UNIVERSIDAD POLITÉCNICA DE MADRID

Escuela Técnica Superior de Ingenieros de Telecomunicación

**CONTRIBUCIÓN AL DESPLIEGUE Y OPERACIÓN
DINÁMICOS Y EFICIENTES DE SISTEMAS DE
VIDEOCONFERENCIA EN ENTORNOS DISTRIBUIDOS**

TESIS DOCTORAL

Álvaro Alonso González
Ingeniero de Telecomunicación

Madrid
Junio 2016

Departamento de Ingeniería de Sistemas Telemáticos

Escuela Técnica Superior de Ingenieros de Telecomunicación

**CONTRIBUCIÓN AL DESPLIEGUE Y OPERACIÓN DINÁMICOS Y EFICIENTES
DE SISTEMAS DE VIDEOCONFERENCIA EN ENTORNOS DISTRIBUIDOS**

TESIS DOCTORAL

Autor: Álvaro Alonso González
Ingeniero de Telecomunicación

Director: Joaquín Salvachúa Rodríguez
Doctor Ingeniero de Telecomunicación

Junio 2016

Tribunal nombrado por el Magnífico y Excelentísimo Sr. Rector de la Universidad Politécnica de Madrid, el día _____ de _____ de 2016.

Presidente: _____

Vocal: _____

Vocal: _____

Vocal: _____

Secretario: _____

Suplente: _____

Suplente: _____

Realizado el acto de defensa y lectura de la Tesis el día _____ de _____ de 2016 en Madrid, habiendo obtenido la calificación de

El presidente,

El secretario,

Los vocales,

Resumen

Durante los últimos años, los servicios y aplicaciones de videoconferencia han experimentado una serie de cambios debido a la evolución de algunas de las tecnologías implicadas. Históricamente dichos servicios y aplicaciones se habían utilizado en escenarios profesionales muy concretos y gracias a plataformas hardware dedicadas.

Sin embargo, la consolidación de tecnologías que permiten el uso de este tipo de sistemas desde plataformas Web ha hecho que sean accesibles desde dispositivos personales como ordenadores, tabletas o teléfonos inteligentes. Esto ha producido un aumento considerable en el número de usuarios que los demandan y en la variedad de casos de uso y escenarios en los que se contemplan. La consecuencia primera de esta ubicuidad es la necesidad de modificar la forma en que se proveen estos servicios.

Desplegar sistemas de videoconferencia software en infraestructuras *Cloud* ofrece a los proveedores una serie de ventajas a la hora de adaptarse a grandes y variables cantidades de usuarios accediendo a sus servicios. El *Cloud* permite gestionar recursos virtuales de forma dinámica abstrayendo ciertas condiciones de escalabilidad que permiten ahorrar costes y aprovechar los recursos de manera más eficiente.

Pero estos despliegues plantean también una serie de retos derivados de la complejidad y características particulares de un sistema de tiempo real como es la videoconferencia. Esta tesis propone una serie de modelos y mecanismos que dan respuesta a estos retos y permiten desplegar sistemas de videoconferencia de manera dinámica y eficiente en sistemas *Cloud*.

Para ello se comienza realizando un estudio detallado de las mencionadas ventajas y retos consiguiendo poner en contexto el problema de cara al diseño de las soluciones. La primera propuesta es una arquitectura que permite escalar de manera dinámica servidores de videoconferencia atendiendo a las variaciones en la demanda. Así, se podrán proveer más o menos recursos según las necesidades de cada momento. Cuando se trata de liberar recursos disponibles es fundamental poder reagrupar las conexiones existentes en un conjunto de servidores para poder apagar el resto. Para conseguirlo se propone un mecanismo de movilidad de flujos multimedia entre servidores sin pérdida de paquetes y de manera transparente para los usuarios.

Utilizar sistemas de videoconferencia distribuidos ofrece a este tipo de despliegues más flexibilidad y granularidad consiguiendo mejorar aún más en eficiencia. Pero esto plantea un nuevo reto consistente en la programación de recursos entre los servidores distribuidos. Se propone también en este trabajo un mecanismo que habilita el uso de algoritmos de programación

avanzados que permiten tomar decisiones de asignación basándose en el estado de cada servidor distribuido. Además, se propone una nueva métrica para estimar el uso de recursos de cada nuevo cliente permitiendo utilizar dichos algoritmos de forma más precisa. Por último se realiza una comparativa de algoritmos de planificación utilizando una serie de coeficientes de evaluación que permiten determinar la idoneidad de los mismos en cada caso de uso y escenario.

Abstract

During the last few years, videoconferencing applications and services have experimented several changes due to the related technologies evolution. In the past, this kind of systems were used in very specific professional scenarios and frequently thanks to dedicated hardware platforms.

However, the consolidation of new technologies that enables the use of such systems in Web-based platforms have made them accessible from personal devices like computers, tablets or smartphones. Consequently, the number of users that demands this type of services has substantially increased. Also the variety of scenarios in which they are used. The main consequence of these changes is that a new way of providing these services is needed.

Deploying software videoconferencing systems in Cloud based infrastructures offers several advantages to the service providers. These deployments help them to adapt the systems to high and variable number of users accessing the services. The Cloud allows the dynamic management of virtual resources and withdraws the providers from the scalability matters. This results in a cost saving and in a more efficient way of resource use.

But these Cloud deployments also imply several challenges due to the complexity and particular characteristics of a real time system like videoconference. This thesis proposes a set of models and mechanisms that answer these challenges and allow dynamic and efficient deployments of videoconferencing systems in the Cloud.

The starting point is a detailed study about the already mentioned advantages and challenges to contextualise the problem. The first proposal is an architecture that allows the dynamic scalability of videoconferencing servers attending demand variations. Thus, more or less resources could be provided depending on the necessities. When a release of resources is required, it is crucial to group the existing connections in a set of servers to be able to turn off the rest of them. To achieve this, a stream mobility mechanism is proposed. It enables the mobility between servers without packet loss and in a transparent way to the users.

Using distributed videoconferencing systems offers more flexibility, granularity and efficiency to these deployments. But the resource scheduling among the distributed servers is not a trivial task. This work also proposes a mechanism that enables the use of advanced scheduling algorithms to take decisions of where to assign new connections based on the state of the distributed servers. Furthermore, a new metric to estimate the resource use of such new connections is proposed. It improves the precision of the assignment decisions. Finally, a comparative between a set of scheduling policies is made. It is performed using several performance coefficients that help to determine the viability of each algorithm depending on the use case and scenario.

Agradecimientos

A todas las personas que son parte de esta tesis. A todos los que estuvieron involucrados de una forma u otra. A todos a los que por concisión o despiste no cito a continuación.

A Juan Quemada y a Joaquín Salvachúa. Por darme la oportunidad de investigar en este campo durante estos años y por guiar los pasos que me trajeron hasta aquí.

A todos mis compañeros del grupo de investigación. A los de la tercera, a los de la primera y a los de la segunda planta. A los que estuvieron y a los que están. Porque no se pueden explicar tantos años sin entender el día a día.

A Pedro Rodríguez y a Javier Cerviño. Por ser tan imprescindibles en cada línea que construyó esta historia. No habría sabido ni por dónde empezar.

A mi hermano, a mis amigos, a mi familia. Con vosotros esperando a la salida todo se ve más nítido desde aquí.

A mi madre y a mi padre. Esto es vuestro. Porque vosotros lo habéis conseguido.

Contenidos

Resumen	
Abstract	
Agradecimientos	
Lista de Ilustraciones	
Lista de Tablas	
Lista de Acrónimos	

1	Introducción	1
1.1	Contexto y motivación	2
1.2	Objetivos	5
1.3	Metodología	6
1.4	Proyectos relacionados	8
1.4.1	Licode	8
1.4.2	LTExtreme	8
1.4.3	MCS5	9
1.4.4	FIWARE, FI-XIFI y FICORE	9
1.4.5	EasyVideo	10
1.4.6	Otros proyectos	10
1.5	Estructura del documento	10
1.5.1	Capítulo 2: Estado del Arte	10
1.5.2	Capítulo 3: Videoconferencia escalable en infraestructuras Cloud	11
1.5.3	Capítulo 4: Asignación de recursos en sistemas de videoconferencia distribuidos	11
1.5.4	Capítulo 5: Validación y Resultados	11
1.5.5	Capítulo 6: Conclusiones	12
2	Estado del Arte	13
2.1	Videoconferencia multiusuario	14
2.1.1	Introducción a la videoconferencia multiusuario	14
2.1.2	Componentes de un sistema de videoconferencia multiusuario	16
2.1.3	Videoconferencia en la Web	20
2.1.4	Sistemas de videoconferencia multiusuario	24
2.2	Cloud Computing	27
2.2.1	Introducción al Cloud Computing	27
2.2.2	Servicios de Cloud Computing	30
2.2.3	Proveedores y plataformas de Cloud Computing	32
2.3	Sistemas distribuidos	37
2.3.1	Programación de recursos en sistemas multimedia	38
2.3.2	Programación de ejecución de tareas	40

2.3.3	Programación de recursos Cloud	42
2.3.4	Asignación de recursos en centros de datos	44
3	Videoconferencia escalable en infraestructuras Cloud	47
3.1	Objetivos	48
3.2	Despliegue de MCUs en el Cloud	48
3.2.1	Oportunidades	48
3.2.2	Retos	52
3.3	Escalabilidad dinámica a la demanda	56
3.3.1	Requisitos y bases de diseño	56
3.3.2	Arquitectura	60
3.3.3	Implementación y caracterización	68
3.4	Movilidad de flujos multimedia	72
3.4.1	Soluciones existentes	73
3.4.2	Descripción del escenario	74
3.4.3	Extensión de la especificación MICE	77
3.4.4	MICE aplicado a un escenario de videoconferencia	83
3.5	Conclusiones	90
4	Asignación de recursos en sistemas de videoconferencia distribuidos	93
4.1	Objetivos	94
4.2	Arquitectura de MCU distribuida	94
4.2.1	dOTM: distributed One To Many	95
4.2.2	Características del escenario	98
4.3	Mecanismo para habilitar la programación de recursos	101
4.3.1	Descripción del modelo	101
4.3.2	Diseño de la solución	103
4.3.3	Implementación y resultados	105
4.4	Estimación de consumo de recursos en MCUs distribuidas	113
4.4.1	Métrica para estimar el uso de recursos	114
4.4.2	Evaluación de resultados	117
4.5	Evaluación de estrategias de programación de recursos	121
4.5.1	Diseño de políticas de asignación	121
4.5.2	Comparativa y discusión	125
4.6	Conclusiones	132
5	Validación y Resultados	135
5.1	Licode	135
5.2	Proyectos Nacionales	136
5.2.1	LTExtreme	136
5.2.2	MCS5	137
5.2.3	EasyVideo	138
5.3	Proyectos Europeos	139
5.3.1	FIWARE, FI-XIFI y FICORE	139
5.4	Otros proyectos	139
5.5	Divulgación de resultados	140
5.5.1	Publicaciones	140

CONTENIDOS

5.5.2	Docencia y seminarios	141
6	Conclusiones	143
6.1	Contribuciones	148
6.2	Trabajos futuros	150
	Bibliografía	153

CONTENIDOS

Ilustraciones

2.1	Comparación de topologías en videoconferencia multiusuario	14
2.2	Arquitectura de WebRTC	25
2.3	Servicios Cloud según Gartner	31
2.4	Arquitectura de servicios de Amazon Web Services	33
2.5	Arquitectura de servicios de Openstack	37
2.6	Arquitectura Mesos con frameworks Hadoop y MPI	42
3.1	Ejemplo de dos MCUs realizando diferentes tareas	50
3.2	Distribución geográfica de MCUs	51
3.3	Proxies reenviando tráfico a MCUs	55
3.4	Flujo de interacción de Nube	59
3.5	Arquitectura de escalabilidad a la demanda	61
3.6	Uso de recursos de una MCU en un escenario de streaming en tiempo real.	70
3.7	Uso de recursos de una MCU en un escenario de videoconferencia	71
3.8	Arquitectura de movilidad de flujos multimedia.	74
3.9	Ejemplo de pérdida de paquetes en movilidad	76
3.10	Configuración de una asignación TURN simple.	84
3.11	Flujo de llamadas entre clientes en movilidad de flujos multimedia.	85
3.12	Configuración de múltiples asignaciones TURN.	88
4.1	Separación de una MCU en procesos.	95
4.2	Arquitectura de control de MCU distribuida	96
4.3	Arquitectura de medios de MCU distribuida	97
4.4	Configuración de extensiones de registro y reporte	104
4.5	Configuración del despliegue	109
4.6	Uso de CPU en los Agents	112
4.7	Uso de CPU en los experimentos	119
4.8	Agent 1 OTMs Linear Regression	120
4.9	Uso de CPU en las estrategias diseñadas	127
4.10	Regresiones lineales y nivel de fragmentación en las estrategias diseñadas	128
5.1	Arquitectura del demostrador LTextreme	136
5.2	Arquitectura de MCS5	138

ILUSTRACIONES

Tablas

3.1	Definición de los recursos en Nuve	58
3.2	Definición de estados de MCUs	64
3.3	Interfaz de decisión de MCUs en Cloud Handler	65
3.4	Mensajes entre los módulos de la arquitectura (I)	67
3.5	Mensajes entre los módulos de la arquitectura (II)	68
4.1	Mensajes Agent Controller - Agents	106
4.2	Instancias utilizadas en los Agents	110
4.3	Interfaz del módulo de gestión de regresiones lineales	118
4.4	Descripción de los métodos usados en los algoritmos	122
4.5	Coefficientes de rendimiento para las estrategias evaluadas.	130

Acrónimos

AMI	Amazon Machine Image
AMQP	Advanced Message Queueing Protocol
API	Application Program Interface
ATM	Asynchronous Transfer Mode
AWS	Amazon Web Services
BPaaS	Business Process as a Service
CSCW	Computer-Supported Cooperative Work
CPU	Central Processing Unit
DaaS	Database as a Service
DTLS	Datagram Transport Layer Security
GING	Grupo de Internet de Nueva Generación
HTML	Hiper Text Markup Language
HTTP	Hiper Text Transport Protocol
HMAC	Hash Message Authentication Code
IaaS	Infraestructure as a Service
ICE	Interactive Connectivity Establishment
IETF	Internet Engineering Task Force
IP	Internet Protocol
JSON	JavaScript Object Notation
MCU	Multipoint Control Unit
NAT	Network Address Translation
OTM	One To Many
PaaS	Platform as a Service
REST	Representational State Transfer

RIA Rich Internet Application

RPC Remote Procedure Call

RTP Real-time Protocol

RTCP RTP Control Protocol

RTC Red Telefónica Conmutada

SaaS Service as a Service

SHA1 Secure Hash Algorithm

SIP Session Initiation Protocol

SDP Session Description Protocol

SFU Selective Forwarding Unit

SRTP Secure Real-time Transport Protocol

STUN Session Traversal Utilities for NAT

TURN Traversal Using Relays around NAT

UPM Universidad Politécnica de Madrid

VNC Virtual Network Computing

W3C World Wide Web Consortium

WebRTC Web Real Time Communications

Capítulo 1

Introducción

Durante los últimos años, los sistemas de videoconferencia colaborativa que encontramos en Internet han evolucionado a una gran velocidad. En la década de los 90 y principios de los 2000, este tipo de sistemas se utilizaban en escenarios muy concretos como tele-reuniones de trabajo o en participación remota en clases o conferencias. Además, para su puesta en marcha y operación se necesitaba la intervención de expertos que configuraran y monitorizaran los sistemas, en ocasiones plataformas hardware específicas para las sesiones.

Sin embargo, hoy en día podemos encontrar aplicaciones de videoconferencia en cualquier ordenador personal, tableta e incluso teléfono inteligente (*smartphone*). Los servicios de Internet que ofrecen comunicaciones de vídeo y audio entre dos o más participantes son infinitos y cada vez son más los usuarios que se comunican utilizando este tipo de aplicaciones cada día.

Estos nuevos servicios y aplicaciones han surgido gracias a la conjunción de dos factores principales. Por una parte, y a raíz del nacimiento de la Web 2.0, surgen una serie de necesidades tecnológicas para mejorar la comunicación entre los usuarios. En un nuevo concepto de Web cuyo fundamento es la colaboración, es imprescindible proveer formas de conexión entre usuarios que vayan más allá de los mensajes escritos existentes hasta el momento. Estas tecnologías llevan por lo tanto la videoconferencia a los navegadores Web y en consecuencia a los ordenadores personales y los dispositivos móviles. El resultado es que cualquier persona con acceso a uno de estos dispositivos puede hacer uso de un servicio de videoconferencia. Y no solamente eso, sino que puede hacerlo desde cualquier lugar.

Por otra parte, y para atender la demanda de este gran y creciente número de usuarios, los proveedores de videoconferencia empiezan a plantearse nuevas formas de ofrecer sus servicios. En sesiones de videoconferencia complejas, donde participan múltiples usuarios o son necesarias tareas avanzadas como la grabación o la conversión de los vídeos compartidos, la comunicación punto a punto entre los participantes no es suficiente. En estos casos es necesario el uso de servidores de videoconferencia que actúan como nodo central de la comunicación y realizan dichas tareas. Estos servidores requieren altos niveles de computación, y para atender la demanda

de un gran y variable número de usuarios, la Computación en la Nube o *Cloud Computing* resulta imprescindible para los proveedores.

El *Cloud Computing* surge también en los 90 para ofrecer un modelo de computación en red bajo demanda en el que sus usuarios pagan únicamente por lo que consumen. Gracias a esta tecnología, los proveedores de servicios de videoconferencia pueden desplegar sus servidores de forma que escalen dinámicamente a la demanda variable de usuarios. Este tipo de despliegues se traducen directamente en ahorro de recursos y por lo tanto en ahorro de costes.

Sin embargo y dadas las características particulares de los servidores de videoconferencia, la manera de realizar estos despliegues no resulta trivial. Es necesario estudiar detenidamente los requisitos de este tipo de sistemas y diseñar una serie de modelos que nos permitan realizar dichos despliegues de forma dinámica y eficiente. Esta tesis pretende cubrir ambas expectativas.

1.1 Contexto y motivación

Para entender la motivación que ha llevado a la elaboración de esta tesis doctoral y poner en contexto las contribuciones que contiene, es necesario remontarnos a mis inicios en el grupo de investigación en el que la he realizado. A principios del año 2010 y mientras terminaba el cuarto curso de mis estudios de Ingeniería de Telecomunicaciones, comienzo a trabajar como becario de investigación en el Grupo de Internet de Nueva Generación (GING). El grupo pertenece al Departamento de Ingeniería de Sistemas Telemáticos (DIT) de la Universidad Politécnica de Madrid (UPM) y sus principales líneas de investigación están relacionadas con las nuevas tecnologías Web, el aprendizaje electrónico (*eLearning*) y los sistemas de videoconferencia multi-usuario.

Al entrar en el departamento mi trabajo va a estar relacionado con esta última línea, los sistemas de videoconferencia multi-usuario. En concreto con el sistema Isabel [De Miguel 1994, Quemada 1996], una plataforma de videoconferencia orientada al mundo educativo y optimizada para realizar conferencias e impartir clases a distancia. Isabel utiliza un esquema multipunto en el que cada cliente puede hacer de *super par* enviando los flujos multimedia (audio y vídeo) de unos participantes a otros. En el momento en que comienzo a trabajar en el proyecto, Isabel puede correr en sistemas Linux mediante una aplicación de escritorio. Mi tarea consiste en desarrollar un cliente móvil que permita acceder a sesiones de Isabel desde dispositivos iOS. La realización de este cliente sería mi Proyecto Fin de Carrera, que

1.1. CONTEXTO Y MOTIVACIÓN

leería en Marzo del año 2012.

Durante esos dos años, por tanto, trabajo en la realización de ese cliente iOS de Isabel. Ya había realizado desarrollos de aplicaciones móviles para iOS en algunas asignaturas de la carrera pero el mundo de la videoconferencia era nuevo para mí. El desarrollo completo del cliente requiere el conocimiento de todas las capas implicadas en un sistema de videoconferencia. He de realizar una capa de control que permita gestionar las sesiones y una capa de señalización que permita la conexión con el resto de clientes para establecer las comunicaciones. Esto implica el estudio de los protocolos de control de Isabel y su implementación en el nuevo cliente. Para realizar las conexiones necesito familiarizarme con las condiciones y componentes de red que intervienen en una conexión punto a punto. Algunos ejemplos son el mecanismo *Network Address Translation* (NAT) o los servidores *Session Traversal Utilities for NAT* (STUN) y *Traversal Using Relays around NAT* (TURN) imprescindibles para la conexión de clientes desde redes privadas.

Por otra parte he de realizar una capa de transporte y media que reciba paquetes de audio y vídeo y los procese hasta poder mostrar imágenes en la pantalla y reproducir sonidos en los altavoces. En el sentido contrario el software debe capturar los medios de la cámara y micrófono y procesarlos hasta encapsularlos en paquetes que se enviarán por la red. Estos procesos implican la codificación y decodificación de flujos de audio y vídeo para lo que tengo que familiarizarme con códecs como H.263, MP3, AAC, etc. Estos datos se encapsulan en paquetes RTP por lo que el manejo de este protocolo de tiempo real es también imprescindible.

En definitiva, a lo largo de esos dos años en el grupo, adquiero una serie de conocimientos que por una parte son fundamentales para el posterior desarrollo de esta tesis y por otra son la raíz de la motivación encontrada para realizarla. No solamente encuentro atractivo el mundo de la videoconferencia y las tecnologías implicadas sino que el hecho de trabajar junto a compañeros que están realizando investigaciones sobre este tema hace que se despierte en mí una inquietud que hará que decida embarcarme también en dichas investigaciones.

Cuando acabo mi Proyecto Fin de Carrera, Isabel ha dejado de ser el punto principal en cuanto a dedicación de esfuerzo en el grupo. El sistema sigue utilizándose por algunas empresas y para la retransmisión de eventos y conferencias en el departamento. Pero el auge de la videoconferencia Web gracias a tecnologías como Adobe Flashⁱ ha hecho que en los últimos años se abra una

ⁱ<http://get.adobe.com/es/flashplayer>

nueva línea de trabajo en la que la videoconferencia se entiende como un servicio provisto bajo demanda y consumido desde clientes Web ligeros en cualquier dispositivo PC o móvil. Así en 2009 había surgido VaaS [Rodríguez 2009, Cerviño 2011b], un sistema diseñado por el propio departamento y como continuación de Marte 3.0 [Cerviño 2008] que permitía proveer salas de videoconferencia virtuales a terceros y bajo demanda para ser integradas en sus propios entornos. Es una nueva forma de entender la videoconferencia con una arquitectura totalmente diseñada para ser desplegada en infraestructuras *Cloud* y escalar de forma dinámica a las necesidades de cada servicio.

Sin embargo, para esas fechas el uso de la tecnología Flash, en la que VaaS está basado, empieza a quedarse a un lado debido a la importancia que empieza a cobrar el estándar HTML5ⁱ, cada vez adoptado por un mayor número de navegadores y con un mayor abanico de posibilidades. Una de ellas, y clave en nuestro contexto, es WebRTC, el estándar definido por el *Internet Engineering Task Force* (IETF) y el *World Wide Web Consortium* (W3C) [Bergkvist 2012] para proveer comunicaciones en tiempo real de audio, vídeo y datos a las aplicaciones web. WebRTC está pensado para realizar comunicaciones entre pares (*peer-to-peer*). Sin embargo, en nuestro grupo de investigación, el equipo encargado de Isabel y que más tarde había creado VaaS, es pionero en la realización de experimentos multi-usuario con el nuevo estándar de RTC. Y a pesar de que la tecnología aún está muy inmadura, a principios de 2012 comienzan a aplicar los conocimientos adquiridos en los proyectos previos para diseñar uno de los primeros sistemas de multiconferencia compatible con WebRTC.

Es en ese momento, y coincidiendo con los últimos meses de mi proyecto, cuando comienzo a trabajar con el equipo de VaaS colaborando en el desarrollo del software del nuevo sistema, que en el futuro se llamaría Licodeⁱⁱ. Siguiendo la experiencia obtenida en los anteriores proyectos, este nuevo sistema se diseñaría desde el principio con una clara orientación Cloud, adaptando el esquema de videoconferencia como servicio de VaaS a las nuevas necesidades tecnológicas impuestas por el WebRTC. Sin embargo, VaaS podía proveer salas de videoconferencia bajo demanda pero una vez desplegado en el Cloud, no tenía definido un esquema de escalabilidad que permitiera adaptarse de forma dinámica a las variaciones en la demanda de usuarios. Había por lo tanto que ir un paso más allá y considerar ciertos factores adicionales de diseño.

ⁱ<http://dev.w3.org/html5/spec>

ⁱⁱ<http://lynckia.com/licode>

1.2. OBJETIVOS

Es entonces cuando me planteo las primeras preguntas a resolver en esta tesis:

1. *¿Cómo debe diseñarse un sistema de multiconferencia Cloud para escalar de forma dinámica a la variación en la demanda?*
2. *¿Cómo y en base a qué criterios debe realizarse dicha escalabilidad?*

Como resultado de las respuestas a estas cuestiones, una de las decisiones tomadas durante el transcurso de la investigación es pasar a un modelo de servidores de videoconferencia distribuidos. Como explicaré más adelante, esto favorece la eficiencia y flexibilidad de los despliegues permitiendo además introducir criterios de escalabilidad avanzados. Frente al nuevo modelo distribuido surgen las siguientes preguntas:

3. *¿Cómo debe diseñarse un sistema de videoconferencia distribuido para permitir la asignación inteligente de recursos?*
4. *¿Cómo y en base a qué criterios debe realizarse dicha asignación de recursos?*

La motivación para el desarrollo de esta tesis doctoral consiste por lo tanto en dar respuesta a estas preguntas. A mediados de 2012 comienzo a trabajar en la investigación que me permitirá ir encontrándolas y que tiene como consecuencia los objetivos que enumero a continuación.

1.2 Objetivos

El objetivo global de este trabajo es proponer una serie de técnicas y arquitecturas que permitan realizar de una forma más eficiente y dinámica el despliegue y operación de sistemas de videoconferencia en entornos Cloud. Para ello es imprescindible diseñar modelos de escalabilidad que permitan adaptar los recursos a la demanda de usuarios. Y cuando se trata de sistemas distribuidos, modelos que permitan asignar los recursos de manera inteligente y acorde a los requisitos de cada escenario.

Este objetivo puede dividirse en los siguientes puntos:

- *Identificar las ventajas y los retos que supone desplegar un sistema de videoconferencia en el Cloud.* Este análisis es fundamental a la hora de diseñar las arquitecturas de despliegue y los modelos de escalabilidad.

- *Proponer e implementar un modelo de arquitectura que permita escalar un sistema de videoconferencia Cloud de forma dinámica y acorde a la demanda.* Este modelo debe permitir que el sistema escale hacia arriba y hacia abajo. Es decir, debe permitir añadir y quitar recursos al sistema según la demanda de usuarios de cada momento.
- *Identificar las características específicas de un sistema de videoconferencia distribuido y sus necesidades a la hora de tomar decisiones de asignación de recursos.* Este estudio debe tener en cuenta las soluciones existentes en otros ámbitos tecnológicos para realizar una comparativa con el escenario objetivo y extraer posibles referencias de diseño.
- *Diseñar e implementar un modelo que habilite la asignación inteligente de recursos en un sistema de videoconferencia distribuido.* La forma de asignar los recursos debe poder adaptarse a las necesidades de cada escenario concreto y debe estar basada en el estado global del sistema, tanto en sus características definidas como en su estado de en cada momento.
- *Proponer y comparar estrategias de asignación de recursos de videoconferencia en entornos distribuidos.* Estas estrategias deben favorecer un uso eficiente de la infraestructura y tener en cuenta las necesidades específicas que dicte cada caso de uso. Además debe proponerse una metodología para evaluar la idoneidad de estas estrategias en cada escenario.

