling algorithm, granularity of elasticity considered in the paper does not match with our desired one. It also does not consider the resource provisioning delay in the proposed schemes. Thus it does not meet the first and second requirements on conference scaling algorithm. It attempts to achieve resource efficiency by minimizing waste of resources, though on a different granularity. So the third requirement is only partially satisfied. The proposed schemes and techniques show good response time which satisfies our fourth requirement.

Reference [66] proposes a lightweight scaling (LS) algorithm to provision resources for applications in an elastic way. It targets transaction-based multitier applications where QoS can be assessed based on the application's response time to each incoming request. The algorithm is given the upper and lower bounds of the required response time of an application. When the observed response time is greater than the upper bound, it performs Lightweight Scaling Up (LSU) algorithm. On the contrary, when it detects response time smaller than the lower bound, it performs Lightweight Scaling Down (LSD) algorithm. The LSU algorithm first tries self-healing scaling which is removing an idle resource from one VM and allocating it to an overloaded VM on the same physical machine. Then it performs resource-level scaling which is allocating resources (CPU, memory) available on a physical machine to an overloaded VM to scale it up. If the required range of response time cannot be achieved through self-healing scaling and resource-level scaling, then the algorithm performs VM-level scaling. On the other hand, the Lightweight Scaling Down (LSD) algorithm first attempts VM-level scaling down, then performs resource-level scaling down. Thus the algorithm incorporates fine-grained (resource-level e.g. CPU, memory etc.) as well as coarse-grained (VM-level) approaches in order to achieve elastic resource scaling. However, it takes a completely reactive approach.

Because of the difference in granularity of elasticity, it does not meet the first requirement on conference scaling algorithm. The second requirement also is not satisfied as the proposed algorithm does not take resource provisioning delay into account. The algorithm aims at achieving resource efficiency by allocating just enough resources to meet the target range of response time; but scaling is done using a different granularity. So it partially meets the third requirement. The authors provide experimental results proving the algorithm's acceptable response time. So the fourth requirement is satisfied.

Table 3-2 summarizes evaluation of related works for conferencing scaling algorithm. The meaning of column values "Yes", "No", "Not Addressed", "Partially Satisfied" are the same as in table 3-1. Based on our review of the state of the art for both conferencing PaaS and conference scaling algorithm, to the best of our knowledge, there is no conferencing PaaS that fulfills our requirements completely. In addition to that, there is no conference scaling algorithm that satisfies the requirements. Some of the works cover part of our requirements but none of them meet all the requirements completely.

Requirements / Related work	**Time instances of scaling and corresponding conference size**	**Scaling delay**	**Minimization of waste of resources in terms of number of participants**	**Response Time**
[63]	No	No	Not addressed	Not addressed
[64]	No	No	Partially	Yes

			satisfied	
[65]	No	No	Partially satisfied	Yes
[66]	No	No	Partially satisfied	Yes

Table 3-2: Summary of Evaluation of Related Works for Conferencing Scaling Algorithm

3.4 Chapter Summary

In this chapter, we presented two motivating scenarios for conferencing PaaS. Then we derived the set of requirements based on the scenarios presented. We divided the requirements into two groups: requirements on conferencing PaaS and requirements on conference scaling algorithm. Next we reviewed and evaluated the state of the art based on the requirements. Finally we come to the conclusion that none of the existing works evaluated in the state of the art meets all of the requirements completely.

Chapter 4

4. Proposed Architecture

In the previous chapter, we derived a set of requirements on a conferencing PaaS. In this chapter, we propose architecture for a conferencing PaaS based on the requirements. We start by explaining the overall architecture. Next, to help conferencing service providers develop different types of conferencing services easily, we propose a set of conferencing Service Development APIs. Then, for a conferencing service, we describe illustrative scenarios of service development, deployment and execution. Next, we discuss how the requirements are met by the architecture. Finally, we summarize this chapter.

4.1 Overall Architecture

In this section, we first discuss the architectural principles that we follow to design the proposed architecture for a conferencing PaaS. Then we describe the architectural components.

4.1.1 Architectural Principles

The first architectural principle is related to composition of conferencing substrates, which are sharable, virtualized and fine-grained building blocks of conferencing. Two widely used compositional approaches are orchestration and choreography [67]. The former is a centralized approach, allowing a central entity to control the component services and their interactions. In contrast, the latter allows the component services to collaborate in a decentralized manner. The first principle of our architecture is to adopt the orchestration approach for the substrate composition because it provides PaaS with a greater control on the substrates and their interactions.

This helps in provisioning complex conferencing services, for example, dial-in audio conference with text-chat for five minutes per hour.

The second principle is to use high-level PaaS/IaaS interfaces rooted in substrates. It contributes to easy conference composition from substrates. This principle also enables PaaS to request IaaSs for scaling conferences in terms of conference size, instead of VM resources. Scaling by PaaS in terms of conference size allows IaaSs, which manage the VMs hosting substrates, to make decisions about necessary VM resource allocation in response to changed conference size. In addition, it helps PaaS to bill in terms of conference size, which is more intuitive than VM for the service providers.

The third principle is to extend the existing PaaS architectures such as Aneka [55] and Cloud Foundry [32]. This allows us to reuse the existing PaaS for the conferencing PaaS implementation.

4.1.2 Architectural Components

The proposed architecture consists of a repository and five components, as shown in figure 4-1. These components deal with three key facets: (i) Conferencing services, (ii) conferences and (iii) substrate information.

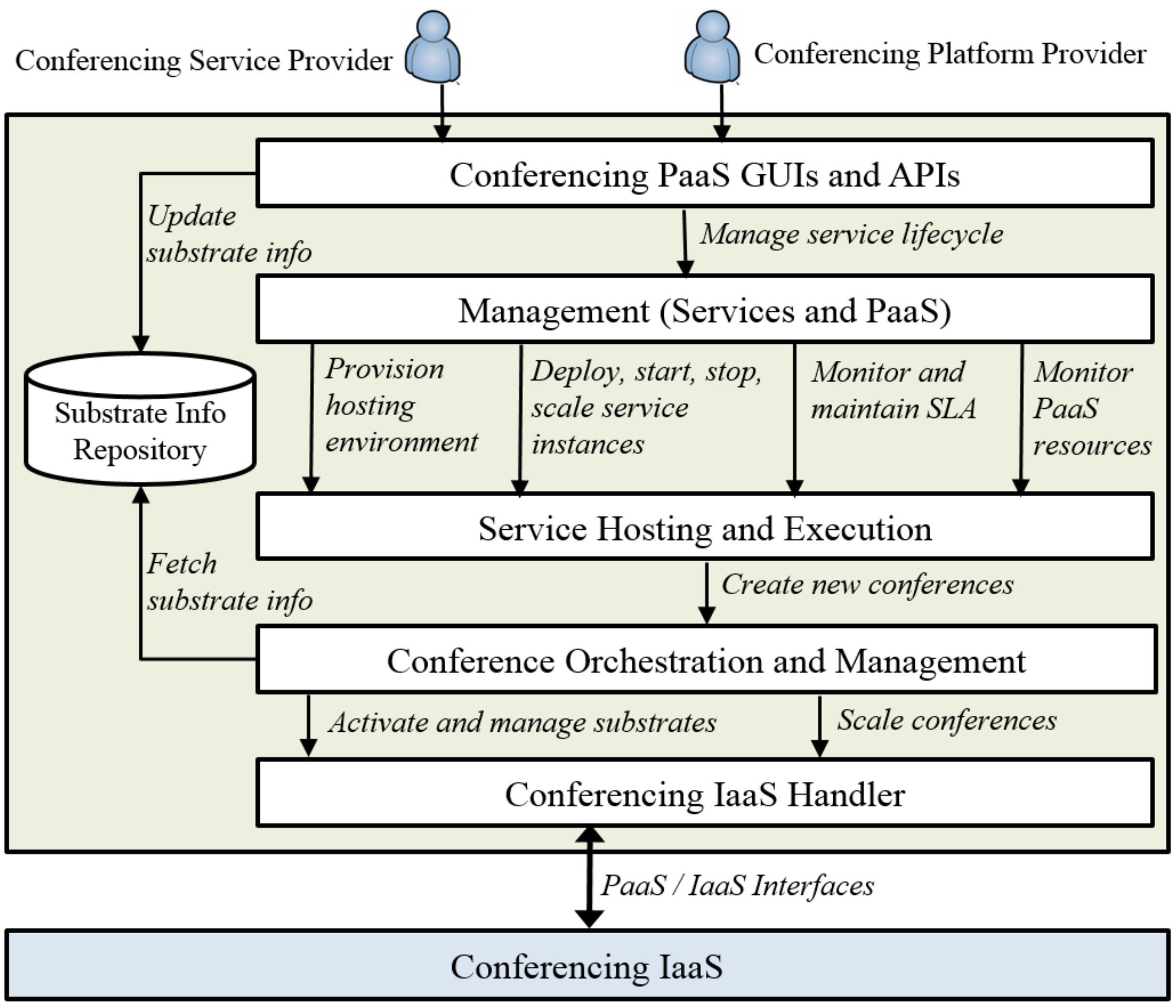

Conferencing Service Provider
Conferencing Platform Provider
Conferencing PaaS GUIs and APIs
Update substrate info
Manage service lifecycle
Management (Services and PaaS)
Substrate Info Repository
Provision hosting environment
Deploy, start, stop, scale service instances
Monitor and maintain SLA
Monitor PaaS resources
Service Hosting and Execution
Fetch substrate info
Create new conferences
Conference Orchestration and Management
Activate and manage substrates
Scale conferences
Conferencing IaaS Handler
PaaS / IaaS Interfaces
Conferencing IaaS

Figure 4-1: Overall Architecture of Conferencing PaaS

1) Components Related to Conferencing Services:

This facet includes service development, deployment and management. *Conferencing PaaS GUIs and APIs* component extends application provisioning front-end of regular PaaS architectures by providing a set of conferencing Service Development APIs. *Management (Services and PaaS)* and *Service Hosting and Execution* components are reused from conventional PaaS architectures.

Conferencing PaaS GUIs and APIs component provides tools for the conferencing service providers. For easy development, service providers use high-level Service Development APIs (discussed in section 4.2), which is novel in this architecture. They also use GUI for service deployment and management, such as starting, updating and stopping services.

Management (Services and PaaS) component manages the conferencing services and monitors their QoS and SLAs. *Service Hosting and Execution* component hosts the conferencing services. It allocates necessary PaaS resources (e.g. server runtime and database drivers) and prepares execution environment before hosting.

Management (Services and PaaS) component receives request from the conferencing PaaS GUI for service deployment and management. It deploys and executes services in *Service Hosting and Execution* component and manages them during execution.

2) Components Related to Conferences:

This facet concerns conference composition and management of created conferences including elastic scaling. *Conference Orchestration and Management* component creates and manages conferences. More explicitly, it performs the following five tasks:

i) It determines the necessary substrate types and their associated requirements by using, for instance, syntactic matching with the categorized API parameters (discussed in section 4.2).

ii) Given the requirements of a substrate, it selects the most suitable conferencing IaaS, by using an algorithm. Existing algorithms for cloud service selection, which has been formulated as a multi-criteria decision problem [68], can be reused in this context. Service selection approaches and algorithms from the state of the art consider various factors, such as cost [69][70] and multiple QoS constraints [71]. Some existing works also consider service composition aspect, for example, transactional properties and QoS characteristics [72], reliability [73] and SLAs [74] of composite service.

iii) It orchestrates conferences from substrates and executes them. Note that conferences are executed in this component. In contrast, the conferencing services that create conferences are executed in the *Service Hosting and Execution* component. We assume that conferencing IaaSs expose substrates as RESTful web services as in [44]. Therefore, existing approaches and techniques [75] for RESTful web service orchestration can be reused. This component uses dynamic binding instead of static because conferencing PaaS selects the most suitable substrates on the fly.

iv) It manages the composed conferences. For example, it can add or remove video from a conference.

v) It monitors the current size of each running conference to make decisions about scaling. If needed, it requests conferencing IaaSs to scale in terms of conference size. However, this decision-making process requires new conference scaling algorithms.

Conferencing IaaS Handler component handles all communications between the conferencing PaaS and the conferencing IaaSs. It realizes the high-level conferencing PaaS/IaaS interfaces proposed in [44], which is reused in this work.

Conference Orchestration and Management component receives requests from conferencing services, which are running in *Service Hosting and Execution* component. Based on the requests received (e.g. create a conference and stop a conference), it takes actions and communicates with IaaSs via *Conferencing IaaS Handler*. Note that *Conference Orchestration and Management* is a novel component while *Conferencing IaaS Handler* is an extension of IaaS communication component in conventional PaaS architectures.

3) **Components Related to Substrate Information:**

To select the best conferencing IaaS for a given substrate, PaaS needs certain information about that substrate, such as substrate type, price, SLA and QoS. Conferencing PaaS provider uses a GUI in *Conferencing PaaS GUIs and APIs* component to manage (e.g. add, remove, update) such information of the substrates. The information is stored in the *Substrate Information Repository*.

4.2 Conferencing Service Development APIs

In order to facilitate conferencing service development by the service providers, *Conferencing PaaS GUIs and APIs* component include Service Development APIs. In this section, the proposed development APIs are discussed.

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 a conference's 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. It also allows to describe the APIs in a generic way.

Using the first principle, we determine the necessary data-set for RESTful API design. The data-set includes conference, participant, media (e.g. audio, video and text), floor and subconference (conference within a conference). The REST resources and their hierarchy can be easily derived then. The top level REST resource, which the service providers deal with, is *list of conferences*. Each individual *conference* resource has several subordinate resources such as *list of participants*, *list of media*, *list of floor* and *list of subconferences*.

Table 4-1 delineates the proposed APIs. It shows the REST resources along with the operations for each. The request parameters and the response contents are also listed.

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, floor control, technology, conference size, QoS requirements, etc.	ID and URI of created conference
	Read IDs of all running conferences	GET:/conferences	None	List of conference IDs and URIs
Conference	Read info of a conference	GET:/conferences/ {conferenceId}	None	Conference description and status
	Terminate a conference	DELETE:/conferences/ {conferenceId}	None	Success or failure indication
List of participants	Add a participant	POST:/conferences/ {conferenceId}/participa nts	Participant description: name, URI	ID and URI of new participant

	Read: IDs of all participants in a conference	GET:/conferences/ {conferenceId}/participants	None	List of participant IDs and URIs
Participant	Remove participant from a conference	DELETE:/conferences/ {conferenceId}/participants/ {participantId}	None	Success or failure indication
	Read description and status of a conference participant	GET:/conferences/ {conferenceId}/ participants/{participantId}	None	Participant status (joined or not), name, URI
List of media	Add a new media later to a conference (e.g. going from audio to video conference)	POST:/conferences/ {conferenceId}/ media	Description of media (e.g. frame rate, resolution, bit rate, codec, latency requirement)	ID and URI of new media
	Get all media used in a conference (e.g. video conference has 2 media- audio and video)	GET:/conferences/ {conferenceId}/ media	None	List of IDs and URIs of media used. These subordinate resources are automatically created when a conference is first instantiated.
Media	Remove media from a conference (e.g. going from video to audio conference)	DELETE:/conferences/ {conferenceId}/media/ {mediaId}	None	Success or failure indication
	Read info of a specific media used in a conference	GET:/conferences/ {conferenceId}/media/ {mediaId}	None	Description of media (e.g. frame rate, resolution, bit rate)
List of floors	Add a floor	POST:/conferences/ {conferenceId}/floors	Floor description: chair, floor participants	ID and URI of new floor added

	Read IDs of all floors in the conference	GET:/conferences/ {conferenceId} /floors	None	List of floor IDs and URIs
Floor	Remove a floor	DELETE:/conferences/ {conferenceId}/ floors/ {floorId}	None	Success or failure indication
	Read info of a conference floor	GET:/conferences/ {conferenceId}/ floors/ {floodId}	None	Floor description: chair, floor participants, floor requests
List of floor chairs	Add a new floor chair	POST:/conferences/ {conferenceId}/floors/{floorId} /floorChairs	Floor chair description: name, URI	ID and URI of new floor chair
	Read IDs of floor chairs	GET:/conferences/ {conferenceId}/ floors/ {floorId}/ floorChairs	None	IDs and URIs of existing floor chairs
Floor chair	Remove a floor chair	DELETE:/conferences/ {conferenceId}/floors/{floorId} /floorChairs/{floorChairId}	None	Success or failure indication
List of floor participants	Add a participant to a floor	POST:/conferences/ {conferenceId}/floors/{floodId} /floorParticipants	Description of floor participant: name, URI	ID and URI of new floor participant
	Read IDs of all floor participants	GET:/conferences/ {conferenceId}/floors /{floodId} /floorParticipants	None	List of floor participant IDs and URIs
Floor participant	Remove participant from a floor	DELETE:/conferences/ {conferenceId}/floors /{floodId} /floorParticipants /{floorParticipantId}	None	Success or failure indication
List of floor requests	Make a floor request	POST:/conferences/ {conferenceId}/floors /{floodId} /floorRequests/	Description of floor request: ID of floor participant who requested floor, timestamp of request	ID and URI of new floor request
	Read IDs of floor requests not handled yet	GET:/conferences/ {conferenceId}/floors /{floodId} /floorRequests/	None	List of floor request IDs and URIs

Floor request	Grant floor	PUT:/conferences/ {conferenceId}/floors /{floodId} /floorRequests /{floorRequestId}	Status: granted	Success or failure indication
	Deny floor	DELETE:/conferences/ {conferenceId}/floors/{floodId} /floorRequests/{floorRequestId}	None	Success or failure indication
List of subconferences	Create a subconference	POST:/conferences /{conferenceId} /subconferences	List of participant IDs to add to the subconference	ID and URI of subconference
	Read IDs of all existing subconferences	GET:/conferences /{conferenceId} /subconferences	None	List of sub-conference IDs and URIs
Subconference	Read information of a subconference	GET:/conferences /{conferenceId} /subconferences /{subconferenceId}	None	List of participants in the sub-conference, creation time etc.
	Remove subconference	DELETE: /conferences /{conferenceId} /subconferences /{subconferenceId}	None	Success or failure indication
	Add a participant	PUT:/conferences /{conferenceId} /subconferences /{subconferenceId} /participants /{participantId}	None	Success or failure indication
	Remove a participant	DELETE:/conferences /{conferenceId} /subconferences /{subconferenceId} /participants /{participantId}	None	Success or failure indication

Table 4-1: Conferencing Service Development APIs

The categorization of API parameters is shown in table 4-2. This table highlights that, to create a conference, the service providers must choose and specify the mandatory conferencing aspects -

one of the three conference models, at least one media and the conferencing technology. It also shows the conditional dependencies of parameters. For example, for WebRTC-based conferencing, signaling protocol must be specified, as WebRTC standard does not mandate any particular signaling protocol [76].

	Categories of Parameters	**Example Values**		
Mandatory Aspects	Conference Model	Pre-arranged conference	Dial-in conference	
			Dial-out conference	
		Ad-hoc conference		
	Media	At least one of audio, video and text		
	Conferencing Technology	SIP-based	Signaling protocol	SIP by default. No need to specify.
			Audio encodings	No mandatory encodings. So, must specify.
			Video encodings	No mandatory encodings. So, must specify.
		WebRTC-based	Signaling protocol	No mandatory protocol. So, must specify.
			Audio encodings	Mandatory: G.711 and Opus. Can specify additional.
			Video encodings	Mandatory: H.264 and VP8. Can specify additional.
		Hybrid (SIP-based + WebRTC-based)	Mandatory protocols and encodings from both technologies apply. Can specify additional.	
Optional Aspects	Floor control	At least one floor control policy, e.g. chair-moderated and round-robin.		
	Subconference	Enabled or not		

Table 4-2: Categorization of API Parameters

The parameters that the service providers can change during runtime are italicized. For example, the service provider can add instant messaging to an audio conference or can remove video from an audio/video conference. However, the conference must have at least one media. A special case is conferencing technology where it is possible only to upgrade from SIP-based or WebRTC-based conferencing to hybrid conferencing.

4.3 Illustrative Scenario

The illustrative scenario consists of several actors. The first actor is a game application where players can talk anytime but can have private text chat for only 5 minutes per hour. The second actor is a service provider offering dial-in audio conferencing service with text chat available only for a certain period of time. The third actor is the conferencing PaaS that subscribes to three conferencing IaaSs A, B and C. The discovery and subscription of the IaaSs are assumed to occur offline. IaaS A and B offer dial-in signaling and audio mixer substrates; IaaS C offers an instant messaging substrate. Conferencing IaaSs represent the fourth actor in this scenario.

We divide the scenario into two parts. The first part concerns conferencing service development and deployment. It illustrates how service providers can use the development APIs to develop complex conferencing services easily. The second part relates to conferencing service execution. It shows how the conferencing PaaS creates a conference when the game application sends requests to the service.

Both parts of the scenario demonstrates the relevant interactions among the different architectural components of the proposed conferencing PaaS architecture. Although the scenario constitutes a subset of all the functionalities of the conferencing PaaS, it helps understand how the proposed architecture works and facilitates conferencing service provisioning.

