A Cloud Infrastructure for Scalable and Elastic Multimedia Conferencing Applications

Flora Taheri[#1], Jerry George[#2], Fatna Belqasmi[#3], Nadjia Kara[*4], Roch Glitho[#5]

[#]*Concordia University, Canada*
[#1]fl_taher@encs.concordia.ca
[#2]je_georg@encs.concordia.ca
[#3]fbelqasmi@alumni.concordia.ca
[#5]glitho@ciise.concordia.ca

[*]*ETS, University of Quebec, Canada*
[*4]nadjia.kara@etsmtl.ca

Abstract – **Multimedia conferencing applications play a critical role in business and everyday life. However, scalability and elasticity remain quite elusive, even though they are the keys to efficiency in resource usage. A cloud-based approach could solve the scalability and elasticity issues and bring other benefits such as an easy introduction of new applications. This paper proposes a cloud infrastructure that relies on fine-grained conferencing substrates. These substrates are virtualized and shared by conferencing applications. They enable scalability and elasticity.**

***Keywords*— Multimedia conferencing, cloud infrastructure, scalable conferencing applications, elastic conferencing applications.**

I. INTRODUCTION

Conferencing [1] can be defined as the conversational/real time exchange of media between several parties. It is the basis of a plethora of multimedia applications and services (e.g. audio/video conferencing, multiparty games and distance learning). Despite having become ubiquitous, these applications still face challenges such as scalability and elasticity. Cloud computing [2] is an emerging paradigm, with three key facets: Software as Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). It has several inherent benefits (e.g. efficient usage of resources, scalability, elasticity, and the rapid development and introduction of new applications). Cloud computing can potentially transform the business landscape in a significant manner, a situation which has led to the proposal of new business models. One example is the business model proposed for cloud-based conferencing in reference [3]. It has six roles: connectivity provider, broker, conferencing substrate provider, conferencing infrastructure provider, conferencing platform provider and conferencing application/service provider. This paper proposes a virtualized infrastructure for cloud-based conferencing. The infrastructure relies on fined-grained sharable conferencing substrates (e.g. audio mixer and dial-out signalling) that can be assembled on the fly to build scalable and elastic multimedia conferencing applications. A scenario is presented in the next section along with the technical challenges. This is followed by a summary of related work, the proposed architecture and its early validation.

II. SCENARIO AND TECHNICAL CHALLENGES

Figure 1 depicts three conferencing applications offered by three different providers over a same cloud conferencing IaaS: an audio/video conference application offered by provider A, a multiparty game application offered by provider B, and a distance learning application offered by provider C. The figure shows as well the conferencing PaaS offered by a conferencing PaaS provider. These applications may have different characteristics (e.g. different Quality of Service (QoS), number of users, cost, etc.). In our vision, they will rely on fine-grained and sharable substrates offered by the conferencing IaaS provider. It should be noted that the conferencing IaaS provider may rely on third party substrate providers, as per the business model proposed in reference[3].

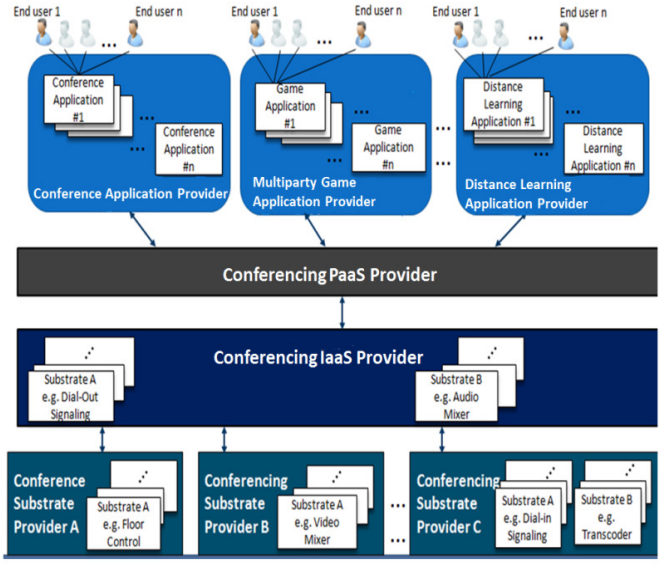

Figure1. Cloud-based Conferencing Illustration

A first challenge is that the IaaS architecture should rely on an open business model. Another one is the ability for all players to publish and discover. Yet another challenge is scalability in terms of the number of end users that use the conferencing applications as well as in terms of the number of applications that can be offered by application providers as SaaS. The last challenge is elasticity to ensure efficiency in resource usage.