1.3 Metodología

Para abordar los objetivos planteados anteriormente ha sido necesario seguir una metodología de trabajo que viene marcada en su mayor parte por los distintos proyectos de investigación en los que he participado. El hilo conductor es el proyecto de código libre Licode, en el que he ido implementando y evaluando cada una de las soluciones propuestas en esta tesis. Licode constituye el núcleo de software que hemos utilizado en el resto de proyectos como base para los nuevos desarrollos requeridos. Estos proyectos, de ámbito privado, nacional y europeo, son los que han ido marcando por lo tanto los requisitos abordados. Sin embargo, en algunas ocasiones, los retos planteados surgen por mi propia inquietud a la hora de buscar mejores soluciones y no estrictamente por necesidades impuestas en ellos.

1.3. METODOLOGÍA

Durante los primeros compases de la investigación la tarea principal fue constituir lo que sería una primera versión del mencionado núcleo de Licode. La referencia de diseño provenía de antiguas plataformas desarrolladas en el departamento como Isabel, VaaS o Marte. Sin embargo, el nuevo sistema se desarrollaría utilizando una nueva tecnología (WebRTC) y la orientación en este caso era desde el inicio hacia un sistema Cloud escalable. Por ello el nuevo diseño hacia plantearse también nuevos retos. Y con estos retos tendría lugar el comienzo de mis investigaciones.

Cabe comentar que durante ese primer periodo los anteriores sistemas Isabel y VaaS aún se utilizaban en proyectos como GLOBALⁱ (*Global Linkage Over Broadband Links*) o SmartStreamⁱⁱ, en los que realicé alguna colaboración puntual. No obstante, los primeros proyectos que servirían como banco de pruebas y validación para mis resultados serían LTextremeⁱⁱⁱ y MCS5 (*Multi-screen Cloud Services in HTML5*). Ambos sirvieron para validar las primeras versiones maduras de Licode, el primero en un escenario LTE y el segundo en un demostrador para reuniones interactivas y escenarios de tele-asistencia.

Más adelante, y ya con el modelo de servidores distribuidos implementado en Licode, incluí un servicio de tele-reuniones y webinarios en el proyecto FIWARE^{iv} para realizar reuniones de trabajo y conferencias de diseminación. Uno de los proyectos más recientes, aún en curso en el momento de escribir estas líneas, es *EasyVideo*. Se trata de una plataforma de colaboración utilizada en el ámbito bancario en la que he realizado un despliegue de servidores de videoconferencia distribuidos para atender la demanda de usuarios en diferentes continentes. La asignación de recursos a estos servidores se realiza siguiendo el modelo propuesto en esta tesis y que explicaré más adelante.

Por último, a largo de estos años he participado en una serie de proyectos de menor envergadura que los anteriores pero igual de importantes a la hora de validar los resultados de mi trabajo. En ellos he realizado tareas de despliegue, configuración y mantenimiento de la infraestructura de videoconferencia para pequeñas empresas de ámbito nacional e internacional. En ocasiones también ha sido necesario implementar algunos módulos adicionales al núcleo de Licode para cubrir necesidades específicas de sus casos de uso.

ⁱhttp://cordis.europa.eu/project/rcn/86659_en.html

ⁱⁱ<http://www.smartstream.es>

ⁱⁱⁱ<http://ltextreme.org>

^{iv}<http://www.fiware.org>

1.4 Proyectos relacionados

En esta sección hago una descripción más detallada de los proyectos mencionados anteriormente. Estos proyectos han sido fundamentales para capturar requisitos en sistemas reales y validar las soluciones propuestas.

1.4.1 Licode

Licode es el un proyecto de código libre cuyo objetivo es proveer comunicaciones avanzadas en tiempo real basadas en el estándar WebRTC. El núcleo de Licode es una Unidad de Control Multipunto (*Multipoint Control Unit (MCU)*) totalmente compatible con este estándar. También provee un módulo de autorización y un API de cliente para el desarrollo sencillo de aplicaciones web. Está diseñado para ser desplegado en entornos Cloud distribuidos de forma que sea inmediato añadir y liberar recursos disponibles para adaptarlos a la variación en la demanda.

Como he introducido anteriormente, este proyecto es fundamental para el desarrollo de mi tesis ya que se trata del hilo conductor y banco de pruebas de todas mis investigaciones. Mi trabajo en Licode ha consistido en la contribución como uno de los tres desarrolladores principales así como el mantenimiento y gestión de la comunidad de software libre. Además he diseñado e implementado cada uno de los módulos propuestos en este documento.

1.4.2 LTEXtreme

El objetivo de LTEXtreme consiste en el diseño y desarrollo de una arquitectura de sistema que optimice los servicios multiusuario y multimedia utilizando los estándares del 3GPP, LTE y LTE-Advanced. LTEXtreme es un proyecto financiado por el Ministerio de Economía y Competitividad.

Se busca mejorar de una forma clara la experiencia del usuario en distintos tipos de aplicaciones, permitiendo medir en tiempo real la calidad, consiguiendo que aplicaciones ofimáticas, videojuegos y otros sistemas interactivos puedan reaccionar ante la información obtenida de estas mediciones, optimizando su funcionamiento.

Las tareas del proyecto pueden dividirse en dos grupos. Por un lado las relacionadas con el estrato radio y por otro lado las de investigación y desarrollo en el nivel de aplicación. Mi participación fue lógicamente en el nivel de aplicación y en concreto aplicando tecnologías WebRTC en maquetas LTE.

1.4. PROYECTOS RELACIONADOS

El objetivo principal era comprobar las ventajas del uso de LTE en la transmisión de flujos multimedia en un entorno multidispositivo a través de la Web. Pero también sirvió para realizar un despliegue de Licode en un entorno con características diferentes a los que había venido realizando hasta entonces. La adaptación a las condiciones de red es fundamental para asegurar la eficiencia del sistema y aunque en este proyecto no se realizaran simulaciones de escalabilidad la experiencia obtenida fue muy útil para el diseño de las soluciones posteriores.

1.4.3 MCS5

El proyecto MCS5 (*Multi-screen Cloud Services in HTML5*) es un proyecto realizado por el grupo de investigación GING para Telefónica España en el ámbito de innovación. Consiste en el desarrollo de un servicio multi dispositivo de televisión social, videovigilancia, teleasistencia y juegos que funciona utilizando el nuevo estándar de comunicaciones web HTML5 WebRTC.

Mi tarea en este proyecto fue la realizar un despliegue de Licode acorde a los requisitos del proyecto. En su momento este despliegue supuso un reto por los múltiples tipos de redes desde las que se realizaría el acceso al sistema. Además, al tratarse de un sistema que contemplaba la videovigilancia y la teleasistencia, hubo que adaptar la plataforma para su acceso desde cámaras IP, televisiones y dispositivos móviles.

1.4.4 FIWARE, FI-XIFI y FICORE

FIWARE (Plataforma central del Internet del futuro, del inglés *Future Internet Core Platform*) es un proyecto europeo que se dedica a la creación de una nueva infraestructura de servicios basada en componentes básicos genéricos y reutilizables denominados Capacitadores Generales (*Generic Enablers, GE*) para la Internet del Futuro. FICORE es una continuación de FIWARE que persigue los mismos objetivos y FI-XIFI pretende extender la infraestructura a diferentes nodos por toda Europa.

Realicé un despliegue de Licode accesible por los diferentes miembros del consorcio para la realización de tele reuniones. Algunas de las charlas y conferencias de diseminación del proyecto se realizan también a través de este sistema. Además Licode es uno de los Habilitadores Genéricos aceptados en FIWARE.

1.4.5 EasyVideo

EasyVideo es un proyecto realizado por el grupo de investigación GING para Produban, del grupo Santander. Se trata de una plataforma de colaboración utilizada por los miembros del grupo para reuniones a distancia con capacidades como la compartición de ficheros o visualización de escritorio remoto.

EasyVideo utiliza Licode en modo distribuido para permitir el acceso a usuarios desde varias partes del mundo optimizando la calidad de sus conexiones. Los servidores están desplegados en la infraestructura interna del banco y se aplican algoritmos de asignación de recursos inteligentes para decidir dónde alojar cada nueva conexión en función de sus características.

Este proyecto ha sido fundamental a la hora de validar los resultados de la última parte de este trabajo, las estrategias de asignación de recursos. Como se verá más adelante, esas estrategias pueden basar sus decisiones en el estado de cada servidor disponible o en la localización de los clientes. Ambos criterios han sido aplicados en este proyecto.

1.4.6 Otros proyectos

Además de los proyectos descritos anteriormente, existen una serie de proyectos con pequeñas y medianas empresas que han servido como campo de pruebas y validación para algunas de las contribuciones de mi investigación. Estos proyectos son de índole muy variada. Un ejemplo es una plataforma para tutorización remota en la que la grabación de las sesiones de videoconferencia era fundamental a la hora de su análisis posterior por los tutores. También participé en un proyecto de una empresa alemana que ofrecía un servicio de entretenimiento con varios miles de usuarios en el que el despliegue escalable era fundamental dada la gran variabilidad de uso. Como último ejemplo pondré el de una universidad estadounidense que realizaba clases remotas para alumnos de todo el mundo. En este caso el despliegue distribuido facilitaba la gestión y programación de recursos antes de producirse cada una de las clases.

1.5 Estructura del documento

1.5.1 Capítulo 2: Estado del Arte

El siguiente capítulo sirve para poner en contexto las tecnologías y disciplinas relacionadas con mis trabajos de investigación. Incluye un recorrido por algunos de los sistemas de

1.5. ESTRUCTURA DEL DOCUMENTO

videoconferencia más relevantes a fecha de hoy explicando sus características y campos de aplicación. Realizo también un pequeño repaso por las tecnologías de Cloud Computing existentes y por los métodos de asignación de recursos en sistemas distribuidos.

1.5.2 Capítulo 3: Videoconferencia escalable en infraestructuras Cloud

El tercer capítulo presenta el primer bloque de contribuciones de esta tesis. Se trata del diseño de un sistema de videoconferencia escalable pensado para su despliegue en infraestructuras Cloud. Para ello hay que tener en cuenta cómo debe comportarse el sistema frente a cambios en la demanda en sentido creciente y decreciente. Propongo por lo tanto un modelo que permite monitorizar el estado de los recursos disponibles para saber cuándo es necesario añadir más. Por otra parte, cuando se requiere eliminar recursos es fundamental realizar una redistribución de las conexiones ya establecidas. Para ello diseño un mecanismo de movilidad de conexiones multimedia entre servidores.

1.5.3 Capítulo 4: Asignación de recursos en sistemas de videoconferencia distribuidos

El capítulo 4 se centra en mis contribuciones al respecto de la asignación de recursos cuando el sistema de videoconferencia está distribuido entre varios servidores. Para ello propongo un modelo que habilita la monitorización y reporte del estado de dichos servidores. Basándose en este estado, el modelo permite la configuración de algoritmos inteligentes de toma de decisiones acerca de donde asignar las nuevas conexiones. Estos algoritmos son personalizables para adaptarse a las estrategias necesarias en cada caso de uso.

1.5.4 Capítulo 5: Validación y Resultados

En este capítulo presento los resultados de la validación de las propuestas explicadas en este documento. Esta validación está enmarcada en los proyectos citados anteriormente. Además hago un repaso por los diferentes resultados desde el punto de vista de diseminación. Esto incluye los trabajos publicados en revistas y congresos de investigación.

1.5.5 Capítulo 6: Conclusiones

En el último capítulo de este documento realizo un pequeño resumen del trabajo realizado extrayendo las principales conclusiones y verificando que se han cumplido los objetivos propuestos al inicio. También hago un ejercicio de análisis de las posibles líneas de investigación futuras abiertas a raíz de esta tesis.

Capítulo 2

Estado del Arte

En esta sección se realiza un repaso de aquellas disciplinas y tecnologías que tienen relación directa con los temas propuestos en la tesis. Se pretende que el lector tenga una perspectiva global del estado actual de dichas disciplinas para entender mejor de qué manera mis propuestas contribuyen al estado del arte actual.

Dado que el tema conductor de esta tesis son los sistemas de videoconferencia, en primer lugar haré una descripción de las características más relevantes de este tipo de sistemas así como su estado actual en el entorno Web. Como he explicado antes, la adopción de la videoconferencia en la Web es una de las razones fundamentales de que aumente significativamente el número de aplicaciones y servicios en los que ésta se encuentra presente y por ende el número de usuarios que requieren de sus ventajas. Este aumento propicia que sea recomendable el despliegue de los sistemas en entornos Cloud escalables.

Por tanto la siguiente tecnología a analizar es el Cloud Computing. En este caso haré una pequeña introducción para resolver la pregunta más inmediata, *¿qué es el Cloud Computing?*. Pasaré entonces a repasar los actuales proveedores de Cloud en todos sus niveles de abstracción.

La última consecuencia de esta evolución hacia la escalabilidad es la distribución de los servidores de videoconferencia en múltiples servidores. Como explicaré más adelante, esta distribución consigue dar un paso de gigante hacia una mayor flexibilidad de los despliegues y con ello un ahorro de recursos sustancial. Pero al distribuir un servicio es crucial decidir cómo deben repartirse los recursos entre los servidores disponibles. No es una tarea trivial y menos aún cuando hablamos de sistemas de tiempo real. Existe una amplia literatura al respecto y al final de esta sección realizaré un análisis de algunos de los métodos de programación de recursos existentes actualmente.

2.1 Videoconferencia multiusuario

2.1.1 Introducción a la videoconferencia multiusuario

Un sistema de videoconferencia multiusuario es aquel en el que participan más de dos usuarios intercambiando sus flujos multimedia (audio, video y/o datos). Es por lo tanto un escenario más complejo que una tradicional llamada uno a uno cuyo objetivo es permitir la colaboración en tiempo real entre grupos de personas. Los casos de uso más típicos son la realización de reuniones a distancia, la impartición de clases o conferencias interactivas o los juegos masivos en red.

Los orígenes de la comunicación multipunto coinciden con los orígenes del dispositivo que la hace posible, la Unidad de Control Multipunto MCU (*Multipoint Control Unit*). Para realizar una comunicación entre varios participantes es necesario que cada uno envíe sus flujos multimedia al resto de participantes y a su vez reciba un flujo por cada uno de ellos. Ésto supone una topología en estrella en la que cada participante está conectado a todos los demás. En una sesión de N participantes, cada uno de ellos estaría enviando $N - 1$ flujos y recibiendo otros tantos. Obviamente este intercambio supone un ancho de banda considerable que puede llegar a ser inasumible en determinadas condiciones de red.

Figura 2.1 : Comparación de topologías en videoconferencia multiusuario

La solución consiste en transformar esta topología de malla en una topología de estrella cuyo nodo central es la MCU, como puede verse en la Figura 2.1. Es en los años 1990 cuando el ITU-T propone la primera definición de este componente [ITU-T 1990]: *"Una pieza de equipamiento localizada en un nodo de la red o en un terminal que recibe varios canales a través de puertos de*

2.1. VIDEOCONFERENCIA MULTIUSUARIO

acceso y, siguiendo ciertos criterios, procesa señales audiovisuales y las distribuye a los canales conectados".

Más adelante, en 1991, INRIA propone un sistema de videoconferencia para Internet basado en multicast para conseguir comunicación multiusuario [Turletti 1996] utilizando un codificador software [Turletti 1993] para evitar la necesidad de hardware específico. En 1992 [Clark 1992] describe una MCU conmutadora de vídeos, dos años más tarde aparece la primera referencia a una MCU mezcladora [Lei 1994] y a una transcodificadora [Willebeek-LeMair 1994].

Con esto tendríamos las principales funcionalidades que hoy en día atribuimos a estos servidores:

- *Distribución:* es la operación básica que realiza una MCU. Se trata de reenviar un flujo de datos desde un participante (comunmente llamado *publisher*) al resto (*subscribers*). Estos *subscribers* reciben cada stream una única vez y los *publishers* solamente lo envían una vez. De esta manera ahorramos ancho de banda en los clientes y conseguimos una mejora en el rendimiento global del sistema.
- *Transcodificación:* en ocasiones se utilizan distintos tipos de dispositivos en una misma sesión de videoconferencia. Esta heterogeneidad puede implicar la necesidad de realizar conversiones de códecs en los flujos multimedia para poder intercomunicar todos los dispositivos. Por otra parte, en caso de heterogeneidad en las redes, codificar los flujos en distintas calidades y tamaños permite que adaptemos el flujo enviado a cada dispositivo en función de las condiciones de su red de acceso. En escenarios en los que queremos exportar los flujos a pasarelas o componentes externos también es imprescindible esta funcionalidad.
- *Composición:* una MCU permite generar un único vídeo o audio a partir de un conjunto de ellos. De esta manera podemos disminuir el ancho de banda de los flujos de salida y hacer que los clientes reciban menos carga.
- *Grabación:* una de las características más demandadas es la grabación de las sesiones para dotar al sistema de persistencia y poder realizar la posterior reproducción de los contenidos almacenados. Como la MCU recibe todos los flujos presentes en una determinada sesión, puede realizar un volcado a disco de forma individual o generando una mezcla como se ha explicado anteriormente.

Por lo tanto, una MCU no resuelve únicamente el problema de ancho de banda derivado de interconectar todos los usuarios de una videoconferencia múltiple. También nos permite realizar tareas avanzadas que sin un componente central serían mucho más arduas si no imposibles.

2.1.2 Componentes de un sistema de videoconferencia multiusuario

Podemos determinar una serie de componentes que son comunes en la mayoría de los sistemas de videoconferencia existentes: el componente de control, el de señalización y el de medios [Firestone 2007]. A continuación hago una descripción de las principales funciones de cada uno de estos componentes. Aprovecharé para comentar en cada caso qué tecnologías y protocolos están implicados en su funcionamiento.

Control

El componente de control se encarga de realizar todas las funciones relacionadas con el control de las conferencias y con el control de medios. El control de una conferencia consiste principalmente en gestionar los usuarios conectados a una sesión y las llamadas entre ellos. Por ejemplo, cuando un usuario se conecta a una determinada sesión, el componente de control se encarga de avisar al resto de su presencia para que comience a realizarse el mecanismo de conexión. De igual forma, este módulo gestiona las desconexiones o posibles fallos en los usuarios para comunicar a la capa de media que tome las determinaciones oportunas.

Entre las tareas de control de la conferencia también se encuentra la gestión de recursos disponibles. Hay sistemas en los que se permite una reserva previa de recursos de videoconferencia. En ese caso, este componente es el que se encarga de realizar la reserva de dichos recursos para su posterior asignación a los solicitantes.

Por otra parte, este componente se encarga de gestionar los medios (audio, vídeo o datos). Al establecerse una sesión y en función de las características de los clientes, se determinan las características de los flujos multimedia y en función de ellos este componente controlará la capa de medios para indicar cómo debe tratarlos. Algunos ejemplos de estas características son la calidad de los vídeos y audios intercambiados, los códecs utilizados, etc.

2.1. VIDEOCONFERENCIA MULTIUSUARIO

Señalización

La capa de señalización es fundamental a la hora de establecer la comunicación entre dos dispositivos. Estos dispositivos pueden ser dos clientes (en caso de comunicación punto a punto), un cliente y una MCU o incluso dos MCUs. Durante la fase de señalización los dispositivos intercambian una serie de mensajes que les permiten ponerse de acuerdo en las condiciones de la comunicación. Un ejemplo de estos mensajes son los definidos por el Protocolo de Descripción de Sesión o SDP (*Session Description Protocol*) [Handley 1998], que permiten negociar diferentes parámetros de una sesión tales como resoluciones o codificaciones soportadas.

A la hora de establecer la comunicación entre dos dispositivos, es fundamental la conectividad entre ellos. En ocasiones, los dispositivos se encuentran en redes privadas o detrás de cortafuegos que dificultan la interconexión con dispositivos en otras redes. Uno de los componentes que encontramos con más frecuencia como barrera para establecer estas conexiones es el *Network Address Translation* (NAT) [Srisuresh 1999].

Se trata de un componente que transforma direcciones IP privadas en públicas y viceversa. Gracias a él, tras una única IP pública podemos establecer una red privada con multitud de direcciones IP privadas. Por lo tanto varios dispositivos están identificados a través de la misma dirección IP pública. El componente NAT se encarga de realizar la traducción entre ambas. Esto es una ventaja dada la escasez de direcciones privadas (cuando hablamos de IP versión 4) y no supone ningún problema cuando los dispositivos de la red privada se conectan a servidores en Internet (red pública). Sin embargo supone un impedimento a la hora de realizar conexiones entrantes hacia las redes privadas ya que desde Internet no podemos identificar de forma unívoca la dirección de cada dispositivo. Y este tipo de conexiones son fundamentales para establecer sesiones de videoconferencia.

Para solventar este problema existen varios protocolos que pueden utilizarse de forma aislada o conjuntamente, dependiendo del tipo de NAT o de la configuración de las redes:

- *Session Traversal Utilities for NAT (STUN)*: Es la solución más sencilla para atravesar NAT. Como se explica en su definición [Rosenberg 2008] (nueva versión de la ya obsoleta [Rosenberg 2003]), permite a un cliente saber si se encuentra "detrás" de uno de estos dispositivos. Para ello el cliente envía una serie de paquetes a un servidor STUN situado en Internet (red pública).

El servidor STUN es capaz de saber qué tipo de NAT se encuentra entre ellos utilizando un algoritmo de caracterización que consiste en unos tests sucesivos con la dirección del cliente. Sin embargo este protocolo únicamente funciona con los tipos de NAT menos restrictivos (*Full-cone NAT*, *Address-restricted Cone NAT* y *Port-restricted Cone NAT*) y no con el NAT más común, el simétrico (*Symmetric NAT*). STUN por sí mismo no constituye por lo tanto una solución absoluta para atravesar NAT.

- *Traversal Using Relay NAT (TURN)*: la especificación de TURN [Mahy 2010] describe un protocolo que permite a un cliente dado enviar datos a través de un servidor (llamado normalmente servidor TURN) que se encuentra en una red pública. De esta manera el cliente utiliza una dirección pública para conectarse con el resto de clientes. Para dichos clientes el primero se encuentra ficticiamente en dicha dirección pública. Es el servidor TURN el encargado de reenviar los flujos de datos a la verdadera IP del cliente, sólo conocida por él.

Con esto podría parecer que el problema está resuelto del todo. Sin embargo el hecho de que todos los flujos atraviesen este servidor hace que su mantenimiento sea muy caro para los proveedores. Así surge el siguiente protocolo, que combina los dos anteriores para ofrecer la solución definitiva en todos los escenarios posibles.

- *Interactive Connectivity Establishment (ICE)*: este protocolo describe un procedimiento [Rosenberg 2010] mediante el cual, utilizando STUN y TURN, es posible establecer comunicación entre dos dispositivos situados en cualquier tipo de red pública o privada. Para ello define una serie de pasos que aseguran encontrar la vía de comunicación más corta entre dichos dispositivos. Solamente se utilizará un servidor de *relay* TURN en el caso en que sea estrictamente necesario.

Para establecer comunicación entre ellos, dos dispositivos A y B siguen los siguientes pasos, explicados a grandes rasgos

1. Cada dispositivo descubre sus pares IP/puerto que pueden llegar a ser utilizados para recibir datos. Estos pares se llaman *candidatos* y para obtenerlos, los dispositivos se apoyarán en un servidor STUN, que les permitirá conocer su IP pública. Además, si existen servidores TURN o relays, las IPs de estos posibles reencaminadores de flujos

2.1. VIDEOCONFERENCIA MULTIUSUARIO

se añadirán a los candidatos.

2. Los candidatos de ambos dispositivos se codifican en un SDP y son intercambiados a través de un medio no definido por el protocolo. ICE define una serie de normas para priorizar unos candidatos sobre otros y transmitir esta información en dicho SDP.
3. Cada dispositivo empareja sus candidatos locales con los recibidos en el SDP y envía pruebas (*checks*) de conexión, por orden de prioridad, utilizando mensajes STUN. Cuando un dispositivo recibe un *check* responde con la IP y el puerto que percibe como origen. Si la respuesta se recibe, el *check* ha sido un éxito.
4. Para coordinar que los dos terminales elijan el mismo par de candidatos ICE define unos roles para los agentes de manera que uno de los dos decide cuándo parar de hacer *checks*.

Medios

El componente de medios es el que se encarga de realizar la gestión de los flujos multimedia (audio, vídeo, datos). Como se ha introducido antes, esta gestión puede consistir en una simple redirección de los paquetes o en tareas más complejas como la composición de vídeos o audios, la transcodificación de medios o la grabación/reproducción de sesiones. Posibles submódulos son por tanto:

- **Encapsulador:** Se encarga de preparar los flujos de datos dividiéndolos en paquetes para enviarlos a la red o de recibirlos y volver a ensamblarlos en destino.
- **Transcodificador:** Cambia las características de los flujos para adaptarse a unas circunstancias específicas. Por ejemplo puede bajar la calidad de un flujo multimedia para que pueda ser enviado a un cliente con mala conexión.
- **Componedor/Mezclador:** Permite agregar varios flujos en uno solo. Aquí se realizan operaciones como mezclar todos los audios de una conferencia para enviarlos a un cliente que no es capaz de recibir o reproducir varios simultáneamente.
- **Reproductor/Grabador:** En un cliente, este módulo se encargará de capturar los flujos provenientes de la webcam y de reproducir los que vengan a través de la red.

Como es de suponer, en la capa de medios intervienen todo tipo de mecanismos de codificación de audio y vídeo. Pero a parte de la gestión y procesamiento interno de los flujos de medios, esta capa también se encarga del envío y recepción de dichos flujos. Debe implementar en consecuencia protocolos de transporte. El más común es el protocolo de transporte tiempo real o RTP (*Real-time Transport Protocol*) [Group 1996]. Su compañero de control es RTCP (*RTP Control Protocol*) [Ott 2006], que complementa a RTP permitiendo mandar información sobre los flujos multimedia tales como estadísticas, retardos o mensajes de sincronización.

Una vez establecidas mediante señalización las direcciones de envío y recepción de flujos multimedia, se utiliza RTP para encapsular los datos de la comunicación propiamente dicha. RTP se envía sobre UDP (*User Datagram Protocol*) y no garantiza la llegada ni el orden de los paquetes. Sin embargo, incluye información para detectar esas posibles pérdidas o desórdenes. RTP y RTCP se implementan siguiendo un *profile* o perfil que define cómo utilizar los campos de los paquetes así como las extensiones de ambos protocolos que se deben utilizar en un determinado ámbito.

2.1.3 Videoconferencia en la Web

Si bien los modelos propuestos en esta tesis son aplicables a sistemas de videoconferencia de diversas índoles, son los basados en la Web los que han motivado de alguna manera el proponerlos. Como se ha introducido anteriormente, el paso de la videoconferencia a la Web ha propiciado que el número de usuarios que hacen uso de ella aumente de forma radical. Y no sólo eso, si no que la posibilidad de acceder a aplicaciones y servicios de videoconferencia desde una gran variedad de dispositivos personales (ordenadores personales, teléfonos móviles, tabletas) hace que ese gran número de usuarios sea además variable e imprevisible. Y con esta variabilidad surge uno de los retos de esta investigación, la adaptabilidad dinámica a la demanda.

Por ello, y para poner el contexto al lector, parece necesario realizar un breve repaso por los sistemas de videoconferencia Web existentes en la actualidad. No sin antes describir la Web 2.0, concepto indispensable para el mencionado salto de las comunicaciones en tiempo real a los sistemas Web.

La Web 2.0

En los orígenes de la Web, los usuarios consumían contenidos en servicios unidireccionales. La mayoría de los portales que podíamos encontrar en Internet, servían páginas estáticas que ofrecían

2.1. VIDEOCONFERENCIA MULTIUSUARIO

una serie de contenidos actualizados de forma periódica por sus proveedores.