4.3.1 Conferencing Service Development and Deployment

In this scenario, we assume that the service provider wants to develop a WebRTC-based conferencing service. The targeted conferencing end-points are capable of handling signaling protocol JSEP (JavaScript Session Establishment Protocol) and can process at least the mandatory audio and video codecs specified by WebRTC standard. We also assume that dial-in substrate offered by IaaSs support WebRTC technology.

The service provider uses high-level conferencing service development APIs in the code. In order to handle requests from service consumers to create conferences, the *create conference* API is used. The API returns an ID after successful creation of a conference. When conference is started, the service receives a notification from the PaaS. We assume that the conferencing service enables private text chat 30 minutes after the conference is started. So, when the *conference started* notification is received, the service provider uses a regular timer function (available in most programming languages) to enable text chat after 30 minutes. The service provider uses another API *add media* to add instant messaging to the conference for 5 minutes. A pseudo-code for this conferencing service is shown in figure 4-2.

```
createConference() {
  // Use conferencing service development API
  // createConference() to create a new
  // WebRTC-based dial-in video conference
  var conferenceId = API.createConference({
    model: "dial-in",
    media: "video",
    technology: "WebRTC",
    signalling: {
      protocols: ["JSEP"]
    },
    maxSize: 1000,
    startTime: new Date("February 17, 2016 15:00:00"), // scheduled conference
    duration: 2*60 // 2 hours
  });
}

onConferenceStarted(conferenceId) {
  // Conference has started. Enable text chat after 30 minutes.
  setTimeout(function () {
    startChat(conferenceId), // track ID
  }, 30*60*1000);
}

startChat(conferenceId) {
  // Use conferencing service development API
  // addMedia() to add chat for only 5 minutes.
  API.addMedia(conferenceId, {
    type: "instant messaging",
    protocol: "SIP/SIMPLE",
    duration: 5
  });

  // Add chat again after one hour.
  setTimeout(startChat, 60*60*1000);
}
```

Figure 4-2: Pseudo-code of dial-in audio conferencing service

When the service provider finishes developing the service, a Conferencing PaaS GUI, such as a simple command line interface (CLI), is used to deploy the service in the PaaS. The GUI conveys the command to *Management (Services and PaaS)* component, which deploys the service in collaboration with *Service Hosting and Execution* component. The interactions of components are illustrated in figure 4-3. After the service is deployed, the service provider, using the same conferencing PaaS GUI, starts the service.

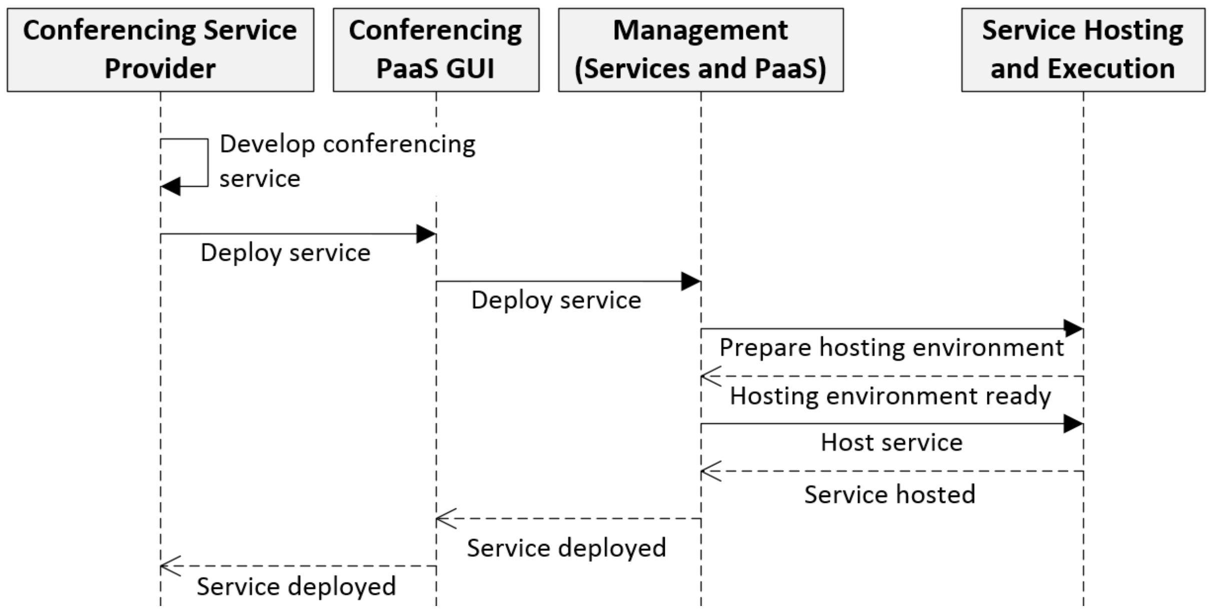

Conferencing Service Provider
Conferencing PaaS GUI
Management (Services and PaaS)
Service Hosting and Execution
Develop conferencing service
Deploy service
Deploy service
Prepare hosting environment
Hosting environment ready
Host service
Service hosted
Service deployed
Service deployed

Figure 4-3: Conferencing Service Deployment Scenario

4.3.3 Conferencing Service Execution

Once the service is started, it can receive requests for creating new conferences. Figure 4-4 depicts the interactions when the service receives a request from the game application for creating a conference. For brevity, the game application actor is omitted in the figure. The service first invokes the *create conference* API in order to create a dial-in audio conference. The service provider used this API in the code to handle such conference creation requests. Handling of API invocation is delegated to *Conference Orchestration and Management* component, which determines necessary substrates (dial-in and audio mixer substrates) and selects appropriate IaaSs. It is assumed that it selects IaaS A for dial-in signaling and IaaS B for audio mixer substrates. Next, it requests IaaSs, via *Conferencing IaaS Handler*, to activate the substrates. Interactions for substrate activation are not shown in the figure. After activation, *Conference Orchestration and Management* component orchestrates a new dial-in audio conference from substrates and then executes it. The orchestrated conference represents a full-fledged conference. It creates individual conferences on the substrates it is composed of. Finally, the ID of the full-fledged conference is returned to the game.

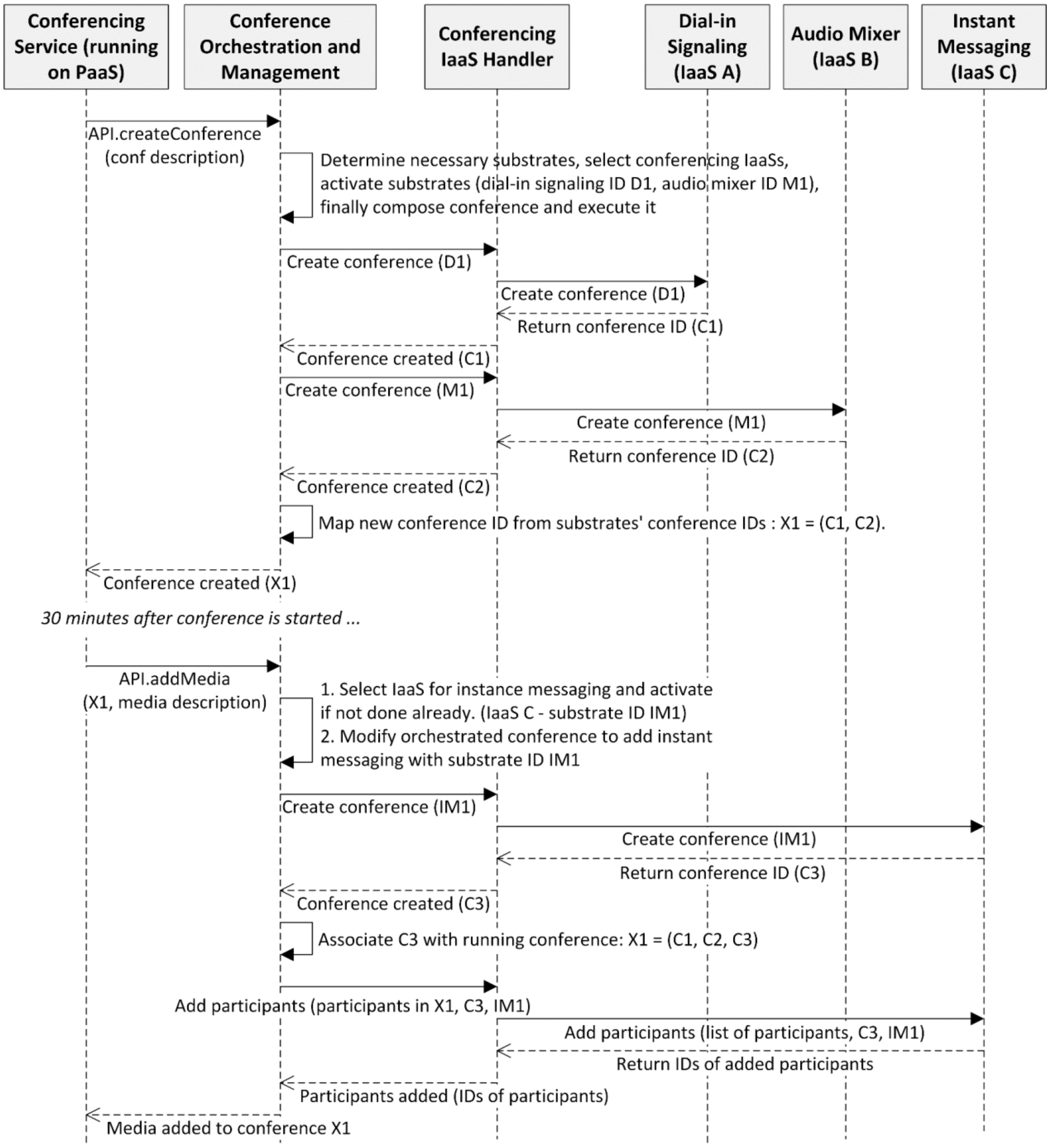

Conferencing Service (running on PaaS)
Conference Orchestration and Management
Conferencing IaaS Handler
Dial-in Signaling (IaaS A)
Audio Mixer (IaaS B)
Instant Messaging (IaaS C)
API.createConference (conf description)
Determine necessary substrates, select conferencing IaaSs, activate substrates (dial-in signaling ID D1, audio mixer ID M1), finally compose conference and execute it
Create conference (D1)
Create conference (D1)
Return conference ID (C1)
Conference created (C1)
Create conference (M1)
Create conference (M1)
Return conference ID (C2)
Conference created (C2)
Map new conference ID from substrates' conference IDs : X1 = (C1, C2).
Conference created (X1)
30 minutes after conference is started ...
API.addMedia (X1, media description)
1. Select IaaS for instance messaging and activate if not done already. (IaaS C - substrate ID IM1)
2. Modify orchestrated conference to add instant messaging with substrate ID IM1
Create conference (IM1)
Create conference (IM1)
Return conference ID (C3)
Conference created (C3)
Associate C3 with running conference: X1 = (C1, C2, C3)
Add participants (participants in X1, C3, IM1)
Add participants (list of participants, C3, IM1)
Return IDs of added participants
Participants added (IDs of participants)
Media added to conference X1

Figure 4-4: Conferencing Service Execution Scenario

The created conference is a scheduled conference. So the individual conferences on the dial-in and audio mixer substrates get started at the specified time. The conferencing PaaS receives notifications from the IaaSs. When both individual conferences on the substrates have started, the PaaS notifies the conferencing service that the conference has started. The conferencing service receives this notification and sets a timer to enable private text chat to the conference after 30 minutes. For brevity, the notifications from IaaSs to the PaaS and to the service are not shown in the figure.

The service invokes another API *add media* after the timer goes off. Using the API, the service adds instant messaging to the conference for 5 minutes. *Conference Orchestration and Management* component selects IaaS C for instant messaging substrate, activates a substrate on IaaS C and modifies the conference to add instant messaging. On the new substrate, individual conference is created for 5 minutes and existing participants are added. The conferencing service as well as the game application are notified that text chat has been enabled. Then, participants can start exchanging text messages. After 5 minutes, the individual conference created on the instant messaging substrate is terminated. The PaaS receives conference termination notification from IaaS C. *Conference Orchestration and Management* component modifies the conference again to exclude instant messaging. The conferencing PaaS can keep the instant messaging substrate activated for other conferences. It can also decide to deactivate the substrate.

4.4 How the Proposed Architecture Meets the Requirements

The proposed architecture of conferencing PaaS satisfies all the requirements mentioned in chapter 3. First, it provides high-level interfaces for provisioning conferencing services by service providers who have programming expertise. For service development, it provides necessary APIs that abstract complexities of conferencing. Moreover, it provides GUI for service deployment and management. Second, Conference Orchestration and Management entity as well as Substrate Information Repository collaborate to determine necessary substrates, select the most suitable conferencing IaaSs offering those substrates and compose the substrates into a conferences. Thus, the requirement of conference composition from substrates is met by the proposed architecture.

Third, in order to ensure elastic scalability of conferences, *Conference Orchestration and Management* component in the architecture monitors the running conferences and performs conference scaling algorithm (discussed in chapter 5). If needed, it requests conferencing IaaSs to scale the ongoing conferences. The PaaS/IaaS interaction interfaces chosen from the state of the art include API for conferencing PaaS to make such requests to IaaS. Fourth, the architecture has components for monitoring and managing both conferencing services and conferences created. The PaaS/IaaS interfaces include interfaces for receiving QoS status notification. Based on the QoS status, conferencing PaaS can take necessary actions, for example, changing conferencing IaaS for a substrate. Measurements for conference runtime operations (e.g. participant joining time), discussed in chapter 6, show that the end-to-end delay is acceptable. Therefore, the architecture meets the requirement of QoS.

4.5 Chapter Summary

In this chapter, we presented the proposed architecture for a conferencing PaaS. We discussed architectural principles that we followed in our design, the main components in our architecture and then the proposed conferencing service development APIs. Next we provided an illustrative scenario, showing how different components of the proposed architecture communicate with each other and how the proposed APIs can help the service providers easily develop conferencing services. Finally we explained how the proposed architecture fulfills the requirements we set in chapter 3. The proposed architecture meets all the requirements on conferencing PaaS whereas the state of the art does not meet all of them. For example, the proposed architecture satisfies the requirement of high-level interfaces for conferencing service providers which related works [32], [42], [43] and [55] do not satisfy. Related work [43] does not meet the requirement of composition of conferences from substrates. The requirement of elastic scalability of conferences is also not met by related works [32], [49], [51], [53] and [55]. The proposed conferencing PaaS architecture meets these requirements. It also meet the requirement of QoS when related works [46], [47] and [48] only partially satisfies the QoS requirement.

The validation of the proposed conferencing PaaS architecture will be discussed in chapter 6.

Chapter 5

5. Conference Scaling Algorithm

In the previous chapter, we propose a conferencing PaaS architecture. The *Conference Orchestration and Management* component of this architecture is responsible for scaling the ongoing conferences in an elastic manner. For elastic scalability, that component needs to run efficient conference scaling algorithm. In this chapter, we first discuss the problem of conference scaling and motivate the need of an algorithm for this with the help of an example. Then we formally state the problem and analyze the nature of the problem. Based on the problem analysis, we then propose a dynamic programming algorithm. After that, we present another greedy algorithm which is faster but produces suboptimal result. Finally, we summarize the chapter.

5.1 Problem Background and Motivation

During a conference, the allocated conference size should be large enough to accommodate all conference participants at any point in time. On one hand, as more participants join, the conference needs to be scaled up. On the other hand, as participants leave the conference, the conference needs to be scaled down; otherwise resources are wasted.

In the cloud-based conferencing business model [1] that we reuse, conferencing IaaS provides substrates, such as dial-in signaling, audio mixer, and video mixer. Conferencing PaaS creates and activates the necessary substrates and then composes conferences from them. During the conference execution, PaaS requests IaaSs to scale these substrates as participants join and leave. On the conferencing IaaS side, conference scaling is not instantaneous. Once PaaS requests for scaling, IaaS needs to perform several functions to realize the request, which takes a certain amount of time. This results in a time lag between sending a scaling request from PaaS and its actual realization in IaaS. Therefore, PaaS cannot send a new scaling request before the current scaling request is met. Hence the time lag constraint should be considered when decisions about scaling are made in the PaaS. We can schedule scaling if we know the prospective number of participants ahead of time.

We assume that a prediction model is available before the conference is started. It provides number of participants during different future time slots. The prediction model can be developed using statistical models of the user growth of cloud services [77] and using the past history of a time-series [78]. The prediction model enables the conferencing PaaS to look ahead and see how number of participants in a conference changes over time and helps it to decide if conference needs to be scaled or not. As the input (number of participants over time) to the conference scaling algorithm is provided by the prediction model, the algorithm's output will be as good as the accuracy of the prediction model. Note that the same prediction model may not be used for different types of conferencing applications. For example, the trends of joining and leaving participants in a conventional audio/video conference and in an online game are not the same and may need different prediction models.

Consider a scenario where predicted numbers of participants are $\{15, 30, 50, 80, 100, 150, 180, 250\}$ at intervals of one time slot. During the first time slot, there will be a maximum of 15 participants, during the second time slot, 15 more participants will join the conference and the number of participants will be 30 and so on.

For simplicity, we assume that the time lag for scaling is a multiple of time slots and in the scenario, we assume it to be 3 time slots. If scaling request is made at the start of the first time slot, the scaled conference is available during the fourth time slot. Therefore, to accommodate predicted number of participants during the fourth time slot, scaling request has to be made during the first time slot. [1] When the duration of time slot and that of scaling time lag are equal, the problem is trivial because conferencing PaaS can send scaling request just before the next time slot. Therefore, in this problem, the time lag's value is greater than one time slot.

For each time slot, the PaaS can decide to scale the conference or to continue with the current conference size. If it decides to scale, it also needs to decide the new conference size. Depending on the time slots during which decisions are made to scale the conference, multiple scaling schedules are possible.

Two possible scaling schedules are shown in Table 5-1. Due to the scaling time lag, the first three time slots must be accommodated with an initial large enough conference size, which we call *initial conference size*. In both schedules, the initial conference size is assumed to be 50. In schedule 1, scaling up is requested, denoted as $U(new\ conference\ size)$, for increased conference size of 150 at the first time slot, which can accommodate time slots from 4 to 6. There will be a maximum of 80 and 100 participants during time slots 4 and 5, respectively. Therefore, wasted resources in terms of number of participants in schedule 1 is 245 (with initial conference size 50: 50 – 15 + 50 - 30 = 55; first scaling to 150: 150 – 80 + 150 – 100 = 120; second scaling to 250: 250 – 180 = 70). In schedule 2, total wasted resource is 265 (with initial conference size 50:

[1] To avoid repetition, we shall state "scaling during a time slot", which will mean "scaling at the start of a time slot".

50 – 15 + 50 – 30 = 55; first scaling to 180: 180 – 80 + 180 – 100 + 180 - 150). Therefore, schedule 1 is better than schedule 2.

Time slot	1	2	3	4	5	6	7	8
Predicted number of participants	15	30	50	80	100	150	180	250
Schedule 1	$U(150)$	-	-	$U(250)$	-	-	-	-
Schedule 2	$U(180)$	-	-		$U(250)$	-	-	-

Table 5-1: An example of predicted number of participants in a conference

For the sake of argument, if the predicted number of participants during the eighth time slot would have been 300 instead of 250, then $U(250)$ would have been $U(300)$. In this case, schedule 2 would be better than schedule 1 with schedule 1 resulting in 295 wasted resources, whereas, schedule 2 results in 265 wasted resources.

In order to achieve cost efficiency, we want to compute a scaling schedule that minimizes waste of resources in terms of difference between allocated conference size and predicted number of participants.

5.2 Problem Statement

Given n time slots, $T = \{1, 2, 3, \dots \dots, n-1, n\}$, $P = \{p_1, p_2, p_3, \dots \dots, p_{n-1}, p_n\}$ are the expected number of conference participants, such that, there will be a maximum of p_i participants during time slot i. It is assumed that P is provided by a prediction model before the conference is started. The time slots in T are equal in length, that is, duration of time slot i equals that of j for $1 \leq i < j \leq n$.

Upon receiving a scaling request from conferencing PaaS, conferencing IaaS allocates or deallocates resources, which takes a certain time. Thus, there is a time lag, denoted as δ, at IaaS layer between the time of receiving the scaling request and the time of having the conference scaled according to the request. Conferencing IaaS is assumed to scale the conferences within the stipulated time lag without affecting QoS requirements of conferencing. In order to make the problem simpler, two assumptions are made about the time lag δ. First, it is a multiple of time slots. Second, time lags for scaling up and scaling down are equal. With these assumptions, if scaling is requested at the beginning of time slot i, the scaled conference is available at time slot $i + \delta$. For example, if δ is 3 time slots and scaling is requested during time slot 2, the scaled conference is available at time slot 5.

Given T, P and δ, the goal is to compute an optimal scaling schedule R,such that amount of wasted resources in terms of number of participants is minimized.