III. RELATED WORK

Conferencing has been extensively studied by standards bodies (e.g. by IETF [4, 5, 6] and 3GPP [7]), especially in non-cloud environments. None of these works fully meets the scalability and elasticity requirements. The same applies to the existing cloud-based products, such as Cisco's WebEx [8]. A few embryonic architectures have been put forward for conferencing in the cloud. Reference [9] proposes an audio/video conference application as SaaS. It does not rely on fine-grained substrates and consequently does not scale in an elastic manner. Reference [8] also offers audio/video conferencing as a cloud-based service. It provides scalability and elasticity. However, it does not rely on an open business model in which third parties could provide conferencing substrates. Reference [10] aims at deploying

video conferencing applications as SaaS in a hybrid cloud, but faces scalability and elasticity issues.

IV. PROPOSED ARCHITECTURE

Figure 2 shows the proposed conferencing architecture. It has two layers, Conferencing IaaS and Conferencing SubaaS (Substrate as a Service), in addition to a broker.

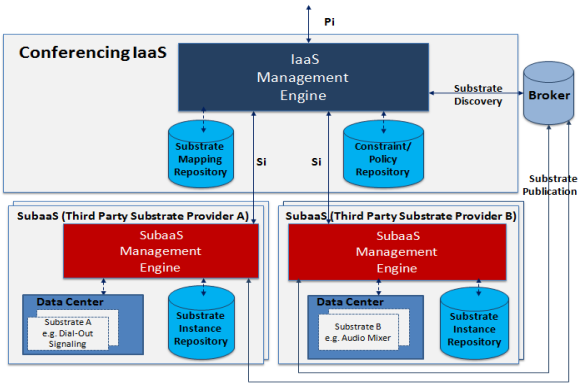

Figure 2. Architecture of Conferencing IaaS

Conferencing substrate providers may offer different types of conferencing substrates. Two third-party conferencing substrate providers are shown in figure 2, but there could be many more. In the figure, substrate provider A is offering a Dial-out signaling substrate and substrate provider B is offering an Audio mixer substrate. The architectural components and interfaces are described in details next, and we end the section with an illustrative scenario.

A. *Architectural Components*

The Broker

The broker is from our previous work [11]. It is a semantic-oriented general framework for the publication and discovery of cloud-based conferencing substrates.

IaaS layer components

The key functional component of the conferencing IaaS is the IaaS Management Engine. This component accepts activation requests for conference applications from the conferencing PaaS. Through the broker, it discovers the adequate substrates and contacts the corresponding substrate providers to activate the required substrates. The IaaS Management Engine also accepts the service execution requests from PaaS and redirects them to the target SubaaSes. There are two other components in this layer: the Substrate Mapping Repository and the Constraint/Policy Repository. The substrate mapping repository contains information on the substrate instances for given conference applications. This information is provided by the substrate providers. Each substrate involved in an application is mapped to a substrate instance provided by a substrate provider. This mapping information is used during execution. The constraint/policy repository contains the constraints on the substrates used for a given conference application.

SubaaS layer components

The key component of SubaaS is the SubaaS Management Engine. This component has several functions: substrate publication and instantiation, substrate instance monitoring, and resource allocation and management. Each substrate provider has an internal repository (Substrate Instance Repository) that contains information pertaining to substrate instances (e.g., type of substrate, the IP address of the Virtual Machines (VM) hosting the substrate, etc.)

Each SubaaS has a data center with the Physical Machines (PMs) that host the VMs running the substrates. To manage resource allocation to substrates, the substrate providers need an appropriate elastic scaling mechanism. It is the SubaaS Management Engine which implements the mechanism. It monitors the usage of substrate instances, and, based on the usage, it may allocate de-allocate resources. There are several requirements that the mechanism should meet. Resource needs in conference applications may fluctuate and change rapidly. A dynamic demand resource allocation mechanism is therefore required to scale up/down. In some cases, the application level information should be taken into account. For example, when scaling down, a substrate instance that still has an ongoing conference/active user should not be totally deleted. The mechanism should also cover a customized support for scaling multitenant substrates where a substrate is shared between several applications. Each individual tenant expects the application to be scalable and the actions of other tenants should not affect the application's performance. None of existing scaling algorithms meets all requirements. We therefore propose a new mechanism which is still at a very preliminary stage.