Con el paso de los años, estos servicios Web evolucionan de tal forma que en algunos portales comienza a haber un retorno por parte del usuario. La comunicación empieza a ser bidireccional de manera que los contenidos no son generados únicamente por los proveedores del servicio si no que los propios usuarios aportan los suyos. Empieza a concebirse entonces de una Web colaborativa creada por proveedores y usuarios.

En 2004 se dan las primeras referencias a esta nueva Web como la Web 2.0, con Tim O'Reilly [O'reilly 2007] como principal divulgador. Según [Andersen 2007] la Web 2.0 establece siete ideas fundamentales:

1. *Producción Individual y contenido generado por usuarios*: Los usuarios son los responsables de generar contenido, muchas veces de manera individual y en muchos casos sin necesidad de pasar por procesos *editoriales*. Un ejemplo de este punto serían los blogs.
2. *Domar el poder de la multitud*: Se trata de aprovechar la colaboración para la generación de mejores contenidos. El ejemplo típico en este sentido es la Wikipedia ⁱ.
3. *Datos a una escala épica*: En la sociedad de la información en que vivimos, los datos son cada vez más importantes. En su artículo original O'Reilly se refiere a empresas como Google y al valor que obtienen con el manejo de los datos. Gran parte de estos datos se generarán por el uso de los servicios de Internet.
4. *Arquitectura de participación*: Los servicios creados deberán ser diseñados de manera que puedan mejorar y facilitar la participación masiva de usuarios. En consecuencia, los servicios serán mejores a medida que más usuarios los utilicen.
5. *Efecto red*: La Web está formada por nodos interconectados (páginas HTML enlazadas) y está construida sobre tecnologías de comunicación y protocolos de Internet. Hay que tener en cuenta por un lado la red tecnológica establecida que da cabida a miles de millones de usuarios y por otro, las conexiones establecidas entre usuarios y entre los usuarios y los contenidos. Un ejemplo de cómo entender e interpretar estas conexiones es el de la Larga Cola (*Long Tail*), descrito en [Anderson 2006].

ⁱ<http://www.wikipedia.org>

6. *Orientación hacia lo abierto*: La Web siempre ha tenido una fuerte tradición hacia lo abierto y la libre circulación de información. En la Web 2.0 esto sigue presente: licencias abiertas, estándares, software libre, etc. Esto permite la colaboración y la mejora colectiva tanto de los contenidos como de las herramientas para crearlos y distribuirlos.

Teniendo en cuenta estas ideas podríamos decir que el concepto de la nueva Web se refiere más a un cambio social y en el modo de uso de las herramientas ya existentes. Sin embargo, para habilitar esta nueva forma de interacción por parte del usuario se necesitan también avances tecnológicos no existentes hasta entonces. Por lo tanto, el concepto de Web 2.0 implica también una serie de novedades técnicas en la Web.

Estas novedades técnicas están presentes tanto en los servidores Web como en los navegadores [Fraternali 2010], cada vez más complejos. En este momento se pasa a hablar de aplicaciones o servicios Web en lugar de simples páginas Web, más referidas a las antiguas páginas estáticas. Se crea el concepto de Aplicaciones Ricas de Internet o RIAs (*Rich Internet Applications*), que necesitan de tecnologías como AJAX (*Asynchronous JavaScript an XML*) para hacer peticiones asíncronas permitiendo dinamizar las páginas web y de multitud de soluciones basadas en *plug-ins* o complementos del navegador.

Una de las funcionalidades que habilitan estas nuevas tecnologías es el tratamiento de contenido multimedia en los navegadores. En un primer momento reproducción de canciones y vídeos pero más adelante captura de imágenes y transmisión de contenidos en tiempo real. Es aquí donde parte nuestro escenario y las dos tecnologías Web clave implicadas, Flash y WebRTC.

Adobe Flash Player

Adobe Flash es un conjunto de aplicaciones multimedia desarrolladas y distribuidas por Adobe Systems desde 1996. Se trata de una plataforma RIA propietaria para añadir funcionalidades interactivas a las aplicaciones web tales como animaciones, vídeo, audio, etc. Flash Player tiene soporte para un lenguaje de programación interpretado conocido como ActionScript basado en el estándar ECMAScript. Desde su origen ActionScript ha pasado de ser un lenguaje muy básico a un lenguaje avanzado con soporte de programación orientada a objetos, comparable en funciones y uso al lenguaje JavaScript.

Originalmente creado para mostrar animaciones vectoriales en 2 dimensiones, ha pasado a utilizarse para crear aplicaciones Web que incluyen flujo de audio y video e interactividad. La

2.1. VIDEOCONFERENCIA MULTIUSUARIO

utilización de gráficos vectoriales le permite disminuir el ancho de banda necesario para la transmisión y, por ende, el tiempo de carga de la aplicación.

Flash utiliza el protocolo RTMP tanto para el transporte de los flujos como para el envío de mensajes. La señalización viene implícita en la plataforma Flash, con un esquema de publicación y suscripción en lugar de llamadas propiamente dichas.

El formato de vídeo utilizado por Flash es FLV (Flash Video), un formato de archivo principalmente diseñado y utilizado para la transmisión de video y audio a través de Internet. Generalmente, la codificación utilizada es una variante de H.263 denominada Sorenson Spark. Pero a partir de la versión 8 existe la posibilidad de usar On2 VP6 permitiendo una mayor calidad con menor ancho de banda consumido. El contra es que requiere una mayor potencia de proceso. Además, desde la versión 9.3 Flash Player permite la reproducción (no la codificación) de H.264, principalmente para mejorar la calidad en los sistemas de vídeo bajo demanda que están floreciendo en Internet.

Para el audio lo más común es utilizar MP3 con la excepción del caso de captura de audio por medio de un micrófono durante la ejecución del Flash Player en cuyo caso existen dos opciones de codificación: Nellymoser o Speex. A partir de 9.3 se incluyó AAC junto a H.264.

Adobe Flash ha venido utilizándose de forma mayoritaria en las aplicaciones de comunicación en tiempo real para la transmisión de vídeo y audio hasta la aparición de HTML5 y WebRTC.

WebRTC

WebRTC [Bergkvist 2012, Loreto 2012] es un estándar en desarrollo por el Internet Engineering Task Force (IETF) y el World Wide Web Consortium (W3C) que pretende definir un framework, protocolos e interfaces de programación que proveerán comunicaciones interactivas y en tiempo real de audio, video y datos a las aplicaciones en los navegadores Web.

Como se explica en [Jennings 2013], la arquitectura en las aplicaciones WebRTC son peer-to-peer de tal forma que únicamente es necesario un servidor externo a los navegadores de los clientes para establecer la comunicación intercambiando los mensajes de señalización al comienzo de la misma. Los APIs de los que consta el protocolo son los siguientes:

- **getUserMedia:** se utiliza para acceder a los recursos multimedia del usuario como su cámara o su micrófono. El resultado será un `MediaStream`.

- **MediaStream:** es un conjunto de pistas de flujos de audio o vídeo con los recursos multimedia del usuario.
- **RTCPeerConnection:** representa un canal de conexión con otro cliente. Un cliente añade a un RTCPeerConnection uno o más MediaStreams para compartirlos con el otro cliente.

Al añadir uno de estos MediaStreams a la conexión comienza la negociación con el otro cliente mediante la cual se negocian ciertos parámetros de la sesión como el formato de los datos (códecs de audio y vídeo), la resolución o el ancho de banda. Esta negociación debe realizarse mediante medios externos al protocolo y consiste en el intercambio de unos mensajes SDP del tipo *offer/answer*.

El protocolo implementa ICE para poder realizar la conexión en escenarios en los que los clientes se encuentren tras un NAT. Una vez establecida la conexión el transporte de datos se realiza sobre SRTP [Baugher 2004], la versión segura de RTP y gracias a RTCP los clientes pueden obtener feedback sobre el estado de la conexión, las posibles pérdidas, etc. Además para la gestión de claves de SRTP se utiliza el protocolo DTLS-SRTP, una extensión de DTLS.

En la Figura 2.2 vemos la arquitectura definida por el estándar. En estos momentos dicho estándar se encuentra cerca de la versión 1.0. Sin embargo está en constante evolución por lo que podría sufrir ligeras modificaciones sobre las características expuestas en este documento.

2.1.4 Sistemas de videoconferencia multiusuario

Una vez explicadas las características de los sistemas de videoconferencia entre múltiples participantes y las principales tecnologías involucradas en su funcionamiento, en este apartado realizo un recorrido por algunas de las soluciones de multiconferencia existentes en la actualidad. Algunos de los sistemas explicados están basados en la Web y otros de ellos no.

Marte 3.0

Marte 3.0 [Cerviño 2008] es un sistema de colaboración multimedia basado en la utilización de clientes ligeros desplegados en la Web. Desarrollado por nuestro grupo de investigación a partir del año 2004 como evolución de Marte 1.0 y Marte 2.0 tiene como objetivo definir una nueva arquitectura de conferencia en Internet centrada en la facilidad de uso e instalación, dotando de conectividad total al usuario final con el resto de participantes.

2.1. VIDEOCONFERENCIA MULTIUSUARIO

Figura 2.2 : Arquitectura de WebRTC

Marte 3.0 está diseñado para proveer salas de videoconferencia bajo demanda. Posee un módulo de gestión de conferencias basado en el estándar *MAuth*, explicado en [Rodríguez 2009] y que también se utilizará en VaaS. La arquitectura de este sistema será el punto de partida para el diseño de los modelos propuestos en esta tesis.

VaaS

VaaS es un servicio de videoconferencia desarrollado también en el ámbito de este grupo. Se trata de un servicio multipunto en la nube basado en Isabel versión 5 que se apoya en un servidor que de acceso a clientes tanto desde PC fijo/portátil, como desde terminales móviles.

El servidor de multiconferencia es capaz de interconectar PCs con terminales móviles, accediendo en ambos a través de un navegador Web y sin necesidad de instalar aplicaciones externas ya que está basado en Flash. El sistema tiene las siguientes características:

1. El servidor es una adaptación de Isabel v5 incluyendo una pasarela flash y otras tecnologías

asociadas para dar soporte a los siguientes tipos de terminales:

- PC con navegador Web compatible con vídeo Flash.
 - Terminal móvil capaz de emitir y recibir vídeo en HTML5.
 - Terminal móvil tipo tableta capaz de emitir y recibir vídeo en HTML5.
2. Capacidad para soportar varias multiconferencias simultáneas e independientes. El número máximo de participantes se determina en función de las características del servidor.
 3. El sistema incluye una gestión mínima de multiconferencias.
 4. El sistema incluye la gestión de estadísticas de códecs, uso de CPU, uso de ancho de banda, resolución, etc.

Google Hangouts

Google Hangouts un servicio de multiconferencia de Google que opera en la Web. Permite acceso desde su aplicación móvil o desde un navegador. La primera se trata de una aplicación nativa que soporta videoconferencias entre dos o más participantes dependiendo de las características del terminal. La versión web soporta hasta diez participantes de manera simultánea en la misma conversación. Además permite compartir vídeos de Youtube en las conversaciones.

En el caso de más de dos participantes se basa en una arquitectura centralizada aprovechando las capacidades de la computación en la nube para ser capaz de escalar a altos números de usuarios.

El sistema se basa en un plug-in desarrollado por Google complementado por tecnología Flash en el navegador. En los últimos meses han migrado su desarrollo a WebRTC ya que Google es uno de los mayores impulsores de dicho proyecto.

Big Blue Button

BigBlueButton es una aplicación web open source para videoconferencia y e-learning o educación a distancia. Es un programa distribuido bajo licencia GNU y que ha surgido de la reutilización de proyectos varios como Asterisk, Flex SDK, Red5, MySQL y otros.

Pero además, BigBlueButton ofrece algunas características adicionales como la posibilidad de que sus usuarios suban archivos PDF o documentos de texto y hojas de cálculo. Ofrece tres roles en sus videoconferencias

2.2. CLOUD COMPUTING

- Presentador, que puede subir presentaciones y compartir su escritorio.
- Espectador, que no tiene autoridad en la videoconferencia y solo puede visualizar los vídeos o chatear.
- Moderador, que puede subir presentaciones, compartir su escritorio y aceptar o expulsar usuarios.

BigBlueButton en su versión web está basado en tecnología Flash y tiene una estructura centralizada.

OpenTok

OpenTok provee un API libre que permite a cualquier desarrollador Web introducir características de chat y vídeo en grupo a sus sitios web. La plataforma está basada en la tecnología Adobe Flash y permite a los usuarios realizar sesiones de videoconferencia con chat y vídeo en vivo de hasta veinte participantes. Permite enviar invitaciones para usar el servicio a través de cuentas en redes sociales como Twitter, Facebook o MySpace.

Está basado en una arquitectura centralizada e inicialmente utilizaba tecnología Flash. Actualmente ofrece sin embargo servicios basados en WebRTC.

2.2 Cloud Computing

Más adelante realizaré un análisis de las ventajas que supone el despliegue de una MCU en una infraestructura Cloud. En esta sección introduzco el término "Computación en la Nube" o *Cloud Computing* describiendo sus principales características y hago un breve repaso de los proveedores comerciales y plataformas de Cloud existentes hoy en día.

2.2.1 Introducción al Cloud Computing

Siempre me gusta explicar el concepto de Cloud Computing con un ejemplo fácilmente entendible. Imaginemos un pequeño empresario que comienza a explotar un servicio Web que ha creado cuyo crecimiento es incierto. En un principio el número de usuarios que acceden a su plataforma es de varias decenas y por tanto para atender sus peticiones le basta con un pequeño servidor de desarrollo que ha configurado para ser accesible desde Internet.

Llega un momento en que su servicio empieza a ser más conocido y dicho número de usuarios empieza a transformarse en centenas de ellos incluso en ocasiones unos pocos miles. Nuestro pequeño empresario decide aumentar por tanto los recursos hardware que dedica a sus servidores y dado que es optimista con el crecimiento de su negocio invierte en una serie de equipos que según sus cálculos le permitirán atender la demanda de centenas de miles de usuarios. Estos nuevos equipos son varios servidores y equipos de red y para poder configurar correctamente los últimos también contrata condiciones especiales con sus proveedores de red.

Tras hacer esta gran inversión sin embargo, nuestro protagonista tiene mala suerte y su negocio no avanza como había imaginado. El número de usuarios no acaba de superar ese pico de unos pocos miles que había conseguido incluso con el paso del tiempo algunos de ellos no vuelven a aparecer. Ha perdido por tanto la inversión que hizo y dispone de un conjunto de caros equipos que no va a aprovechar.

Este contratiempo no se hubiera producido si el empresario hubiera conocido las ventajas que ofrece la Computación en la Nube. Volviendo al pasado y cuando el empresario había detectado que su pequeño servidor de desarrollo no era suficiente para atender la demanda de usuarios, existía una alternativa a la gran inversión en equipos de procesamiento y red. Dicha alternativa consiste en desplegar el servicio Web en una infraestructura Cloud provista por un proveedor de Cloud Computing.

Dicho proveedor de Cloud dispone de una gran granja de servidores y equipos hardware en los que virtualiza máquinas para alquilar a sus clientes. De esta forma, nuestro empresario contrataría un pequeño servidor virtual algo más potente que su inicial servidor de desarrollo. La capacidad de computación de este nuevo servidor puede variarse de forma dinámica e instantánea con un único *click* o llamada a un API por lo que, atento a la variación del número de usuarios en su servicio, podría ir adaptándola a sus necesidades. Y en todo momento estaría pagando únicamente por la capacidad contratada.

Los recursos ofrecidos por estos proveedores de Cloud no se limitan a capacidad de computación. También ofrecen redes virtuales, capacidad de almacenamiento o servicios de administración y configuración. Un ejemplo típico y muy útil para nuestro pequeño empresario es la posibilidad de monitorizar la cantidad de recursos consumidos y configurar alarmas para aumentarlos o disminuirlos automáticamente según las necesidades de cada momento.

Haciendo entonces una definición estricta del término, Cloud Computing es un modelo para

2.2. CLOUD COMPUTING

proveer acceso bajo demanda a recursos de computación como redes, servidores, almacenamiento, aplicaciones y servicios de forma configurable y a través de la red.

Las características del modelo de Cloud Computing según [Mell 2011] son las siguientes:

- **Servicio bajo demanda:** Los usuarios pueden disponer de capacidades de computación (CPU, ancho de banda, almacenamiento, etc.) según lo necesiten.
- **Acceso a través de la red:** Estas capacidades están disponibles en la red en diferentes posiciones y son servidas a través de mecanismos estándar.
- **Asignación de recursos:** El Cloud sigue un modelo *multi-tenant* por el cual pueden asignarse recursos a diferentes usuarios.
- **Flexibilidad:** Las capacidades pueden ser provistas y liberadas de forma automática para adaptarse a la demanda de los usuarios.
- **Servicio monitorizado:** Los recursos son controlados de forma automática y monitorizados y reportados por mecanismos de medida.

Por otra parte y según [Wang 2010] el Cloud Computing se distingue de otras tecnologías en los siguientes aspectos:

- **Interfaces centrados en los usuarios:** Los recursos de Cloud Computing son accedidos de manera sencilla por los usuarios. En la mayoría de los casos, se trata de interfaces Web de fácil manejo.
- **Recursos provisionados bajo demanda:** Las plataformas de Cloud ofrecen sus recursos computacionales bajo demanda. Los usuarios pueden configurar y adaptar los entornos posteriormente, por ejemplo, instalando software o configurando la red.
- **Oferta de calidad de servicio garantizada:** El entorno que provee la computación puede garantizar cierta calidad de servicio a los usuarios en términos de rendimiento de hardware.
- **Sistemas autónomos:** El Cloud es un sistema autónomo que se maneja de manera transparente para el usuario. El hardware, el software y los datos dentro de la nube pueden ser automáticamente reconfigurados, orquestados y consolidados.

- **Adaptabilidad a la demanda y flexibilidad:** Son dos de los aspectos que más han contribuido al auge del Cloud Computing. Los servicios y las plataformas que ofrecen los proveedores de Cloud Computing pueden adaptarse a picos en demanda de varias maneras: con distribución geográfica, aumento de capacidad de hardware o configuración de software.

Por su parte [Vaquero 2008] define la Nube como una gran reserva de recursos virtualizados (hardware, plataformas de desarrollo o servicios) fácilmente accesibles. Estos recursos pueden ser reconfigurados dinámicamente de cara a adaptarse a una carga variable, permitiendo al mismo tiempo una utilización de los recursos óptima. Esta reserva de recursos es típicamente ofrecida en un modelo de pago por uso.

Como vemos cada autor añade ciertos matices en su definición pero la idea fundamental en todas ellas es la misma. Tras esta pequeña introducción parece evidente que el despliegue Cloud de servidores de videoconferencia ofrece grandes ventajas. Se trata de servidores que consumen muchos recursos de computación y en los que la variación en el número de usuarios afecta de forma drástica a dicho consumo.

Pero para aprovechar las ventajas que un despliegue de estas características puede ofrecer, es necesario realizar un diseño minucioso mediante el que aseguremos también que no gastamos más recursos de los necesarios. Y como veremos más adelante este diseño no es trivial. Ya algunos autores han realizado estudios de esta complejidad en otros ámbitos [Armbrust 2010, Hobfeld 2012, Chou 2015], en esta tesis me ocuparé de realizarlos en el de la videoconferencia.

2.2.2 Servicios de Cloud Computing

Los servicios bajo demanda que ofrece el Cloud Computing pueden ser de diversas naturalezas: computación, software, información, redes, etc. De hecho una forma de definir el Cloud Computing podría ser "todo como servicio" o *Everything as a Service*. Sin embargo suelen hacerse tres categorías para dividir los tipos de servicios ofrecidos.

En la figura 2.3 podemos ver estas categorías y una relación de servicios pertenecientes a cada una de ellas (según Gartner en 2009).

- **Infrastructure as a Service (IaaS):** este tipo de servicios proveen básicamente recursos hardware: almacenamiento, servidores virtuales, redes, etc. Los consumidores pueden

2.2. CLOUD COMPUTING

Figura 2.3 : Servicios Cloud según Gartner

utilizar IaaS para construir sus aplicaciones y servicios directamente sobre estos recursos. En el caso de los servidores virtuales lo más normal es que las instancias se provean con un sistema operativo preinstalado. Ejemplos de este tipo de servicios son Amazon Web Services, Rackspace, GoGrid, Microsoft Azure, Google Compute, CloudSigma, etc. Aunque existen también tecnologías que permiten construir Clouds privados a organizaciones que deseen utilizar su propia infraestructura. Por ejemplo OpenNebula, Eucalyptus u OpenStack.

- **Platform as a Service (PaaS):** se trata de la capa correspondiente a la plataforma. Ofrece facilidades para el desarrollo de aplicaciones, diseño, testing, despliegue de servicios, etc. Algunas de las plataformas ofrecen también funcionalidades extra como herramientas de colaboración para equipos de desarrollo, gestión de APIs, seguridad o integración con bases de datos. Ejemplos comerciales de PaaS son Google App Engine, Microsoft Azure, Force.com, Heroku...
- **Software as a Service (SaaS):** corresponde con las capas de servicio y aplicación. Los proveedores de Cloud proporcionan en este caso aplicaciones como servicios remotos. Típicamente estas aplicaciones están basadas en interfaces Web y los usuarios acceden a ellas a través de un cliente ligero. En otros casos consisten en aplicaciones que interactúan con parte del servicio a través de internet. En ambos casos los usuarios no requieren

ejecutar toda la carga necesaria para el funcionamiento de la aplicación en sus computadoras cliente. Ejemplos de este tipo de servicios son Salesforce, Gmail, Google Docs, Hotmail, etc.

Pero aparte de esta clasificación hay quién habla de servicios más concretos. Algunos ejemplos son:

- **Database as a Service (DaaS):** provee el software necesario para correr una base de datos y el almacenamiento necesario de forma remota y como servicio. Los usuarios pueden acceder al servicio pagando por el uso realizado. El ejemplo más conocido sería Amazon Web Services S3 pero también hay otros proveedores como Heroku o AppHarbour que ofrecen características similares.
- **Process as a Service (PaaS):** también conocido por las siglas BPaaS (*Business Process as a Service*) se refiere a un recurso remoto capaz de ejecutar procesos de negocio de manera global. Los recursos utilizados pueden encontrarse en el mismo proveedor o en uno remoto y el resultado es un proceso de negocio. Estos procesos de negocio son más fácilmente actualizables que en las aplicaciones estándar por lo que ofrecen ventajas al respecto. Ejemplos de estos servicios son Appian Anywhere, Akemma y Intensil.
- **Videoconference as a Service (VaaS):** se trata de ofrecer comunicación de vídeo y audio entre usuarios remotos. Los recursos necesarios para procesar estos medios se encuentran en la infraestructura Cloud. En ocasiones no solamente es necesario procesamiento sino también almacenamiento (caso de grabación). Las comunicaciones se realizan usando protocolos como RTP o RTMP. Algunos ejemplos son BigBlueButton, Flash Meeting y Tok Box.

2.2.3 Proveedores y plataformas de Cloud Computing

En esta sección se realiza un repaso por las características de los principales proveedores y plataformas de Cloud Computing existentes en la actualidad.

2.2. CLOUD COMPUTING

Amazon AWS

Los Servicios Web de Amazon (*Amazon Web Services*) están compuestos por una serie de servicios IaaS y PaaS que constituyen hoy en día la mayor plataforma de su clase en términos de uso. De hecho para [Reese 2009] el término "infraestructura cloud" es sinónimo de Amazon EC2 y Amazon S3, los dos servicios principales de este proveedor.

En un principio Amazon no tenía pensado ofrecer su infraestructura como un servicio bajo demanda. Desplegaron un Cloud privado para alojar la infraestructura de negocio necesaria pero más tarde descubrieron que tenían una serie de recursos que estaban siendo desaprovechados. Es entonces cuando decidieron ofrecerlos como servicio y al darse cuenta del gran éxito entre los usuarios decidieron lanzar su línea de negocio en este sentido (año 2006).

Figura 2.4 : Arquitectura de servicios de Amazon Web Services

En la Figura 2.4 podemos ver la arquitectura de servicios de AWS, agrupadas por niveles desde la pura infraestructura dividida en regiones hasta las aplicaciones que despliega el usuario. Típicamente los dos principales servicios de AWS son:

- **Amazon S3:** se trata de un sistema de almacenamiento persistente. Es independiente del resto de servicios de AWS por lo que aplicaciones externas podrían utilizarlo sin necesidad de estar desplegadas en la infraestructura de Amazon. El acceso a Amazon S3 puede realizarse mediante APIs REST con la posibilidad de crear grupos de objetos, asignarles metadatos, etc.
- **Amazon EC2:** es el servicio más utilizado y se encarga de gestionar los servidores virtuales que arrancan los usuarios. Además gestiona una serie de herramientas imprescindibles para acceder a dichos servidores virtuales como son los grupos de seguridad o los *keypairs*.

Los servidores virtuales se arrancan a partir de unas imágenes *Amazon Machine Image (AMI)* que normalmente contienen únicamente un sistema operativo. Sin embargo los usuarios pueden crear sus propias imágenes haciendo copias de servidores que ya tienen configurados. De esta forma se facilita la automatización de despliegues con máquinas virtuales preconfiguradas.

Rackspace

Conocido originalmente como Mosso LLC (2006) pasó a llamarse Rackspace Cloud en el año 2009 y finalmente simplemente Rackspace. Ofrece capacidades para alojar máquinas virtuales al estilo de Amazon EC2 así como almacenamiento, plataformas Web o infraestructura Cloud.

De igual forma que AWS los servicios que ofrece Rackspace son multiregión por lo que el usuario puede decidir en qué zona geográfica desplegar sus recursos. Los servicios más relevantes son:

- **Cloud Files:** ofrece almacenamiento y *Content Delivery Network (CDN)*. Es muy similar al ofrecido por Amazon S3.
- **Cloud Servers:** provee la infraestructura Cloud de computación. La tecnología que hay tras este servicio fue adquirida por Rackspace en 2008 cuando compraron Slicehost.
- **Cloud Sites:** ofrece funcionalidades de plataformas Web soportando PHP, Python, Perl y .NET entre otras tecnologías Web.

2.2. CLOUD COMPUTING

Microsoft Azure

Se trata de la competencia ofrecida por Microsoft en cuanto a proveedores de infraestructura como servicio. En un principio su orientación era mayormente hacia PaaS pero en la actualidad intenta competir con los grandes proveedores de IaaS (Amazon y Google). De igual forma que Amazon provee recursos de computación y almacenamiento bajo demanda, un servidor de SQL llamado SQL Azure y Windows Azure AppFabric, una colección de APIs que se encuentra en modo deprecado en la actualidad.

En la literatura podemos encontrar algunos estudios que miden las capacidades de Azure y dan recomendaciones sobre buenas prácticas en los despliegues y escalabilidad. Dos ejemplos serían [Hill 2010] y [Lu 2010].

Google Cloud Platform

Google Cloud Platform es la plataforma Cloud en la que Google ofrece computación como servicio. Para ello Google utiliza la misma infraestructura en la que hospeda sus servicios como su buscador o Youtube. Ofrece también almacenamiento bajo demanda y APIs de traducción y predicción así como su famoso servicio PaaS llamado Google App Engine.

Uno de los puntos fuertes del servicio de Google son las herramientas de Big Data o procesamiento masivo de datos basado en Hadoop. Dispone también de componentes de Machine Learning y procesamiento de datos y sus últimas mejoras de interfaz e integración hacen que hoy en día sea uno de los más serios competidores de Amazon Web Services.

Joyent

El servicio clave que ofrece Joyent es un entorno PaaS basado en la tecnología Node.js. Originalmente se centraban en un catálogo de aplicaciones de gestión de correo y calendarios, pero actualmente incluyen también servicios IaaS como parte de su oferta comercial.