The value of δ can vary for different IaaSs. If a conferencing PaaS provisions services from multiple IaaSs, then it just computes an optimal schedule for each value of δ corresponding to the different IaaSs. Table 5-2 delineates the notations used in the problem.

Notation	Definition
T	Time slots, $T = \{1, 2, 3, \ldots \ldots, n-1, n\}$
P	Expected number of conference participants $\{p_1, p_2, p_2, \ \ldots \ \ldots \ , p_{n-1}, p_n\}$ such that during time slot i, there is a maximum of p_i participants for $1 \leq i \leq n$.
δ	The time lag, stipulated in conferencing IaaSs' Service Level Agreement (SLA), for meeting scaling request.
R	A scaling schedule $\{r_1, r_2, \ldots \ldots, r_{n-\delta}\}$, where $r_i = \begin{cases} m > 0, & \text{if scaling is requested for } m \text{ participants during time slot } i \\ 0, & \text{otherwise} \end{cases}$
S	Allocated conference sizes, $S = \{s_1, s_2, s_3, \ldots \ldots, s_{n-1}, s_n\}$, where $s_i$is the allocated size during time slot i. The values of s_i can vary depending on scaling schedule R. Here, s_1 is the initial conference size.

Table 5-2: List of notations

Constraints:

The first constraint is that, due to time lag δ at conferencing IaaS, two consecutive scaling requests from conferencing PaaS must be separated by δ. In a scaling schedule $R = \{r_1, r_2, r_3, \ldots \ldots, r_{n-\delta}\}$, if one scaling request is made during time slot i and the next during k, that is, if $r_i > 0, r_{i+1} = 0, \ldots \ldots, r_{k-1} = 0, r_k > 0$, then $k - i \geq \delta \; \forall \; 1 \leq i < k \leq n - \delta$.

The second constraint is that, the allocated conference size during time slot $i, s_i \in S$, must accommodate the predicted number of participants $p_i \in P$, that is, $s_i \geq p_i \; \forall \; 1 \leq i \leq n$.

Given a scaling schedule $R = \{r_1, r_2, r_3, \ldots \ldots, r_{n-\delta-1}\}$ and predicted number of participants $P = \{p_1, p_2, \ldots \ldots, p_n\}$, the values of $s_i \in S$ can be derived. If the first scaling request is made during time slot k, that is, if $r_1 = 0, \ldots \ldots, r_{k-1} = 0, r_k > 0$, then the allocated conference size, $s_j = \max \{p_j\} \; \forall \; 1 \leq j < k + \delta$.

If one scaling request is made during time slot i and the next one during k, that is, if $r_i > 0, r_{i+1} = 0, r_{i+2} = 0, \ldots \ldots, r_{k-1} = 0, r_k > 0$, then the allocated conference size from time slot $(i + \delta)$ to $(k + \delta - 1)$ will equal r_i. Therefore, $s_j = r_i \; \forall \; i + \delta \leq j < k + \delta$, where $r_i > 0, r_{i+1} = 0, r_{i+2} = 0, \ldots \ldots, r_{k-1} = 0, r_k > 0$.

Both cases of s_j can be summarized as follows:

$$s_j = \begin{cases} \max\{p_j\}, & \forall \; 1 \leq j < k + \delta, \text{when } r_1 = 0, \ldots \ldots, r_{k-1} = 0, r_k > 0 \\ r_i, & i + \delta \leq j < k + \delta, \text{when } r_i > 0, r_{i+1} = 0, r_{i+2} = 0, \ldots \ldots, r_{k-1} = 0, r_k > 0 \end{cases}$$

Objective:

The allocated conference sizes $S = \{s_1, s_2, \dots\dots, s_n\}$ for time slots $T = \{t_1, t_2, \dots\dots, t_n\}$ can be computed from a scaling schedule $R = \{r_1, r_2, r_3, \dots\dots, r_{n-\delta}\}$ and predicted number of participants $P = \{p_1, p_2, \dots\dots, p_n\}$. Waste of resources during time slot i is the difference between allocated conference size s_i and predicted number of participants p_i, that is, $(s_i - p_i)$. The objective is to compute an optimal scaling schedule, which minimizes the amount of wasted resources during the entire conference, i.e., the sum of differences between allocated conference size, $s_i \in S$ and predicted number of participants, $p_i \in P$ for $1 \leq i \leq n$, that is,

$$minimize\left(\sum_{i=1}^{n} s_i - p_i\right)$$

5.3 Problem Analysis

In order to determine the minimum amount of wasted resources for time slots from i to j, where $1 \leq i \leq j \leq n$, the amounts of wasted resources for all possible time ranges in-between time slot i and j (inclusive) are required. To determine amount of wasted resources, number of participants as well as allocated conference size are needed. Predicted number of participants *P* is an input to the problem. As for allocated conference size, it depends on an optimal scaling schedule, which needs to be computed. However, according to the second constraint, the allocated conference size for time slot i must be greater than or equal to predicted number of participants for that time slot. Therefore, required conference size for time slots from i to j must be greater than or equal to predicted numbers of participants $\{p_i, p_{i+1}, \dots\dots, p_{j-1}, p_j\}$.

Let $c[i,j]$ denote the required conference size for time slots from i to j. In order to satisfy the second constraint, which is $s_i \geq p_i \; \forall \; 1 \leq i \leq n$, required conference size is the maximum number of participants predicted from time slot i to j. Therefore, $c[i,j]$ can be computed using the following equation.

$$c[i,j] = max\,\{p_k\}\ \forall\, 1 \le i \le k \le j \le n \quad \dots\dots\dots (1)$$

Let $w[i,j]$ denote the amount of wasted resources from time slot i to j when one and only one scaling request accommodates predicted number of participants from time slot i to j. From Eq. (1), it is clear that the conference must be scaled to size $c[i,j]$. Thus the value of $w[i,j]$ can be computed as in Eq. (2).

$$w[i,j] = \sum_{k=i}^{j} (c[i,j] - p_k)\ \ \forall\, 1 \le i \le k \le j \le n \ \dots\dots\dots (2)$$

A trivial case is $i = j$ where $w[i,i] = 0$ because $c[i,i]$ equals p_i.

Let $m[i,j]$ denote minimum amount of wasted resources for time slots from i to j. This is the cost of optimal solution of the problem for time slots from i to j. Note that, $m[i,j]$ may result from multiple scaling requests. $m[i,j]$ may be achieved by a single scaling request that accommodates participants for all time slots from i to j. $m[i,j]$ may also be achieved by having one scaling request accommodating participants for time slots from i to k and later scaling request(s) accommodating participants for the remaining time slots from $k+1$ to j. Therefore, when the cost of an optimal solution to $m[i,j]$ involve more than one scaling requests, the following can be stated:

$$m[i,j] = w[i,k] + m[k+1,j] \quad \dots\dots\dots (3)$$

From Eq. (3), it is clear that in order to find $m[i,j]$, the range of time slots must be split between time slots k and $k+1$ for some integer k in the range $i + \delta - 1 \le k < j$. Finding minimum waste of resources for time slots from i to j involves finding minimum waste of resources for the time slots that are accommodated by subsequent scaling requests. The solution to the problem is essentially reduced to finding the solution to a smaller subproblem. Thus, the problem shows optimal substructure property.

Proof of the Problem's Optimal Substructure Property:

The optimal substructure property of the problem is proved as follows. Suppose, to achieve minimum waste of resources from time slot i to j, the time range is split between time slot k and $k+1$, meaning, this split leads to an optimal solution for $m[i,j]$. Then the solution to the subproblem $m[k+1,j]$ must be an optimal one. Because, if there were a solution to the subproblem that leads to less waste of resources, then solution to the original problem would have resulted into less waste of resources than the optimum. This contradicts the assumption that $m[i,j]$ is an optimal solution. Thus, the solution to the subproblem $m[k+1,j]$ must be an optimal solution.

Overlapping Subproblems Property:

Let k_1, k_2 and k_2 denote three of the many time slots in between the time slots i and j, that is, $i < k_1 < k_2 < k_3 < j$. This is illustrated in figure 5-1. Assume that the time slots i, k_1, k_2, k_3 and j are separated by at least δ, that is, $k_1 - i \geq \delta, k_2 - k_1 \geq \delta, k_3 - k_2 \geq \delta$. To find an optimal solution to $m[i,j]$, three possible splits around k_1, k_2 and k_3 time slots are possible, which lead to subproblems $m[k_1+1,j], m[k_2+1]$ and $m[k_3+1,j]$, respectively. While solving subproblem $m[k_1+1,j]$, two possible splits are around k_2 and k_3 as $k_1 < k_2 < k_3 < j$, which leads to subproblems $m[k_2+1,j], m[k_3+1,j]$. While solving subproblem $m[k_2+1,j]$, one possible split is around k_3 as $k_2 < k_3 < j$, which leads to subproblem $m[k_3+1,j]$. Thus, it is observed that the same subproblems, such as $m[k_2 + 1, j]$ and $m[k_3 + 1, j]$ are solved repeatedly. Therefore, this problem demonstrates the overlapping subproblems property.

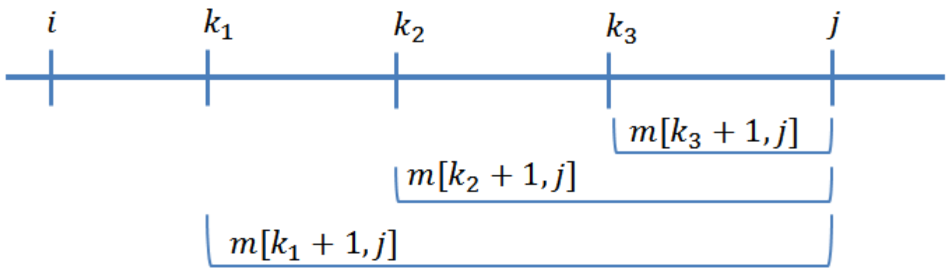


Figure 5-1: Overlapping subproblems

Proof of Overlapping Subproblems property:

The overlapping subproblems property can be proved as follows. We follow the scenario in figure 5-1. Finding optimal solutions to both $m[i, j]$ and $m[k_1 + 1, j]$ must lead to subproblem $m[k_2 + 1, j]$. If $m[k_2 + 1, j]$ were not a subproblem of $m[i, j]$ and $m[k_1 + 1, j]$, then $k_2 - i < \delta$ and $k_2 - k_1 < \delta$, respectively. This contradicts the assumptions that $k_2 - k_1 \geq \delta$. In addition to that, $k_1 - i \geq \delta$ and $k_2 > k_1$, then $k_2 - i \geq \delta$. Thus, $m[k_2 + 1]$ is a subproblem of both $m[i, j]$ and $m[k_1 + 1, j]$.

The problem thus shows two hallmark properties (optimal substructure and overlapping subproblems) of an optimization problem that can be solved by dynamic programming. In the next section, we provide an overview of dynamic programming. For the sake of continuity, we analyze the problem further in this section and provide the problem's recurrence relation below.

Recurrence Relation:

Now, in order to apply dynamic programming, the minimum waste of resources $m[i,j]$ is defined recursively as follows. If $i = j$, the problem is trivial. Similar to the trivial case of of Eq. (2), there is only one time slot to consider and hence minimum waste of resources is zero, that is, $m[i,i] = 0$ for $i = 1,2,\ldots,n$. When $i < j$, minimum waste of resources can be achieved by either one or multiple scaling requests. So, the minimum of the two waste of resources for these two cases is chosen. When one scaling request results in the minimum waste of resources, $m[i,j]$ equals $w[i,j]$ because that scaling request accommodates all time slots from i to j. When multiple scaling requests lead to the minimum waste of resources, Eq. (3) applies. However, Eq. (3) assumes that the value of k is known, when it is unknown. Number of possible values for k is $j - i - \delta + 1$, which are $i + \delta - 1, i + \delta, i + \delta + 1, \ldots \ldots, j - 1$. The optimal solution to $m[i,j]$ chooses one of these values of k.Thus, $m[i,j]$ can be defined recursively as follows:

$$m[i,j] = \begin{cases} 0, & \text{if } i = j \\ \min\limits_{i+\delta-1 \le k < j} \{w[i,j], w[i,k] + m[k+1,j]\}, & \text{if } i < j \end{cases} \quad \ldots\ldots\ldots (4)$$

The range of values for k is dictated by the first constraint of time lag.

Minimum waste of resources for all time slots T can be derived from Eq. (4) as follows:

$$m[1,n] = \begin{cases} 0, & \text{if } n = 1 \\ \min\limits_{\delta \le k < n} \{w[1,n], w[1,k] + m[k+1,n]\}, & \text{if } n > 1 \end{cases} \quad \ldots\ldots\ldots (5)$$

In our analysis of the problem, we have observed that the problem demonstrates two hallmark properties of dynamic programming, that is, optimal substructure and overlapping subproblems. We have proven the recurrence relation of the optimal substructure property. We have also defined the minimum waste of resources $m[i,j]$ recursively. In the next section, we design a dynamic programming algorithm to solve the problem where we memoize the solutions to subproblems and reach an optimal value of $m[1,n]$ from its subproblems. We also show how a scaling schedule $\{r_1, r_2, \dots \dots, r_{n-\delta}\}$ can be constructed from the values of $m[i,j]$.

5.4 Proposed Dynamic Programming Algorithm

In this sub-section, we first give a brief overview of dynamic programming. Then we provide a high-level view of the proposed dynamic programming algorithm for conference scaling. Next, detailed description of the algorithm and the analysis of its time complexity are presented.

5.4.1 A Brief Overview of Dynamic Programming

Dynamic programming [79] divides a problem into many smaller subproblems and reaches an optimal solution to the problem by combining the solutions to the subproblems. However, the subproblems may overlap, meaning the same subproblem may occur as a subproblem of two different larger subproblems. Therefore, instead of repeatedly solving the overlapping subproblems, dynamic programming solves each subproblem only once and stores the results in a table for future lookup.

Dynamic programming is typically used in optimization problems. The problems that can be solved by applying dynamic programming must have two characteristics. The first characteristic is optimal substructure property, meaning if a solution to a problem is optimal, then the solutions to its subproblems must be optimal. The second characteristic is overlapping subproblems, that is, subproblems share subsubproblems.

5.4.2 High Level View

Based on the equations derived in section 5.3, the following is a high-level view of the dynamic programming algorithm.

1. For all possible time ranges $[i, j]$ where $1 \leq i \leq j \leq n$, compute required conference sizes and store in a table c.

2. For all possible time ranges $[i, j]$ where $1 \leq i \leq j \leq n$, compute waste of resources if a time range is accommodated by only one scaling request. Store the results in a table w.

3. Now consider accommodating a time range with one or more scaling requests. If more than one scaling requests results in minimum waste of resources, there will be an optimal split of that time range. Using the tables c and w, compute minimum waste of resources for all possible time ranges. Store the optimal splits in table K and the minimum waste of resources in table m.

4. Using the table K that stores optimal splits, construct an optimal scaling schedule and compute initial conference size.

5.4.3 Detailed Algorithm Description

Compute Required Conference Sizes:

The procedure for computing required conference sizes for time slots from i to j, where $1 \leq i \leq j \leq n$, uses Eq. (1) in section 5.3. It is described below.

```
COMPUTE-REQUIRED-CONFERENCE-SIZES ($P,\ n$)
1.  let $c[1 \dots n, 1 \dots n]$ be a new table that stores required conference sizes for all possible time ranges
```

```
2.  for $i = 1$ to $n$
3.      $max = -\infty$
4.      for $j = i$ to $n$
5.          if $P[j] > max$
6.              $max = P[j]$
7.          $c[i,\ j] = max$
8.  return  $c$
```

Procedure COMPUTE-REQUIRED-CONFERENCE-SIZES takes predicted number of participants P and number of time slots n as input. It computes the required conference sizes for all possible time ranges, stores the results in a table and finally outputs the table.

Compute Waste of Resources:

The following procedure computes waste of resources for all possible time ranges when a single scaling request accommodates predicted numbers of participants for that time range, i.e., time slots from i to j, where $1 \le i \le j \le n$. It uses Eq. (2) in section 5.3.

```
COMPUTE-WASTE-OF-RESOURCES $(P,\ n,\ c)$

1.  let $w[1 \dots n, 1 \dots n]$ be a new table that stores waste of resources when a single scaling request
    accommodates the whole time range.
2.  for $i = 1$ to $n$
3.      $w[i,\ i] = 0$
4.      for $j = i + 1$ to $n$
5.          $w[i,\ j] = w[i,\ j-1] + (c[i,\ j] - c[i,\ j-1]) \times (j - i) + (c[i,\ j] - P[j])$
6.  return  $w$
```

Compute Minimum Waste of Resources:

The following procedure computes minimum waste of resources for all possible time ranges (considering both single and multiple scaling requests). It also computes optimal splits or divisions of time slots that lead to the minimum waste of resources, so that an optimal scaling schedule can be constructed later.

```
COMPUTE-MIN-WASTE-RESOURCES ($P, n, \delta, c, w$)

1.  let $m[1 \dots n, 1 \dots n]$ and $K[1 \dots n, 1 \dots n]$ be two new tables. $m$ will store minimum waste of
    resources for all possible time ranges, where a time range can be accommodated by multiple scaling
    requests. $K$ will store the next optimal split for all possible time ranges.
2.  for $i = 1$ to $n$
3.      for $j = i$ to $n$
4.          $m[i, j] = \infty$
5.          $K[i, j] = -1$
6.  $min\_resource\_waste$ = RECURSIVE-COMPUTE-MIN-WASTE-RESOURCES ($P, \delta, c, w, m, K, 1,$
    $n$)
7.  return $m$ and $K$
```

The following recursive algorithm uses Eq. (4), which is the recurrence relation of this conference scaling problem.

```
RECURSIVE-COMPUTE-MIN-WASTE-RESOURCES ($P, \delta, c, w, m, K, i, j$)

1.  if $m[i][j] \neq \infty$
2.      return $m[i][j]$
3.  else if $i = j$
4.      $m[i][i] = 0$
5.      $K[i][i] = i$
6.      return $m[i][j]$
7.  else
8.      $min = w[i][j]$
9.      $\min\_k = j$
10.     for $k = i + \delta - 1$ to $j - 1$
11.         $min2 = w[i][k]$ + RECURSIVE-COMPUTE-MIN-WASTE-RESOURCES ($P, \delta, c, w, m, K, k +$
    $1, j$)
12.         if $min2 < min$
13.             $min = min2$
14.             $\min\_k = k$
15.     $m[i][j] = min$
16.     $K[i][j] = \min\_k$
17.     return $m[i][j]$
```

Construct Optimal Schedule:

The following algorithm constructs an optimal schedule R. It uses a table which stores next optimal split of time slots for all possible time ranges.

CONSTRUCT-OPTIMAL-SCHEDULE ($n,\ \delta,\ K$)

```
1. $k = K[1][n]$
2. $initial\_size = c[1][k]$
3. let $R[1 \dots n]$ be a new array to store scaling schedule.
4. for $i = 1$ to $n$
5.     $R[i] = 0$
6. RECURSIVE-CONSTRUCT-OPTIMAL-SCHEDULE ($n,\ \delta,\ K,\ k+1,\ n$)
7. return $R$ and $initial\_size$
```

The following procedure recursively constructs scaling schedule from a given table K, which stores next optimal split of time slots for all possible time ranges.

RECURSIVE-CONSTRUCT-OPTIMAL-SCHEDULE ($n,\ \delta,\ K,\ i,\ j$)

```
1. $k = K[i][j]$
2. if $k = n$
3.     $R[i-\delta] = c[i][j]$
4. else
5.     $R[i-\delta] = c[i][k]$
6.     RECURSIVE-CONSTRUCT-OPTIMAL-SCHEDULE ($n,\ \delta,\ K,\ k+1,\ j$)
```

5.4.4 Time Complexity Analysis

Both COMPUTE-REQUIRED-CONFERENCE-SIZES procedure and COMPUTE-WASTE-OF-RESOURCES procedure have nested loops. The outer loop iterates at most n times in both procedures. The inner loop iterates at most $n-i+1$ times and $n-i$ times in COMPUTE-REQUIRED-CONFERENCE-SIZES procedure and COMPUTE-WASTE-OF-RESOURCES procedure, respectively. Therefore, the running time of both procedures is $O(n^2)$.

RECURSIVE-COMPUTE-MIN-WASTE-RESOURCES procedure solves each subproblem only once. Because it returns result immediately when the subproblem is already solved and $m[i][j]$ stores the result (lines 1 to 2). Lines 3 to 6 deals with the case when the subproblem is trivial because $i = j$, that is, minimum waste of resources is zero when only one time slot is considered. Apart from these two cases, it iterates for $(j-i-\delta+1)$ times in order to find an optimal waste of resources and the associated value of k. To solve the problem of size n, i.e. when $i = 1, j = n$, it iterates $(n - \delta)$ times (line 10), making recursive calls for subproblems of sizes $n - \delta, n - \delta - 1, n - \delta - 2, \dots\dots, 3, 2, 1$. Similarly, to solve a subproblem of size $n - \delta$, i.e. when $i = \delta + 1, j = n$, the procedure iterates $n - 2\delta$ times (line 10) for subproblems of sizes $n - 2\delta, n - 2\delta - 1, n - 2\delta - 3, \dots\dots, 3, 2, 1$.Therefore, the total number of iterations (lines 10 to 14) over all recursive calls leads to an arithmetic series $(n - \delta) + (n - 2\delta) + (n - 3\delta) + \cdots + x$, where x is n modulo δ. The series gives a total of $O(n^2)$ iterations. Hence, the running time of RECURSIVE-COMPUTE-MIN-WASTE-RESOURCES procedure is $\mathrm{O}(n^2)$.