Most dynamic on-demand resource allocation approaches are based on VM-level elasticity, wherein scaling is performed by increasing or decreasing the number of VMs that serve an application [12]. The drawback of this approach is that when the application is not using newly-created VMs efficiently, considerable computing resources will be wasted, resulting in extra cost. Also, creating, shutting down, and removing the VMs dynamically at run time will increase overhead [12].

Another approach is fine-grained resource-level elasticity [12] which focuses on changing the VM capacity at the level of the underlying resources components (e.g. CPU, memory, I/O) instead of using VMs as basic units for dynamic resource provisioning. Based on the fact that VM-level elasticity may not be always required, and in some scenarios, lightweight resource-level elasticity can be sufficient, our resource allocation mechanism performs a fine-grained resource-level scaling based on the level of resource utilization to improve efficiency in the resource utilization of conferencing substrates.

B. *Interfaces*

REST is the technology used for the interface implementation. The interfaces are *Substrate Publication* interface for the the publication of substrates to broker, *Substrate Discovery* interface for substrates discovery from the broker, *Pi* interface for PaaS and IaaS interactions and *Si* interface for IaaS and SubaaS interactions. As an illustration, table 1 depicts a subset of the Pi interface. It is the subset used for the execution of a dial-out audio conference application that relies on two substrates: dial-out signaling and audio mixer. This subset contains operations for creation and deletion of a conference.

Operation	Method	Path	Request body parameters	Response
CreateConference	POST	/DialOutSignaling/Conferences	Conference Information	ConferenceId
CreateConference	POST	/AudioMixer/Conferences	Conference Information	ConferenceId
DeleteConference	DELETE	/DialOutSignaling/ Conferences/{ConferenceId}	ConferenceId	200 OK
DeleteConference	DELETE	/AudioMixer/ Conferences/{ConferenceId}	ConferenceId	200 OK

Table 1. A Subset of the Pi interface APIs

C. Illustrative Scenario

We consider a dial-out audio conference application in which a conference chair can perform different requests, such as *Create Conference, Add Participant to conference, Delete Participant from conference* and *Delete Conference*. We assume that the application has already been created using an application creation tool provided by the conferencing PaaS. The activation process deals with the instantiation of the substrates. The PaaS starts the process by sending a request to the IaaS Management Engine for the activation of the substrates required for that application (dial-out signaling and audio mixer substrates in this scenario). The request comes with the characteristics of the substrates (e.g., QoS requirements). Upon receipt of the request, the IaaS Management Engine searches the broker to find the appropriate substrates. The broker returns the search result to the IaaS Management Engine.

Let us assume that the discovered substrates are a dial-out signalling substrate offered by substrate provider A and an audio mixer substrate offered by substrate provider B. The IaaS Management Engine will contact the SubaaS Management Engines of the two substrate owners to request the instantiation of the substrates. Each of the two SubaaS Management Engines instantiate its substrate, saves the instance information in the internal substrate instance repository, and returns back the activation result to the IaaS Management Engine. When both substrates are successfully activated, the IaaS Management Engine sends the activation response back to the conferencing PaaS. The substrate mapping information will also be saved in the substrate mapping repository of the IaaS for later use. Now that the back end of the application has been provided, the execution becomes feasible. Conferencing PaaS has the logic/workflow of the application requests (this workflow is produced at the time of service creation). Upon reception of the requests from the conference chair, the conferencing PaaS sends execution requests to the IaaS Management Engine, which in turn directs the requests to the target SubaaS Management Engines. The requests are eventually forwarded to the actual substrate instances by the SubaaS Management Engines. The execution of a request may involve several substrates. Depending on the request, the request workflow may contain several sub-processes which will be executed one after another until the initial request has been executed completely. Figure 3 shows the execution sequence of *Create Conference* request based on the REST interfaces.