Joyent tiene un departamento en el que se realizan muchos proyectos de código libre. Una de los más destacados es SmartOS, su sistema operativo distribución de Illumos liberado en el año 2011.

Openstack

Openstack es la tecnología de infraestructura Cloud más reconocida en la actualidad. Se trata de un proyecto de código libre originalmente impulsado por Rackspace y que actualmente tiene una comunidad de miles de miembros y empresas colaboradoras. Openstack está formado por una serie de subproyectos dedicados a cada una de las tareas IaaS:

- **Nova:** Nova es el proyecto encargado de la computación. Gestiona máquinas virtuales, grupos de seguridad, *keypairs*, etc.
- **Glance:** en el caso de Openstack la gestión de imágenes está en un componente propio, Glance. Básicamente se trata de un repositorio de imágenes virtuales donde se pueden subir nuevos sistemas operativos o servidores virtuales preconfigurados y que se comunica con Nova para instanciarlos.
- **Cinder:** Cinder se encarga de la gestión de volúmenes de datos virtuales. Gracias a él se puede añadir persistencia a las máquinas virtuales.
- **Swift:** en este caso la gestión de contenedores y objetos se encuentra separada de la de volúmenes y a cargo del componente Swift. También pueden realizarse copias de seguridad de volúmenes en este sistema.
- **Neutron:** se encarga de la gestión de redes virtuales, subredes, pasarelas, routers y todos los componentes relacionados con las redes.
- **Keystone:** es el gestor de identidad y componente fundamental de Openstack. Aparte de realizar gestión de usuarios y grupos de usuarios se encarga de autenticar a éstos y al resto de servicios.

Además de los mencionados servicios Openstack está añadiendo en los últimos años componentes de niveles Cloud superiores, principalmente PaaS, como podrían ser orquestadores de servicios (proyecto Heat), motorización (Ceilometer) o gestores de software (proyecto Murano). En la figura 2.5 podemos ver el modelo de servicios y la comunicación entre ellos.

2.3. SISTEMAS DISTRIBUIDOS

Figura 2.5 : Arquitectura de servicios de Openstack

2.3 Sistemas distribuidos

Como introducía al principio de esta sección y explicaré con detalle más adelante, una de las decisiones tomadas durante el desarrollo de esta tesis para contribuir al despliegue y escalabilidad eficientes de MCUs es la distribución de las mismas entre múltiples servidores. Sin embargo, una vez distribuidas, cómo repartir la carga entre ellas no es una tarea trivial.

En la literatura podemos encontrar estudios sobre cómo realizar programación de recursos en servidores de vídeo bajo demanda o de *streaming* así como en sistemas de otros ámbitos no relacionados con el procesamiento de media. Sin embargo, los sistemas de tiempo real tienen características especiales que hacen que estos mecanismos no puedan aplicarse. En el Capítulo 4 se hará una descripción detallada de estas características y un análisis de por qué las soluciones

existentes no son aplicables.

A continuación realizo un repaso por las más relevantes de las mencionadas soluciones en otros ámbitos.

2.3.1 Programación de recursos en sistemas multimedia

Los estudios que existen en relación a la programación de recursos en sistemas multimedia se refieren a sistemas de vídeo bajo demanda o *streaming*. Como se explica en [Ghose 2000], los retos que suponen este tipo de sistemas son diferentes a los de los sistemas de transmisión de ficheros tradicionales. Esto se debe principalmente a las necesidades especiales de ancho de banda y computación que requieren los servidores de vídeo.

Sistemas de vídeo bajo demanda

Los autores del trabajo mencionado anteriormente [Ghose 2000] realizan un análisis y comparativa entre un conjunto de políticas basadas en los siguientes principios:

- **Broadcasting:** la aproximación utilizando criterios de *broadcasting* o centrados en el servidor es más eficiente que la basada en cliente-servidor o centrada en usuario. Usando un único canal puede darse servicio a un número elevado de usuarios (asumiendo suficiente ancho de banda en las redes utilizadas).
- **Batching:** en los sistemas de vídeo bajo demanda el cuello de botella suele ser el ancho de banda de entrada/salida a disco. Utilizando una aproximación centrada en cliente habría que asignar cada petición a un flujo de entrada/salida. De esta manera el sistema se saturaría rápidamente con el crecimiento del número de usuarios.

La única alternativa es una solución centrada en datos realizando un multicast del mismo mensaje a múltiples clientes sin causar sobrecargas en el servidor. Teniendo esto en cuenta, el *batching* agrupa las peticiones a un mismo recurso que se reciben en periodos de tiempo parecidos (llamados *ventana de batching*) y sirviéndolas en un único flujo de datos.

- **Caching:** se trata de un método simple pero muy efectivo para reducir el ancho de banda de entrada/salida en disco. Es especialmente útil en entornos de *batching* con ventanas de tiempo determinísticas. La idea del *caching* es bien conocida, consiste en guardar los datos

2.3. SISTEMAS DISTRIBUIDOS

previamente servidos para utilizarlos en futuras peticiones en lugar de acceder de nuevo a disco. Obviamente, la eficiencia en costes de este método se encuentra en la diferencia entre el soporte a discos y el soporte a memorias de caché.

- **Piggybacking adaptativo:** esta técnica (explicada por los autores de [Golubchik 1996]) consiste en la compartición de datos entre flujos del mismo vídeo. En este caso no se utilizan *buffers* y no es necesaria la existencia de varias peticiones del mismo recurso para atenderlas (como en los casos de *caching* y *batching* respectivamente).

El principio de este mecanismo consiste en adelantar o retrasar la reproducción de vídeos para intentar igualar la fase de varios vídeos emitidos en intervalos de tiempo diferentes. De esta forma y con el paso del tiempo puede llegar a conseguirse dicha igualación y transmitir todos ellos mediante un único flujo multimedia.

- **Esquema de programación Look-ahead:** en este caso los autores [Yu 1995] proponen un esquema similar al de *batching* pero algo más refinado para habilitar un uso más eficiente de la capacidad de los discos. Mediante este mecanismo es posible integrar un comportamiento de parada y reanudación de transmisión con los ya mencionados mecanismos de *batching* y *caching*.

Por otra parte, los autores de [Wu 2001] discuten sobre un conjunto de algoritmos de programación de recursos en servidores de vídeo paralelos. Su objetivo es el de encontrar los requisitos para optimizar la asignación de dichos recursos y conseguir cumplir las condiciones de calidad de servicio adecuadas. Para ello se centran en el análisis de cuatro ideas:

- **Eliminación de conflictos de asignación:** basado en el método *Hungarian* [Bondy 1976] permite evitar conflictos cuando varias peticiones se reciben al mismo tiempo.
- **Minimización del retardo:** mediante un algoritmo *peg-and-hole* se consigue hacer mínimo el retardo entre las peticiones atendidas.
- **Reasignación de peticiones:** en caso de cargas no balanceadas deben aplicarse técnicas de reasignación de recursos para intentar balancear los servidores. Y esta reasignación debe ser lo menos costosa posible en términos de consumo de recursos y latencia.

- **Control de admisión:** únicamente cuando las tres condiciones anteriores se cumplen, debe admitirse una petición para ser atendida. En caso contrario, se tendría que retrasar su atención perjudicando a posibles peticiones posteriores.

Por último, en [Hong 2009], los autores proponen añadir datos semánticos a los paquetes de datos para cuantificar su importancia desde un punto de vista de la compresión. Esta importancia influye de cara a programar la asignación de recursos a dichos paquetes predominando la transmisión de subconjuntos de paquetes con mayor importancia acorde a la métrica propuesta, llamada *Packet Significance*.

2.3.2 Programación de ejecución de tareas

Cuando hablamos de computación, la programación de tareas (*task scheduling* en inglés) consiste en la distribución de trabajos que los consumidores necesitan ejecutar entre el conjunto de recursos disponibles. Típicamente estos trabajos pueden ser elementos de computación virtuales como hebras, procesos o flujos de datos que deben programarse para ser ejecutados en recursos hardware como procesadores o elementos de red.

En estos sistemas, el componente encargado de gestionar la asignación de tareas a recursos se llama "programador" (*scheduler*). Los programadores pueden implementarse para hacer una distribución homogénea de los recursos entre las tareas a ejecutar, para compartir recursos entre diferentes tareas, o para asegurar una cierta calidad de servicio. Sin embargo, aparte de estas configuraciones estándar, podemos pensar en algoritmos de programación más avanzados que tengan en cuenta prioridades o se adapten a las necesidades de cada momento o escenario.

La programación de recursos es fundamental para los sistemas de computación y sin ella no serían concebibles los sistemas multitarea que conocemos hoy en día. Si hablamos de entornos de tiempo real, los programadores deben también asegurar que se cumplen límites de tiempo y de esa forma garantizar la estabilidad de los sistemas.

Como vemos, este tipo de sistemas se asemeja al escenario de asignación de recursos en un sistema de videoconferencia distribuido. Por ello, resulta interesante analizar las soluciones existentes de cara a extraer posibles ideas a la hora de plantear soluciones.

2.3. SISTEMAS DISTRIBUIDOS

Algoritmos adaptativos

Existe un gran abanico de algoritmos de asignación de tareas adaptativos. Este tipo de algoritmos se adaptan de forma dinámica a las condiciones de cada entorno y en ocasiones basándose en la propia información obtenida del estado actual del mismo.

Los autores de [Li 2010] proponen y comparan dos algoritmos adaptativos, el *Adaptive List Scheduling* y el *Adaptive Min-Min Scheduling*. Estos algoritmos están basados respectivamente en los algoritmos *Critical Nodes Parent Trees* [Hagras 2003] y *Min-Min Greedy Algorithm* [Ibarra 1977] y consisten en establecer prioridades en la ejecución de tareas. Estas prioridades tienen en cuenta parametros como el tiempo de ejecución de las tareas o los requisitos concretos de los recursos necesarios para cada una de ellas. Además dan prioridad a trabajos que han sido programados con antelación realizando una especie de reserva de recursos.

Mesos

Mesos [Hindman 2011] es una plataforma diseñada por la Universidad de Berkley para compartir capacidad de computación en *clusters* entre diferentes *frameworks*. Ejemplos de estos *frameworks* son Hadoop o *Message Passing Interface* (MPI). Mesos está basado en lo que sus diseñadores llaman *resource offers* u "ofertas de recursos". Un proceso maestro se encarga de gestionar los nodos *cluster* disponibles y tiene una lista de los recursos libres en cada uno de ellos. Con esta información decide qué tipo recursos y qué cantidad de ellos ofrecer a cada *framework* en cada momento.

En la figura 2.6 podemos ver la arquitectura de Mesos con los dos *frameworks* más típicos. Mesos permite a los desarrolladores configurar algoritmos de decisión personalizados y adaptados a los requisitos específicos de cada caso de uso. De esta forma, las "ofertas de recursos" a los *frameworks* variarán según el escenario concreto. También puede utilizarse un modelo de incentivos para dar prioridad a ciertos *frameworks*. Por ejemplo, un *framework* que utiliza tareas de corta duración o que nunca acepta recursos que no va a utilizar, tendrá prioridad respecto a otros en el programador del nodo maestro.

Figura 2.6 : Arquitectura Mesos con frameworks Hadoop y MPI

2.3.3 Programación de recursos Cloud

En despliegues de servicios en infraestructuras Cloud es muy frecuente que los servidores se encuentran distribuidos entre varias máquinas virtuales. La manera en que se distribuyen las peticiones de los clientes entre los servidores disponibles es crucial a la hora de garantizar una buena calidad de servicio.

Ant Colony Optimisation

En [Lu 2011] se propone el mecanismo *Ant Colony Optimisation* (ACO) para resolver los dos problemas que los autores identifican como clave para resolver la programación de recursos en servicios Cloud: la detección de puntos calientes (*hot spots*) y la estrategia de programación de recursos adaptativa. Siempre teniendo en cuenta que existen una serie de nodos entre los que distribuir las peticiones de alojamiento de recursos recibidas por los clientes. Los autores basan su trabajo en el pensamiento de adaptación de carga explicado en [Qiang 2009].

- **Detección de puntos calientes:** este problema consiste en cómo saber cuándo uno de los nodos ha alcanzado un umbral de uso de recursos (CPU, memoria o ancho de banda). Para resolverlo, los autores proponen un mecanismo de detección de pulsos mediante el cual cada nodo envía a un nodo central de control su estado en cada momento.

2.3. SISTEMAS DISTRIBUIDOS

Este nodo de control recopila el estado de todos los nodos "hijos" para determinar cuándo alguno de los valores recibidos es mayor que el umbral establecido. En ese momento ese nodo se marca como punto caliente y dicha característica se tendrá en cuenta a la hora de tomar decisiones de asignación. Para evitar puntos calientes temporales, el nodo central necesita recibir una señal de detección un número determinado de veces antes de considerar a un nodo como tal.

- **Estrategia de programación de recursos adaptativa:** Una vez que un punto caliente es detectado, ACO se basa en el llamado Algoritmo de la Colonia de Hormigas (*Ant Colony Algorithm*) para programar los recursos Cloud y compartir la carga entre los nodos. La idea se basa en cómo las hormigas patrullan los caminos en la naturaleza dejando un rastro de feromonas y marcando el mejor recorrido a seguir.

Resumiendo, cuando un nodo alcanza el límite de recursos utilizados, busca un nuevo nodo enviando un mensaje a los que se encuentran a su alrededor. Este proceso se repite entre todos los nodos de manera sucesiva hasta que se encuentra un nodo disponible (i.e. que no está marcado como punto caliente). Estos mensajes enviados entre los nodos realizan la función de las feromonas de las hormigas en la naturaleza. Dichos mensajes van marcando el recorrido con mayor probabilidad de éxito en encontrar un nodo disponible.

Como veremos más adelante, algunas de estas ideas resultarán muy útiles a la hora de diseñar nuestros algoritmos de programación de recursos en sistemas de videoconferencia distribuidos. El reporte del estado de cada nodo a la capa de toma de decisiones es fundamental en entornos de tiempo real.

Data Envelopment Analysis

Por otra parte, los autores del trabajo [Cho 2014] destacan el hecho de que los métodos de programación de recursos Cloud existentes están normalmente basados en parámetros de entrada únicos y principalmente operacionales. Esta condición resulta en un problema de desequilibrio con la consecuente pérdida de recursos.

Su propuesta frente a esto es aprovechar los estudios de *Data Envelopment Analysis* para analizar múltiples factores en lugar de solamente uno. Estos factores pueden ser la demanda de usuario, la distancia, el uso de CPU o su coste. De esta manera construyen un sistema con

múltiples entradas y múltiples salidas que les permiten crear políticas de asignación de recursos complejas y más adecuadas a los requisitos de sistemas avanzados.

2.3.4 Asignación de recursos en centros de datos

En los centros de datos donde se despliegan infraestructuras Cloud existen una serie de recursos hardware fijos que hay que utilizar para alojar los recursos virtuales que se ofrecen a los usuarios. La forma en que los recursos físicos se asignan para cubrir la demanda de los virtuales es clave e influye de forma directa tanto en la eficiencia de los servicios ofrecidos como en los costes asociados a los despliegues. Esto resulta fundamental para los modelos de negocio de los proveedores de servicios Cloud. Existen por lo tanto muchos estudios sobre mecanismos para mejorar la eficiencia de dichas asignaciones.

En estas condiciones, uno de los factores clave a tener en cuenta es el tiempo de asignación. Este tiempo es el que transcurre desde el momento en que un cliente solicita un recurso virtual hasta el momento en que éste está listo. En caso de que haya más de un centro de datos donde alojar este recurso, la decisión de en cuál de ellos hacerlo puede ser crítica.

[Jung 2011] propone un modelo de asignación adaptativo que toma las decisiones basándose en dos factores: 1) la distancia geográfica entre el consumidor y los centros de datos y 2) la carga de trabajo de cada centro de datos. Según su investigación son estos dos parámetros los que influyen de manera más directa en el tiempo de asignación.

- **Cálculo de la distancia geográfica:** para calcular la distancia geográfica existente entre un consumidor y un centro de datos dado, los autores dividen las localizaciones en zonas y comprueban la latencia de red entre ellas.
- **Cálculo de la carga de trabajo:** para realizar el cálculo de la carga de trabajo de un centro de datos cuentan el número de CPUs virtuales que un nodo está atendiendo en un momento determinado. Entonces, dado el número de CPUs virtuales que una nueva petición requiere, para decidir donde asignarla, se comprueba cuál de los centros de datos tiene espacio suficiente para atenderla. Los nodos que no tienen suficiente espacio son descartados de forma automática.

La forma de calcular la carga de las MCUs a la hora de asignar peticiones es una de las contribuciones principales de esta tesis. Como veremos en el Capítulo 4, no se trata de un cálculo

2.3. SISTEMAS DISTRIBUIDOS

trivial y es necesario diseñar una nueva métrica adecuada a los requisitos de los sistemas de videoconferencia distribuidos.

Capítulo 3

Videoconferencia escalable en infraestructuras Cloud

En los últimos años se han producido una serie de circunstancias tecnológicas y sociales que han hecho que el uso de aplicaciones y servicios de videoconferencia sea cada vez más frecuente. La heterogeneidad en los dispositivos que son compatibles con este tipo de servicios ha hecho que el acceso sea mas ubicuo que en el pasado y además que se utilice en ámbitos muy diferentes. Tradicionalmente se utilizaba en escenarios muy concretos y casi siempre profesionales pero hoy en día cualquier persona dispone de aplicaciones de videoconferencia en sus ordenadores personales, teléfonos móviles y tabletas.

Como adelantaba en la introducción, este crecimiento en el número de usuarios hace que los sistemas tradicionales de videoconferencia no se adapten a los requisitos actuales, o en caso de que sí lo hagan, que la forma en que se han venido desplegando no sea adecuada para las variaciones dinámicas en la demanda causadas por las nuevas condiciones de uso.

En este capítulo realizo un análisis de las ventajas que supone un despliegue basado en el Cloud para los sistemas de videoconferencia multiusuario y teniendo en cuenta estas nuevas condiciones. En concreto el despliegue de MCUs, componente que recibe la carga mayoritaria en cuanto a uso de recursos en estos sistemas. Pero una vez claras estas oportunidades, hay una serie de retos a tener en cuenta de cara al diseño de las nuevas arquitecturas necesarias. Teniendo en cuenta este análisis, me centro en resolver alguno de los retos planteados.

En concreto propongo un modelo de arquitectura diseñado para escalar MCUs de manera dinámica a las variaciones en la demanda de usuarios. Como veremos, a la hora de escalar frente a disminuciones de la demanda, la movilidad de flujos multimedia entre MCUs es fundamental para asegurar el uso eficiente de recursos. Propongo también por tanto un nuevo mecanismo que permitirá realizar esta movilidad de manera transparente para los usuarios y sin afectar a la calidad de las comunicaciones.

3.1 Objetivos

Teniendo en cuenta esta pequeña introducción, los objetivos planteados en este capítulo son:

- *Identificar las ventajas y los retos que supone desplegar un sistema de videoconferencia en el Cloud.* El primer paso consiste en hacer un análisis de las ventajas que nos aporta el despliegue de MCUs en infraestructuras Cloud. De esta manera tendré claro qué aspectos debo explotar a la hora de diseñar mis modelos. Pero también es importante tener en cuenta los retos con los que voy a encontrarme. Tendremos por tanto una caracterización global del escenario.
- *Proponer e implementar un modelo de arquitectura que permita escalar un sistema de videoconferencia Cloud de forma dinámica y acorde a la demanda.* Este modelo debe permitir que el sistema escale hacia arriba y hacia abajo. Es decir, debe permitir añadir y quitar MCUs al sistema según la demanda de usuarios en cada momento. Para ello es fundamental explorar las mejores vías para realizar movilidad de flujos multimedia entre estos servidores.

3.2 Despliegue de MCUs en el Cloud

Como hemos visto en la Sección 2.1.2 un sistema de videoconferencia entre múltiples usuarios suele constar de una serie de capas que se encargan de la gestión del control, señalización y medios. Y estas tareas son realizadas por un componente central llamado MCU. Las MCUs suelen hacer un uso bastante intensivo de recursos de computación debido principalmente a la gestión de medios. Esto es especialmente acusado cuando hablamos de MCUs transcodificadoras o grabadoras. Y en casos en los que el número de usuarios conectados fluctúa de forma dinámica estas necesidades de recursos pueden variar drásticamente en periodos de tiempo muy cortos. Todo parece indicar que las necesidades de estos sistemas cuadran con las oportunidades que ofrecen los sistemas Cloud y que se han enumerado en el capítulo anterior (Sección 2.2.1). Vamos a analizar con detalle cada una de estas oportunidades y los retos que supondría un despliegue basado en el Cloud.

3.2.1 Oportunidades

Un componente MCU puede requerir diferentes características de computación en función del número de participantes, las condiciones de la sesión (grabación, redirección, transcodificación,

3.2. DESPLIEGUE DE MCUS EN EL CLOUD

composición) y la posición física de los participantes. Además estas condiciones pueden variar de forma dinámica durante la sesión de videoconferencia. En una infraestructura Cloud la sesión va a poder adaptarse de forma sencilla y dinámica a las variaciones de este tipo de condiciones y en función de los requisitos de cada momento.

Escalabilidad a la demanda de usuarios

Los sistemas de videoconferencia ofrecen a sus usuarios la posibilidad de unirse a una conferencia antes y durante la sesión. Además los usuarios pueden abandonar la sesión mientras está en curso. Dependiendo del tipo de sesión y de las características del servicio esta variación en el número de usuarios puede ser más o menos frecuente. Un alto número de usuarios normalmente significa más ancho de banda, consumo de memoria y CPU. En otras palabras, una MCU demandará más o menos capacidades a su infraestructura de computación en función del número de usuarios conectados a una sesión.

En un entorno tradicional el proveedor debería preparar previamente las máquinas físicas que dedicaría al sistema y éstas deberían ser suficientes para poder asumir los picos de demanda que se produjeran. No obstante esta solución implica desperdiciar una alta cantidad de recursos cuando la demanda por parte de los usuarios es pequeña.

En un entorno Cloud, por el contrario, el proveedor puede proveer de forma dinámica máquinas virtuales y liberarlas cuando no sean necesarias. Esto se suele realizar normalmente encendiendo o apagando estas máquinas virtuales dependiendo de los recursos necesarios, en relación a los participantes o usuarios que estén conectados en cada momento. Esto puede también conseguirse incrementando de forma dinámica el rendimiento de una única máquina. Podemos por ejemplo incrementar el nivel de CPU o memoria de una máquina virtual que ya está corriendo. El ancho de banda disponible a la salida y a la entrada puede también modificarse variando el tamaño de las máquinas virtuales. Por ejemplo Amazon Web Services permite diferentes tipos de máquinas virtuales con diferentes capacidades de rendimiento.

Escalabilidad a los requisitos de la sesión

La operación de una MCU también depende del tipo de sesión que esté gestionando. Cada tipo de funcionalidad o tarea que desarrolle requiere diferentes capacidades de computación: redireccionamiento de flujos multimedia, composición de vídeos, transcodificación, grabación de

las sesiones, etc. Un componente MCU básico solamente redirecciona flujos de un participante al resto y requiere bajos niveles de computación. Sin embargo estos niveles (como el uso de memoria y CPU) aumentan considerablemente si la MCU realiza tareas avanzadas. La necesidad de realizar estas tareas puede variar de forma dinámica durante el transcurso de una sesión dependiendo de factores como el número de usuarios, el tamaño de los vídeos generados, los códecs que manejan los usuarios en sus dispositivos, etc.

Por ejemplo un alto número de usuarios puede forzar a la MCU a realizar una composición de vídeos en uno solo a partir de los de un conjunto de usuarios. Por otro lado en escenarios en los que los usuarios se conectan desde diferentes dispositivos, la MCU deberá adaptar los flujos que envía a cada uno de ellos en función de ciertas características de los dispositivos como el tamaño de la pantalla, la conexión de red, etc. Por último la MCU puede tener que grabar la sesión multimedia o parte de ella para realizar una reproducción posterior.

Entornos virtualizados en sistemas Cloud ayudan a la MCU a adaptarse a las variaciones de los requisitos de estas funcionalidades. De igual forma que en el caso anterior podríamos encender una nueva máquina para realizar nuevas tareas y apagarla cuando no sea necesaria. O añadir capacidad a las máquinas que ya se encuentran corriendo en la infraestructura cuando sea necesario.

Figura 3.1 : Ejemplo de dos MCUs realizando diferentes tareas

Otra posibilidad que ofrece el Cloud es la de configurar diferentes tipos de máquinas virtuales dependiendo de las tareas que vayan a desempeñar. En la Figura 3.1 se puede observar un ejemplo en el que la máquina encargada de redireccionar los flujos multimedia necesita un alto nivel de

3.2. DESPLIEGUE DE MCUS EN EL CLOUD

CPU y memoria. Por otro lado la máquina encargada de grabar la sesión consumirá poca memoria y CPU si recibe un único flujo de vídeo con la composición de la sesión.

Resumiendo, el despliegue de una MCU en una infraestructura Cloud hace que sea sencillo y dinámico gestionar la configuración de diferentes tipos de máquinas virtuales adaptando sus características a las necesidades del escenario concreto. Así podremos proveer un servicio adaptativo que utilizará de manera eficiente los recursos disponibles reduciendo costes y mejorando el rendimiento del sistema.

Flexibilidad geográfica

Otro factor crítico en las aplicaciones de tiempo real que afecta directamente a la experiencia de usuario es la latencia de las paquetes que viajan entre los usuarios conectados a una sesión. Esta latencia normalmente depende de la distancia geográfica entre los clientes. En los escenarios con un dispositivo MCU todos los flujos multimedia atraviesan la máquina en la que ésta está corriendo por lo tanto resulta crucial la posición en la que se encuentra.

Figura 3.2 : Distribución geográfica de MCUs

Gracias a un sistema basado en el Cloud podemos correr MCUs en diferentes localizaciones geográficas conectando cada una de ellas con los usuarios que están utilizando el servicio en cada región. Por ejemplo, Amazon Web Services provee centros de datos en varios continentes e inclusive en varias zonas dentro de dichos continentes. Los proveedores Cloud suelen además permitir la interconexión de MCUs en diferentes regiones por lo que pueden ofrecerse sesiones en todo el mundo conectando usuarios a la MCU más cercana e interconectando las MCUs entre

ellas. En la Figura 3.2 puede observarse un ejemplo de esta distribución.

3.2.2 Retos

Para realizar un despliegue que aproveche todas las posibilidades que ofrece el Cloud y que han sido descritas en la sección anterior es necesario realizar un diseño minucioso y teniendo en cuenta todas las características del sistema que manejamos. En esta sección se analizan todos los retos que plantea este problema y se esbozan posibles soluciones de las cuales algunas serán resueltas en esta tesis y otras quedarán pendientes para futuros trabajos. En la literatura podemos encontrar soluciones genéricas de escalabilidad [Vaquero 2011] pero como veremos nuestro escenario tiene ciertas peculiaridades que hacen no sean aplicables.

Caracterización del sistema

Caracterizar el rendimiento de la MCU es el primer paso para conseguir un despliegue efectivo en el Cloud. Dependiendo de la tarea (transcodificar, grabar, etc.) que vaya a realizarse en una MCU el hardware y ancho de banda necesario varían significativamente. Midiendo el rendimiento de los dispositivos en entornos conocidos y acotados podemos extrapolar el comportamiento a otros escenarios para aproximar las características que serán necesarias en cada momento. Así sabremos la cantidad de CPU, memoria o ancho de banda que serán necesarias en las futuras sesiones.