COMPUTE-MIN-WASTE-RESOURCES procedure has nested loops in lines 3 and 4, resulting in $O(n^2)$ running time. Call to RECURSIVE-COMPUTE-MIN-WASTE-RESOURCES procedure also adds $O(n^2)$ running time. Therefore, the running time of COMPUTE-MIN-WASTE-RESOURCES procedure is $O(n^2)$.

RECURSIVE-CONSTRUCT-OPTIMAL-SCHEDULE procedure calls recursively in line 6. In each call, the subproblem size is reduced by at least δ, because the next earliest scaling after this can be requested after δ time slots. Therefore, RECURSIVE-CONSTRUCT-OPTIMAL-SCHEDULE procedure is called at most $(\frac{n}{\delta} + 1)$ times. Thus, the running time of RECURSIVE-CONSTRUCT-OPTIMAL-SCHEDULE procedure is $O(n)$.

CONSTRUCT-OPTIMAL-SCHEDULE procedure has a loop in line 4, which contributes to $O(n)$ running time. The single call to RECURSIVE-CONSTRUCT-OPTIMAL-SCHEDULE procedure adds $O(n)$ running time. Therefore, the running time of CONSTRUCT-OPTIMAL-SCHEDULE procedure is $O(n)$.

The procedures used in the algorithm have running time at most $O(n^2)$ and there is no iteration of the high-level procedures used in the algorithm. Therefore, the running time of the algorithm is $O(n^2)$.

5.5 An Alternative Greedy Algorithm

The time complexity of the proposed dynamic programming algorithm is $O(n^2)$. This algorithm produces an optimal solution. However, for comparison purpose, we develop an alternative greedy algorithm which is faster but produces a sub-optimal solution. This section starts with a brief overview of greedy algorithm. Then the greedy heuristic used is motivated with an example. Finally, a high-level view of the greedy algorithm is presented. The appendix to this thesis contains the detailed description and time complexity analysis of the proposed greedy algorithm.

5.5.1 A Brief Overview of Greedy Algorithm

Greedy algorithm [79] is usually used to solve optimization problems. While solving a problem, it always makes a choice that seems best at the moment. In other words, it hopes to reach globally optimal solution by making locally optimal choices. However, this heuristic strategy does not always lead to an optimal solution.

While making locally optimal choices, a greedy algorithm may depend on choices made so far. However, it does not depend on any future choices. This is in contrast to dynamic programming, which solves all subproblems before making a choice. Greedy algorithm makes a choice and proceeds to the next subproblem. A dynamic programming algorithm solves a problem in bottom-up manner. On the other hand, a greedy algorithm usually advances in a top-down fashion while making greedy choices one by one.

5.5.2 Motivation for the Greedy Heuristic Used

Given the problem statement in the beginning of this chapter, a naïve approach of scaling a conference in terms of number of participants is to request scaling after every δ time slots, where δ is the time lag. Consider the following predicted number of participants. Figure 5-2 depicts the change in number of participants over time.

$P = \{\dots, 10, 25, 35, 40, 60, 70, 75, 85, 95, 110, 90, 70, 45, 40, 30, 20, 15, \dots\}$.

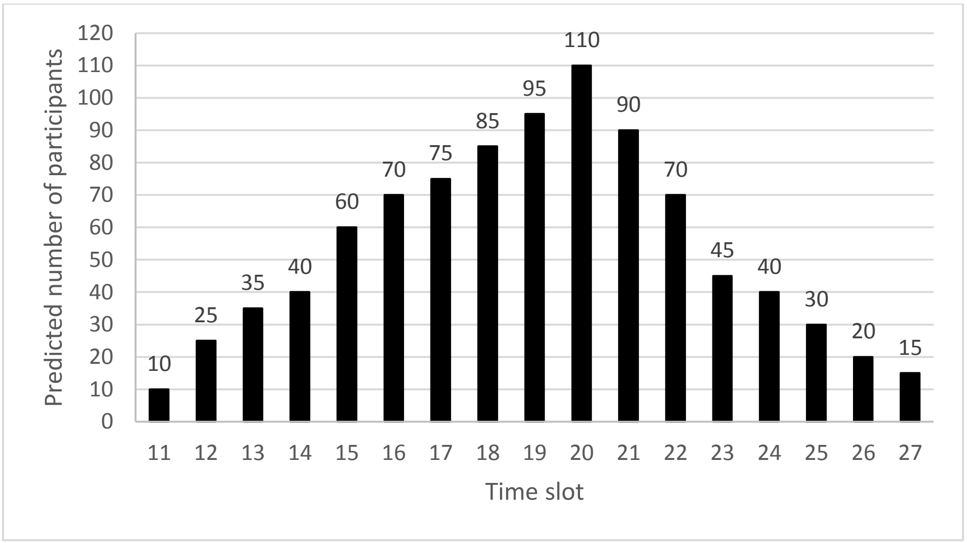
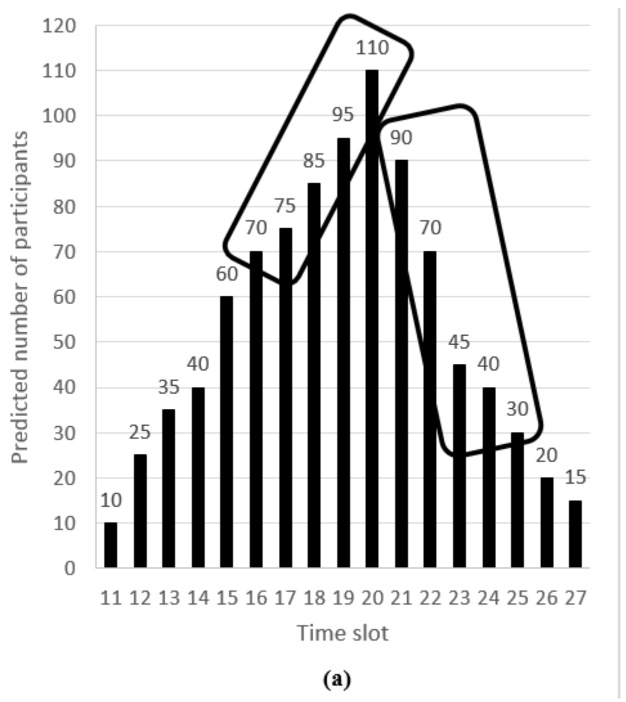


Figure 5-2: An example of predicted number of participants

In this example of P, the number of participants first increases up to 110, then starts decreasing. It is assumed that $\delta = 5$. The naïve approach does not take the changes in P over time into account. As a result of this, as illustrated in figure 5-3(a), it is possible to send scaling request (scaling up to 110) to accommodate time slots from 16 to 20. The next scaling request (scaling down to 90) is made to accommodate time slots from 21 to 25. Waste of resources for the first scaling request is $\sum_{i=16}^{20} 110 - P_i$ = 115. Waste of resources for the second scaling request is $\sum_{i=21}^{25} 90 - P_i$ = 175. Thus, total waste of resources in naïve approach is 290 (115+175). In this approach, the required number of participants for both scaling requests are chosen from two of the time slots (20 and 21) near the local maxima.

It is possible to optimize waste of resources by choosing, instead of two, only one time slot near the maxima, such that the maxima and its nearby time slots are accommodated. Figure 5-3(b) illustrates this case. Keeping the maxima in the middle, time slots from 18 to 22 (scaling up to 110) are accommodated by one scaling request $s2$. The previous scaling request $s1$ (scaling up to 75) accommodates time slots up to 17. The next one $s3$ (scaling down to 45) covers time slots from 23 to 27. To compare with waste of resources for the naïve approach, we shall calculate waste for time slots from 16 to 25. Waste of resources from time slots 16 to 17 by $s1$ is $\sum_{i=16}^{17} 75 - P_i$ = 5, by $s2$ is $\sum_{i=18}^{22} 110 - P_i$ = 100, by s3 is $\sum_{i=23}^{25} 45 - P_i$ = 20. Therefore, the total waste of resources is 145 (5+120+20), which is less than the naïve scaling approach (290). The same observation applies to local minima. This local extrema optimization will lead to a better global optimization than the naïve scaling approach when the number of participants increases and decreases over time.



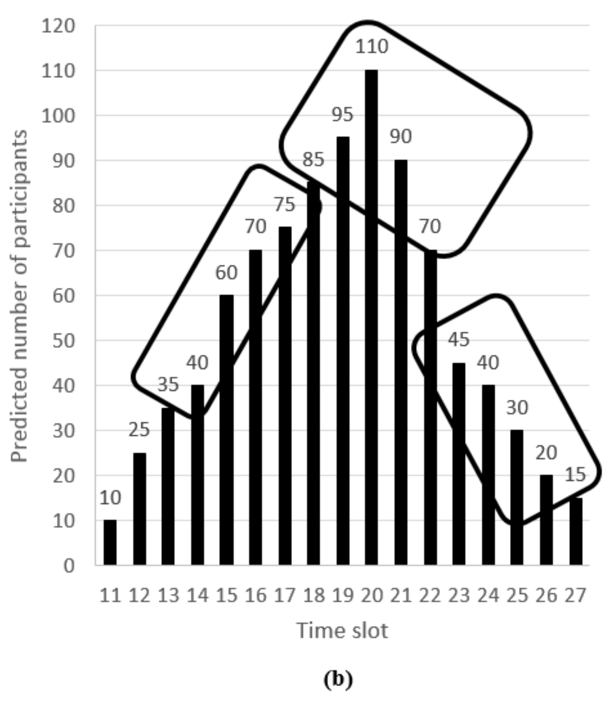


Figure 5-3: (a) Scaling with naive approach (b) Scaling with extrema optimization

Using this local extrema optimization heuristic, a greedy algorithm can be designed.

5.5.3 High Level View

Following is the high-level view of the proposed greedy algorithm:

HIGH-LEVEL-GREEDY-OPTIMIZATION(P, n, δ)

1. Divide P, predicted numbers of participants into upward and downward slopes. Upward slope consists of consecutive time slots with non-decreasing number of participants. Downward slope consists of consecutive time slots with decreasing number of participants.

2. Schedule scaling requests around local maxima (upward slope on the left and downward slope on the right) as well as local minima (downward slope on the left and upward slope on the right).

3. Schedule scaling requests for the time slots which are not accommodated by step 2, following the naïve scaling approach.

An edge case is that predicted number of participants drops sharply after reaching local maxima (or rise sharply after local minima). In that case, keeping the local maxima in the middle of accommodated time slots results in more waste of resources than the naïve approach. This is demonstrated in figure 5-4. Therefore, in step 2, the waste of resources needs to be checked before deciding to schedule around local extrema.

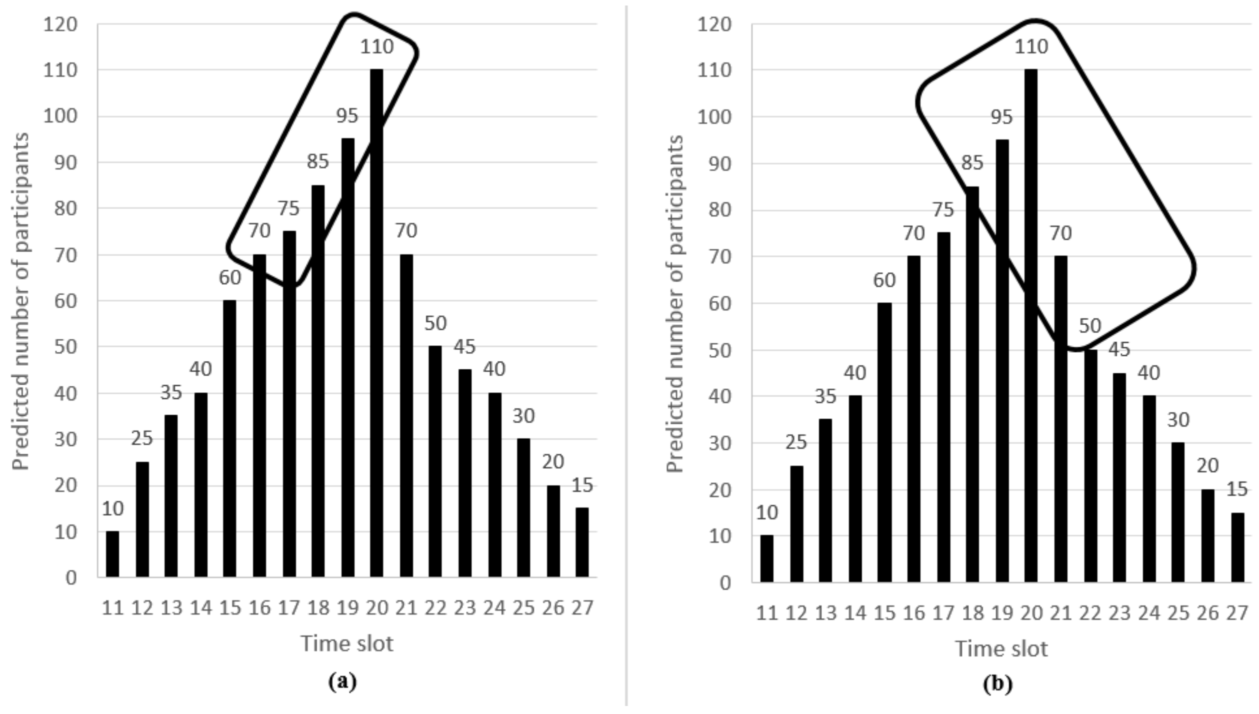


Figure 5-4: (a) Waste of resources is 115 (b) Waste of resources is 140

5.6 Chapter Summary

In this chapter, we address the problem of conferencing scaling. We formally state the problem and analyze it. Then we present two algorithms to minimize waste of resources in terms of number of participants. One is a dynamic programming (DP) algorithm and it provides an optimal scaling schedule. We discuss the algorithm in details and analyze its time complexity. The other is a greedy algorithm. It provides a suboptimal schedule but is faster. We present a high-level view of the greedy algorithm and leave the details in appendix. According to the requirements on conference scaling algorithm, the proposed algorithms are practical when they can produce output within a second for the given input size. The measurements of the proposed conference scaling algorithms, discussed in the next chapter, show that they perform well below one second for a big enough input size of 100 time slots.

Chapter 6

6. Validation: Prototype and Evaluation

In chapter 4, we have proposed a general architecture of conferencing PaaS for multimedia conferencing service provisioning. In chapter 5, we propose algorithms to scale conferences in an elastic manner while minimizing waste of resources in terms of number of participants. In this chapter, we discuss the design of software architecture of conferencing PaaS, a prototype to validate the software architecture, the performance measurements of the prototype as well as the proposed algorithms.

This chapter first presents the overall software architecture of conferencing PaaS. This is followed by a discussion of proof-of-concept prototype. Next, the prototype setup and performance measurements are discussed. After that, performance measurements of the proposed conference scaling algorithm are described. Lastly, the chapter summary is presented.

6.1 Overall Software Architecture

Figure 6-1 shows the overall software architecture that we propose for conferencing service provisioning. This software architecture is derived from the general architecture (discussed in chapter 4) by breaking down the components of general architecture into smaller software components. In the next sub-sections, the software components of each high-level component in general architecture are described. This is followed by a sub-section that illustrates the interactions among the software components. *Conferencing IaaS Handler* and *Substrate Info Repository* components remain the same as in general architecture. Therefore, their discussion is omitted.

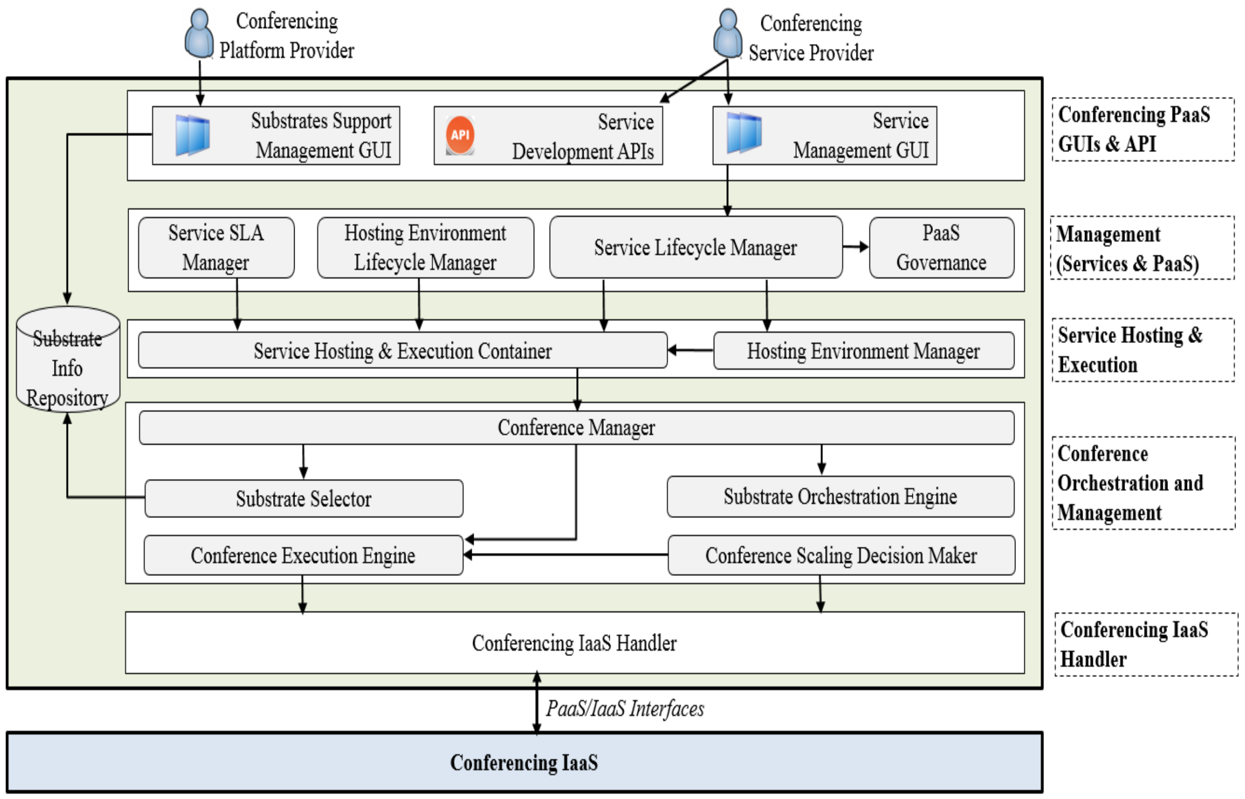

Conferencing Platform Provider
Conferencing Service Provider
Substrates Support Management GUI
Service Development APIs
Service Management GUI
Conferencing PaaS GUIs & API
Service SLA Manager
Hosting Environment Lifecycle Manager
Service Lifecycle Manager
PaaS Governance
Management (Services & PaaS)
Substrate Info Repository
Service Hosting & Execution Container
Hosting Environment Manager
Service Hosting & Execution
Conference Manager
Substrate Selector
Substrate Orchestration Engine
Conference Execution Engine
Conference Scaling Decision Maker
Conference Orchestration and Management
Conferencing IaaS Handler
Conferencing IaaS Handler
PaaS/IaaS Interfaces
Conferencing IaaS

Figure 6-1: Software architecture of conferencing PaaS

6.1.1 Conferencing PaaS GUIs and APIs

The components include *Service Development APIs*, *Service Management GUI* and *Substrates Support Management GUI*. The components *Service Development APIs* and *Substrates Support Management GUI* are extensions to typical PaaS software architecture.

Conferencing platform providers use *Substrates Support Management GUI* to manage (e.g., add, remove, update) information of the substrates that they have subscribed to. Conferencing service providers use *Service Development APIs* and *Service Management GUI*. Conferencing PaaS provides the service providers with *Service Development APIs* as a programming library (e.g., JAR file in Java and NPM module in JavaScript). Conferencing service providers, who have programming expertise, can use this library to easily develop conferencing services. For service deployment and management, conferencing service providers use *Service Management GUI*. The GUI can be as simple as a command line interface (CLI).