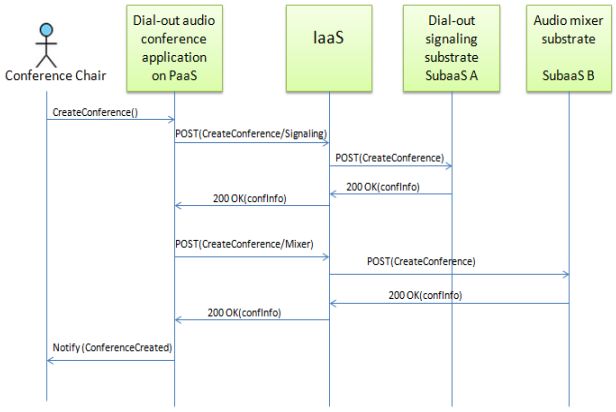

Figure 3. Sequence of the *Create Conference* Process

V. PROOF OF CONCEPT PROTOTYPE AND PERFORMANCE EVALUATION

A. Software Architecture

Figure 4 shows the software architecture. In the conferencing IaaS, the IaaS Management Engine includes Request Manager, Service Activation Engine, Substrate Discovery/Selection Engine and Service Execution Enigine as software entities. The Activation Request Manager realizes the activation interface provided by IaaS (Pi1). The Service Activation Engine searches for the substrates using a substrate discovery engine, and after finding them contacts the corresponding substrate providers for substrate instantiation. The Execution Request Manager receives the execution requests from PaaS (through Pi2 interface) and sends them to the corresponding service execution engine instance. A service execution instance provides the required interface for redirecting execution requests to the target substrates. There is a service exution instance for each application.

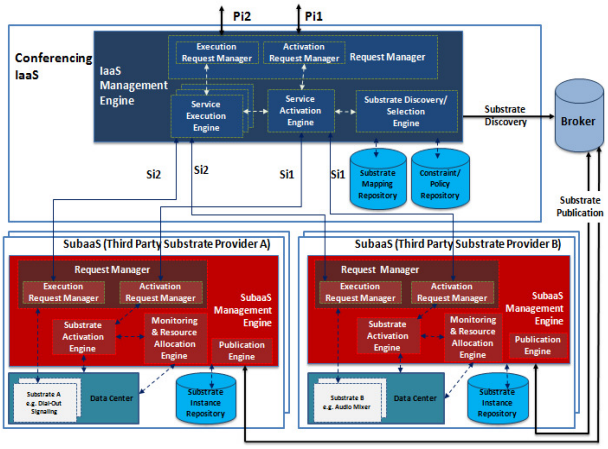

Figure 4. Software Architecture

A SubaaS Mangement Engine in SubaaS includes Publication Engine, Request Manager, Substrate Activation Engine, and Monitoring and Resource Allocation Engine as software entities. The Publication Engine publishes the description of the substrates provided by the substrate provider to the broker through the RESTful publication interface. The ActivationRequest Manager receives the activation requests from IaaS (through the Si1 interface) and sends them to the substrate activation engine.

The Substrate Activation Engine checks the availbility of resourses upon receiving an instantiation request and then instantiates the substrate. The Execution Request Manager receives the execution requests from IaaS (through an Si2 interface) and sends them to a substrates instance to be executed. There can be several instances of a substrate serving an application. The Monitoring and Resource Allocation Engine monitors the substrates. It also implements the resource allocation mechanim proposed earlier.

B. Prototype

We have implemented a prototype as a proof of concept. The implemented scenario for our prototype consists of a conference application provider, a platform provider, an Infrastructure provider and 2 substrate providers: substrate provider A offering a dial-out signalling substrate, and substrate provider B offering an audio mixer substrate. There are also several end users and a conference chair. The type of conference application offered is a dial-out audio conference hosted on conferencing PaaS and offered as SaaS by the application provider. It is offered as a web

application that provides a GUI to be used by the conference chair. We also provide a simple web GUI through which the application can be activated. The workflow of the execution requests is hard coded since we do not address service composition in this paper. We have implemented a subset of components from the software architecture. We assume that IaaS already knows the two substrates. Hence there is no substrate discovery engine implemented in IaaS. The monitoring and resource allocation engine is not implemented. The resource allocation to substrates is static. No request manager is used in IaaS, as there is only one activation request and one service execution instance. No request manager is used in SubaaS, since there is only one substrate instance. RESTful interfaces are provided to realize the roles of an IaaS service activation engine and a SubaaS substrate activation engine. All these REST interfaces used in the prototype are implemented using Jersey. To simulate a PM in a datacenter environment, we used a Xen server to host and virtualize the substrates. Upon receiving an activation request, the substrate activation engine creates a substrate instance from a predefined VM template. Java binding of Xen's XML-RPC based API is used for the VM management operations (e.g. creation, starting, etc.). Dial-out signaling and audio mixer substrates run on two separate VMs. VMs are configured so that they deploy the substrate programs at the start up (using init scripts). Each VM sends a notification back to the requester through a REST POST request to indicate that the VM has started. All of the prototype's components are located in a local network. For a dial-out signaling substrate, we used Medooze conference server implemented with SIP servlets. For the audio mixer, we used Medooze media mixer, and for end-users as SIP clients we used X-Lite softphones. The softphones register with the dial-out signaling server. The end users', participants' and the conferences' information is stored in a MySQL database. The conference chair can see the list of registered users, create a conference and select users to add to the conference as participants by using the web GUI.