Una vez completada la caracterización del sistema es interesante encontrar relaciones entre los recursos físicos y recursos de más alto nivel como el número de usuarios o el número de salas activas en la sesión. Encontrando esta correlación podremos simplificar el trabajo a la hora de realizar la monitorización y resultará muy útil para las tareas que se explicarán en secciones siguientes, con el escalado horizontal y vertical. Sabiendo la incidencia de cada nuevo usuario conectado combinado con la monitorización continua de los recursos consumidos por la MCU podremos reaccionar de forma efectiva a los requisitos y los cambios en la demanda del sistema. Por supuesto esto implica conocer el efecto que un nuevo usuario conectado tiene sobre las capacidades y requisitos del sistema completo y cada MCU.

Existe también un reto impuesto por el despliegue de cualquier sistema en el Cloud en relación a las posibles interferencias entre diferentes máquinas virtuales. Hay mucha literatura acerca de este tema [Tickoo 2010, Koh 2007], así como sobre posibles formas de caracterizar el problema [Do 2011]. No obstante en el marco de esta investigación asumiré que estas interferencias no van

3.2. DESPLIEGUE DE MCUS EN EL CLOUD

a ser relevantes a la hora de la caracterización de nuestro sistema.

Escalabilidad hacia arriba

Cuando los recursos disponibles en una sesión alcanzan su límite podemos aprovechar las características del Cloud aumentando estos recursos para seguir ofreciendo un servicio con la calidad adecuada. Para ello existen dos tipos de escalabilidad: horizontal y vertical. El escalado horizontal consiste en añadir nuevas máquinas virtuales a las ya arrancadas en el sistema y el escalado vertical en aumentar las características de las máquinas que ya están corriendo.

En una MCU ambos métodos tienen sus ventajas. Si los nuevos recursos se necesitan para hacer una nueva tarea en la sesión, como por ejemplo grabar una sesión multimedia, la escalabilidad horizontal puede ser más beneficiosa. Sin embargo si los recursos se necesitan porque ha habido un incremento del número de usuarios en la sesión puede ser mejor realizar un escalado vertical para que la sesión siga gestionándose por la misma máquina virtual. Esto nos dará más facilidades en caso de que sea necesario escalar hacia abajo en el futuro como se explicará en el siguiente apartado.

Suele ser en general más conveniente tener todos los usuarios de una sala en la misma MCU. Sin embargo, no todos los proveedores de Cloud públicos ofrecen la posibilidad de realizar un escalado vertical. Por lo tanto un primer reto importante relacionado con el escalado es el de elegir qué tipo va a utilizarse cuando en el sistema se requiere un mayor nivel de recursos para gestionar una sesión.

Por otro lado ambos tipos de escalado implican una latencia desde que se realiza la acción de aumentar los recursos hasta que éstos están disponibles para su uso. Para lograr una buena experiencia de usuario es fundamental que no haya ningún tipo de interrupción en las comunicaciones y por ello es necesario anticiparse a los picos de demanda que pueda haber en el sistema para actuar en consecuencia con un tiempo que garantice la transparencia frente al usuario.

Una primera aproximación para solucionar el problema es la de utilizar algoritmos que, basándose en una monitorización del sistema, calculen cuándo va a ser necesario incorporar más recursos a las máquinas virtuales mediante cualquiera de los dos tipos de escalabilidad explicados anteriormente. La MCU deberá monitorizar constantemente el estado del sistema analizando los diferentes factores mostrados anteriormente. Si establecemos los límites del

sistema y la correlación con el número de usuarios y salas será sencillo determinar cuando es necesario añadir recursos al sistema con la antelación necesaria. Un ejemplo de un sistema parecido puede observarse en [Galán 2009]. En la siguiente sección hago un diseño de arquitectura que permite implementar este mecanismo.

Pero podemos ir un poco más allá y utilizar modelos predictivos para anticiparnos a los cambios en los requisitos de las MCUs basándonos en el análisis de datos previos. Con este tipo de modelos podemos analizar patrones de comportamiento del sistema para predecir lo que sucederá en un momento determinado. Estos patrones pueden obtenerse principalmente de dos formas. Bien utilizando el comportamiento previo del propio sistema o el de sistemas similares. La primera opción es la óptima pero no siempre podemos disponer de esos datos por lo que puede ser necesario acudir a la segunda. En [Caron 2010] puede observarse un buen punto de partida en el cual el problema se resuelve con un algoritmo que predice el uso de recursos utilizando un matching de patrones.

Escalabilidad hacia abajo

De la misma forma que durante una sesión puede ser necesario incrementar los recursos disponibles puede ocurrir que en un momento dado un pico de demanda haya desaparecido y estemos desperdiciando recursos. Como hemos visto antes existen diferentes métodos de incrementar los recursos de un módulo MCU. En el caso del escalado hacia abajo ocurre lo mismo pero en sentido contrario.

Por lo tanto el primer reto que plantea el escalado hacia abajo es similar al caso anterior. Cuando detectamos que según la demanda del sistema estamos ofreciendo más recursos de los necesarios debemos elegir la mejor forma de disminuirlos. Podemos realizar un escalado vertical y reducir la capacidad de computación de las máquinas que ya están corriendo o realizar un escalado horizontal reduciendo el número de máquinas arrancadas. Pero en el segundo caso, escalado horizontal hacia abajo, se plantean dificultades adicionales. Debe tenerse en cuenta que en la máquina que vamos a apagar muy probablemente está realizándose la gestión de usuarios, flujos y salas multimedia por lo que deberemos distribuir estos elementos entre el resto de máquinas que seguirán arrancadas. Esta distribución no es un problema trivial.

Un cliente que está participando en una sesión está enviando y recibiendo varios flujos multimedia desde y hacia la MCU. Si esta MCU va a ser apagada es necesario redirigir estos

3.2. DESPLIEGUE DE MCUS EN EL CLOUD

Figura 3.3 : Proxies reenviando tráfico a MCUs

flujos a otra MCU y esta redirección debe realizarse de forma transparente para el usuario. Además para optimizar el uso de recursos debe realizarse una correcta distribución de los usuarios y las salas de videoconferencia procurando agrupar estos en las mismas MCUs.

Una posible solución a este problema, mostrado en la Figura 3.3, es la de incluir un *proxy* entre los clientes y el módulo MCU. De esta forma cuando una máquina va a ser apagada el proxy comenzará a duplicar los flujos entre la antigua MCU y la nueva. Cuando los flujos estén preparados el proxy cambiará el envío y recepción al cliente de la antigua MCU a la nueva y en ese momento podrá apagarse la antigua. En la sección 3.4 diseño un mecanismo de movilidad de flujos que se basa en esta idea.

Distribución geográfica

De la mano de la flexibilidad que da la distribución geográfica permitida por los proveedores de Cloud viene el reto de localizar de forma óptima las instancias de MCUs para obtener el mejor servicio posible. Esta decisión puede ser trivial cuando todos los usuarios están localizados en el mismo continente o zona del proveedor de Cloud. Pero decidir cómo actuar cuando los usuarios están distribuidos en distintos lugares puede determinar la calidad de la sesión.

Para tomar esta decisión debe tenerse en cuenta el número de usuarios en cada región geográfica pero también la calidad de sus conexiones. Para caracterizar las conexiones entre las

regiones es interesante realizar medidas de ancho de banda, jitter, pérdidas de paquetes y RTT (*Round-Trip delay Time*). Probando la conexión de cada usuario con las diferentes regiones podemos decidir dónde será mejor la calidad. Como puede comprobarse en [Cerviño 2011a] las conexiones de red entre instancias de Amazon en diferentes regiones rinden mejor que las conexiones de Internet medias. Debe tenerse esto en cuenta a la hora de realizar el diseño y despliegue del sistema.

3.3 Escalabilidad dinámica a la demanda

Como acabamos de comprobar, una de las grandes oportunidades que nos brinda el despliegue de MCUs en infraestructuras Cloud es la de poder adaptar los recursos disponibles a la demanda de usuarios de una manera dinámica y prácticamente instantánea. La forma de adaptar estos recursos puede realizarse modificando el número de máquinas virtuales disponibles o modificando las capacidades de las ya existentes. Pero como sabemos, una MCU consta de varias capas de operación y control que hay que tener presentes a la hora de decidir cómo realizamos esta adaptación.

En esta sección propongo un modelo de arquitectura que permite monitorizar el estado de los recursos disponibles en un sistema de videoconferencia y actuar en consecuencia añadiendo o quitando recursos al mismo. El modelo propuesto, además, permite configurar el uso de los APIs de proveedores de IaaS para gestionar de forma automática los recursos.

3.3.1 Requisitos y bases de diseño

Los requisitos de una arquitectura de videoconferencia que permita escalar los recursos de forma dinámica a la demanda son los siguientes:

- **Control centralizado:** es necesaria la existencia de un módulo de control que gestione los recursos disponibles en el sistema para poder asignar las nuevas conexiones a las MCUs pertinentes. Hay que tener en cuenta que en un modelo de MCUs centralizadas una sesión de videoconferencia debe estar siempre gestionada por una misma MCU. Por lo tanto no es factible un modelo en el que el control esté descentralizado.
- **Balanceo de carga:** en un escenario en el que existen varias MCUs disponibles es crucial la manera en que se programan los recursos en dichas MCUs. Cuando se requiere asignar

3.3. ESCALABILIDAD DINÁMICA A LA DEMANDA

una nueva conexión (un nuevo participante se ha unido a una sesión de videoconferencia), el sistema debe permitir tomar la decisión de en cuál de las MCUs disponibles hacerlo. Esta decisión debe poder tomarse en función del estado de cada MCU o de ciertas características prefijadas.

- **Actuador de escalabilidad:** en un sistema en el que queremos añadir y quitar recursos en función de las necesidades de cada momento es necesario automatizar dicha tarea. Debe existir un módulo que se encargue de arrancar y parar máquinas virtuales cuando sea necesario y para ello debe realizar llamadas a las APIs correspondientes de los proveedores IaaS.
- **Monitorización de recursos disponibles:** si escalamos el servicio añadiendo MCUs como unidad básica de recursos, el sistema debe ser consciente del estado de dichas MCUs en cada momento para tener la capacidad de actuar en consecuencia. Esto es necesario tanto para la asignación de recursos a las MCUs disponibles (balanceo de carga) como para la decisión de arranque o parada de nuevas MCUs (actuador de escalabilidad).
- **Movilidad de flujos:** cuando descende la demanda de uso y tenemos disponibles más recursos de los necesarios es frecuente la necesidad de parar MCUs y retirarlas del conjunto de recursos disponibles. Sin embargo puede ocurrir que los recursos sobrantes no estén concentrados en las mismas MCUs sino repartidos entre varias de ellas. En ese caso tenemos que reagrupar las conexiones en un subconjunto de MCUs para liberar el resto y poder apagarlas. Esta reagrupación implica mover los flujos multimedia entre MCUs y esta movilidad debe realizarse de forma transparente para los usuarios y sin pérdidas en la transmisión de datos.

Teniendo en cuenta estos requisitos voy a tomar como punto de partida para el diseño de la arquitectura el modelo de videoconferencia como servicio de Marte 3.0, descrito en la Sección 2.1.4. Esta arquitectura es posible gracias a Nuve [Rodríguez 2009], un componente que permite proveer de forma segura salas de videoconferencia a terceros. El detalle de Marte 3.0 puede encontrarse en [Cerviño 2008] y el de su integración con Nuve en la tesis doctoral [Cerviño 2012], desarrollada también en el ámbito de mi grupo de investigación.

Nuve: salas de videoconferencia como servicio

Nuve define un interfaz para integrar un sistema de videoconferencia en otros servicios de tal forma que éstos puedan desentenderse de la gestión de los recursos de videoconferencia. Además provee un sistema de autorización llamado *MAuth* que permite securizar el acceso al sistema de videoconferencia. Según sus autores, los objetivos que Nuve consigue son los siguientes:

- La gestión de los usuarios y su autorización la lleva a cabo la aplicación de terceros, no el núcleo de videoconferencia.
- Se centra en la gestión de salas de videoconferencia, siendo una sala de videoconferencia un espacio virtual donde los usuarios comparten sus flujos multimedia.
- La interacción entre las aplicaciones y el núcleo de videoconferencia se realiza de manera segura.
- Permite el despliegue del núcleo de videoconferencia de manera escalable en un entorno de computación en la nube. Esto permite asegurar un mínimo de calidad de servicio para las salas de videoconferencia aprovechando las capacidades de los proveedores de *Cloud Computing*.

Tabla 3.1 : Definición de los recursos en Nuve

Recurso	Definición
Service	Es una aplicación de terceros que tiene posibilidad de gestionar salas y pedir acceso a las mismas para sus usuarios.
User	Representa a un usuario presente en el sistema de conferencia. Este usuario, a través de un servicio, accede a salas de videoconferencia.
Room	Representa una sala de conferencia donde los usuarios provenientes de los servicios se conectan para publicar y suscribirse a flujos multimedia.
Token	Es la ficha usada para delegar la autorización de los usuarios a los servicios.

En resumen, Nuve propone convertir el núcleo de videoconferencia en un *proveedor de salas de comunicación en tiempo real*. En la tabla 3.1 podemos ver una relación de los recursos que

3.3. ESCALABILIDAD DINÁMICA A LA DEMANDA

gestiona Nuve. El funcionamiento de un componente que implemente Nuve puede entenderse mejor con un ejemplo de flujo de interacción entre un servicio, un usuario, Nuve y el sistema de videoconferencia, una MCU.

Figura 3.4 : Flujo de interacción de Nuve

En primer lugar el servicio es registrado en Nuve, con lo que obtiene unas credenciales que le permiten gestionar recursos de videoconferencia de forma autenticada y nominal. A partir de ahora todos los recursos que cree este servicio pertenecerán únicamente a él mismo. El servicio puede entonces crear una sala de videoconferencia en Nuve haciendo uso de sus credenciales. Cuando un usuario o cliente quiere conectarse a dicha sala, comienza el proceso de conexión. En la Figura 3.4 podemos ver el detalle de la interacción entre los componentes durante este proceso:

1. El cliente necesita utilizar un *token* para conectarse a la sala. Por lo tanto le pedirá al servicio en cuestión que le consiga uno.
2. El servicio pedirá a Nuve un *token* que autorice al usuario a conectarse a la sala. Para proceder a la creación de este *token* Nuve comprobará que el servicio tiene los permisos adecuados y la capacidad de conectar más usuarios a la sala. Además este *token* será creado para un usuario en concreto, con determinados roles o permisos sobre la sesión y únicamente autorizado para conectarse a esa sala en concreto.
3. Una vez creado devolverá el *token* al servicio

4. y el servicio al cliente.
5. El cliente que va a conectarse a una sala del sistema de videoconferencia deberá utilizar el *token* durante el proceso de conexión por lo que lo incluirá en su petición de conexión a la MCU.
6. El sistema de videoconferencia (la MCU) comprobará entonces la validez del *token*. En primer lugar mediante una firma de seguridad y en segundo lugar preguntando a Nuve si es válido para acceder a la sala solicitada.
7. En caso afirmativo el sistema de videoconferencia dispondrá además de información sobre el usuario para el que el *token* ha sido creado. Esta información incluye los roles asignados en su creación mediante los cuales el sistema de videoconferencia podrá determinar qué acciones puede o no puede realizar el usuario sobre la sala (por ejemplo publicar un flujo multimedia).

Como podemos ver, incluir Nuve en mi arquitectura hace que se cumpla una parte del primer requisito expuesto al principio, disponemos de un componente central que gestiona los recursos de videoconferencia. En concreto Nuve tiene constancia de las salas disponibles en el sistema por lo tanto parece que puede ser éste componente el encargado de gestionar la asignación de dichas salas a cada una de las MCUs. Tomaré esta idea como punto de partida para el diseño de la arquitectura.

3.3.2 Arquitectura

Partiendo por tanto de la idea de Nuve y su provisión de salas de videoconferencia como servicio, propongo una arquitectura que extiende este modelo para cumplir los requisitos enunciados al comienzo de esta sección. El resultado queda reflejado en la Figura 3.5. Como vemos, he añadido el módulo Cloud Handler al núcleo de Nuve con su submódulo IaaS Controller. Como es lógico ahora disponemos de un conjunto de MCUs y toman parte en la arquitectura los proveedores de recursos Cloud correspondientes, en este caso de infraestructura. La última novedad es un bus de mensajes que como explicaré a continuación servirá de medio de comunicación entre las MCUs, Nuve y Cloud Handler.

3.3. ESCALABILIDAD DINÁMICA A LA DEMANDA

Figura 3.5 : Arquitectura de escalabilidad a la demanda

Nuve

Nuve tiene las mismas funciones que las explicadas anteriormente con una única diferencia. Ahora para realizar tareas sobre una sala necesita saber en qué MCU se encuentra gestionada. Esto ocurre para las acciones de borrado de una sala (desconexión de todos sus participantes), borrado de un usuario (desconexión de ese participante), consulta de los usuarios conectados a una sala y creación de *tokens* de conexión para un cliente.

Ahora en un *token* Nuve debe incluir información sobre la MCU a la que el cliente tiene que conectarse. Para ello preguntará a Cloud Handler qué MCU debe utilizar para conectar a un cliente a la sala solicitada. En caso de que ya haya usuarios conectados a esa sala, la MCU será la que ya esté gestionándola. En caso de que sea una nueva sala, Cloud Handler decidirá a qué MCU debe asignar su gestión. Para borrar una sala o un usuario en una MCU así como para consultar información sobre los usuarios conectados a una sala, Nuve también pedirá a Cloud Handler información sobre la MCU donde ésta se encuentra alojada. De esta forma podrá enviar a la MCU las órdenes pertinentes.

Service

Se trata de un servicio registrado en Nuve y que provee salas de videoconferencia a sus usuarios. Responde a la definición de servicio dada en la sección anterior.

Client

Se trata de un cliente (usuario) que se conecta a salas de videoconferencia. Responde a la definición de usuario dada en la sección anterior.

Message Bus

Para el intercambio de mensajes entre los diferentes componentes de esta arquitectura, se utiliza un bus de mensajes tipo *publish/subscribe* de tal forma que en un despliegue cada componente únicamente debe conocer la dirección de este bus para poder comunicarse con el resto. Esto resulta útil en una arquitectura en la que el tipo y número de componentes es dinámico.

MCUs

Una MCU constituye el núcleo de videoconferencia del sistema y tiene los componentes indicados en la Sección 2.1.2. Como he explicado antes, una MCU recibe peticiones de conexión a salas de videoconferencia por parte de los clientes. Estas peticiones van autenticadas por un *token* de conexión y una vez comprobada localmente su validez, la MCU pregunta a Nuve (a través del bus de mensajes) si ese *token* tiene los permisos adecuados para proceder a la conexión. En ese momento puede comenzar a realizar las tareas de señalización para establecer la conexión y una vez concluidas satisfactoriamente comenzar con el intercambio de medios según los requerimientos del cliente.

En este modelo, varias MCUs pueden coexistir y un cliente se conectará a una u otra en base a lo que Nuve haya indicado durante la creación del *token*. Como hemos visto es Cloud Handler el componente encargado de decidir y decirle a Nuve qué MCU debe indicar, más adelante veremos según qué criterios. Pero lo que está claro es que Cloud Handler debe tener constancia de las MCUs disponibles para poder tomar dicha decisión. Es la propia MCU la que se registra como tal en este componente.

Así, cuando una nueva MCU arranca y está lista para comenzar a recibir peticiones de

3.3. ESCALABILIDAD DINÁMICA A LA DEMANDA

conexión, envía un mensaje a Cloud Handler para registrarse como una nueva MCU disponible. En este mensaje la MCU indica ciertos parámetros necesarios para ser alcanzada por los clientes (datos de conexión) y otros metadatos configurables que pueden ser útiles para Cloud Handler a la hora de tomar la decisión de asignación de recursos. Ejemplos de estos metadatos podrían ser la localización de la MCU, el proveedor IaaS en el que se encuentra corriendo, etc.

Una vez registrada, la MCU envía a Cloud Handler mensajes de *keep alive* periódicos para indicar que sigue levantada y disponible para recibir conexiones. Cuando Cloud Handler deja de recibir estos mensajes durante un tiempo determinado interpreta que la MCU ya no está disponible y la elimina de su registro de MCUs (registro que también explicaré más adelante).

Las decisiones de asignación que toma Cloud Handler pueden estar basadas en los datos de una MCU configurados durante el registro o en datos sobre el estado de una MCU en tiempo real. Como he adelantado en 3.2.2, caracterizar el estado de una MCU según su consumo en términos de computación no es una buena solución para la toma de decisiones de asignación de recursos. Una cantidad de recursos dada puede servir para alojar diferente número de conexiones dependiendo del tipo de sala de videoconferencia, la calidad de los flujos multimedia intercambiados, las funciones que se realicen, etc. Por lo tanto es necesario caracterizar la MCU según otras métricas que nos den una idea clara de la cantidad de conexiones que una MCU puede gestionar. En función de esta caracterización y el número de unidades métricas que esté gestionando una MCU en un momento determinado, ésta reportará a Cloud Handler su estado según unos baremos establecidos.

Por lo tanto esta métrica debe permitirnos extrapolar el consumo de recursos calculado en unas condiciones conocidas al consumo que se producirá frente a otras condiciones. Como podremos comprobar en la sección 3.3.3, en un modelo de MCU centralizada donde una misma sala de videoconferencia se gestiona siempre en una única MCU, esta unidad métrica es la sala de videoconferencia. Sabiendo el consumo de recursos de una MCU cuando gestiona una sala de un número determinado de participantes, podemos extrapolar el consumo que se producirá para N salas.

Teniendo en cuenta esta caracterización y por lo tanto el número de salas máximo que una MCU puede alojar, defino los estados reflejados en la Tabla 3.2 y que indican la capacidad de una MCU para alojar nuevas salas. Cada vez que una MCU reciba la petición de alojar una nueva sala o deje una sala vacía, enviará un mensaje a Cloud Handler actualizando su estado con uno de los

Tabla 3.2 : Definición de estados de MCUs

Estado	Definición
Not available	La MCU se encuentra al límite de su capacidad y por lo tanto no puede alojar nuevas salas de videoconferencia.
Warning	La MCU se encuentra en estado crítico de uso de recursos y por lo tanto no es conveniente asignar nuevas salas de videoconferencia.
Available	La MCU tiene recursos disponibles y por lo tanto puede alojar nuevas salas de videoconferencia.

valores definidos. Cloud Handler podrá utilizar esta información para balancear la carga y para tomar decisiones de asignación.

Cloud Handler

Cloud Handler es el componente encargado de balancear la carga entre MCUs y de gestionar los recursos disponibles. Para el balanceo de carga dispone de un registro de las MCUs disponibles en el sistema y una relación de las salas asignadas a cada una de ellas en cada momento. Además y como ya hemos visto, recibe de manera periódica un reporte con el estado de cada MCU de tal manera que es capaz de decidir en cuál de ellas asignar una nueva sala.

Para tomar esta decisión Cloud Handler dispone de dos tipos de información sobre las MCUs. Por una parte de la información fija configurada en proceso de registro de cada MCU. Esta información puede resultar útil cuando las decisiones se basan en el tipo de MCU, en su localización, en el proveedor de Cloud en el que está desplegada, etc. En definitiva se trata de parámetros que no van a variar a lo largo del tiempo ni en función del estado de las sesiones.

Por otra parte dispone de información sobre el estado de las MCUs en tiempo real. Esta información se recibe en los reportes que envía cada MCU actualizando su estado según los definidos arriba. Al recibir un reporte, Cloud Handler actualiza una lista de prioridades que le indica en todo momento el orden de asignación atendiendo a criterios de capacidad. Esta actualización se realiza según el Algoritmo 1. Como podemos ver, al final de la ejecución del algoritmo tenemos una lista ordenada según el estado de cada MCU. Frente a una petición de Nuve para asignar una nueva sala de videoconferencia, Cloud Handler puede combinar la información fija de cada MCU con la información extraída de esta lista de prioridades para tomar

3.3. ESCALABILIDAD DINÁMICA A LA DEMANDA

Algorithm 1 Cálculo de prioridad de MCUs

Require: *mcus*

```

1: priorityQueue  $\leftarrow$  newArray
2: for all mcus do
3:   status  $\leftarrow$  getMcuStatus(mcu)
4:   if status == AVAILABLE then
5:     priorityQueue  $\leftarrow$  mcu
6:   end if
7: end for
8: for all mcus do
9:   status  $\leftarrow$  getMcuStatus(mcu)
10:  if status == WARNING then
11:    priorityQueue  $\leftarrow$  mcu
12:  end if
13: end for
14: for all mcus do
15:   status  $\leftarrow$  getMcuStatus(mcu)
16:   if status == NOT_AVAILABLE then
17:     priorityQueue  $\leftarrow$  mcu
18:   end if
19: end for
20: return priorityQueue

```

Tabla 3.3 : Interfaz de decisión de MCUs en Cloud Handler

<i>selectMcu</i>	
Interfaz que dada una sala, la lista de MCUs disponibles, la información configurada para cada una de ellas y la cola de prioridad según recursos, devuelve la MCU elegida para alojar dicha sala.	
Parameters	room, MCU list, MCU priority queue
Returns	selected mcu

la decisión de asignación.

En este modelo las políticas de decisión son configurables por lo que pueden adaptarse a cada caso de uso y escenario concreto. Cloud Handler define una interfaz con los parámetros de entrada y salida descritos en la Tabla 3.3. A la hora de seleccionar una MCU para asignar una nueva sala ejecutará la orden *selectMCU* dando la información de la sala, las MCUs disponibles y la cola de prioridades y esperará obtener una de las MCUs como resultado de la decisión. En caso de que no se especifique una política concreta siempre elegirá la primera según la lista de prioridad.

Tras la ejecución del algoritmo de cálculo de prioridades y en función del nuevo estado del conjunto de las MCUs disponibles, Cloud Handler puede decidir que es necesario añadir o eliminar

MCUs en dicho conjunto. En su caso, pedirá a su submódulo IaaS Controller que proceda con la acción pertinente, arrancar o parar una MCU. En caso de parada la eliminará de su registro de MCUs. En caso de arranque la MCU será la que se registre de forma automática una vez esté lista, como he explicado antes. Esta acción puede realizarse sobre diferentes proveedores IaaS y arrancando MCUs con diferentes características como su localización, número de CPUs, ancho de banda necesario, etc.

De igual manera que las políticas de decisión de asignación son configurables, la decisión de cómo y dónde arrancar MCUs también lo es. Y para tomar esta decisión pueden tenerse en cuenta diversos parámetros como la eficiencia de las MCUs en cada proveedor y en cada momento, las localizaciones disponibles o los precios de las instancias virtuales, redes, etc. Se trata por tanto de una decisión compleja y muy dependiente de cada caso de uso. No es objeto de esta tesis entrar en detalles sobre estas decisiones, solamente proveer las herramientas necesarias para poder tomarlas.

Para enviar las peticiones a cada proveedor IaaS, Cloud Handler dispone de una interfaz común que le permite realizar las acciones de parar o arrancar una MCU de forma unificada e independiente de los requisitos específicos de cada uno de ellos. Esta interfaz es IaaS Controller, que dispone de la definición de cada API de gestión del proveedor. Dispone también de una relación de cada MCU y el proveedor en el que se ha arrancado lo que le permite saber a dónde mandar la petición de parada en caso necesario.

En la petición de arranque de una MCU puede incluirse cierta información de contexto para la configuración de la MCU. La información fundamental es la dirección donde se encuentra el Bus de Mensajes ya que es ahí donde tendrá que enviar el mensaje de registro una vez lista. En algunos proveedores IaaS las instancias virtuales no tienen acceso a sus direcciones IP públicas. Y esta información es imprescindible para Cloud Handler, que ha de incluirla como parte de los datos de conexión cuando Nuve le pide una MCU para incluir en un token. Dispone Cloud Handler de una llamada que puede hacer a IaaS Controller para preguntar a los proveedores IaaS por la dirección pública de una instancia dada.