6.1.2 Management (Services & PaaS)

The components are *Service Lifecycle Manager*, *Hosting Environment Lifecycle Manager*, *Service SLA Manager* and *PaaS Governance*. These are typical management components in conventional PaaS architectures. *Service Lifecycle Manager* receives requests from Service Management GUI and takes actions based on the type of the received requests. It manages different lifecycle events such as conferencing service deployment, execution and stop. *Hosting Environment Lifecycle Manager* is responsible for managing PaaS resources (e.g., runtime, DBMS instance) used to host services. It also monitors and controls lifecycle of different PaaS components. *Service SLA Manager* monitors the services' performances, compares them with the stipulated SLA of conferencing PaaS and takes actions (e.g., scaling up service instances) as necessary. *PaaS Governance* relates to user authentication and authorization, billing etc.

6.1.3 Service Hosting & Execution

The components are *Service Hosting & Execution Container* and *Hosting Environment Manager*. Similar to software components in *Management (Services & PaaS)*, these two components are also typical of traditional PaaS software architectures. The deployed services are stored and executed in *Service Hosting & Execution Container*. *Hosting Environment Manager* is responsible for preparing environment for hosting conferencing services.

6.1.4 Conference Orchestration and Management

The components include *Substrate Selector*, *Substrate Orchestration Engine*, *Conference Execution Engine*, *Conference Scaling Decision Maker* and *Conference Manager*. All of these software components are novel to the proposed software architecture. *Substrate Selector* chooses the most suitable conferencing IaaS, given the substrate requirements. *Substrate Orchestration Engine* composes the selected substrates into a full-fledged conference. *Conference Execution Engine* hosts the conferences. *Conference Scaling Decision Maker* monitors running conferences and requests scaling when needed. *Conference Manager* receives requests from northbound component and coordinates other subcomponents to serve the requests.

6.1.5 Operational Procedures

Based on the proposed software architecture, two procedures for a dial-in audio conference are illustrated in this sub-section. One procedure is conferencing service development and deployment. The other procedure pertains to conferencing service execution and only conference creation is illustrated for brevity.

Figure 6-2 shows interactions of software components for conferencing service development and deployment. Conferencing service provider uses Conferencing Service Development APIs to develop the service. Once the service is developed, service provider deploys the service in conferencing PaaS.

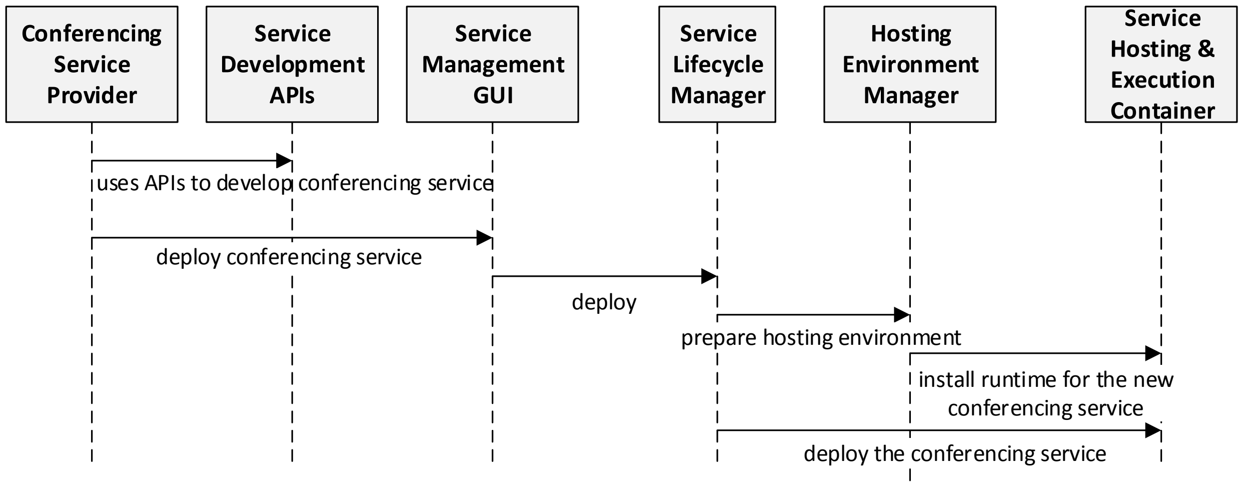
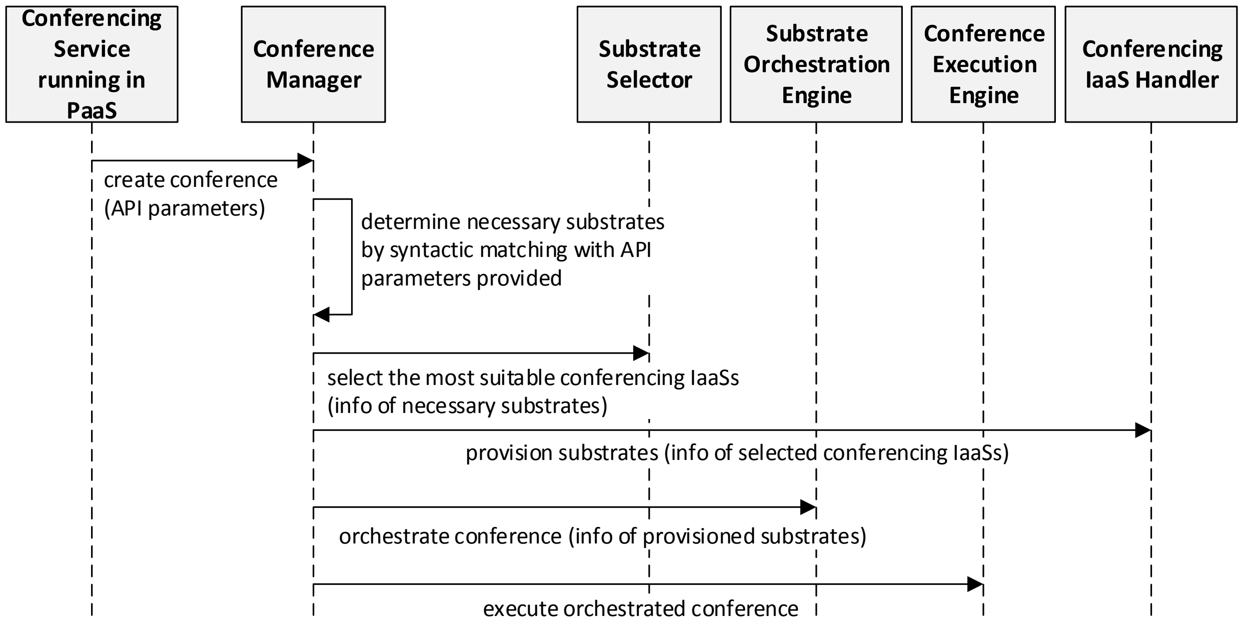

Conferencing Service Provider
Service Development APIs
Service Management GUI
Service Lifecycle Manager
Hosting Environment Manager
Service Hosting & Execution Container
uses APIs to develop conferencing service
deploy conferencing service
deploy
prepare hosting environment
install runtime for the new conferencing service
deploy the conferencing service

Figure 6-2: Interactions of software components for conferencing service development and deployment

After the dial-in audio conferencing service is started, it can receive requests from the conferencing applications such as game applications. Figure 6-3 illustrates the interactions for creating a conference when the service receives such a request.



Conferencing Service running in PaaS
Conference Manager
Substrate Selector
Substrate Orchestration Engine
Conference Execution Engine
Conferencing IaaS Handler
create conference (API parameters)
determine necessary substrates by syntactic matching with API parameters provided
select the most suitable conferencing IaaSs (info of necessary substrates)
provision substrates (info of selected conferencing IaaSs)
orchestrate conference (info of provisioned substrates)
execute orchestrated conference

Figure 6-3: Interactions of software components for creating a conference

6.2 Prototype

This subsection first presents the implemented scenario. The implementation scope and the prototype architecture are then discussed. Next, the software tools used for prototype implementation are described briefly.

6.2.1 Implemented Scenario

The prototype implements subset of both scenarios discussed in chapter 3 – (1) conferencing service development and deployment and (2) conferencing service execution. The implemented scenario includes a service provider offering dial-in audio conferencing service and a game application consuming that service. The scenario also includes the conferencing PaaS and two conferencing IaaSs – both providing dial-in signaling and audio mixer substrates. While developing the game application, the developer uses dial-in audio conferencing SaaS API provided by the conferencing service provider. The service provider uses a conferencing PaaS to develop and deploy the dial-in audio conferencing service. During execution, the service receives request from the game application to create and manage conferences. The conferencing PaaS provisions substrates from the two conferencing IaaSs. The subscription to conferencing IaaSs by the conferencing PaaS is done offline. Two use-cases are considered. In one use-case, conferencing PaaS selects substrates from the same IaaS. In the other use-case, substrates are selected from different IaaSs.

6.2.2 Implementation Scope

A subset of the components from the proposed software architecture is implemented in the prototype. In *Conferencing PaaS GUIs and APIs*, a subset of APIs from the proposed *Conferencing Service Development APIs*, which is needed to perform measurements, is implemented. *Service Management GUI* is also implemented to deploy and start the service. *Substrates Support Management GUI* is not implemented as we assume that conferencing PaaS subscribes to two conferencing IaaSs for dial-in signaling and audio mixer substrates. In *Management (Services & PaaS)*, all components except *Service SLA Manager* are implemented to host and execute the service. *Service SLA Manager* is not implemented as the prototype scenario considered do not require this component.

In *Conference Orchestration & Management* layer, *Conference Manager*, *Substrate Orchestration Engine* and *Conference Execution Engine* are implemented to create, host and execute conferences. *Substrate Selector* is not implemented. Because, in one use-case, we assume that conferencing PaaS selects both dial-in signaling and audio mixer substrates from the same IaaS. In the other use-case, conferencing PaaS is assumed to select one IaaS for dial-in signaling substrate and the other IaaS for audio mixer substrate. *Conference Scaling Decision Maker* component is not implemented as the prototype scenario does not cover runtime scaling of conferences. However, the proposed conference scaling algorithm is evaluated with different test cases and its performance measurements are discussed later in this chapter.

Conferencing IaaS Handler implements a subset of Conferencing PaaS/IaaS interfaces that are necessary for realizing the prototype scenario.

For proof-of-concept prototype, the REST requests from the game application to the dial-in audio conferencing service to create conferences are simulated by open-source REST clients.

A colleague in Telecommunications Service Engineering (TSE) Lab, who is working on conferencing IaaS, has implemented a stripped-down version of conferencing IaaS. This IaaS implementation is used in the prototype.

6.2.3 Prototype Description

Figure 6-4 shows the prototype architecture. Cloud Foundry PaaS [32] is extended to implement the conferencing PaaS prototype. Cloud Foundry provides the implementation of typical PaaS components. The components which are specific to conferencing PaaS are implemented using other open-source libraries and frameworks.

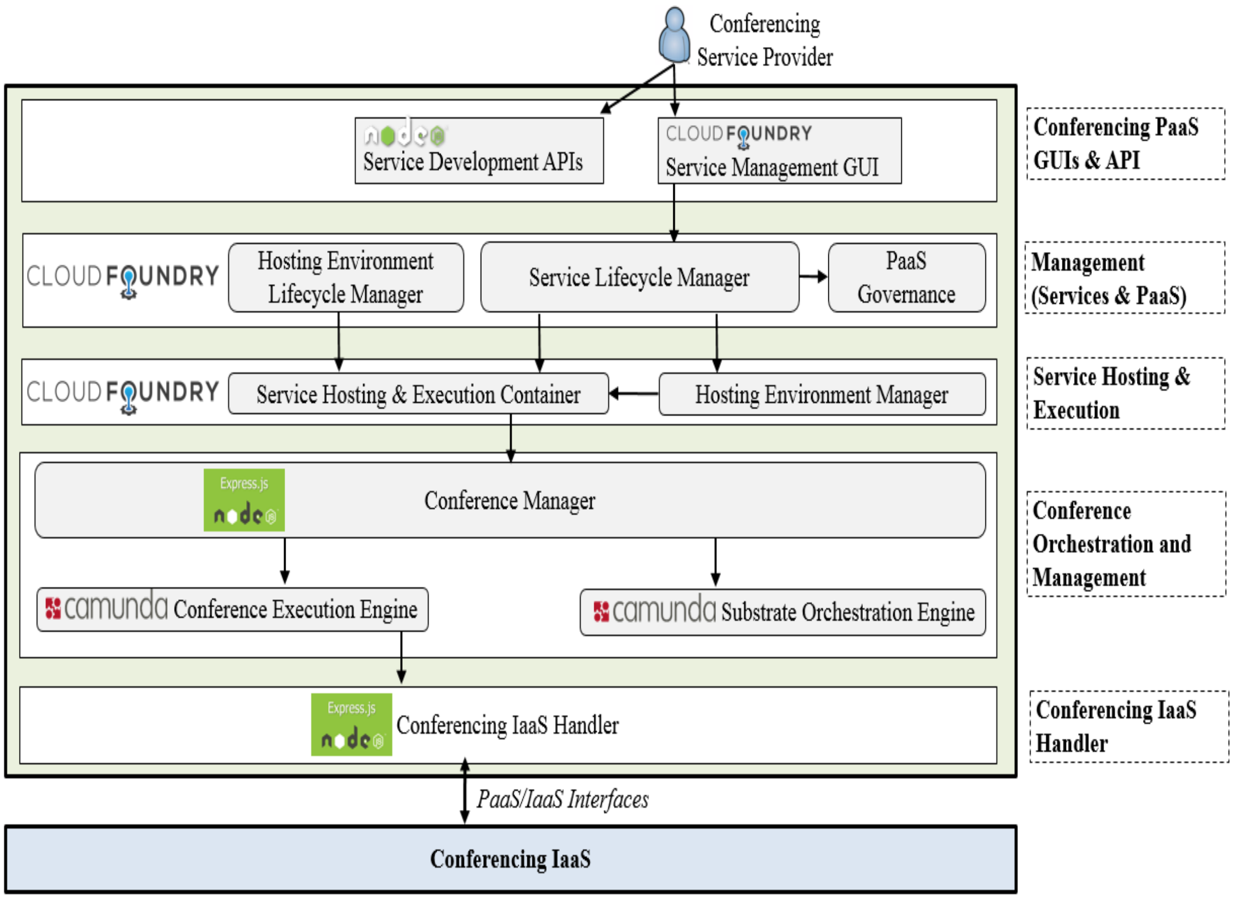

Conferencing Service Provider
Service Development APIs
Service Management GUI
Conferencing PaaS GUIs & API
Hosting Environment Lifecycle Manager
Service Lifecycle Manager
PaaS Governance
Management (Services & PaaS)
Service Hosting & Execution Container
Hosting Environment Manager
Service Hosting & Execution
Conference Manager
Conference Execution Engine
Substrate Orchestration Engine
Conference Orchestration and Management
Conferencing IaaS Handler
Conferencing IaaS Handler
PaaS/IaaS Interfaces
Conferencing IaaS

Figure 6-4: Prototype architecture of conferencing PaaS

Service Development APIs are implemented and exposed as a Node.js [80] module. Conferencing service provider imports this module and uses the APIs to develop conferencing services. *Service Management GUI* is reused from Cloud Foundry. The components of *Management (Services & PaaS)* and *Service Hosting & Execution layers* are also reused from Cloud Foundry. The conferencing service is deployed and executed in Cloud Foundry.

For *Substrate Orchestration Engine* and *Conference Execution Engine* components, open-source Camunda tool [81] is reused. In this prototype, the platform provider orchestrates the conferences as a BPMN workflow and stores in a workflow repository of *Substrate Orchestration Engine*. Upon receiving request to create a conference, appropriate workflow is selected based on the *createConference* API parameters. Since we use only dial-in audio conference in the prototype, only one workflow was created. The workflow is configured with the selected conferencing IaaSs and deployed in *Conference Execution Engine*.

Conference Manager and *Conferencing IaaS Handler* are implemented as REST servers using Express.js framework [82]. Advanced REST Client [83] is used to simulate conferencing SaaS API invocations by the game.

6.2.4 Software Tools

This section briefly discusses the software tools used in prototype implementation.

6.2.4.1 Cloud Foundry

Cloud Foundry [32] is one of the most popular free and open-source Platform-as-a-Service. We reuse and extend it to implement the conferencing PaaS prototype. Of many languages and frameworks that it supports for development, we use JavaScript language and Node.js runtime [80]. For deployment, Cloud Foundry provides a command line interface named CF CLI [84] that we reuse for service deployment. The software components in *Management (Services & PaaS)* and *Service Hosting & Execution* layers, which we implement in conferencing PaaS prototype, are reused from different Cloud Foundry components.

6.2.4.2 Camunda

Camunda [81] is a free and open-source implementation of BPMN engine. The Business Process Model and Notation (BPMN) [85] is the de-facto standard for graphical representation of the business processes or workflows. Moreover, BPMN supports orchestration of RESTful web services with a concept of service task. In our prototype, the workflow is created graphically using Camunda BPMN Modeler and then stored in a workflow repository. During conference orchestration, the workflow is instantiated, configured with the selected IaaSs and then executed in *Conference Execution Container*. This container is implemented by reusing Camunda BPMN Engine, a process execution engine.

6.2.4.3 Node.js

Node.js [80] is a widely used free and open-source cross-platform JavaScript runtime. It adopts an event-driven, asynchronous I/O approach which makes it lightweight. It is used to develop server-side applications in JavaScript language. There are a lot of popular libraries and frameworks based on Node.js runtime. One of them is Express.js [82]web application framework. It is very lightweight and can be used to implement REST servers as well as clients. In conferencing PaaS prototype, Express.js is used to develop the dial-in audio conferencing service. It is also used to implement *Conference Manager* component in *Conference Orchestration & Management* layer and *Conferencing IaaS Handler* component.

6.2.4.4 OpenStack

OpenStack is a collection of open source software projects that cloud providers can use to setup and run their infrastructure. It has a community with researchers, developers and enterprises, with a common goal to create simple, scalable and feature-rich infrastructure [86]. OpenStack provides services such as compute, object storage and block storage. It also has identify service and VM image service. It provides a graphical dashboard for managing virtual machines and networks.

6.2.4.5 SAVI testbed

Smart Applications on Virtual Network (SAVI) [87] is collaboration among Canadian industry, academia, research and education networks. Its goal is to investigate key elements of future application platforms. The SAVI testbed provides flexible, virtualized converged infrastructure to support experimental research. This testbed is implemented using OpenStack. The conferencing PaaS prototype is deployed on SAVI testbed.

6.2.4.5 Additional Software Tools

Advanced Rest Client [83] is a graphical tool used to test REST APIs. It is an extension of Google's chrome browser. In conferencing PaaS, the game application's invocations of the conferencing service SaaS APIs are simulated with Advanced REST client. CF NISE installer [88] is a tool to install Cloud Foundry easily on a single machine. Since conferencing PaaS prototype is implemented by extending Cloud Foundry, the Cloud Foundry components are installed on a SAVI testbed VM using CF NISE installer.

6.3 Prototype Setup and Performance Measurements

This sub-section starts with a short description of the prototype setup. It then discusses the performance metrics and the results.

6.3.1 Prototype Setup

The conferencing PaaS prototype along with conferencing IaaSs are deployed on SAVI testbed. Conferencing PaaS prototype is deployed on two VMs. One VM hosts the Cloud Foundry instance. The other VM hosts the conferencing PaaS-specific components (e.g., *Conference*

Manager, *Conferencing IaaS Handler*). Each of these two VMs have 8 GB RAM, 4 vCPUs, 80 GB storage and runs Ubuntu 14.04 operation system.

The conferencing IaaSs are hosted on separate VMs. The IaaSs provision the substrates dynamically on machines with 4 GB RAM and two vCPUs running Ubuntu 14.04 LTS.

6.3.2 Performance Measurements

In this section, we first describe the performance comparison scenarios. Next, performance metrics and the results obtained from the conferencing PaaS prototype are presented. It is noteworthy that in the prototype setup, the conferencing PaaS and conferencing IaaSs are in the same SAVI testbed network. In a real world scenario, they can be spread across different geographical locations. This will add external network latency to the measurements. However, as discussed later in this sub-section, the observations comparing between different scenarios, are not affected.

6.3.2.1 Comparison Scenarios

Three performance comparison scenarios are considered. Two of them concern cloud-based conferencing, where conferencing PaaS is leveraged.

i) **Non-cloud conferencing (NCC):** Resources are allocated beforehand in this scenario. Therefore, there may always be some idle and unutilized resources.

ii) **Cloud single IaaS provider (CSIP):** Conferencing PaaS selects the required substrates for a conference from the same IaaS. It is assumed that the conferencing IaaSs host the substrates on a single VM in this scenario.

iii) **Cloud multiple IaaS provider (CMIP):** Conferencing PaaS chooses substrates from different IaaSs. Since substrates are from different IaaSs, they are hosted on separate VMs.

6.3.2.2 Performance metrics

The following three metrics are considered. These metrics help validate that the proposed conferencing PaaS architecture meets the requirement of QoS.

i) **Conference start time:** This is the time required to get a conference ready upon the receipt of a request. It is calculated from the time conferencing service receives a request to create a conference to the time it receives a response.

ii) **Participant joining time:** This is the time required to add a participant to a running conference.

iii) **Resource allocation:** This is the total amount of allocated resources, such as RAM and CPU, to accommodate all participants. This metric pertains to only cloud-based scenarios as resources are allocated upfront in non-cloud scenario. We consider RAM to compare resource allocation.