C. *Performance Evaluation*

We measured the time delay for substrate activation. The substrates' activation time starts when an activation request is sent from IaaS to SubaaS for the instantiation of a conferencing substrate, and continues until the substrate instance has been created and the notification of a successful instantiation has been sent back to the requester. The time delays for the activation of a dial-out signaling and an audio mixer instance are shown in table 2. They are acceptable, as activation is a onetime operation which happens before execution.

Substrate Name	VM Specification	Activation Time (ms)
Audio mixer	Ubuntu 12.10 Storage: 9.5 GB Memory: 1024 MB CPU: Intel core 2 (2.53 GHz)	41533
Dial-out signaling	Windows 7 (64 bit) Storage: 9.5 GB Memory: 1024 MB CPU: Intel core 2 (2.53 GHz)	53032

Table 2. . Substrate Activation Time Delay

VI. CONCLUSIONS

We propose a scalable and elastic infrastructure for cloud-based conferencing applications. Our architecture enables various conferencing applications to be built using virtualized conferencing substrates that can be provided by different substrate providers. As a proof of concept we implemented a prototype to validate the feasibility of the proposed architecture. We have also proposed a very early scaling mechanism to be used for scaling conferencing substrates.

ACKNOWLEDGMENT

This work was supported in part by the new researcher start up program of Fond de Recherche du Quebec – Nature et Technologies (FQRNT), and by the Natural Science and Engineering Council of Canada (NSERC) SAVI Research Network.

REFERENCES

[1] R. Even and N. Ismail, Conferencing scenarios, IETF RFC 4597, July 2006.

[2] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, 'A break in the clouds: towards a cloud definition', ACM SIGCOMM Computer Communication Review, vol. 39, no. 1, pp. 50–55, 2008.

[3] R. H. Glitho, 'Cloud-based Multimedia Conferencing: Business Model, Research Agenda, State-of-the-Art', IEEE 13th Conference on Commerce and Enterprise Computing (CEC), 2011, pp. 226 – 230.

[4] M. B. nortel, C. B. Avaya, and O. Levin, 'A Framework for Centralized Conferencing', IETF FRC 5239, June 2008.

[5] J. Rosenberg, 'A Framework for Conferencing with the Session Initiation Protocol (SIP)', IETF RFC 4353, February 2006.

[6] A. Buono, S. Loreto, L. Miniero, and S. P. Romano, 'A distributed IMS enabled conferencing architecture on top of a standard centralized conferencing framework', IEEE Communications Magazine, vol.45, No 3, March 2007.

[7] 3GPP TS 24.147, 'Conferencing Using the IP Multimedia (IM) Core Network (CN)', Stage 3, Release 11, September 2012.

[8] J. Li, R. Guo, and X. Zhang, 'Study on Service Oriented Cloud Conferencing', Third IEEE International Conference on Computer Science and Information Technology, 2010.

[9] P. Rodriguez, D. Gallego, J. Cerviiio, F. Escribano, J. Quemada, and J. Salvachua, 'VaaS: Videoconferencing as a Service', 5th International Conference on Collaborative Computing: Networking, Application and Work sharing, 2009.

[10] J. Cervino, F. Escribano, P. Rodriguez, I. Trajkovska, and J. Salvachua, 'Videoconference Capacity Leasing on Hybrid Clouds', IEEE 4th International Conference on Cloud Computing, 2011.

[11] J. George, F. Belqasmi, R. Glitho, and N. Kara, 'A Semantic-Oriented Description Framework and Broker Architecture for Publication and Discovery of Cloud Based Conferencing', 4th Canadian Semantic Web Symposium, July 2013.

[12] R. Han, L. Guo, M. M. Ghanem, and Y. Guo, 'Lightweight Resource Scaling for Cloud Applications', Department of Computing Imperial College London, 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2012.