IaaS Provider

Se trata de los diferentes proveedores Cloud IaaS en los que Cloud Handler puede pedir arrancar MCUs. Típicamente estos proveedores tendrán configurada una imagen de MCU lista para ser arrancada con cierta configuración de contexto que IaaS Controller proporcionará en sus

3.3. ESCALABILIDAD DINÁMICA A LA DEMANDA

Tabla 3.4 : Mensajes entre los módulos de la arquitectura (I)

<i>getMcuForRoom</i>	
Pregunta a Cloud Handler en qué MCU asignar una petición de conexión a una sala. En caso de ser una sala nueva Cloud Handler usará sus políticas de decisión para determinarlo.	
From	Nuve
To	Cloud Handler
Parameters	room id
Returns	mcu info
<i>getUsersInRoom</i>	
Pide la lista de usuarios en una sala.	
From	Nuve
To	MCU
Parameters	room id
Returns	user list
<i>deleteRoom</i>	
Solicita el borrado de una sala. Implica la desconexión de la sala de todos los clientes presentes.	
From	Nuve
To	MCU
Parameters	room id
Returns	result
<i>deleteUser</i>	
Solicita el borrado de un usuario, implica la desconexión de ese usuario.	
From	Nuve
To	MCU
Parameters	user id
Returns	result
<i>validateToken</i>	
Solicita la validación de un token de conexión a una sala de videoconferencia.	
From	MCU
To	Nuve
Parameters	token
Returns	user info
<i>addNewMcu</i>	
Registra una nueva MCU en Cloud Handler.	
From	MCU
To	Cloud Handler
Parameters	mcu info
Returns	mcu id

peticiones. IaaS Controller dispondrá por tanto del identificador de esta imagen en cada uno de los proveedores así como una implementación del API de gestión correspondiente.

Mensajes entre los módulos

En las Tablas 3.4 y 3.5 podemos ver una relación de los diferentes mensajes intercambiados entre los módulos descritos anteriormente. Para cada mensaje se indica el componente origen, el

Tabla 3.5 : Mensajes entre los módulos de la arquitectura (II)

<i>keepAlive</i>	
Envía un mensaje de <i>keep alive</i> indicando que la MCU sigue disponible para recibir conexiones.	
From	MCU
To	Nuve
Parameters	mcu id
Returns	result
<i>updateStatus</i>	
Actualiza el estado según la tabla de estados definida.	
From	MCU
To	Cloud Handler
Parameters	status
Returns	result
<i>launchMCU</i>	
Pide a IaaS Controller que arranque una MCU en un IaaS Provider determinado y con ciertos requisitos. Puede incluir información de contexto para la configuración de la MCU.	
From	Cloud Handler
To	IaaS Controller
Parameters	provider, requirements, context
Returns	result
<i>stopMCU</i>	
Pide a IaaS Controller que pare una MCU en el IaaS Provider en el que se encuentra.	
From	Cloud Handler
To	IaaS Controller
Parameters	mcu id
Returns	result
<i>getMcuPublicAddress</i>	
Solicita la dirección IP pública de una determinada MCU.	
From	Cloud Handler
To	IaaS Controller
Parameters	mcu id
Returns	public ip

componente destino, los parámetros de entrada y los de salida así como una pequeña descripción que resume lo explicado antes.

3.3.3 Implementación y caracterización

La solución propuesta ha sido implementada como parte del proyecto Licode, ya introducido en la Sección 1.4.1. La primera implementación de este proyecto se realizó de acuerdo a este modelo, utilizando como MCU *Erizo*, una MCU modular compatible con WebRTC y explicada con detalle en la tesis [Rodríguez 2016a].

Erizo se divide en dos grandes partes, la capa de medios (el núcleo) y la capa de control, llamada *Erizo Controller*. El núcleo de Erizo se encarga de todas las tareas relacionadas con el

3.3. ESCALABILIDAD DINÁMICA A LA DEMANDA

manejo de flujos multimedia. Es modular porque estas tareas se realizan combinando componentes de dos tipos, nexos y operaciones. Los nexos se encargan de transformar protocolos externos al protocolo que entienden las operaciones, RTP. Como es compatible con WebRTC, los flujos de media entre el Cliente y Erizo se intercambian mediante SRTP por lo que el núcleo implementa este protocolo así como el protocolo DTLS, para el intercambio de claves de cifrado. Las operaciones se encargan de manejar estos flujos (ya en RTP) para transcodificar, grabar, multiplexar, etc. Para establecer la conexión de medios con los clientes implementa el protocolo ICE

Por su parte Erizo Controller se encarga de implementar la capa de control del sistema, realizar la señalización y establecer las diferentes configuraciones del núcleo en función del escenario. En control se basa en salas de videoconferencia por lo que su integración con Nuve es casi inmediata. En cuanto a la señalización, Erizo implementa un protocolo basado en SDPs. Este es el componente que se comunica con los clientes para permitir a éstos la conexión a salas de videoconferencia. Por último, para facilitar la tarea a los desarrolladores de aplicaciones Web, existe un API de cliente que permite la comunicación entre éste y Erizo Controller a través de un protocolo de Web Sockets.

En Licode, la implementación de Cloud Handler y de una versión de Nuve actualizada es tarea mía y esta implementación inicial será la base para los siguientes pasos de mi investigación. Ambos módulos están implementados usando la tecnología Node.js. Incluyo también una serie de librerías para facilitar las llamadas al API de Nuve, en este caso en tecnologías Node.js, Python y Ruby. De esta manera cualquier servicio implementado con una de estas tres tecnologías podrá gestionar recursos de Nuve usando estas librerías.

Para el Bus de Mensajes Licode utiliza el protocolo AMQP (*Advanced Message Queuing Protocol*), en concreto la implementación RabbitMQ con un mecanismo de llamadas a procedimiento remoto RPC (*Remote Procedure Call*) para el intercambio de mensajes. En un primer momento el módulo IaaS Controller era compatible con Amazon Web Services, en concreto con el API de EC2.

A continuación explico un ejemplo de caracterización de Licode (en concreto de Erizo) que muestra cómo debe realizarse el perfil de una MCU para establecer las métricas que nos permiten determinar el número máximo de salas que puede gestionar. Para el experimento despliego una MCU Erizo en una máquina virtual pequeña de AWS EC2, equivalente a un procesador Intel

Xeon a 1GHz. Diseño dos escenarios que son los más comunes en sistemas de videoconferencia: un streaming de vídeo en tiempo real y una videoconferencia multiusuario. En ambos sistemas realizo una monitorización del consumo de memoria y CPU así como del uso de ancho de banda entrante y saliente.

(a) Uso de CPU

(b) Consumo de ancho de banda

Figura 3.6 : Uso de recursos de una MCU en un escenario de streaming en tiempo real.

En el primer escenario, streaming en vivo, uno de los clientes está publicando su flujo de audio y vídeo en la sesión y los clientes que se suscriben a él van añadiéndose de forma gradual. En la Figura 3.6 podemos observar cómo el uso de CPU en la MCU aumenta de forma lineal con el incremento de usuarios que se suscriben al streaming. Esto ocurre porque el estándar WebRTC, como ya se ha explicado, utiliza SRTP para la transmisión de paquetes y por lo tanto la MCU tiene que desproteger y volver a proteger los paquetes para realizar la transmisión desde el cliente que publica hacia los que se suscriben.

3.3. ESCALABILIDAD DINÁMICA A LA DEMANDA

También podemos observar cómo el ancho de banda entrante es constante durante toda la sesión y el saliente aumenta también de forma lineal debido a que cada nuevo cliente conectado implica un nuevo flujo de salida. Acerca de la memoria utilizada aumenta también de forma lineal pero con una variación mínima durante la sesión (de unos 10 MB), y por tanto no relevante para el experimento. Finalmente puede observarse una pequeña anomalía cuando se conecta el usuario número 8 y el número 13 que probablemente sea debida a un error en el rendimiento del cliente que publica los datos, que también corre en una máquina de Amazon EC2.

En el segundo escenario, la videoconferencia multiusuario, cada usuario que se conecta a la sesión publica su flujo de audio y vídeo y además se suscribe al resto de usuarios que estaban conectados previamente. Se ha establecido un límite de seis usuarios ya que es el número que comúnmente se utiliza en este tipo de sesiones.

(a) Uso de CPU

(b) Consumo de ancho de banda

Figura 3.7 : Uso de recursos de una MCU en un escenario de videoconferencia

En la Figura 3.7 podemos observar cómo en este caso el consumo de ancho de banda entrante aumenta linealmente con el incremento de usuarios en la sesión debido al hecho de que cada nuevo usuario publica su flujo en el sistema. Sin embargo el ancho de banda saliente y el consumo de CPU se incrementan de forma cuadrática debido a que por cada nuevo usuario la MCU debe redirigir el nuevo flujo al resto de usuarios. Por lo tanto el número de flujos de salida aumentará siguiendo la siguiente ecuación:

$$N = n(n - 1) \quad (3.1)$$

Donde n es el número de usuarios en la sala. El uso de memoria en este escenario también aumenta de forma cuadrática pero de nuevo con una variación insignificante para este estudio. Para este escenario por tanto podríamos establecer el límite de una sala como máximo para una MCU desplegada en una máquina virtual de estas características. Obviamente se trata de un ejemplo en el que he utilizado una instancia con capacidad muy limitada. Simplemente se trata de ilustrar el mecanismo a seguir en un despliegue real.

3.4 Movilidad de flujos multimedia

En la anterior sección he resuelto todos los requisitos planteados para el diseño de una arquitectura de videoconferencia escalable a la demanda de usuarios excepto uno, la movilidad de flujos multimedia. El modelo propuesto nos permite añadir recursos de computación cuando la demanda de usuarios aumenta. Sin embargo, dado que hablamos de MCUs monolíticas en las que una misma sala de videoconferencia ha de estar gestionada por la misma MCU, en caso de que una vez alcanzado el límite de capacidad de una MCU sea necesario seguir añadiendo participantes, deberemos mover la gestión de esa sala a una máquina con más capacidad.

Por otra parte, frente a una bajada en la demanda, el modelo también nos permite eliminar recursos mediante la parada de MCUs. Sin embargo, esta acción no siempre es trivial dado que los recursos libres tras esa bajada en la demanda pueden no estar concentrados en las mismas MCUs sino repartidos entre ellas. Por lo tanto podemos tener un escenario en el que múltiples MCUs se encuentran utilizando una parte reducida de sus recursos. El comportamiento ideal es el de reagrupar las conexiones en un subconjunto de MCUs y apagar las restantes.

Como vemos, existen diversos escenarios en los que la movilidad de flujos entre MCUs es

3.4. MOVILIDAD DE FLUJOS MULTIMEDIA

necesaria. Sin embargo, los sistemas de videoconferencia imponen ciertos requisitos a la hora de mover conexiones entre MCUs ya que los flujos multimedia de vídeo y audio no deben ser interrumpidos durante el proceso. La solución que propongo se basa en la especificación MICE (*Mobility with ICE*), que extiende las especificaciones TURN y ICE para realizar movilidad en escenarios genéricos. Como veremos a continuación MICE no resuelve de manera total nuestro caso de escalabilidad de MCUs hacia abajo por lo que es necesario adaptarlo a los requisitos concretos del escenario añadiendo algunos atributos y parámetros a los mensajes STUN. Pero vayamos paso a paso analizando el escenario y las posibles soluciones al problema.

3.4.1 Soluciones existentes

Para poder abordar el problema con conocimiento de causa, lo primero que voy a hacer es realizar un breve repaso por las soluciones de movilidad existentes en la literatura. No existen soluciones específicas para la movilidad de flujos multimedia entre MCUs, pero sí algunas que resuelven el problema de movilidad entre pares en escenarios de comunicación en tiempo real. La mayoría de ellas están orientadas a la movilidad de dispositivos en redes heterogéneas.

Por ejemplo los autores de [Corici 2008] proponen un servicio de movilidad para plataformas IMS (*IP Multimedia Subsystem*). Este servicio analiza además parámetros de red como la calidad de servicio (QoS). Por otra parte [Mineno 2008] propone un mecanismo de pre-negociación para SIP centrado en teléfonos móviles que se mueven entre redes de comunicación y [Maltz 1998] hace algo similar pero utilizando varias interfaces de red al mismo tiempo. Por último en [Kristiansson 2004] se define una abstracción a la capa de aplicación para poder añadir o quitar *sockets* de comunicación sin afectar a las aplicaciones que corren en los dispositivos.

Este tipo de mecanismos podrían aplicarse a un escenario de videoconferencia en el que, en definitiva, una MCU actúa como un par más en la comunicación implementando protocolos de conexión como SIP o ICE. Sin embargo, realizar una re-negociación suele implicar latencia en la comunicación e interrupciones en los datos multimedia recibidos por los usuarios. Además, todas estas propuestas implican implementaciones específicas en los pares lo que puede ser un problema en escenarios con dispositivos heterogéneos. Por ejemplo, en un escenario de videoconferencia Web habría que modificar la implementación de los navegadores para soportar nuevos protocolos, cosa inviable. La solución óptima es una en la que la movilidad sea transparente para los pares y el único nodo que tenga que implementar modificaciones sea la MCU.

La especificación MICE [Wing 2013] propone dos mecanismos para realizar movilidad. La primera de ellas se basa en ICE añadiendo atributos de STUN para soportarla. Por lo tanto asume que todos los pares involucrados soportan MICE, cosa que nos lleva al mismo problema enunciado antes. El segundo mecanismo se basa en TURN y considera que solamente uno de los pares soporta la nueva especificación. Este será el punto de partida para describir mi solución. Pero el mecanismo presenta algunos problemas que veremos a continuación.

3.4.2 Descripción del escenario

Como hemos visto, la solución de movilidad de flujos multimedia que mejor se adapta al escenario de videoconferencia es la especificación MICE. No obstante, es necesario adaptarla a las necesidades concretas de este escenario añadiendo algunas características y modificando algunas partes de su comportamiento. A continuación voy a analizar las carencias de MICE que ponen de manifiesto esta necesidad de adaptación.

Figura 3.8 : Arquitectura de movilidad de flujos multimedia.

En la Figura 3.8 podemos observar un escenario donde el *Peer A* y el *Peer B* están conectados a la *MCU 1* y preparados para intercambiar paquetes de datos con ella. Ambos participantes podrían estar conectados directamente desde una red pública o en una red privada a través de un NAT. Para establecer la conexión con el dispositivo MCU han utilizado el protocolo ICE. El *Peer A* está enviando un flujo de datos a la *MCU 1* y ésta lo está redirigiendo al *Peer B*. En otras palabras, el *Peer A* publica un flujo multimedia y el *Peer B* se suscribe a él. Mi objetivo es mover la gestión

3.4. MOVILIDAD DE FLUJOS MULTIMEDIA

de ese flujo multimedia de *MCU 1* a *MCU 2* para que el *Peer A* envíe los paquetes a *MCU 2* y ésta los redirija a *Peer B* (como muestra la línea de puntos en la figura). Como he explicado antes este proceso resulta de gran utilidad en escenarios en los que realizamos escalabilidad de MCUs. Si necesitamos apagar la máquina en la que está alojada *MCU 1* y *MCU 2* tiene recursos disponibles, de esta manera lograremos redistribuir los recursos utilizados.

Este es un escenario básico en que el *Peer A* solamente envía datos y el *Peer B* únicamente los recibe. Sin embargo en caso de que ambos envíen y reciban flujos de datos el problema es el mismo pero duplicado. De igual manera en configuraciones complejas con más de dos participantes conectados a una misma MCU las características del escenario pueden extrapolarse. En un escenario de *streaming* uno de los participantes publica su flujo y la MCU lo redirige al resto. Por su parte, en una sesión de multi videoconferencia, cada uno de los participantes publican y la MCU redirige al resto. Dicho esto, en esta sección voy a analizar el caso simple explicado en la Figura 3.8 y su solución podrá extrapolarse fácilmente al resto de escenarios y configuraciones.

Como he introducido en la Sección 3.4.1, la especificación MICE describe dos mecanismos para realizar movilidad de participantes entre redes en tiempo real. La segunda de ellas puede aplicarse en escenarios donde solamente uno de los participantes soportan MICE. El caso de movilidad de flujos entre MCUs es muy parecido a éste ya que solamente queremos modificar la dirección IP de uno de los extremos, la MCU. Sin embargo es importante recordar que hablamos de escenarios de tiempo real en los que la experiencia de usuario debe ser la mejor posible. Esto significa que cuando movamos los flujos multimedia de *MCU 1* a *MCU 2*, el *Peer 2* no debe experimentar ningún tipo de interrupción en la recepción de paquetes. Esto hace que MICE no sea válido por sí mismo.

El mecanismo propuesto por MICE extiende la especificación TURN descrita en [Mahy 2010]. Para poner en contexto al lector explicaré brevemente en que consiste un servidor TURN actuando de *relay*. Normalmente se utiliza este tipo de configuración TURN para disponer de un nodo intermedio entre dos pares localizado en una red pública y al que ambos tienen acceso. De esta forma solamente debe ser conocida la dirección de este servidor y puede conseguirse conectividad en escenarios con dispositivos NAT restrictivos. Así, el servidor TURN establece una *asignación* (*TURN allocation*) entre la dirección del par llamado *cliente* y la dirección de *relay*. Cada asignación se identifica mediante la dirección de *relay* y tiene asociada una estructura de datos (*5-tuple*) con la información del cliente.

En nuestro escenario, colocando un servidor TURN actuando a modo de *relay* entre los extremos, los participantes corresponden con la dirección de *relay* y las MCUs serían los clientes, cuya información se encuentra en las tuplas de cada asignación. En esta configuración, MICE permite refrescar una asignación TURN con un nuevo parámetro (MOBILITY-TICKET) que indica una variación en la dirección IP y puerto del cliente (la MCU). De esta manera el servidor TURN modifica la tupla asociada a la asignación reemplazándola con una nueva que contiene la nueva dirección IP/puerto.

A partir de ese momento todos los paquetes que provienen de la dirección/puerto de *relay* correspondiente a la asignación (los que provienen de los participantes) serán enviados a la nueva IP/puerto de cliente (una nueva MCU). Por otra parte todos los datos de aplicación que provienen de la nueva dirección IP/puerto serán enviados a través de la dirección de *relay* a los participantes. Pero en el momento de actualizar la tupla algunos paquetes pueden haber sido enviados a la antigua MCU y por lo tanto no serán procesados por el servidor de TURN cuando vuelvan porque la antigua tupla no existirá.

Figura 3.9 : Ejemplo de pérdida de paquetes en movilidad

La figura 3.9 muestra un ejemplo que permite entender con más claridad este problema. El servidor de TURN envía paquetes a *MCU 1*. Justo antes de enviar el paquete número 3 (*P3* en la figura) se cambia la tupla de la asignación de *MCU 1* a *MCU 2*. Por lo tanto el paquete número 4 se envía a *MCU 2*. Los paquetes 1, 2 y 3 están siendo procesados por la MCU y por lo tanto aún no han llegado al servidor de TURN. Cuando finalmente llegan la tupla de *MCU 1* no existirá

3.4. MOVILIDAD DE FLUJOS MULTIMEDIA

y estos paquetes serán ignorados. El resultado es una pérdida de paquetes en el cliente con la correspondiente interrupción y corte en la recepción del flujo multimedia.

En la siguiente sección propongo un mecanismo que, extendiendo la actual especificación MICE, permite resolver este problema. Durante la descripción de esta solución utilizaré algunos términos definidos en la especificación TURN y MICE. Intentaré explicar cada paso de tal forma que sea entendible sin ser expertos en dichas especificaciones. Pero en cualquier caso éstas han de servir como posible referencia en caso de duda [Mahy 2010, Wing 2013].

3.4.3 Extensión de la especificación MICE

Volviendo al ejemplo mostrado en la Figura 3.9, propongo un modelo en el que ambas tuplas (correspondientes a *MCU 1* y *MCU 2*) se mantienen asociadas a la misma asignación hasta que todos los paquetes enviados desde el TURN a *MCU 1* han sido procesados por ésta y han regresado al servidor TURN. Pero el hecho de que una misma asignación tenga dos tuplas de manera simultánea debe implicar solamente el canal duplicado en uno de los sentidos. El servidor TURN podrá recibir datos de aplicación de dos clientes diferentes y enviarlos a través de la misma dirección/puerto de *relay*. Sin embargo, en el otro sentido esto no debe ocurrir. Cuando el TURN recibe datos a través de la dirección/puerto de *relay* únicamente debe enviarlos al cliente correspondiente a la nueva tupla.

Además puede ocurrir que en el sistema se necesite realizar un nuevo proceso de movilidad desde *MCU 2* a una nueva MCU (*MCU 3*) antes de que *MCU 1* haya acabado de enviar datos de aplicación al servidor de TURN. En este caso la nueva MCU refrescará la asignación añadiendo su tupla y manteniendo las tuplas de *MCU 1* y *MCU 2*. De esta forma habilitamos una forma de realizar la movilidad de una forma rápida y dinámica. Pero la extensión propuesta deber soportar que una única asignación tenga asociada al mismo tiempo una tupla tradicional y múltiples tuplas especiales que solamente funcionan en una dirección, recibiendo datos de aplicación de los clientes y enviándolos a los participantes a través de la misma dirección/puerto de *relay* (según he explicado antes). Llamaré a estas tuplas especiales *deprecated-5-tuples*.

Son deprecadas porque su tiempo de vida debe ser limitado. Debido a razones de seguridad no es recomendable tener este tipo de tupas habilitadas de forma indefinida. Por lo tanto el servidor de TURN borrará una *deprecated-5-tuple* de una asignación tras un periodo de tiempo determinado. Este periodo de tiempo se especifica en un nuevo parámetro de configuración del

servidor de TURN, *deprecated-5-tuple-LIFETIME*. En cualquier caso el cliente al que pertenece una *deprecated-5-tuple* puede eliminarla de la asignación en cualquier momento enviando una petición *Refresh Request* con el atributo *LIFETIME* asignado con el valor 0. Si un servidor TURN no quiere soportar el uso de este tipo especial de tuplas puede configurar el tiempo de vida *deprecated-5-tuple-LIFETIME* a valor 0.

Como he introducido anteriormente, MICE define un nuevo atributo STUN llamado *MOBILITY-TICKET*. Un cliente que quiere soportar movilidad debe incluir este atributo a la hora de crear una nueva asignación en el servidor de TURN. Esto provoca que el servidor de TURN genere un ticket y lo incluya en el atributo con ese nombre de la respuesta *Allocate Success* que envía al cliente. Cuando el cliente quiere refrescar una asignación manteniendo la misma dirección/puerto de *relay* pero con una nueva dirección/puerto de cliente, debe incluir dicho ticket en la petición *Refresh Request*. El servidor de TURN validará el ticket internamente para asegurar que el cliente es el propietario de la asignación y en caso de que todo vaya bien procederá al cambio de IP/puerto de cliente manteniendo la misma dirección/puerto de *relay*.

En esta extensión defino un nuevo atributo de STUN llamado *SHARED-MOBILITY-TICKET* que, incluido en una petición *Refresh Request*, indica al servidor TURN que debe mantener la *deprecated-5-tuple* al refrescar la asignación. En el ejemplo anterior *MCU 1* obtiene el *SHARED-MOBILITY-TICKET* cuando crea su asignación en el servidor TURN y se lo envía a *MCU 2* antes de proceder a la movilidad de flujos. *MCU 2* incluye este ticket en su petición *Refresh Request* para autenticar el proceso de movilidad. Por lo tanto el *SHARED-MOBILITY-TICKET* está diseñado para ser compartido entre clientes para soportar movilidad entre ellos. Razón por la que se llama ticket de movilidad compartido. El ticket tradicional *MOBILITY-TICKET* y el nuevo ticket *SHARED-MOBILITY-TICKET* son compatibles por lo que un cliente puede pedir ambos tickets cuando crea una nueva asignación. Con el primero de ellos podrá realizar movilidad entre redes y con el segundo podrá realizarla con otro cliente.

La manera en la que un *SHARED-MOBILITY-TICKET* se intercambia entre clientes está fuera de la definición de esta especificación pero debe garantizarse que se hace de forma segura para evitar ataques de intermediario. Por último y para asegurar que la movilidad de flujos se realiza de manera satisfactoria, se recomienda que el nuevo cliente envíe algún tipo de notificación al antiguo en el momento en el que recibe la confirmación *Refresh Success*. De esta manera el antiguo cliente puede eliminar su *deprecated-5-tuple* de la asignación correspondiente. Para ello

3.4. MOVILIDAD DE FLUJOS MULTIMEDIA

enviará la ya mencionada petición *Refresh Request* con el atributo *LIFETIME* a 0. De otra forma podría ocurrir que se perdieran paquetes porque el servidor de TURN dejara de recibir paquetes del antiguo cliente antes de establecer la conexión con el nuevo.

Una vez explicado el funcionamiento general de la extensión propuesta, las especificaciones MICE y TURN resultan de la siguiente manera:

Creando una asignación (*Allocation*)

1. Enviando una petición *Allocate Request*

Además del proceso descrito en la Sección 5.1.1 de [Wing 2013], el cliente puede incluir el atributo *SHARED-MOBILITY-TICKET* con longitud 0. Esto indica que el cliente es un nodo que requiere movilidad de flujos multimedia entre clientes y necesita un ticket.

2. Recibiendo una petición *Allocate Request*

Además del proceso descrito en la Sección 5.1.1 de [Wing 2013], el servidor comprueba si el atributo *SHARED-MOBILITY-TICKET* está incluido en la petición. Si su longitud es 0 y la movilidad entre clientes está deshabilitada por las políticas locales (el parámetro de configuración del TURN *deprecated-5-tuple-LIFETIME* es 0) el servidor debe denegar la petición devolviendo el nuevo código de error *Shared Mobility Forbidden*. Si el servidor no puede entender el atributo *SHARED-MOBILITY-TICKET*, lo ignora.

Si el servidor puede procesar de forma satisfactoria la petición procede de la manera explicada en MICE, creando un nuevo ticket e incluyéndolo cifrado en el atributo de STUN *SHARED-MOBILITY-TICKET* de la respuesta.

3. Recibiendo una respuesta *Allocate Response*

El proceso es el mismo que el explicado en las secciones 5.1.3 y 5.1.4 de [Wing 2013] para la respuesta *Allocate Success* y para la respuesta *Allocate Error*. Pero en este caso el cliente guarda el ticket *SHARED-MOBILITY-TICKET* en lugar del *MOBILITY-TICKET* y recibe el código de error *Shared Mobility Forbidden* en lugar del *Mobility Forbidden* en caso de fallo.

Refrescando una asignación (*Allocation*)

1. Enviando una petición *Refresh Request*

Si un cliente necesita eliminar una asignación existente o refrescarla con un nuevo tiempo de expiración (*time-to-expire*) el proceso es el mismo que el explicado en la Sección 7.1 de [Mahy 2010]. Si un cliente necesita realizar movilidad de flujos multimedia entre clientes debe incluir el ticket *SHARED-MOBILITY-TICKET* obtenido por el cliente al que pertenece la asignación cuando fue creada. Este *SHARED-MOBILITY-TICKET* es válido una única vez por lo que en cada transacción de refresco el cliente debe utilizar el último ticket creado.

2. Recibiendo una petición *Refresh Request*

Cuando el servidor TURN recibe una petición *Refresh Request* con un atributo *SHARED-MOBILITY-TICKET* comprueba la validez del ticket como se explica en la Sección 5.2.2 de [Wing 2013] para los *MOBILITY-TICKET* y en caso de fallo en la validación envía el correspondiente código de error.

Si la petición *Refresh Request* contiene un valor de vida (*life-time*) deseado distinto de 0 y la *5-tuple* obtenida del paquete corresponde con una *deprecated-5-tuple* de una asignación, significa que el cliente está tratando de refrescar una *deprecated-5-tuple* y el servidor TURN debe denegar la petición.