The above performance metrics include response time or delay of two conference runtime operations – conference start time and participant joining time. Since cloud-based conferencing has virtualization overhead as well as notification overhead between cloud layers, it is possible that delay in cloud-based scenarios is higher than that in non-cloud conferencing. Therefore, it is necessary to measure resource allocation (the third metric) to see the benefits of resource efficiency provided by cloud-based conferencing.

6.3.2.3 Performance Results

Figure 6-5 shows the comparison of conference start times. NCC takes the least time to start a new conference because it does not have virtualization overhead. Cloud-based scenarios (CSIP and CMIP) take longer because VMs need to be instantiated for substrates. Since substrates need to connect over network in CMIP, it takes more time than in CSIP.

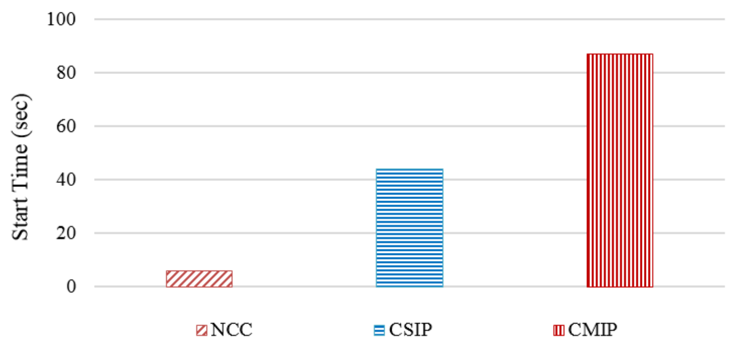


Figure 6-5: Average conference start time

Comparison of participant joining time is depicted in figure 6-6. Participant joining time is the least in NCC. Cloud-based scenarios take more time because of the notification overhead between IaaSs, PaaS and the game server. When a new participant joins the conference, conferencing IaaS notifies conferencing PaaS, which forwards the notification to the game server. However, this is a one-time operation for a participant and does not contribute to the participant's communication delay. Moreover, based on International Telecommunication Union (ITU) standards [89], this time is acceptable as long as it is below 400 msec. Participant joining time of the two cloud-based scenarios are close as IaaSs can notify PaaS in parallel. This shows that the proposed architecture satisfies the QoS requirement.

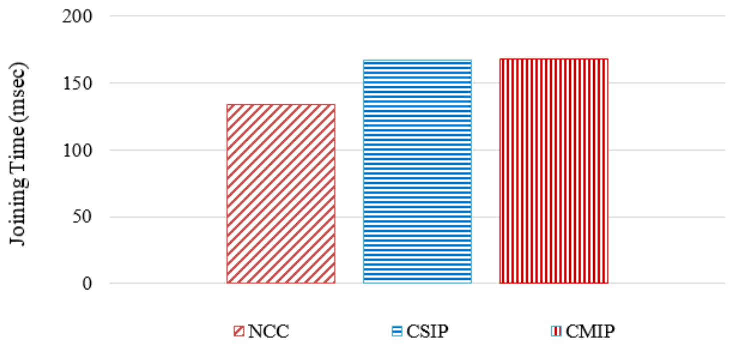


Figure 6-6: Average participant joining time

Although in cloud-based scenarios, conference start time and participant joining time are more than those in NCC, it helps to achieve resource efficiency and reduce costs. Figure 6-7 shows the allocated amount of RAM for a conference with between 1 and 3000 participants. To simulate conference scaling, conference size is increased by 200 participants every 10 minutes. The results are based on the observed resource usage per participant. IaaSs are assumed to scale up and out VMs while maintaining QoS requirements. In NCC, there are always some idle and non-utilized resources because of upfront resource provisioning. Hence, it is not shown in the figure. CSIP scales better than CMIP (i.e. allocates less resources) for smaller conferences whereas CMIP wins for bigger conferences, because in CMIP, substrates are hosted on separate VMs as they are chosen from different IaaSs. For smaller conferences, it leads to more VMs and more non-utilizable resources (e.g., resources consumed by operating system) than in CSIP. However, with the increase of conference size, CMIP achieves better scalability because of the less VMs and more utilizable resources than in CSIP.

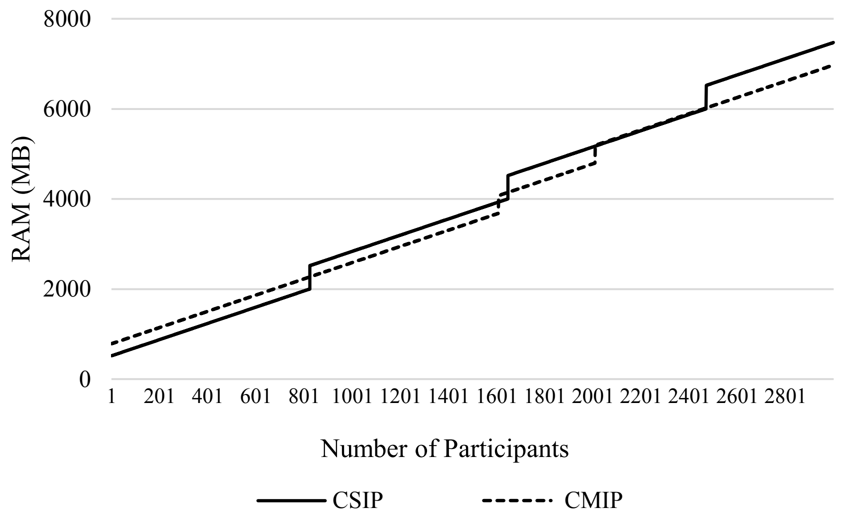


Figure 6-7: Resource Allocation Evaluation

6.4 Conference Scaling Algorithm and Performance Measurements

This section starts with a discussion of implementation of conference scaling algorithms (Dynamic Programming algorithm and Greedy algorithm). Next, the performance comparison metrics and the results are presented.

6.4.1 Algorithm Implementation and Test Sets

The conference scaling algorithms are implemented in C++. A few C++ 11 standard libraries, such as `chrono,` are used. Therefore, C++ 11 standard is enabled while compiling the implementation using GNU Compiler Collection (gcc).

Five large test sets, which represent predicted number of participants provided by the assumed prediction model, are generated. Each test set consists of 100 time slots. The values of predicted number of participants at different time slots are chosen using random integer generator.

6.4.2 Performance Metrics

In chapter 5, we propose a dynamic programming algorithm that produces optimal result i.e., minimum waste of resources in terms of number of participants. We also propose a faster greedy algorithm but it produces suboptimal result. We consider the following three metrics to compare these two algorithms:

i) **Effects of scaling time lag (delta) on waste of resources in terms of number of participants:** If the time lag (delta) between consecutive scaling requests increases, less number of scaling requests can be made. With decreased scaling requests, the elasticity also decreases. Therefore, with increase of time lag value, waste of resources is expected to increase for both algorithms.

ii) **Elastic allocation of conference sizes**: As the predicted number of participants varies over time, the conference sizes allocated by the proposed dynamic programming and the greedy algorithms should also increase and decrease accordingly.

iii) **Running time**: The time complexity of the proposed DP algorithm is polynomial whereas that of the proposed greedy algorithm is linear. This should be reflected in the graph when we plot the calculated running time of the algorithms. For each test set, we calculate the running times in milliseconds for both algorithms. Each individual test set is divided into 10 parts, with increment of 10 time slots.

In chapter 3, we derived four requirements on conference scaling algorithm. Among the three metrics above, the first two metrics relate to the third requirement of minimizing waste of resources. The third metric (running time) pertains to the fourth requirement of acceptable response time. We do not need performance metrics for the first two requirements. Because the outputs of the proposed algorithms include when to scale (future time slots) and how much to scale (conference size to allocate). They also consider scaling time lag while scheduling scaling requests.

6.4.3 Performance Results

The results obtained from evaluating the conferencing algorithms are described below:

i) **Effects of scaling time lag (δ) on waste of resources in terms of number of participants:** Figure 6-8 shows that, with increase of scaling time lag, waste of resources increase for both algorithms. This observation is aligned with the expectation for this metric discussed in the previous sub-section. It also shows that the greedy algorithm, as a result of being sub-optimal, leads to more waste of resources.

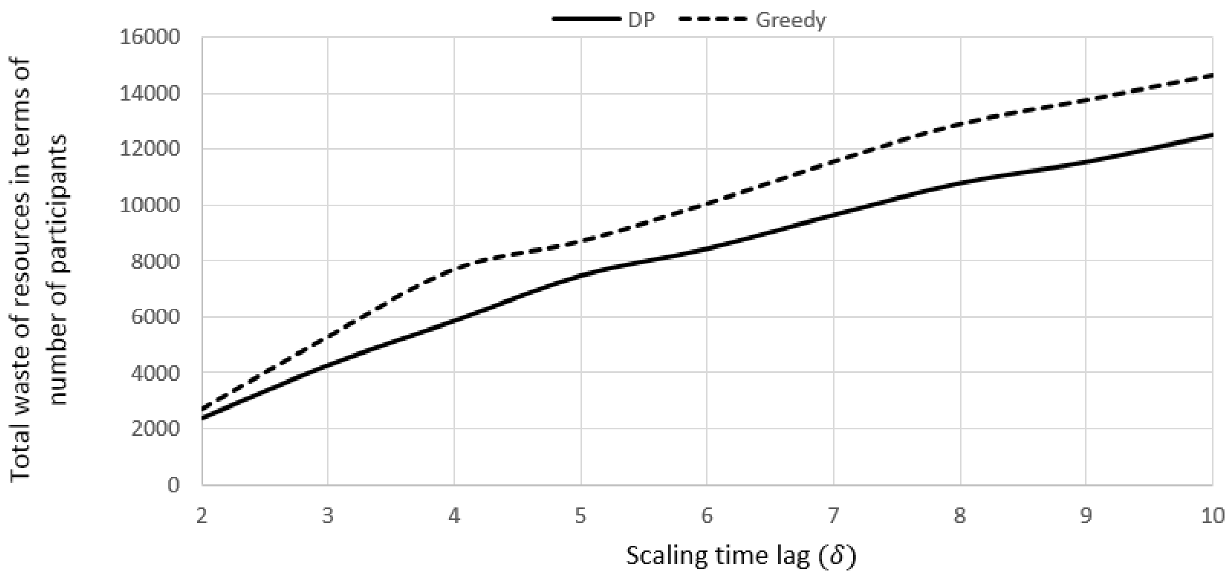


Figure 6-8: Effects of scaling time lag (δ) on waste of resources in terms of number of participants

Figure 6-9 shows the difference in resource wastage between the DP and the greedy algorithm. The difference does not follow any particular pattern because minimization of resource wastage in greedy algorithm is affected by the value of delta as well as the variable trends of P, predicted number of participants. Note that the proposed greedy algorithm tries to minimize resource waste around the local maxima and minima, taking time lag δ value into consideration. However, when δ is 2, the difference is very small compared to higher δ values. Because with delta of 2, it is possible to scale at every other time slot, which leads to greater optimization by both algorithms and results into very small difference.

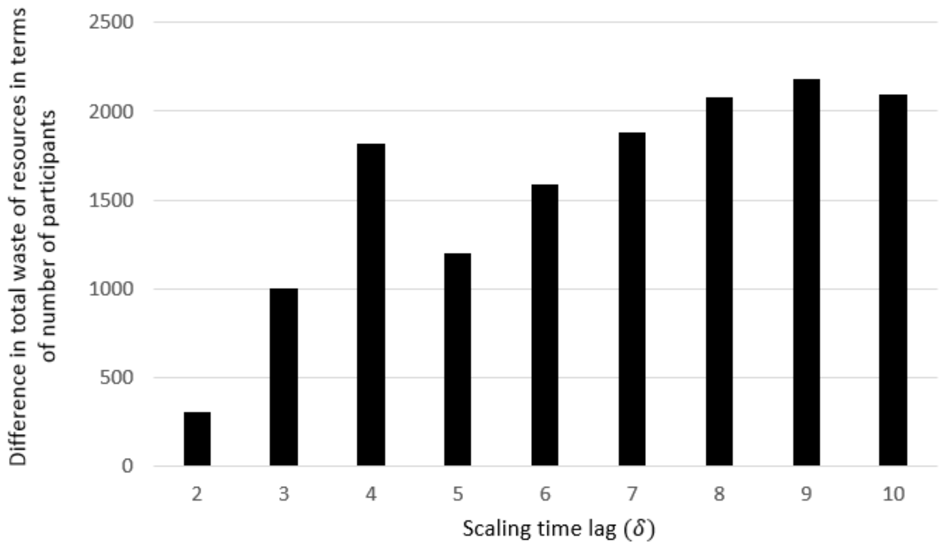


Figure 6-9: Difference in waste of resources between DP and greedy algorithms

ii) **Elastic allocation of conference sizes:** Figure 6-10 and 6-11 derived from measurements demonstrate that the proposed algorithms ensure elasticity. Figure 6-12 shows the comparison of allocated conference sizes between DP and greedy algorithms. Since DP algorithm produces the optimal result, allocated conference size by DP algorithm is less than that by greedy algorithm most of the time. However, when DP algorithm allocates more than the greedy one, it does so to avoid local optimization, while striving to achieve global optimization of wasted resource. For example, in figure 6-12, greedy algorithm allocates less than DP between time slots 25 and 30. Between time slots 20 and 40, predicted number of participants has three spikes at time slots 25, 32 and 38 (referring to figure 6-10 and 6-11). The greedy algorithm is myopic and minimizes waste locally for the first spike. In doing so, it could not optimize the subsequent spikes due to time lag constraint. On the other hand, the DP algorithm optimizes globally, leading to less waste of resources in the long term.

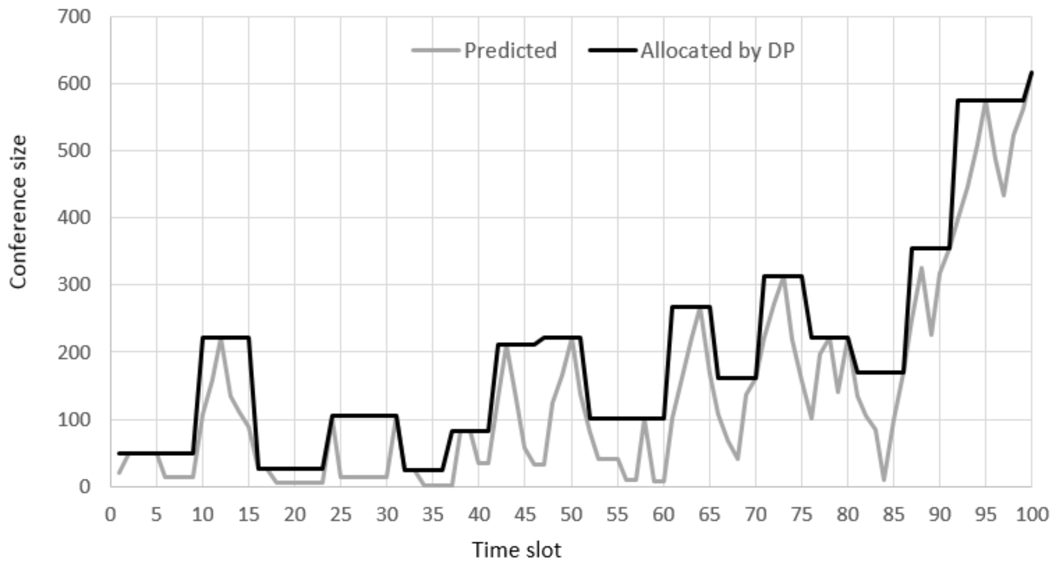


Figure 6-10: Elastic conference size allocation by DP algorithm

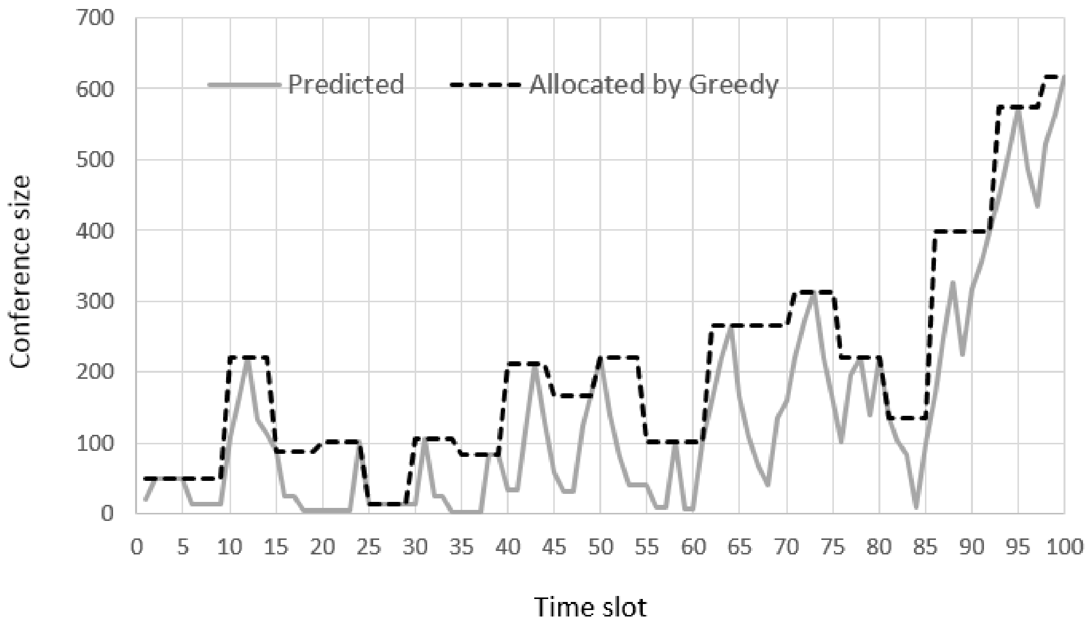


Figure 6-11: Elastic conference size allocation by greedy algorithm

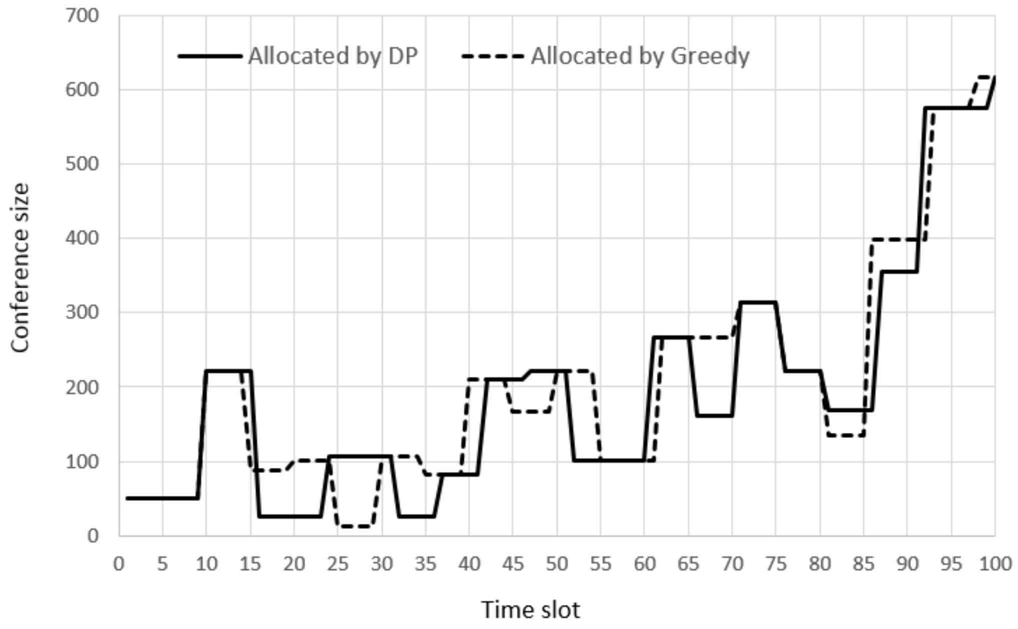


Figure 6-12: Comparison of elastic conference size allocation between DP and greedy algorithms

iii) **Running time:** Figure 6-13 derived from the measurements of running time demonstrates the proposed greedy algorithm is polynomial and the greedy algorithm is linear. The X-axis is the number of time slots and the Y-axis is the running time. The vertical bar shows the confidence interval with 95% confidence level. The figure shows that, with increase of time slots, the running time increases according to the time complexities of the algorithms.