Si todas estas comprobaciones pasan el servidor TURN entiende que el cliente quiere comenzar un proceso de movilidad entre clientes y obtiene la *5-tuple* del *SHARED-MOBILITY-TICKET* en lugar de obtenerla de la petición. A continuación busca la asignación asociada a esa tupla y añade la tupla a la lista de tuplas deprecadas de la asignación. Ahora la *5-tuple* que identifica la asignación es la tupla del nuevo cliente extraída de la petición. Por último el servidor de TURN calcula un nuevo ticket a partir de la nueva tupla y lo envía en el parámetro de STUN *SHARED-MOBILITY-TICKET* como parte de la respuesta *Refresh Success*.

A partir de este momento todo el tráfico recibido a través de la dirección IP/puerto de *relay* correspondiente a la asignación TURN será enviado al cliente dueño de la nueva tupla. Sin embargo el servidor TURN procesará datos de aplicación que provengan de todos los clientes cuyas tuplas formen parte de la lista de tuplas deprecadas en la asignación y los enviará a los participantes a través de la misma dirección/puerto de *relay*. De este modo cuando se reciben datos de aplicación el servidor de TURN no debe comprobar únicamente la *5-tuple* principal de la asignación si no también la lista de tupas deprecadas asociadas.

3.4. MOVILIDAD DE FLUJOS MULTIMEDIA

Si la petición *Refresh Request* contiene un valor de vida (*life-time*) deseado igual a 0 y la tupla extraída del paquete corresponde con una *deprecated-5-tuple* de cualquier asignación, el servidor de TURN elimina esa tupla de la lista de tuplas deprecadas.

Si la petición *Refresh Request* contiene un valor de vida (*life-time*) deseado igual a 0 y la tupla extraída del paquete identifica una asignación con una o más *deprecated-5-tuples*, el servidor de TURN elimina la asignación y todas las tuplas asociadas.

3. Recibiendo una respuesta *Refresh Response*

El proceso es el mismo que el descrito en la Sección 5.2.3 de [Wing 2013] pero en este caso el cliente guarda el ticket *SHARED-MOBILITY-TICKET*.

Asignaciones (*Allocations*)

Además de los datos de estado descritos para las asignaciones TURN en la Sección 5 de [Mahy 2010], en esta extensión describo un nuevo campo que contiene una lista de las estructuras de datos con la información de las tuplas deprecadas de la asignación. Cada una de estas estructuras de datos tiene la siguiente forma:

- *deprecated-5-tuple* (dirección IP del cliente, puerto del cliente, dirección IP del servidor, puerto del servidor, protocolo de transporte)
- *time-to-expiry*

La variable *time-to-expiry* indica el tiempo (en segundos) restante hasta que la tupla es borrada de la asignación. Cuando se asigna tiene el valor configurado en el parámetro *deprecated-5-tuple-LIFETIME* del servidor TURN. Como he explicado antes un cliente puede borrar una tupla asignando el valor 0 a esta variable mediante una petición *Refresh Request*.

La lista de permisos y canales correspondientes a cada tupla deprecada no están incluidos en esta estructura de datos ya que son los mismos que los de la tupla principal a los que están asociadas.

Con esto los datos de estado de una asignación TURN resultan como sigue:

- dirección de transporte de *relay*

- *5-tuple* (dirección IP del cliente, puerto del cliente, dirección IP del servidor, puerto del servidor, protocolo de transporte)
- información de autenticación
- *time-to-expiry*
- lista de permisos
- lista de canales enlazados a los participantes
- lista de estructuras de datos de las *deprecated-5-tuples* asociadas

Nuevo parámetro de configuración de TURN

El servidor de TURN tiene un nuevo parámetro de configuración, *deprecated-5-tuple-LIFETIME*, que indica el tiempo de vida de las *deprecated-5-tuples*. Cuando una *deprecated-5-tuple* se añade a la lista de tuplas deprecadas de una asignación TURN, dicha tupla existirá durante el tiempo indicado en este parámetro o hasta que el cliente la elimine explícitamente mediante un *life-time* deseado igual a 0.

Borrando tuplas *deprecated-5-tuples* expiradas

El servidor de TURN comprueba de manera periódica si existe alguna *deprecated-5-tuple* con un tiempo de vida mayor que el configurado en el parámetro *deprecated-5-tuple-LIFETIME*. Si existe significa que esa tupla ha expirado y el servidor de TURN la elimina de forma permanente de la lista de tuplas deprecadas de la asignación TURN correspondiente.

Nuevo atributo de STUN *SHARED-MOBILITY-TICKET*

Además de los atributos de STUN definidos en [Rosenberg 2008], el atributo *SHARED-MOBILITY-TICKET* se utiliza para realizar procesos de movilidad de flujos multimedia entre clientes de servidores de TURN. Este ticket se intercambia entre el servidor TURN y los clientes para permitir esta movilidad. Tiene las mismas propiedades de seguridad y encriptación que el *MOBILITY-TICKET* descrito en la Sección 5.3 de [Wing 2013].

3.4. MOVILIDAD DE FLUJOS MULTIMEDIA

Nuevo código de error de STUN (*Error Response Code*)

Además de los códigos de error de STUN definidos en [Rosenberg 2008], el nuevo código de error *Shared Mobility Forbidden* indica que una petición de movilidad entre clientes es válida pero no está permitida debido a restricciones dadas por las políticas locales del servidor de TURN.

3.4.4 MICE aplicado a un escenario de videoconferencia

Con la extensión de MICE descrita disponemos de las herramientas necesarias para solucionar el problema planteado al inicio, la movilidad de flujos multimedia entre MCUs en una sesión de videoconferencia. A continuación explico cómo aplicar esta solución a un escenario concreto de uso en dos posibles configuraciones. De esta forma puede entenderse mejor el funcionamiento de la especificación. La primera de ellas consiste en una configuración básica en la que una MCU utiliza una única asignación TURN (*allocation*) para todos los flujos de la sesión. En la segunda se muestra una configuración más compleja en la que una MCU utiliza múltiples asignaciones TURN.

Configuración de asignación TURN única

En la Figura 3.10 se puede observar un escenario simple de videoconferencia en el que el *Peer A* se encuentra publicando un flujo de datos multimedia en la *MCU 1* y el *Peer B* está suscrito recibiendo dicho flujo. El comportamiento es el mismo que el explicado en la Sección 3.4.2. El servidor de TURN se utiliza como *relay* para habilitar la movilidad de flujos mediante el protocolo MICE. Para forzar la comunicación a través del servidor TURN ambos participantes deben ignorar los candidatos ICE que no corresponden a la dirección de *relay* del TURN. No se desea modificar la implementación de los clientes utilizados por los participantes por lo que para conseguir esto se eliminarán dichos candidatos durante el proceso de negociación entre la MCU y los participantes al establecer la conexión.

En un momento determinado durante el transcurso de la sesión deseamos mover la gestión del flujo multimedia desde *MCU 1* a *MCU 2* de tal forma que el *Peer A* envíe su flujo a *MCU 2* y el *Peer B* pueda suscribirse a ella para recibirlos. Esto puede ocurrir porque necesitemos apagar la máquina en la que actualmente tenemos corriendo *MCU 1* para escalar el sistema hacia abajo ahorrando recursos. En tal caso deberemos proceder de esta manera con todos los flujos

Figura 3.10 : Configuración de una asignación TURN simple.

multimedia de todas las sesiones que esté gestionando *MCU 1*. En la arquitectura ilustrada se incluye un componente central *Controller* que se comunica con ambas MCUs para controlar la sesión y coordinar el proceso de movilidad.

En la Figura 3.11 se detalla el flujo de mensajes intercambiados entre estos componentes durante el proceso, desde el momento en que comienza la comunicación entre los participantes hasta el momento en que la movilidad a la *MCU 2* se completa. Como ya he comentado suponemos que se trata de una configuración básica en la que cada MCU utiliza una única asignación TURN para enviar y recibir datos de los participantes.

El proceso consta de los siguientes pasos:

1. Establecimiento de la comunicación con soporte a movilidad.

Para comenzar la comunicación con los participantes de la sesión (*Peers*), la *MCU 1* pide una nueva asignación (*Allocation*) al servidor de TURN. Dado que quiere soportar movilidad de flujos multimedia entre clientes incluye en la petición *Allocate Request* el parámetro *SHARED-MOBILITY-TICKET* con longitud 0.

3.4. MOVILIDAD DE FLUJOS MULTIMEDIA

Figura 3.11 : Flujo de llamadas entre clientes en movilidad de flujos multimedia.

Si la movilidad entre clientes está permitida y el servidor TURN procesa correctamente la petición, éste crea una asignación (*Allocation 1*) con la 5-tuple de *MCU 1* (*5-tuple 1*) y le asocia una dirección IP/puerto de *relay* (*RelayedAddress 1*). El servidor genera también un *SHARED-MOBILITY-TICKET* para dicha asignación (*Ticket 1*) y envía una respuesta *Allocate Success* a *MCU 1* incluyendo el ticket. *MCU 1* guarda *Ticket 1* asociado al flujo de datos publicado.

En este momento *Peer A* comienza a publicar su flujo de datos multimedia (*Stream A*) enviando datos a la dirección/puerto de *relay* de *Allocation 1*. El servidor de TURN obtiene la 5-tuple asociada a esta asignación (*5-tuple 1*) que corresponde a *MCU 1*, por lo que envía los datos a *MCU 1*. Entonces *MCU 1* procesa los datos publicados y los reenvía a los suscriptores, en este caso únicamente *Peer B*. Para ello envía los datos de aplicación al

servidor TURN encapsulados en un mensaje de STUN. La mejor manera de hacer esto es utilizando un *TURN Channel Binding* como se describe en la Sección 11 de [Mahy 2010]. El servidor de TURN procesa los datos que provienen de *MCU 1 (5-tuple 1)*, busca la asignación correspondiente (*Allocation 1*) y envía los datos al *Peer B* a través de la dirección/puerto de *relay* asociada a dicha asignación (*RelayedAddress 1*).

2. Inicio del proceso de movilidad.

En un momento determinado de la sesión y debido a requisitos del sistema, el controlador (*Controller*) decide mover la gestión de *Stream A* de *MCU 1* a *MCU 2*. Para ello envía un mensaje a *MCU 1* pidiendo el *SHARED-MOBILITY-TICKET* de su flujo de datos. Al recibir el mensaje *MCU 1* busca el ticket correspondiente a *Stream A* y se lo devuelve al controlador. Se trata del ticket asociado a *Allocation 1*, *Ticket 1*.

Una vez tiene el ticket, el controlador envía un mensaje a *MCU 2* indicando que debe comenzar un proceso de movilidad de la asignación correspondiente a dicho ticket (*Ticket 1*) para lo cual lo incluye en el mensaje. Como se ha explicado antes, el protocolo utilizado para realizar este intercambio de mensajes entre el controlador y las MCUs es irrelevante en la descripción de este mecanismo.

Una vez recibido el ticket y la orden de comenzar el proceso, *MCU 2* envía al servidor de TURN una petición *Refresh Request* incluyéndolo en el atributo *SHARED-MOBILITY-TICKET*. El servidor TURN comprueba si la petición *Refresh Request* es válida siguiendo los pasos descritos en la sección anterior. En caso de serlo extrae la *5-tuple* del ticket (*5-tuple 1*) y la busca en su base de datos determinando así que corresponde con la asignación *Allocation 1*. A continuación añade *5-tuple 1* a la lista de tuplas deprecadas la asignación y configura como tupla principal la *5-tuple* que extrae de la petición. Esta nueva tupla (*5-tuple 2*) corresponde a *MCU 2*. Por último el servidor de TURN genera un nuevo *SHARED-MOBILITY-TICKET* con la nueva tupla (*Ticket 2*) y lo incluye en la respuesta *Refresh Success* que envía a *MCU 2*.

Ahora la asignación *Allocation 1* tiene la tupla *5-tuple 2* como tupla principal y la tupla *5-tuple 1* como tupla deprecada. Esto quiere decir que los paquetes recibidos en *RelayedAddress 1* son enviados a la dirección IP/puerto de la tupla *5-tuple 2* (correspondiente a *MCU 2*). Pero por otra parte tanto los datos recibidos desde la dirección

3.4. MOVILIDAD DE FLUJOS MULTIMEDIA

de la tupla *5-tuple 1* (correspondientes a *MCU 1*) como desde la dirección de la tupla *5-tuple 2* (correspondientes a *MCU 2*) son enviados a los participantes a través de la dirección *RelayedAddress 1*.

3. Fin del proceso de movilidad.

Al recibir la respuesta *Refresh Success* del servidor TURN, *MCU 2* guarda el ticket *Ticket 2* para un posible futuro proceso de movilidad y envía un mensaje al controlador indicando que el proceso actual ha concluido.

El controlador espera entonces un tiempo que permita asegurar que todos los paquetes enviados a *MCU 1* antes del proceso han llegado de nuevo al TURN. Pasado este tiempo envía un mensaje a *MCU 1* pidiendo que elimine su tupla del servidor de TURN. Para ello *MCU 1* envía una petición *Allocate Request* con el parámetro *LIFETIME* asignado a 0.

Al recibir esta petición el servidor de TURN extrae la tupla (*5-tuple 1*) y la busca en su base de datos. En este caso la tupla corresponde a una tupla deprecada (y no a una principal) por lo que la elimina de la asignación a la que pertenece, en este caso *Allocation 1*. En caso de que el tiempo de vida de esta tupla deprecada hubiera expirado antes de recibir esta petición (debido a haber sobrepasado el tiempo indicado en *deprecated-5-tuple-LIFETIME*), la tupla ya habría sido eliminada por el propio TURN. Una vez eliminada el servidor TURN devuelve la respuesta *Refresh Success* a *MCU 1*.

A partir de este instante la asignación *Allocation 1* no tiene ninguna tupla deprecada en su lista por lo que su funcionamiento consiste en enviar los datos provenientes del *Peer A* a través de la dirección *RelayedAddress 1* a la dirección IP/puerto correspondientes a la tupla *5-tuple 2* (*MCU 2*) y en recibir datos de la misma dirección enviándolos al *Peer B* de nuevo a través de *RelayedAddress 1*.

Una vez finalizado este proceso el *Peer A* se encuentra publicando datos a *MCU 2* y el *Peer B* suscrito a ella para recibirlos, por lo tanto el proceso de movilidad está completado. En este ejemplo una misma asignación TURN gestiona varios flujos multimedia enviando y recibiendo paquetes a través de la misma dirección IP/puerto de *relay*. Pero esto no siempre es así, en ocasiones y debido a la configuración del servidor de TURN, necesitamos mover flujos de datos asociados a más de una asignación. Y esos casos hay que tener en cuenta algunas consideraciones.

Configuración de múltiples asignaciones TURN

En la Figura 3.12 puede observarse un escenario en el que se utiliza más de una asignación TURN para establecer la comunicación entre los participantes. En este caso el participante *Peer A* publica un flujo multimedia en *MCU 1* y los participantes *Peer B* y *Peer C* se suscriben a él. Pero cada uno de ellos utiliza una dirección de *relay* diferente (líneas de puntos). Esto implica que si queremos mover la gestión de este flujo a *MCU 2* debemos mover las tres asignaciones TURN.

Figura 3.12 : Configuración de múltiples asignaciones TURN.

El orden en el que se realiza la movilidad de estas asignaciones es crucial para asegurar la calidad en la comunicación. Llamando *Allocation A*, *Allocation B* y *Allocation C* a las asignaciones de los participantes *Peer A*, *Peer B* y *Peer C* respectivamente, en caso de mover *Allocation A* antes de mover las otras dos el resultado será la pérdida de un conjunto de paquetes de datos. Los paquetes recibidos a través de la dirección de *relay* de *Allocation A* desde el participante *Peer A* se enviarían a *MCU 2* y al volver al servidor de TURN las asignaciones *Allocation B* y *Allocation C* estarían esperando recibir paquetes de *MCU 1*. Por lo tanto se perderían.

3.4. MOVILIDAD DE FLUJOS MULTIMEDIA

La conclusión es que cuando van a moverse varias asignaciones TURN asociadas al mismo flujo de datos deben moverse en primer lugar las correspondientes a los suscriptores. Dicho esto y suponiendo que

- *MCU 1* ha asignado *Allocation A*, *Allocation B* y *Allocation C* con direcciones de *relay* para comunicarse con los participantes *Peer A*, *Peer B* y *Peer C* respectivamente,
- dichas asignaciones están identificadas por las tuplas *5-tuple 1A*, *5-tuple 1B* and *5-tuple 1C* respectivamente,
- las tuplas tienen asociada la dirección IP de *MCU 1* con un puerto diferente cada una,
- *MCU 1* tiene el ticket *SHARED-MOBILITY-TICKET* de cada una de las tres asignaciones, *Allocation A*, *Allocation B* y *Allocation C*,
- siendo dichos tickets *Ticket A*, *Ticket B* y *Ticket C* respectivamente,

deben realizarse los siguientes pasos para conseguir el proceso de movilidad en esta configuración (los pasos están simplificados porque se supone que los detalles de cada uno de los mensajes se conocen de los ejemplos anteriores):

1. Movilidad de las asignaciones de los suscriptores

El controlador pide a *MCU 1* los tickets *SHARED-MOBILITY-TICKET* de ambos suscriptores, obteniendo *Ticket B* y *Ticket C* y enviándoselos a *MCU 2*. Utilizando estos tickets *MCU 2* envía dos peticiones *Refresh Request* al servidor de TURN, una por cada asignación a mover. Al recibirlas el servidor TURN configura en *Allocation B* una tupla de *MCU 2* (*5-tuple 2B*) como tupla principal y la tupla *5-tuple 1B* como tupla deprecada. Hace lo mismo para *Allocation C* utilizando *5-tuple 2C* como principal y añadiendo *5-tuple 1C* a la lista de deprecadas. Cuando *MCU 2* recibe la respuesta de éxito de ambas peticiones el servidor TURN está preparado para recibir datos de ambas MCUs.

2. Movilidad de la asignación que publica

MCU 2 notifica al controlador que puede proceder con la movilidad de la asignación restante (del flujo de datos publicado, *Allocation A*) para lo que éste obtiene el ticket (*Ticket A*) de *MCU 1* y lo envía a *MCU 2*. Con este ticket *MCU 2* pide al servidor TURN el refresco de la

asignación *Allocation A*. Igual que anteriormente el servidor TURN configura como tupla principal de dicha asignación una nueva tupla de *MCU 2 (5-tuple 2A)* por lo que los datos recibidos de *Peer A* son ahora enviados a *MCU 2* y recibidas por las ya configuradas *5-tuple 2B* y *5-tuple 2C*.

3. Final del proceso

Cuando *MCU 2* recibe la confirmación de éxito de la última asignación comunica a *MCU 1* a través del controlador el final del proceso. Entonces *MCU 1* pide al servidor de TURN que elimine las tuplas deprecadas de las tres asignaciones.

Utilizando esta metodología puede conseguirse la movilidad de flujos de datos multimedia en cualquier configuración de TURN y en escenarios de videoconferencia complejos.

3.5 Conclusiones

Los modelos y arquitecturas propuestos en este capítulo pretenden solucionar el problema de cómo escalar sistemas de videoconferencia de manera dinámica en infraestructuras Cloud.

Para ello, en primer lugar he realizado un análisis de las oportunidades que plantea el despliegue de estos sistemas en este tipo de infraestructuras. Tras constatar que efectivamente la flexibilidad que ofrecen las infraestructuras Cloud pueden resultar muy beneficiosas para el uso eficiente de los recursos y la escalabilidad dinámica de dichos sistemas, he analizado los retos que plantea llevar a cabo estos despliegues. En esta tesis resuelvo solamente algunos de ellos, los que considero más críticos y que deben resolverse en primera instancia, facilitando avances posteriores en el resto de ellos.

El primer y fundamental reto es el de diseñar una arquitectura que permita realizar la escalabilidad de MCUs (componente nuclear de los sistemas de videoconferencia multiusuario) atendiendo a sus características concretas. Para el diseño de esta arquitectura he tomado como punto de partida el componente Nuve, que torna el paradigma de la videoconferencia en un modelo en el que se proveen salas como servicio.

La arquitectura propuesta permite desplegar escenarios con múltiples MCUs entre las que se balancea la carga de manera inteligente teniendo en cuenta el consumo de recursos de cada una de ellas y ciertas características previamente configuradas. Esto es posible haciendo una caracterización de las capacidades de estas MCUs y extrapolando los resultados a datos de

3.5. CONCLUSIONES

consumo reales. Para ilustrar esta caracterización he mostrado también un ejemplo en un caso de uso real y sobre una implementación de la arquitectura.

Esta arquitectura permite también la gestión de los recursos disponibles en el sistema siendo capaz de arrancar o parar MCUs cuando es necesario. Estas MCUs pueden arrancarse en diferentes proveedores IaaS gracias a un adaptador que abstraer los APIs de cada uno de ellos. Sin embargo a la hora de parar MCUs surge el problema de la movilidad de conexiones entre ellas, necesaria cuando éstas se encuentran alojadas en una MCU que queremos apagar.

En el último bloque de esta sección he propuesto una extensión de la especificación de movilidad de flujos MICE solventando sus carencias en casos de tiempo real en los que no queremos que se produzcan pérdidas de paquetes. Así, propongo una serie de parámetros y mecanismos adicionales que permiten alcanzar ese objetivo.

La extensión de MICE que propongo no está implementada y validada porque, como veremos en el siguiente capítulo, en un escenario de MCUs distribuidas la movilidad de flujos deja de ser crítica. Y precisamente es a un escenario de este estilo a donde me llevaron los pasos de mi investigación considerando que, gracias al nuevo modelo, la implementación de la extensión de MICE no era necesaria. Sin embargo, la especificación MICE original está validada en otros ámbitos de movilidad y mi extensión no añade ningún componente que haga pensar que puede no cumplir con los requisitos del escenario. En cualquier caso, dejaré para trabajos futuros la validación de esta extensión.

Capítulo 4

Asignación de recursos en sistemas de videoconferencia distribuidos

En el capítulo anterior he abordado el problema de cómo escalar de forma dinámica MCUs en función de las variaciones en la demanda de usuarios. El modelo propuesto permite monitorizar el estado de cada MCU desplegada y actuar en consecuencia añadiendo o eliminando MCUs al sistema según las necesidades de cada momento. Sin embargo, desplegar MCUs monolíticas impone una restricción considerable en términos de escalabilidad: una misma sala de videoconferencia ha de estar siempre gestionada por la misma MCU.

De esta forma, en caso de llegar al límite de capacidad de procesamiento de una MCU y querer seguir añadiendo participantes a una sala deberíamos mover la gestión de dicha sala a otra MCU con mayor cantidad de recursos. La movilidad de conexiones es una solución que, si bien hasta ahora no estaba resuelta de forma completa, gracias a la extensión de MICE que he propuesto resulta realizable. No obstante, implica la introducción de un nuevo componente en la arquitectura (servidor TURN) que añade complejidad, costes y un nuevo punto de posibles fallos. Otra posible solución es la de reservar recursos en cada MCU anticipándose a la posible unión de nuevos participantes en el futuro. Obviamente esto implica un gasto de recursos innecesario por lo que de nuevo no es la solución óptima.

La respuesta final consiste en distribuir una MCU de tal manera que olvidamos el concepto de MCU monolítica y permitimos gestionar una misma sala de videoconferencia repartida entre varios servidores distribuidos. De esta forma la unidad de escalabilidad ya no es la sala de videoconferencia sino una unidad granular de gestión de flujos multimedia llamada dOTM (*distributed One To Many*) y que explicaré más adelante.

En un escenario de MCUs distribuidas las posibilidades de escalabilidad y despliegue aumentan considerablemente y el modelo de reporte y distribución de carga propuesto anteriormente tiene que ser ampliado también. Frente a una mayor flexibilidad y granularidad en los despliegues, la manera en que se programan y asignan recursos es mucho más compleja pero a la vez se abre una puerta hacia una mayor eficiencia acompañada de un ahorro de costes mucho más sustancial.

4.1 Objetivos

Teniendo en cuenta el nuevo escenario de MCUs distribuidas, los objetivos a conseguir en este capítulo son los siguientes:

- *Identificar las características específicas de un sistema de videoconferencia distribuido y sus necesidades a la hora de tomar decisiones de asignación de recursos.* Se trata de realizar un análisis de las consecuencias que tiene el despliegue de una MCU distribuida en términos de escalabilidad y programación de recursos. Este análisis debe tener en cuenta las soluciones existentes en otros ámbitos tecnológicos para realizar una comparativa con el escenario objetivo y extraer posibles referencias de diseño.
- *Diseñar e implementar un modelo que habilite la asignación inteligente de recursos en un sistema de videoconferencia distribuido.* La forma de asignar los recursos debe poder adaptarse a las necesidades de cada escenario concreto y debe estar basada en el estado global del sistema, tanto en sus características definidas como en su estado en cada momento. Para poder tomar las decisiones de asignación, un reto fundamental es el de caracterizar la nueva unidad de escalabilidad surgida al distribuir la MCU.
- *Proponer y comparar estrategias de asignación de recursos de videoconferencia en entornos distribuidos.* Estas estrategias deben favorecer un uso eficiente de la infraestructura y tener en cuenta las necesidades específicas que dicte cada caso de uso. Además es necesario diseñar herramientas que permitan evaluar estas estrategias para determinar su idoneidad en cada escenario.

4.2 Arquitectura de MCU distribuida

En objetivo de este capítulo no es el de diseñar un modelo de MCU distribuida si no el de afrontar los retos que plantea la programación de recursos en este tipo de modelos. Para ello voy a tomar como referencia el modelo de MCU distribuida basado en los llamados *distributed One To Many* o DOTMs. Se trata de una propuesta de MCU distribuida en cuyo diseño he participado de forma activa pero que forma parte de las contribuciones de otra tesis doctoral desarrollada en el mismo grupo de investigación que ésta y cuyo detalle podemos encontrar en [Rodríguez 2016a] y en [Rodríguez 2014, Rodríguez 2016b].

4.2. ARQUITECTURA DE MCU DISTRIBUIDA

4.2.1 dOTM: distributed One To Many

El modelo dOTM considera una MCU realizando la tarea de replicación de paquetes, la más común entre todas las posibles y que consiste en la recepción de flujos de datos de los participantes que publican y su replicación para enviarlos a los participantes que se suscriben. La idea fundamental de este modelo es dividir esta tarea en procesos independientes de tal forma que en una sesión de videoconferencia en la que todos los participantes publican sus flujos y se suscriben a los del resto, existe un proceso realizando la tarea por cada participante conectado a la sala.

Figura 4.1 : Separación de una MCU en procesos.

En la Figura 4.1 podemos ver ilustrado un ejemplo en que una sala con tres participantes conectados queda dividida en tres componentes que reciben paquetes de un participante y los reenvían a los otros dos. Si arrancamos cada uno de estos componentes como un proceso independiente en servidores diferentes tendremos una MCU distribuida. Pero al hablar de replicación de paquetes de los flujos multimedia estamos únicamente hablando de la parte de medios de una MCU y es importante recordar que existen más componentes involucrados: la señalización y el control. Lo que propone el modelo dOTMs es dividir la MCU en tres capas correspondientes a cada uno de los tres componentes, aunque a efectos prácticos la señalización se realiza a través de la capa de control.

El resultado de esta arquitectura para la capa de control puede observarse en la Figura 4.2, con los siguientes componentes:

Figura 4.2 : Arquitectura de control de MCU distribuida

- **OTM:** se trata del componente de replicación de paquetes y nueva unidad atómica de despliegue en la MCU distribuida. Como adelantaba es un proceso independiente a cargo de recibir un flujo multimedia de un único participante y reenviarlo a los demás. Para desempeñar su función, debe ser capaz de intercambiar mensajes de señalización e implementa una pila completa de protocolos para transmisión de medios compatible con la utilizada por los clientes.
- **Agent:** es el componente encargado de arrancar, parar y gestionar procesos OTM bajo demanda. Un Agent debe ser instanciado en cada servidor en que queramos arrancar estos componentes, es decir, en cada servidor que forme parte del despliegue de MCU distribuida. Los Agents son automáticamente descubiertos en el Controller gracias al bus de mensajes.
- **Controller:** es la implementación de la capa de control del sistema, encargada de gestionar las salas de videoconferencia. Se trata del punto de entrada y el interfaz de control con

4.2. ARQUITECTURA DE MCU DISTRIBUIDA

los participantes. Así, dirige a los participantes a la sala adecuada y mantiene en memoria información sobre las salas disponibles y los usuarios conectados. El control de cada sala queda delegado a los *RoomControllers*, un módulo diferenciado pero que forma parte del propio Controller. Para mantener una única conexión para control y señalización, Controller se encargará de reenviar todos los mensajes de señalización entre los participantes y los OTMs correspondientes.