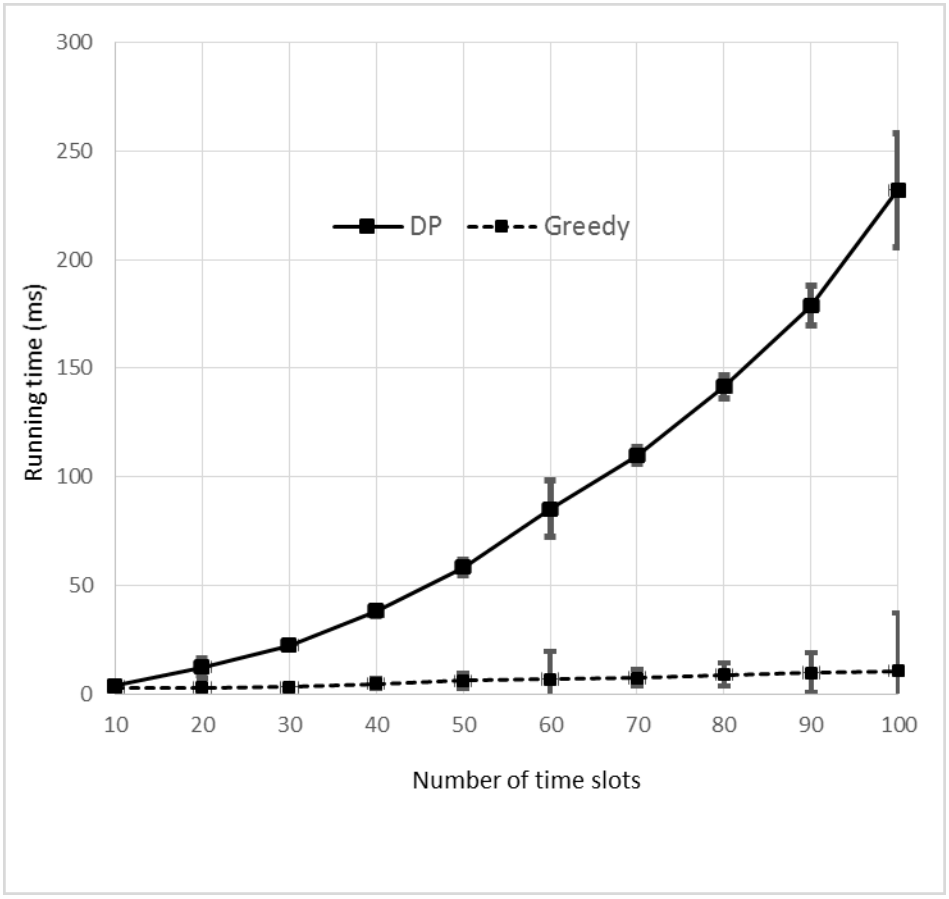


Figure 6-13: Comparison of running time between DP and greedy algorithms

The above performance results demonstrate the trade-offs between the proposed DP and greedy algorithms for conference scaling. While the greedy algorithm is faster, the DP algorithm produces better scaling. Waste of resources incurred by the schedule of greedy algorithm depends on two factors – value of time lag and trend of predicted number of participants over time. Future works with the greedy algorithm includes adding more heuristics and determining an upper bound of waste of resources for the greedy algorithm.

6.5 Chapter Summary

In this chapter, we have presented software architecture of conferencing PaaS. We have explained the software components and their interactions for two scenarios. We have also discussed the prototype and the software tools used to implement the prototype. Then we presented the performance measurements of the prototype with the comparison scenarios, performance metrics and results. After that, we have evaluated performances of the proposed conference scaling algorithms. The performances of DP and greedy algorithms for conference scaling are compared. In the next chapter, we shall summarize contribution of this thesis and then propose several future research directions.

Chapter 7

7. Conclusion and Future Work

In this chapter, we summarize the contributions of this thesis. We also provide several research directions for future works on conferencing PaaS.

7.1 Contribution Summary

Multimedia Conferencing is an important part of a wide range of conferencing applications such as massively multi-player online games and distance learning applications. Conferencing application developers can use third-party conferencing services (e.g., dial-out video conference, dial-in audio conference) to speed up development and to save cost. However, conferencing service provisioning (i.e. service development, deployment and management) still remains very difficult for the conferencing service providers. One challenge is that service providers need to know the complicated low-level details of conferencing technologies, protocols and their interactions. Another challenge is elastic resource provisioning. Participants join and leave during the conference. In order to achieve cost efficiency, it is critical to allocate and deallocate resources in an elastic manner.

This thesis relies on a business model from the state of the art. In addition to the conventional roles broker and connectivity provider, the business model followed consists of the following roles: conferencing service provider, conferencing platform provider, conferencing infrastructure provider, conferencing substrate provider. This thesis has focused on conferencing PaaS provider role. It is assumed that conferencing infrastructure provider also plays the role of substrate provider. From a motivating scenario for cloud-based conferencing service provisioning, we have derived a set of requirements on the conferencing PaaS. We have also derived requirements on conference scaling algorithm, which conferencing PaaS performs to scale conferences in an elastic manner. We have reviewed the state of the art and evaluated them against the requirements. We have found that none of them meets all of our requirements.

We have proposed a conferencing PaaS architecture that facilitates conferencing service provisioning. As part of this architecture, a set of high-level service development APIs for the conferencing service providers is also proposed. Thus, the proposed architecture tackles the challenge of service providers to master low-level conferencing details. For another challenge elastic conference scaling, we have designed two algorithms. One of them produces optimal scheduling of scaling requests, given the predicted number of participants for a certain period of time. The other algorithm is faster but gives suboptimal scheduling.

We have designed a software architecture based on the proposed general architecture of conferencing PaaS. A prototype is implemented to validate the architecture. The implemented scenario includes a conferencing service provider offering dial-in audio conferencing service, a game application consuming that service, a conferencing PaaS and two conferencing IaaSs providing dial-in signaling and audio mixer substrates. For performance evaluation, three comparison scenarios (non-cloud conferencing and two cloud-based conferencing scenarios) have been considered. Performance results show resource efficiency of cloud-based conferencing with acceptable penalty in response times. The proposed conference scaling algorithms are implemented and evaluated with large test sets. They are compared for different scenarios (different time lag constraint, different sizes of test set). Evaluation shows that both of them ensures elastic conference scaling with different trade-offs (speed versus optimal result).

7.2 Future Work

In the proposed conferencing PaaS architecture, management of conferencing service SLA is included as part of a bigger management component. One interesting research direction is to explore the SLA issues specific to conferencing services and to extend the proposed architecture.

Conferencing substrates are offered as web services by conferencing IaaS providers. This thesis assumes that subscription of different substrate services by the conferencing PaaS provider is done offline. A future work is to integrate substrate discovery into the conferencing PaaS architecture.

After executing conferencing services in the PaaS, the service providers publish their services for the conferencing application developers. In future, the issues involved in automatic publication of executed conferencing services can be investigated and the conferencing PaaS architecture can be extended for that.

Before creating a requested conference, conferencing PaaS selects the most suitable IaaSs that provides the necessary substrates. A future work is to investigate existing cloud service selection algorithms that can be reused for substrate selection. There are a few future works related to the proposed conference scaling algorithms. They are relaxing assumption of time lag constraint being a multiple of time slot duration, adding more heuristics to improve the proposed greedy algorithm and deriving upper bound for the waste of resources.

While designing conference scaling algorithms, this thesis assumes that a prediction model provides the future number of participants in a conference. A future work is to investigate the issues involved in designing suitable prediction models for different types of conferencing.

Appendix

1. Detailed Description of the Proposed Greedy Algorithm for Conference Scaling

The main algorithm corresponds directly to the steps in high-level view of the algorithm.

```
GREEDY-OPTIMIZATION-SUBOPTIMAL ($P, n, \delta$)

1.  ($slopes, total\_slopes$) = DIVIDE-INTO-SLOPES($P, n, \delta$)
2.  $schedule$ = SCHEDULE-EXTREMA($P, n, \delta, slopes, total\_slopes$)
3.  $schedule$ = SCHEDULE-NAIVE($P, n, \delta, schedule$)
```

Each step of this main algorithm uses some other procedures, which will be described next. The first procedure is DIVIDE-INTO-SLOPES.

```
DIVIDE-INTO-SLOPES ($P, n, \delta$)

1.  Let $slopes$ be the list of slopes found in $P$
2.  $start = \delta + 1$
3.  $total\_slopes = 0$
4.  Until $start < n$
5.      ($upward\_slope\_found, end$) = CHECK-UPWARD-SLOPE($start, P, n$)
6.      If upward slope is not found
7.          ($downward\_slope\_found, end$) = CHECK-DOWNWARD-SLOPE($start, P, n$)
8.      Add ($start, end$) to list $slopes$
9.      $total\_slopes = total\_slopes + 1$
10.     $start = end$
11. Return ($slopes, total\_slopes$)
```

DIVIDE-INTO-SLOPES iterates over the predicted number of participants, looking for upward or downward slopes. After finding a slope, it adds the slope's starting and ending time slot number to the list. CHECK-UPWARD-SLOPE and CHECK-DOWNWARD-SLOPE procedures are as follows:

```
CHECK-UPWARD-SLOPE ($start, P, n$)

1.  $end = start + 1$
2.  Until $end \leq n$ and $P[end] \geq P[end - 1]$
```

```
3.       $end = end + 1$
4.  If $end > start + 1$
5.       Return $(yes, end - 1)$
6.  else return $(no, -1)$
```

CHECK-DOWNWARD-SLOPE $(start, P, n)$

```
1.  $end = start + 1$
2.  Until $end \leq n$ and $P[end] < P[end - 1]$
3.       $end = end + 1$
4.  If $end > start + 1$
5.       Return $(yes, end - 1)$
6.  else return $(no, -1)$
```

Both of the above procedures differ only on line 8. CHECK-UPWARD-SLOPE checks for non-decreasing sequence whereas CHECK-DOWNWARD-SLOPE does for decreasing sequence.

After dividing the time slots into a list of slopes, the next step is to schedule the time slots near extrema. SCHEDULE-EXTREMA performs this task and is described below:

SCHEDULE-EXTREMA $(P, n, \delta, slopes, total\_slopes)$

```
1.  Initialize a list $schedule[1 \dots n]$ with $nil$
2.  $min\_left = \delta + 1$
3.  $min\_right = \frac{(\delta+1)}{2}$
4.  $current\_slope = 1$
5.  Until $current\_slope < total\_slopes$
6.       $left = slopes[current\_slope]$
7.       $right = slopes[current\_slope + 1]$
8.       $total\_left = left.end - left.start + 1$
9.       $total\_right = right.end - right.start + 1$
10.      If $total\_left \geq min\_left$ and $total\_right \geq min\_right$
11.           $x =$ COMPUTE-WASTE-NAIVE($P, n, \delta, left.end$)
12.           $y =$ COMPUTE-WASTE-EXTREMA($P, n, \delta, left.end$)
13.           If $y < x$
14.                  $mid\_time\_slot = left.end$
15.                  $left\_time\_slot = left.end - \frac{\delta}{2}$
16.                  $right\_time\_slot = left.end + (\frac{\delta+1}{2}) - 1$
17.                  $schedule[left_{time_{slot}} - \delta] =$
                     $max(P[mid\_time\_slot], P[left\_time\_slot], P[right\_time\_slot])$
18.     $current\_slope = current\_slope + 1$
```

```
19. Return $schedule$
```

Each extrema consists of a left slope and a right slope. For example, in case of local maxima, there is an upward slope on the left and a downward slope on the right. A local maxima is followed by a local minima and vice versa. SCHEDULE-EXTREMA iterates over each slope, taking the current slope as the left slope and the next one as the right slope. In order to deal with edge cases, such as sharp rise or fall after extrema value, it also assesses the waste of resources and then schedules only if the waste is optimized using extrema optimization.

The helper procedures COMPUTE-WASTE-NAÏVE and COMPUTE-WASTE-EXTREMA are straight-forward. In order to compute total waste of resources at the end of a slope, these just add the difference with the maximum number of participants. These two procedures are given below:

```
COMPUTE-WASTE-NAÏVE ($P, n, \delta, left\_slope\_end$)

1. $start = left\_slope\_end - \delta + 1$
2. $end = left\_slope\_end$
3. $largest = max(P[start], P[end])$
4. $waste = 0$
5. for $i = start$ to $end$
6.       $waste = waste + (largest - P[i])$
7. Return $waste$
```

```
COMPUTE-WASTE-EXTREMA ($P, n, \delta, left\_slope\_end$)

1. $mid = left\_slope\_end$
2. $start = mid - \frac{\delta}{2}$
3. $end = mid + \left(\frac{\delta+1}{2}\right) - 1$
4. $largest = max(P[start], P[mid], P[end])$
5. $waste = 0$
6. For $i = start$ to $end$
7.       $waste = waste + (largest - P[i])$
8. Return $waste$
```

After scheduling the local extrema, the last step is to schedule the remaining time slots using the naïve scaling approach. SCHEDULE-NAIVE procedure performs this and is described below:

```
SCHEDULE-NAIVE ($P, n, \delta, schedule$)

1. $first$ = NEXT-SCHEDULED-TIME-SLOT($1, schedule, n$)
2. If $first$ is $nil$
3.     $m = n - \delta$
4.     $total\_scaling\_requests = \frac{m+\delta-1}{\delta}$
5.     SCHEDULE-INTERVAL($P, n, \delta, \delta + 1, n, total\_scaling\_requests, schedule$)
6. else SCHEDULE-BEFORE-FIRST($P, n, \delta, first, schedule$)
7.     $last$ = SCHEDULE-BEFORE-LAST($P, n, \delta, left, schedule$)
8.     SCHEDULE-AFTER-LAST($P, n, \delta, last, schedule$)
```

SCHEDULE-NAIVE procedure finds the first time slot at which a scaling request has been scheduled. If there is none, it means no scaling request has been scheduled using extrema optimization approach. Therefore, the whole conference is scheduled using the naïve scaling approach. On the other hand, if a scheduled time slot $first$ is found, the naïve scaling approach is applied in three steps. First, schedules are made to accommodate the beginning time slots till $first + \delta - 1$. Second, the time slots between the first and the last schedule, which are not accommodated yet, are covered. Last, the time slots after the last schedule are accommodated.

Next, the helper procedures used in SCHEDULE-NAIVE are described. The first helper procedure is NEXT-SCHEDULED-TIME-SLOT. Given the starting time slot, it iterates until it either finds a scheduled time slot or hits the end. The complete procedure is given below.

```
NEXT-SCHEDULED-TIME-SLOT ($start, schedule, n$)

1. for $i = start$ to $n$
2.     If $schedule[i]$ is not $nil$
3.         Return $i$
4. Return $nil$
```

Given total scaling requests, a starting and an ending time slots, the next helper procedure SCHEDULE-INTERVAL schedules the given number of scaling requests between the starting and the ending time slots using the naïve scaling approach. The complete SCHEDULE-INTERVAL procedure is given below.

SCHEDULE-INTERVAL $(P, n, \delta, start, end, total\_scaling\_requests, schedule)$

```
1. for $i = 1$ to $total\_scaling\_requests$
2.     $x = start$
3.     If this is the last scaling request
4.         $y = end$
5.     Else $y = start + \delta - 1$
6.     $schedule[start - \delta] =$ MAX-PARTICIPANTS$(P, n, x, y)$
7.     $start = start + \delta$
```

On line 6, SCHEDULE-INTERVAL procedure uses another helper procedure MAX-PARTICIPANTS which calculates the maximum predicted participant, given a range of time slots. This helper procedure is straight-forward and is given below:

MAX-PARTICIPANTS $(P, n, start, end)$

```
1. $max = -1$
2. for $i = start$ to $end$
3.     if $P[i] > max$
4.         $max = P[i]$
5. Return $max$
```

The next important helper procedure used by SCHEDULE-NAIVE is SCHEDULE-BEFORE-FIRST. It is described below:

SCHEDULE-BEFORE-FIRST $(P, n, \delta, first, schedule)$

```
1. $m\_temp = first + \delta - 1$
2. $m = m\_temp - \delta$
3. If $m \geq \delta$
4.     $start = \delta + 1$
5.     $end = first + \delta - 1$
6.     $total\_scaling\_requests = \frac{m}{\delta}$
7.     SCHEDULE-INTERVAL$(P, n, \delta, start, end, total\_scaling\_requests, schedule)$
```

SCHEDULE-BEFORE-FIRST procedure schedules the targeted time slots by reusing helper procedure SCHEDULE-INTERVAL. Line 1 and 2 calculates the number of time slots that need to be scheduled. The first scaling request is scheduled on time slot $first$, meaning this scaling will accommodate time slots from $first + \delta$. Therefore, SCHEDULE-BEFORE-FIRST procedure needs to take care of time slots up to $first + \delta - 1$ (line 1). However, the initial conference size covers the first δ time slots (line 2). Because of the time lag constraint, there must be at least δ time slots that need to be scheduled. Otherwise, those time slots have to be accommodated by the initial conference size. This is checked on line 3. Line 4 sets the starting time slot for the interval. The first δ time slots are accommodated by the initial conference size. Therefore, the starting time slot is $\delta + 1$. As discussed already, the ending time slot is $first + \delta - 1$ and set on line 5. Line 6 calculates the total number of required scaling requests. Line 7 uses SCHEDULE-INTERVAL to schedule scaling requests within the given range of time slots.

Another important helper procedure used by SCHEDULE-NAIVE is SCHEDULE-AFTER-LAST. It is given below:

```
SCHEDULE-AFTER-LAST ($P, n, \delta, last, schedule$)

1.  $m = n - (last + 2 * \delta) + 1$
2.  If $m > 0$
3.      $start = last + 2 * \delta$
4.      $end = n$
5.      $total\_scaling\_requests = \frac{m+\delta-1}{\delta}$
6.      SCHEDULE-INTERVAL($P, n, \delta, start, end, total\_scaling\_requests, schedule$)
```

The last time slot $last$, which has a scaling request scheduled, accommodates time slots from $last + \delta$ to $last + 2 * \delta - 1$. SCHEDULE-AFTER-LAST procedure schedules time slots from $last + 2 * \delta$ to n, where n is the total number of time slots. Line 1 calculates the number of time slots that need to be scheduled. If there is at least one time slot, line 3 and 4 sets the starting and the ending time slots. Line 5 calculates the total scaling requests required. The remaining time slots m may not be evenly divisible by δ. This means that we may have time slots fewer than δ at the end, which should be accommodated as well. That's why $\delta - 1$ is added to m on line 5 to round up to the nearest total scaling requests.

The last helper procedure used in SCHEDULE-NAIVE is SCHEDULE-BEFORE-LAST. Given the first time slot which is scheduled by the extrema optimization approach, it iterates forward till the last time slot scheduled by SCHEDULE-EXTREMA. While performing iteration, it schedules the time slots which are not accommodated yet. The complete SCHEDULE-BEFORE-LAST procedure is described below:

```
SCHEDULE-BEFORE-LAST ($P, n, \delta, start, schedule$)

1.  $left = start$
2.  $right$ = NEXT-SCHEDULED-TIME-SLOT($left + 1, schedule, n$)
3.  Until $right$ is not $nil$
4.      $m = (right + \delta - 1) - (left + 2 * \delta) + 1$
5.      If $m > 0$
6.          If $m \geq \delta$
7.              $x = left + 2 * \delta$
8.              $y = right + \delta - 1$
9.              $total\_scaling\_requests = \frac{y-x+1}{\delta}$
10.             SCHEDULE-INTERVAL($P, n, \delta, start, end, total\_scaling\_requests, schedule$)
11.         Else
12.             $x = left + \delta$
13.             $y = right + \delta - 1$
14.             $schedule[left]$ = MAX-PARTICIPANTS($P, n, x, y$)
15.     $left = right$
16.     $right$ = NEXT-SCHEDULED-TIME-SLOT($left + 1, schedule, n$)
17. Return $left$
```

During each iteration, SCHEDULE-BEFORE-LAST procedure takes two consecutive time slots $left$ and $right$ scheduled by SCHEDULE-EXTREMA procedure. Then it calculates the number of time slots which are not accommodated yet. If more than δ time slots need to be accommodated, new scaling requests has to be scheduled. For fewer than δ time slots, the scaling request scheduled on the $left$ has to be updated.

Line 1 and 2 sets the initial values of $left$ and $right$. Lines 3 to 16 iterates over the scaling requests scheduled by SCHEDULE-EXTREMA. Line 4 calculates the number of time slots that are not accommodated by SCHEDULE-EXTREMA. Schedule at time slot $left$ covers time slots from $left + \delta$ to $left + 2 * \delta - 1$. Schedule at time slot $right$ covers time slots starting from $right + \delta$. Therefore, time slots from $left + 2 * \delta$ to $right + \delta - 1$ need to be accommodated. If more than δ time slots need to be accommodated, lines 7 to 10 add new scaling requests to the schedule. Otherwise, it is not possible to add a new scaling request due to the time lag constraint. So, lines 12 to 14 updates the schedule on the $left$.

After SCHEDULE-EXTREMA and SCHEDULE-NAIVE procedures schedules scaling requests, the initial conference size can be calculated easily. INITIAL-CONFERENCE-SIZE procedure returns the initial size. It finds the first time slot fts at which a scaling request is scheduled. Then it just checks the maximum participants from the first time slot to $fts + \delta - 1$. The complete INITIAL-CONFERENCE-SIZE procedure is given below:

```
INITIAL-CONFERENCE-SIZE($P, n, \delta, schedule$)

1.  $fts$ = NEXT-SCHEDULED-TIME-SLOT($1, schedule, n$)
2.  $size$ = MAX-PARTICIPANTS($P, n, 1, fts + \delta - 1$)
3.  Return $size$
```

2. Time Complexity Analysis of the Proposed Greedy Algorithm

In order to derive the time complexity of the greedy algorithm GREEDY-OPTIMIZATION-SUBOPTIMAL, the time complexities of the used procedures are determined first. Figure 5-5 shows the call graph of GREEDY-OPTIMIZATION-SUBOPTIMAL algorithm and figure 5-6 shows the call graph for SCHEDULE-NAIVE procedure. Following a bottom-up approach, time complexities of the procedures in figure 5-5 will be determined first. Next, the procedures in figure 5-6 will be determined.

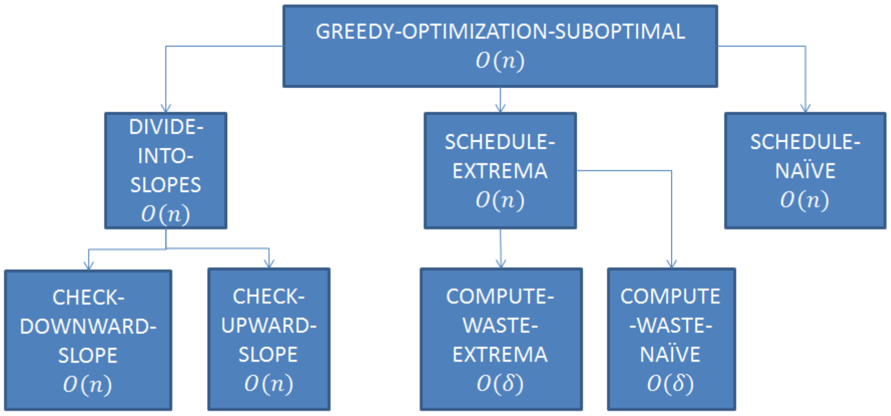

GREEDY-OPTIMIZATION-SUBOPTIMAL $O(n)$
DIVIDE-INTO-SLOPES $O(n)$
SCHEDULE-EXTREMA $O(n)$
SCHEDULE-NAÏVE $O(n)$
CHECK-DOWNWARD-SLOPE $O(n)$
CHECK-UPWARD-SLOPE $O(n)$
COMPUTE-WASTE-EXTREMA $O(\delta)$
COMPUTE-WASTE-NAÏVE $O(\delta)$

Figure A-1: Call graph of GREEDY-OPTIMIZATION-SUBOPTIMAL algorithm

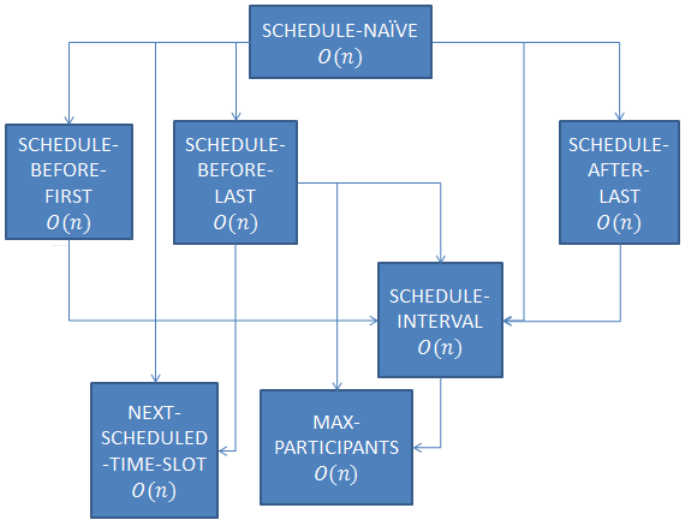

SCHEDULE-NAÏVE $O(n)$
SCHEDULE-BEFORE-FIRST $O(n)$
SCHEDULE-BEFORE-LAST $O(n)$
SCHEDULE-AFTER-LAST $O(n)$
SCHEDULE-INTERVAL $O(n)$
NEXT-SCHEDULED-TIME-SLOT $O(n)$
MAX-PARTICIPANTS $O(n)$

Figure A-2: Call graph of SCHEDULE-NAIVE procedure

Both CHECK-UPWARD-SLOPE and CHECK-DOWNWARD-SLOPE iterates at most $(n - start)$ times, which gives their running time $O(n)$. DIVIDE-INTO-SLOPES procedure iterates from lines 4 to 10. However, at line 10, increment in iteration is a direct result of checking slopes at lines 5 and 7. Since the iteration counter $start$ jumps from one slope to the next, DIVIDE-INTO-SLOPES iterates at most n times. Therefore, the runtime complexity is $O(n)$.

Both COMPUTE-WASTE-NAIVE and COMPUTE-WASTE-EXTREMA procedures iterate at most δ times. So, their runtime complexity is $O(\delta)$. SCHEDULE-EXTREMA iterates over each slope from lines 5 to 18. On line 11 and 12, COMPUTE-WASTE-NAIVE and COMPUTE-WASTE-EXTREMA procedures are used, which for each iteration, contributes 2δ to runtime complexity. Given n time slots and time lag δ, maximum number of slopes can be $\frac{n}{\delta}$. So the time complexity of SCHEDULE-EXTREMA is $O\left(\left(\frac{n}{\delta}\right) * 2\delta\right) = O(2n) = O(n)$.

To derive the time complexity of SCHEDULE-NAIVE, we use call graph in figure 5-6. NEXT-SCHEDULED-TIME-SLOT iterates at most $(n - start + 1)$ times. So, its runtime complexity is $O(n)$. MAX-PARTICIPANTS iterates at most $(end - start + 1)$ times. So, its runtime complexity is $O(n)$.

SCHEDULE-INTERVAL iterates for $total\_scaling\_requests$ times. Given n time slots and time lag δ, the maximum number of scaling requests possible is $\frac{n}{\delta}$. For each iteration, the call to MAX-PARTICIPANTS(P, n, x, y) contributes 2δ to runtime complexity, because $(y - x + 1) < 2\delta$. So, the time complexity of SCHEDULE-INTERVAL is $O\left(\left(\frac{n}{\delta}\right) * 2\delta\right) = O(2n) = O(n)$.

SCHEDULE-BEFORE-FIRST and SCHEDULE-AFTER-LAST procedures do not have iteration and uses SCHEDULE-INTERVAL. Therefore, their time complexities will be the same as SCHEDULE-INTERVAL, which is $O(n)$.

SCHEDULE-BEFORE-LAST procedure iterates only over the time slots scheduled by SCHEDULE-EXTREMA. Maximum number of such time slots can be the maximum number of slopes. During each iteration, SCHEDULE-BEFORE-LAST procedure uses either SCHEDULE-

INTERVAL or MAX-PARTICIPANTS procedure. In either case, the time slots not accommodated by SCHEDULE-EXTREMA in-between $left$ and $right$ (lines 7-8 and lines 12-13) are iterated. Then, the iteration jumps to the next slope (line 15). Thus, call to SCHEDULE-INTERVAL or MAX-PARTICIPANTS contributes at most n to the runtime complexity over all iterations. However, the call to NEXT-SCHEDULED-TIME-SLOT procedure costs another full iteration from left to right time slot, contributing n to the runtime complexity over all iterations. As NEXT-SCHEDULED-TIME-SLOT, SCHEDULE-INTERVAL and MAX-PARTICIPANTS are called in the same iteration, the runtime complexity will be added. Therefore, the complexity of SCHEDULE-BEFORE-LAST procedure is $O(n + n) = O(2n) = O(n)$.

SCHEDULE-NAIVE uses NEXT-SCHEDULED-TIME-SLOT, SCHEDULE-INTERVAL, SCHEDULE-BEFORE-FIRST, SCHEDULE-AFTER-LAST and SCHEDULE-BEFORE-LAST procedures without any iteration. Therefore, its time complexity is the maximum of the complexity of the procedures, which is $O(n)$.

Now that we know the runtime complexity of all three steps of GREEDY-OPTIMIZATION-SUBOPTIMAL algorithm, we can derive its complexity. The algorithm uses the main steps without any iteration. The complexity is then the maximum of the complexity of the main steps, which is $O(n)$.

Bibliography

[1] 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.

[2] P. Koskelainen, H. Schulzrinne, and X. Wu, “A SIP-based conference control framework,” in *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, 2002, pp. 53–61.

[3] J. Rosenberg and H. Schulzrinne, “Models for Multi Party Conferencing in SIP.” [Online]. Available: https://tools.ietf.org/html/draft-ietf-sipping-conferencing-models-01. [Accessed: 01-Jun-2016].

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

[6] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, “A break in the clouds: towards a cloud definition,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, 2008.

[7] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, “SIP: session initiation protocol,” 2002.

[8] H. Liu and P. Mouchtaris, “Voice over IP signaling: H. 323 and beyond,” *Commun. Mag. IEEE*, vol. 38, no. 10, pp. 142–148, 2000.

[9] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A transport protocol for real-time applications,” 2003.

[10] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman, “The secure real-time transport protocol (SRTP),” 2004.

[11] J. Rosenberg, *RFC 4353—A framework for conferencing with the session initiation protocol*. February, 2006.

[12] H. Khartabil, P. Koskelainen, and A. Niemi, “The conference policy control protocol (CPCP),” *Draft-Ietf-Xcon-Cpcp-01 Work Prog.*, 2004.

[13] P. Koskelainen, X. Wu, H. Schulzrinne, and J. Ott, “Requirements for floor control protocols,” 2006.

[14] G. Camarillo, K. Drage, and J. Ott, “The binary floor control protocol (BFCP),” 2006.

[15] A. Amirante, T. Castaldi, L. Miniero, and S. P. Romano, “On the seamless interaction between webRTC browsers and SIP-based conferencing systems,” *Commun. Mag. IEEE*, vol. 51, no. 4, pp. 42–47, 2013.

[16] F. Andreasen, M. Arango, C. Huitema, R. Kumar, S. Pickett, I. Elliott, B. Foster, and A. Dugan, “Media gateway control protocol (MGCP) version 1.0,” 2003.

[17] C. Boulton, S. McGlashan, and T. Melanchuk, “Media Control Channel Framework,” 2011.

[18] J. Van Dyke, E. Burger, and A. Spitzer, “Media Server Control Markup Language (MSCML) and Protocol,” 2007.

[19] A. Amirante, T. Castaldi, L. Miniero, and S. P. Romano, “Performance analysis of the Janus WebRTC gateway,” in *Proceedings of the 1st Workshop on All-Web Real-Time Systems*, 2015, p. 4.

[20] P. Rodríguez Pérez, J. Cerviño Arriba, I. Trajkovska, and J. Salvachúa Rodríguez, “Advanced Videoconferencing based on WebRTC,” 2012.

[21] L. Lopez Fernandez, M. P. Diaz, R. Benitez Mejias, F. J. Lopez, J. Santos, and others, “Catalysing the success of WebRTC for the provision of advanced multimedia real-time communication services,” in *Intelligence in Next Generation Networks (ICIN), 2013 17th International Conference on*, 2013, pp. 23–30.

[22] W. Elleuch, “Models for multimedia conference between browsers based on WebRTC,” in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on*, 2013, pp. 279–284.

[23] B. Sredojev, D. Samardzija, and D. Posarac, “WebRTC technology overview and signaling solution design and implementation,” in *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on*, 2015, pp. 1006–1009.

[24] P. Segec, P. Paluch, J. Papán, and M. Kubina, “The integration of WebRTC and SIP: way of enhancing real-time, interactive multimedia communication,” in *Emerging eLearning Technologies and Applications (ICETA), 2014 IEEE 12th International Conference on*, 2014, pp. 437–442.

[25] R. Buyya, J. Broberg, and A. M. Goscinski, *Cloud computing: principles and paradigms*, vol. 87. John Wiley & Sons, 2010.

[26] N. R. Herbst, S. Kounev, and R. Reussner, “Elasticity in Cloud Computing: What It Is, and What It Is Not.,” in *ICAC*, 2013, pp. 23–27.

[27] I. Sriram and A. Khajeh-Hosseini, “Research agenda in cloud technologies,” *ArXiv Prepr. ArXiv10013259*, 2010.

[28] 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.

[29] “What is platform as a service (PaaS)?” [Online]. Available: http://www.thoughtsoncloud.com/2014/02/what-is-platform-as-a-service-paas/. [Accessed: 04-Nov-2015].

[30] M. Boniface, B. Nasser, J. Papay, S. C. Phillips, A. Servin, X. Yang, Z. Zlatev, S. V. Gogouvitis, G. Katsaros, K. Konstanteli, and others, “Platform-as-a-service architecture for real-time quality of service management in clouds,” in *Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on*, 2010, pp. 155–160.

[31] V. Gonçalves and P. Ballon, “Adding value to the network: Mobile operators’ experiments with Software-as-a-Service and Platform-as-a-Service models,” *Telemat. Inform.*, vol. 28, no. 1, pp. 12–21, 2011.

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

[33] “Buildpacks.” [Online]. Available: http://docs.cloudfoundry.org/buildpacks/index.html. [Accessed: 04-Nov-2015].

[34] “Custom Buildpacks.” [Online]. Available: http://docs.cloudfoundry.org/buildpacks/custom.html. [Accessed: 04-Nov-2015].

[35] “Boilerplates.” [Online]. Available: https://www.ng.bluemix.net/docs/starters/boilerplates.html. [Accessed: 05-Nov-2015].

[36] “Services Overview.” [Online]. Available: http://docs.cloudfoundry.org/devguide/services/. [Accessed: 04-Nov-2015].

[37] "Service Broker API v2.7." [Online]. Available: http://docs.cloudfoundry.org/services/api.html. [Accessed: 04-Nov-2015].
[38] "Using cf CLI Plugins." [Online]. Available: http://docs.cloudfoundry.org/devguide/installcf/use-cli-plugins.html. [Accessed: 04-Nov-2015].
[39] "Cloud Foundry Eclipse Plugin." [Online]. Available: https://docs.cloudfoundry.org/buildpacks/java/sts.html. [Accessed: 05-Nov-2015].
[40] "BOSH." [Online]. Available: https://bosh.io/docs/about.html. [Accessed: 04-Nov-2015].
[41] "Deploying Cloud Foundry." [Online]. Available: http://docs.cloudfoundry.org/deploying/. [Accessed: 04-Nov-2015].
[42] 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.
[43] 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.
[44] 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.
[45] M. Marzolla, S. Ferretti, and G. D'angelo, "Dynamic resource provisioning for cloud-based gaming infrastructures," *Comput. Entertain. CIE*, vol. 10, no. 1, p. 4, 2012.
[46] 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.
[47] 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.
[48] 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.
[49] "VidyoWorks^{TM} Integrated Vidyo Enabled Applications." [Online]. Available: http://info.vidyo.com/rs/vidyo/images/WP-VidyoWorks.pdf. [Accessed: 08-Nov-2015].
[50] "VidyoWorks^{TM} Datasheet." [Online]. Available: http://www.vidyo.com/wp-content/uploads/2013/12/DS-VidyoWorks.pdf. [Accessed: 08-Nov-2015].
[51] "Web Conferencing: Unleash the Power of Secure, Real-Time Collaboration." [Online]. Available: http://www.webex.com/includes/documents/security_webex.pdf. [Accessed: 08-Nov-2015].
[52] "WebEx Event Center - Pricing." [Online]. Available: http://www.webex.com/products/webinars-and-online-events.html#pricing. [Accessed: 08-Nov-2015].
[53] "Blue Jeans : Saas Video Conferencing." [Online]. Available: https://www.bluejeans.com/video-collaboration/saas-video-conferencing. [Accessed: 09-Nov-2015].
[54] "Blue Jeans Data Sheet." [Online]. Available: http://bluejeans.com/sites/default/files/pdf/BJN-Datasheet.pdf. [Accessed: 09-Nov-2015].

[55] 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.
[56] “What Is Google App Engine?” [Online]. Available: https://cloud.google.com/appengine/docs/whatisgoogleappengine?hl=en. [Accessed: 05-Nov-2015].
[57] “Heroku - Cloud Application Platform.” [Online]. Available: https://www.heroku.com/. [Accessed: 10-Nov-2015].
[58] “What Is Elastic Beanstalk and Why Do I Need It? - Elastic Beanstalk.” [Online]. Available: http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html. [Accessed: 10-Nov-2015].
[59] “Salesforce App Cloud | The Leading Enterprise PaaS | Salesforce Developers.” [Online]. Available: https://developer.salesforce.com/platform. [Accessed: 10-Nov-2015].
[60] “OpenShift Enterprise 3.” [Online]. Available: https://enterprise.openshift.com. [Accessed: 10-Nov-2015].
[61] “Pivotal Cloud Foundry.” [Online]. Available: http://pivotal.io/platform. [Accessed: 10-Nov-2015].
[62] “IBM Bluemix - Create, Deploy, Manage Your Applications in the Cloud.” [Online]. Available: http://www.ibm.com/cloud-computing/bluemix/index-b.html. [Accessed: 10-Nov-2015].
[63] A. Pessoa 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.
[64] 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.
[65] 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.
[66] 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.
[67] 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.
[68] Z. ur Rehman, F. K. Hussain, and O. K. Hussain, “Towards Multi-criteria Cloud Service Selection,” in *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, 2011, pp. 44–48.
[69] W. Zeng, Y. Zhao, and J. Zeng, “Cloud Service and Service Selection Algorithm Research,” in *Proceedings of the First ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, New York, NY, USA, 2009, pp. 1045–1048.
[70] L. Zhao, Y. Ren, M. Li, and K. Sakurai, “Flexible service selection with user-specific QoS support in service-oriented architecture,” *J. Netw. Comput. Appl.*, vol. 35, no. 3, pp. 962–973, May 2012.
[71] 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.

[72] J. El Hadad, M. Manouvrier, and M. Rukoz, “TQoS: Transactional and QoS-Aware Selection Algorithm for Automatic Web Service Composition,” *IEEE Trans. Serv. Comput.*, vol. 3, no. 1, pp. 73–85, Jan. 2010.
[73] S.-Y. Hwang, E.-P. Lim, C.-H. Lee, and C.-H. Chen, “Dynamic web service selection for reliable web service composition,” *Serv. Comput. IEEE Trans. On*, vol. 1, no. 2, pp. 104–116, 2008.
[74] H. Wada, P. Champrasert, J. Suzuki, and K. Oba, “Multiobjective Optimization of SLA-Aware Service Composition,” in *IEEE Congress on Services - Part I, 2008*, 2008, pp. 368–375.
[75] M. Garriga, C. Mateos, A. Flores, A. Cechich, and A. Zunino, “RESTful service composition at a glance: A survey,” *J. Netw. Comput. Appl.*, vol. 60, pp. 32–53, 2016.
[76] S. Loreto and S. P. Romano, “Real-Time Communications in the Web: Issues, Achievements, and Ongoing Standardization Efforts.,” *IEEE Internet Comput.*, vol. 16, no. 5, 2012.
[77] G. Zhao, J. Liu, Y. Tang, W. Sun, F. Zhang, X. Ye, and N. Tang, “Cloud computing: A statistics aspect of users,” in *IEEE International Conference on Cloud Computing*, 2009, pp. 347–358.
[78] T. Lorido-Botrán, J. Miguel-Alonso, and J. A. Lozano, “Auto-scaling techniques for elastic applications in cloud environments,” *Dep. Comput. Archit. Technol. Univ. Basque Ctry. Tech Rep EHU-KAT-IK-09-12*, 2012.
[79] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, vol. 6. MIT press Cambridge, 2001.
[80] “Node.js.” [Online]. Available: https://nodejs.org/en/. [Accessed: 25-Jun-2016].
[81] “camunda BPM.” [Online]. Available: https://github.com/camunda. [Accessed: 27-Feb-2016].
[82] “expressjs/express,” *GitHub*. [Online]. Available: https://github.com/expressjs/express. [Accessed: 25-Jun-2016].
[83] “Advanced REST client.” [Online]. Available: https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo. [Accessed: 27-Feb-2016].
[84] “cloudfoundry/cli,” *GitHub*. [Online]. Available: https://github.com/cloudfoundry/cli. [Accessed: 25-Jun-2016].
[85] M. Chinosi and A. Trombetta, “BPMN: An introduction to the standard,” *Comput. Stand. Interfaces*, vol. 34, no. 1, pp. 124–134, Jan. 2012.
[86] X. Wen, G. Gu, Q. Li, Y. Gao, and X. Zhang, “Comparison of open-source cloud management platforms: OpenStack and OpenNebula,” in *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 2012, pp. 2457–2461.
[87] “Smart Applications on Virtual Infrastructure.” [Online]. Available: http://www.savinetwork.ca/. [Accessed: 27-Feb-2016].
[88] “yudai/cf_nise_installer,” *GitHub*. [Online]. Available: https://github.com/yudai/cf_nise_installer. [Accessed: 25-Jun-2016].
[89] O. T. Time, “Itu-t recommendation g. 114,” *ITU-T May*, 2000.