- **Room Controller:** Como parte de Controller, implementa la gestión del control para una única sala. Room Controller se comunicará con los Agents para pedir nuevos OTMs según sean necesarios por la llegada de nuevos participantes que quieran publicar en la sala. Además, guardará la correspondencia para que los clientes que quieran suscribirse puedan encontrar el OTM adecuado.

Figura 4.3 : Arquitectura de medios de MCU distribuida

Por otra parte, la Figura 4.3 muestra el camino de medios de la MCU distribuida. De acuerdo con lo explicado anteriormente, Controller, Room Controllers y Agents no están presentes en esta figura ya que no intervienen en este nivel. Tras una negociación de la conexión exitosa, los OTMs y los participantes de la sesión se comunican directamente intercambiando los paquetes de datos multimedia. En la figura los participantes A, B y C se encuentran en la misma sala de

videoconferencia. Los participantes A y B están publicando sus flujos multimedia utilizando dos OTMs de los servidores A y B respectivamente. Ambos están suscritos para recibir el flujo del otro y un tercer participante C lo está a ambos dos.

El modelo de escalabilidad propuesto en el capítulo anterior sigue siendo válido para esta arquitectura distribuida. Pero como puede observarse ahora tenemos una unidad más de escalabilidad, los OTMs. Por lo tanto la gestión realizada desde Cloud Handler corresponde a instancias del módulo Controller (con la unidad de escalabilidad de sala) y será este módulo el que a su vez se encargue de gestionar Agents (con la unidad de escalabilidad de OTM).

Sin embargo, en el modelo dOTMs no está contemplado cómo deben programarse los recursos entre los Agents disponibles. Cuando hablábamos de distribución de carga entre MCUs monolíticas, las posibilidades eran más limitadas y una caracterización basada en el consumo por número de salas resultaba suficiente ya que una vez asignada una sala, todas las conexiones correspondientes a una sala irían a esa misma MCU. Sin embargo, al ser capaces de repartir conexiones de una misma sala entre varios servidores tenemos que asignar conexiones (en la práctica OTMs) de manera independiente, con lo que las posibilidades se multiplican y el problema es mucho más complejo.

4.2.2 Características del escenario

En los servicios Web tradicionales basados en HTTP, cuando queremos repartir las conexiones entre varios servidores, normalmente es suficiente con realizar una distribución homogénea entre ellos utilizando balanceadores de carga estándar. Estos balanceadores están adaptados a los requisitos especiales de servicios avanzados, con mecanismos de gestión de sesiones, etc. Pero en el caso de los sistemas de videoconferencia distribuidos no existe una solución estándar.

En estos sistemas el uso de recursos en cada uno de los nodos distribuidos puede depender de diferentes parámetros aparte del número de usuarios o conexiones. Esto puede entenderse con un ejemplo muy sencillo. Imaginemos un escenario con una MCU distribuida entre dos servidores gestionando seis usuarios cada una. En la primera MCU los seis usuarios están conectados a la misma sala de videoconferencia y en la segunda cada usuario está conectado a una sala diferente y sin ningún otro usuario más. Obviamente la primera MCU está consumiendo más recursos ya que está recibiendo paquetes de cada uno de los usuarios y procesándolos para enviarlos al resto. Mientras que la segunda MCU únicamente está recibiendo paquetes de cada usuario sin ningún

4.2. ARQUITECTURA DE MCU DISTRIBUIDA

tipo de procesamiento que hacer.

Por lo tanto, para resolver el problema de la programación de recursos en este tipo de sistemas distribuidos, debemos entender en primer lugar qué tipo de criterios debemos tener en cuenta. Para ello es imprescindible realizar una caracterización del escenario teniendo en cuenta los requisitos y peculiaridades de la arquitectura. A continuación explico los aspectos más relevantes de cara a comenzar a proponer un modelo de programación de recursos para la arquitectura dOTM.

Requisitos de tiempo real no crítico

Como se explica en [Sha 2004], los sistemas de tiempo real "relajados" o no críticos (*soft real-time systems*) tienen diferentes características que los escenarios de programación de recursos tradicionales. El criterio *hard-deadline* usado frecuentemente en otros sistemas no es válido aquí debido a las grandes diferencias entre los valores de uso peor y medio.

El caso de la videoconferencia donde el consumo de ancho de banda puede experimentar picos de uso durante una sesión es un buen ejemplo para entender esto. Sin embargo, suele ocurrir que no es necesario cumplir todos los límites impuestos en la ejecución y que la pérdida de tareas es admisible en cierta medida. En este caso los requisitos impuestos para una sesión de vídeo dependen de la calidad de servicio (QoS) deseada. De esta forma la latencia o incluso la pérdida de paquetes puede ser aceptada siempre que la calidad recibida por los usuarios sea aceptable.

Asignación inmediata

En Cloud Computing hay tres maneras de proveer recursos por parte de los proveedores [Li 2010]: mediante reserva previa, en modo *best-efford* o de manera inmediata. En el caso de videoconferencia podríamos pensar en escenarios en los que los recursos se reservan con anterioridad a las sesiones. Por ejemplo cuando se programa una reunión de trabajo o una conferencia. Sin embargo el escenario que presenta un reto para esta investigación es aquel en el que necesitamos atender las peticiones en tiempo real y proveer los recursos de forma inmediata. De esta forma además no tendremos nunca recursos reservados y no utilizados haciendo un uso más eficiente de la infraestructura.

No petición - respuesta

El paradigma más común que encontramos en los servicios Web tradicionales es el de petición - respuesta. En este modelo cada petición enviada por un cliente es independiente de la anterior. En ocasiones el sistema guarda la sesión de un usuario para mantener cierto estado en las conexiones y guardar un histórico de la actividad de cada usuario. Pero se trata únicamente de información de contexto añadida en las peticiones subsecuentes. En este modelo el tiempo de procesamiento y la carga del sistema puede ser estimada a partir del número de usuarios y peticiones. Y por lo tanto en la mayoría de los casos es suficiente con un reparto homogéneo de la carga entre los servidores disponibles.

Sin embargo y como anticipaba arriba, en los sistemas de videoconferencia el uso de los recursos puede depender de otros factores aparte del número de usuarios. Normalmente la computación necesaria depende del tamaño de las salas, del número de usuarios conectados a cada una de estas salas o de la calidad de los datos intercambiados entre los clientes. Por ejemplo, el uso de recursos de diez participantes no es el mismo si se encuentran agrupados en salas de dos usuarios o si están todos conectados a la misma sala. Tampoco si están enviando un vídeo de alta calidad o solamente un flujo de audio.

En otras palabras y volviendo al modelo dOTMs, el uso de recursos por un OTM dependerá del número de suscriptores y de la cantidad de ancho de banda utilizado por cada uno entre otros factores. Además, la duración de las sesiones en un sistema de videoconferencia es difícil de predecir y anticiparse a la carga es complicado incluso aunque supiéramos la demanda de usuarios prevista.

Por último, cuando programamos recursos en sistemas distribuidos, es bastante frecuente establecer un umbral de uso máximo para evitar la saturación de los servidores. Pero a pesar de esto y de manera circunstancial, pueden producirse picos de uso temporales que hagan que se sobrepase este límite. En el caso de videoconferencia las conexiones son permanentes por lo que estos posibles picos pueden extenderse en el tiempo. Si no estimamos de manera precisa la cantidad de recursos necesarios podemos saturar los servidores durante largos periodos de tiempo.

4.3. MECANISMO PARA HABILITAR LA PROGRAMACIÓN DE RECURSOS

Múltiples consumidores

Una de las ventajas de la distribución de los servicios es hacerla siguiendo criterios geográficos. En otro tipo de sistemas distribuidos hay un único tipo de consumidor por lo que el programador de recursos toma la decisión de asignación basándose en la localización de este consumidor. Sin embargo, en el caso de los sistemas de videoconferencia los consumidores pueden ser usuarios que publican o que se suscriben a flujos multimedia. Esto hace que la distribución geográfica no pueda gestionarse de manera trivial. Resulta complejo y deben tenerse en cuenta nuevas consideraciones ya que en ocasiones puede ser beneficioso localizar al proveedor cerca de un tipo de consumidor o de otro.

Control centralizado

Algunas de las soluciones de programación de recursos que hemos analizado en la Sección 2.3 están pensadas para entornos puramente distribuidos. En nuestro caso tenemos un componente central que facilita la señalización y el control entre el resto. También es una característica que hay que tener en cuenta a la hora de diseñar los mecanismos de programación y asignación de recursos.

4.3 Mecanismo para habilitar la programación de recursos

Una vez analizadas las características de los sistemas de videoconferencia distribuidos y en concreto del modelo dOTMs, el primer paso para afrontar el problema de la programación eficiente de recursos es el de habilitar un mecanismo que nos permita realizar la asignación basándonos en políticas adaptables a las necesidades de cada escenario. A continuación voy a realizar una descripción del modelo necesario para habilitar este mecanismo y después extenderé la arquitectura dOTM incluyendo los módulos y configuraciones necesarios para cumplir con dicho modelo.

4.3.1 Descripción del modelo

Como introducía anteriormente, los servidores MCU distribuidos se diferencian de los servicios Web comunes en varios aspectos. La videoconferencia no es un servicio de petición-respuesta y la cantidad de recursos disponibles por usuario dependen del tamaño de las sesiones. De acuerdo

ASIGNACIÓN DE RECURSOS EN SISTEMAS DE VIDEOCONFERENCIA DISTRIBUIDOS

con esto, no podemos distribuir los usuarios de forma homogénea entre los servidores. Cuando conectamos un nuevo usuario al sistema necesitamos saber cierta información acerca del estado actual de cada uno de los servidores disponibles para decidir cuál conviene utilizar. Una vez que la decisión está tomada, necesitamos poder asignar la conexión a la MCU seleccionada. Por lo tanto el modelo que propongo debe cumplir tres aspectos principales:

Capa de decisión

En un escenario de MCU distribuida, es necesario un componente central encargado de decidir en cuál de los nodos distribuidos asignar cada conexión. Este componente debe ser configurable con algoritmos de decisión personalizables que tomen como entrada la lista de MCUs disponibles y la información de cada una de ellas y devuelvan la MCU seleccionada. Esta información puede ser información de contexto configurada en la MCU o información sobre el estado de la MCU recibida en tiempo real.

Una vez que la decisión está tomada, este componente debe ser capaz de comunicarse de manera directa con la MCU seleccionada para asignar los recursos en ella. Por último, los algoritmos deben poder modificarse en tiempo real teniendo en cuenta la retroalimentación recibida desde las MCUs.

Registro de MCU

Para poder comunicarse con una MCU en concreto, la capa de decisión debe tener una lista con todas las MCUs disponibles en el sistema. Para ello, una MCU tiene que ser registrada en el componente central al añadirse al conjunto de MCUs. Durante el registro, la MCU puede especificar una serie de características sobre sí misma que podrán ser utilizadas por los algoritmos a la hora de tomar las decisiones de asignación.

Reporte de MCU

La información de contexto configurada durante el registro de las MCUs no es suficiente para poder tomar decisiones de asignación. En la capa de decisión es necesaria también información en tiempo real sobre el estado de las MCUs. De esta forma los algoritmos podrán basarse también en parámetros como el uso de memoria o CPU de cada una de ellas. Por todo esto, una vez

4.3. MECANISMO PARA HABILITAR LA PROGRAMACIÓN DE RECURSOS

establecida la comunicación desde la capa de decisión a cada MCU, es necesario un canal de retorno que permita este reporte.

4.3.2 Diseño de la solución

A continuación voy a extender el modelo dOTM para soportar estos tres requisitos de diseño. Analizando las características de este modelo en relación a los requisitos podemos observar lo siguiente:

Selección de Agent

Cuando se necesita un nuevo OTM en una sesión, el módulo Controller selecciona un Agent en modo *round-robin* pidiendo la petición de creación de OTM a uno de los Agents disponibles cada vez. Según los requisitos enunciados es necesario soportar la creación de OTMs en Agents concretos basándonos en los algoritmos de decisión. Para habilitar esta funcionalidad hay que introducir una manera de permitir al Controller comunicarse de manera directa con cada Agent.

Registro de Agent

En el modelo actual de dOTMs, el módulo Controller no tiene conocimiento de los Agents existentes en el sistema. Para comunicarse con ellos simplemente envía un mensaje al bus y es éste el que redirecciona el mensaje a uno de los Agents suscritos. Ahora es necesario añadir un mecanismo que provea al Controller de una lista de los Agents disponibles en cada momento. En otras palabras, es necesario que cada Agent se registre en el Controller cuando se añada al conjunto de Agents disponibles.

Reporte de Agent

Para tomar la decisión de qué Agent seleccionar para crear un nuevo OTM, el módulo Controller necesita información sobre los Agents disponibles. Es especialmente interesante tener información acerca del estado de cada Agent en tiempo real. En la configuración actual, una vez que un OTM es creado el controlador le envía los mensajes relacionados con la sesión directamente a él por lo que nunca más se comunica con el Agent. Para soportar el reporte en tiempo real por parte

de los Agents es necesario habilitar un canal de comunicación persistente entre cada agente y el controlador.

Figura 4.4 : Configuración de extensiones de registro y reporte

La Figura 4.4 ilustra las extensiones que propongo añadir al modelo dOTM. En primer lugar y en el bus de mensajes, introduzco una nueva cola para cada uno de los Agents disponibles (*Agent id*). Estas colas están configuradas en modo *unicast* directo y las utiliza el controlador para comunicarse directamente con un agente. Cuando un nuevo Agent se añade al sistema crea una nueva cola en el bus para poder recibir mensajes. Por otra parte introduzco el módulo Agent Controller, encargado de mantener el registro de agentes y comunicarse con ellos para obtener los reportes. Además, cuando un RoomController necesite un nuevo OTM, ahora se lo pedirá a Agent Controller y será éste, siguiendo la política de decisión establecida, el que se encargue de su creación en el agente elegido.

Todos los agentes siguen estando suscritos a la cola común (*Agent*) pero ahora esa cola está configurada para soportar mensajes de tipo *broadcast*. Agent Controller la utilizará para enviar un mensaje periódico a todos los agentes pidiéndoles su estado. En la respuesta a estos mensajes cada agente incluye información de tres tipos:

4.3. MECANISMO PARA HABILITAR LA PROGRAMACIÓN DE RECURSOS

- **Información de contacto:** se trata de la información necesaria para contactar con el agente. Básicamente incluye el identificador de la cola de mensajes creada por el agente en el bus.
- **Información fija:** se trata de información constante configurada por el agente en tiempo de creación (durante el registro).
- **Información de tiempo real:** es la información sobre el estado del agente en tiempo real.

Recibiendo esta información de manera periódica, Agent Controller dispone en todo momento de una lista actualizada con los agentes disponibles en el sistema. Además, cuando se requiere un nuevo OTM para conectar un participante, puede basarse en la información de cada agente (la fija y la de tiempo real) para decidir cuál de ellos utilizar para su creación. Una vez tomada la decisión, envía el mensaje de creación utilizando la cola correspondiente.

La cola común puede seguir utilizándose como antiguamente para crear OTMs en modo *round-robin* entre los Agents disponibles. Puede ser de utilidad cuando no se quiere especificar ningún tipo de criterio de asignación. En este caso los OTMs se distribuirán de manera homogénea entre los servidores disponibles.

Por último, el mensaje periódico enviado por el controlador a los agentes sirve también como mensaje de *heartbeat* para asegurar que un agente determinado sigue estando disponible. El controlador dispondrá de una cuenta de mensajes de *heartbeat* no contestados para determinar cuando un agente no puede seguir considerándose para la asignación de nuevas conexiones.

En la Tabla 4.1, puede observarse la especificación detallada de los mensajes necesarios para habilitar el mecanismo propuesto. Se trata de una extensión de los mensajes originales del modelo que pueden consultarse en [Rodríguez 2016b]. Los mensajes *createOTM* and *deleteOTM* son los mismos que en dicha especificación original pero en esta extensión añadido un nuevo mensaje *createOTM* para crear un OTM en un Agent concreto (enviado mediante la cola *unicast* correspondiente) y el mensaje *getAgents*, de tipo *broadcast*, para obtener el estado de los agentes disponibles. Además, como ya he dicho antes, ahora los mensajes se envían desde el módulo Agent Controller.

4.3.3 Implementación y resultados

En la sección anterior he propuesto un mecanismo que, teniendo un sistema de videoconferencia distribuido entre varios nodos de computación, permite tomar decisiones acerca de cómo

Tabla 4.1 : Mensajes Agent Controller - Agents

<i>createOTM</i>	
Solicita la creación de un nuevo OTM a un Agent. El Agent es seleccionado en modo <i>round-robin</i> .	
<i>Queue</i>	Agent
<i>Type</i>	unicast
<i>Parameters</i>	-
<i>Returns</i>	OTM id
<i>createOTM</i>	
Solicita la creación de un nuevo OTM en un Agent específico.	
<i>Queue</i>	Agent_id
<i>Type</i>	unicast
<i>Parameters</i>	-
<i>Returns</i>	OTM id
<i>deleteOTM</i>	
Solicita la destrucción de un OTM. El mensaje se envía a todos los Agents pero solamente el Agent que dispone del OTM en cuestión será el que proceda a su borrado.	
<i>Queue</i>	Agent
<i>Type</i>	broadcast
<i>Parameters</i>	OTM id
<i>Returns</i>	Result code
<i>getAgents</i>	
Solicita información a todos los Agents disponibles en el sistema.	
<i>Queue</i>	Agent
<i>Type</i>	broadcast
<i>Parameters</i>	-
<i>Returns</i>	Stats object

programar los recursos entre ellos de forma dinámica. Para validar su funcionamiento he realizado una implementación del modelo en el proyecto Licode y realizado una prueba en un entorno real desplegado en servidores de varios proveedores Cloud. A continuación explico los detalles de esta implementación así como el experimento realizado y sus resultados.

4.3. MECANISMO PARA HABILITAR LA PROGRAMACIÓN DE RECURSOS

Implementación

He realizado la implementación del mecanismo propuesto anteriormente sobre una versión de Licode que implementa el modelo dOTM. Esta versión modifica la implementación de la MCU Erizo para soportar su distribución entre varios servidores. Para ello se modifica el concepto original de MCU separándolo en componentes que se encuentran distribuidos, lo que complica sustancialmente el modelo.

Para redirigir cada cliente al OTM adecuado se requiere una nueva capa de control que gestiona todos los recursos asociados (flujos, OTMs y salas). RoomController, conectado por un lado directamente con los clientes y por otro con los OTMs a través de un bus de mensajes, cumple esta función. La pieza monolítica Erizo debe dejar de serlo, separando sus dos componentes (ErizoController y el núcleo) en dos procesos diferentes comunicados a través del bus de datos. Las llamadas a procedimientos locales pasan a ser llamadas remotas a través de dicho bus. El núcleo ahora está limitado a una única operación por proceso del sistema operativo. Además aparecen dos entidades nuevas RoomController y ErizoAgent, ambos son los equivalentes en Licode a los módulos explicados antes.

En este punto empieza la implementación de mi extensión de la arquitectura introduciendo el módulo Agent Controller, que en esta implementación recibe el nombre de ECCH (*Erizo Controller Cloud Handler*). Modifico también los mensajes intercambiados entre el controlador (ErizoController) y los agentes (ErizoAgents) que ahora se intercambiarán entre ECCH y los agentes. Para enviar los mensajes *unicast* implemento llamadas a procedimiento remoto (RPC) usando *direct exchanges* de AMQP. Para los mensajes de *broadcast* utilizo *exchanges* tipo *topic*.

Cuando ECCH envía el mensaje *getAgents* de tipo *broadcast* para obtener la información sobre los agentes disponibles, éstos responden proporcionando la siguiente información:

- **Agent_id**: identificador único del agente.
- **rpc_id**: identificador de la cola de mensajes AMQP a la que el agente está suscrito. Debe utilizarla el controlador para contactar con este agente.
- **Objeto de metadatos**: objeto tipo JSON con la información de contexto configurada al registrar el agente.
- **Objeto de estado**: objeto de tipo JSON con información sobre el estado del agente en

tiempo real. Incluye información sobre el uso de memoria y CPU.

- **Contador de Keep Alive:** un contador utilizado para asegurar que el agente responde a peticiones de creación de OTMs. Cuando el contador alcanza un límite establecido el agente se elimina del registro del controlador.

Para tomar la decisión de qué agente elegir para crear un nuevo OTM, el controlador utiliza por defecto el modo *round-robin*. Pueden introducirse políticas de decisión personalizadas mediante *scripts* programáticos. Uno de estos *scripts* recibe como parámetro un objeto con los agentes disponibles y su información, toma la decisión basándose en dicha información y devuelve el agente seleccionado.

Descripción del experimento

Utilizando esta implementación del modelo propuesto voy a realizar un experimento que probará que el mecanismo de registro, reporte y toma de decisiones es válido en un escenario de videoconferencia distribuida desplegado en un entorno Cloud. Para ello propongo un caso de uso muy común en el que imaginamos ser un proveedor de videoconferencia como servicio que necesita provisionar salas de videoconferencia bajo demanda a un número de usuarios variable.

Para alojar los recursos necesarios poseemos un conjunto de servidores fijos en los que, tratando de facilitar el despliegue, hemos configurado un entorno Cloud privado que nos permitirá arrancar máquinas virtuales bajo demanda. Estas máquinas virtuales tendrán configurados los diferentes componentes de Licode como veremos más adelante. En caso de que de manera eventual necesitemos más recursos de los disponibles tenemos la posibilidad de arrancar máquinas virtuales adicionales en dos proveedores Cloud públicos. En este caso estas máquinas solamente alojarán Agents ya que son los OTMs el cuello de botella cuando hablamos de consumo de recursos. Como podemos comprobar en [Rodríguez 2014], los OTMs limitan las capacidades de las máquinas virtuales en términos de CPU. El consumo de memoria y ancho de banda no es un factor crítico en instancias de baja capacidad de procesamiento en este tipo de proveedores de Cloud. Por ello para este experimento voy a simplificar las condiciones teniendo únicamente en cuenta las medidas de consumo de CPU reportadas por los agentes.

El uso de recursos en los proveedores públicos implica un coste económico que no se contempla en el caso del privado ya que las máquinas físicas nos pertenecen. Es por esto por lo

4.3. MECANISMO PARA HABILITAR LA PROGRAMACIÓN DE RECURSOS

que el criterio de asignación de recursos consiste en priorizar el uso del Cloud privado y sólo cuando sea estrictamente necesario por carecer de recursos en él, asignar conexiones en los públicos. Sin embargo, en este caso vamos a también a suponer que queremos disponer siempre de una pequeña reserva de recursos en nuestra infraestructura privada para alojar ciertas sesiones especiales. Por lo tanto configuraremos un umbral de uso de CPU que no podremos sobrepasar nunca.

Una vez que la configuración de entorno esté lista el experimento consiste en ir conectando usuarios de forma consecutiva. Los primeros se asignarán siempre al Cloud privado y cuando el umbral configurado sea alcanzado, el programador de recursos comenzará a enviar las conexiones a los públicos. Para decidir en cuál de los dos entornos disponibles asignar una determinada conexión seguiré un criterio de carga utilizando siempre el que menos consumo de CPU tenga en cada momento. Defino también un factor de peso entre ambos entornos públicos que puede utilizarse para configurar una prioridad entre ellos.

Configuración del entorno

Figura 4.5 : Configuración del despliegue

Para el despliegue del escenario he elegido Openstack Compute ⁱ como el Cloud privado y

ⁱ<http://www.openstack.org/software/openstack-compute/>

Tabla 4.2 : Instancias utilizadas en los Agents

Cloud Platform	Instance Type	CPU	Memory
Openstack	m1.small	1 vCPU	2 GB
Google Cloud	f1-micro	1 vCPU (Shared GCPU)	0.60 GB
Amazon EC2	t2.micro	1 vCPU (Variable ECU)	1 GB

Amazon Web Services EC2 ⁱ y Google Compute Engine ⁱⁱ como los públicos. En la Figura 4.5 puede observarse un diagrama con los diferentes componentes incluidos en este despliegue.

He desplegado el controlador (Erizo Controller) en una máquina aislada. Por otra parte se encuentran el servidor de autorización (con Nuve), el servicio Web que se comunica con éste para la creación de salas y que sirve la aplicación Web y el servidor de RabbitMQ para la cola de mensajes AMQP. La infraestructura en que corren estos componentes no es relevante para este experimento. Tampoco las características de las máquinas virtuales en las que están desplegados.

Por otro lado he desplegado un Agent en una máquina virtual de cada uno de los proveedores Cloud mencionados anteriormente, Openstack, Google Cloud y Amazon EC2. Estos serán los encargados de arrancar OTMs cuando el controlador se lo solicite. En la Tabla 4.2 puede observarse una descripción de las características de cada una de dichas máquinas virtuales. El sistema operativo utilizado es Ubuntu 14.04 LTS para todas ellas. El pequeño tamaño y la baja capacidad de computación de las máquinas virtuales que he elegido no es un problema en el ámbito de este experimento. De hecho es conveniente tener una baja capacidad de computación para saturar de manera sencilla los servidores y hacer el experimento más ágil y descriptivo. En relación a la cantidad de memoria y ancho de banda, he comprobado que el disponible en estas máquinas es suficiente para el número de usuarios que voy a conectar.

Los clientes se conectan a las salas de videoconferencia utilizando el navegador Chromium ⁱⁱⁱ, la versión de código abierto de Google Chrome, en concreto la versión 45. Para poder conectar un número elevado de clientes arranco estos navegadores en una máquina virtual de Amazon EC2

ⁱ<http://aws.amazon.com/ec2/>

ⁱⁱ<http://cloud.google.com/compute/>

ⁱⁱⁱ<http://www.chromium.org>

4.3. MECANISMO PARA HABILITAR LA PROGRAMACIÓN DE RECURSOS

Algorithm 2 Decisión de selección de Agent

Require: *cpuThreshold*, *weightFactor*, *osAgent*, *googleAgent*, *ec2Agent*

```
1: osCPU  $\leftarrow$  getCurrentCPU(osAgent)
2: if osCPU < cpuThreshold then
3:   return osAgent
4: else
5:   googleCPU  $\leftarrow$  getCurrentCPU(googleAgent)
6:   ec2CPU  $\leftarrow$  getCurrentCPU(ec2Agent)
7:   if googleCPU * weightFactor < ec2CPU then
8:     return googleAgent
9:   else
10:    return ec2Agent
11:  end if
12: end if
```

con las mismas características que la del Agent y configurada para conectar un número de clientes de forma automática y consecutiva.

El Algoritmo 2 muestra la política de decisión que he configurado en el controlador. Ésta es ejecutada cada vez que se necesita un nuevo OTM para un nuevo usuario. En el objeto de metadatos de cada agente podemos encontrar información sobre el proveedor de Cloud en el que se encuentra. De esta forma obtenemos los valores *osAgent*, *googleAgent* y *ec2Agent*. Este parámetro se configura al arrancar cada Agent y forma parte de su información de contexto. Por otra parte, el objeto de estado tiene la información de tiempo real y permite obtener el consumo de CPU en cada momento (método *getCurrentCPU()* en el algoritmo). Para este experimento he configurado las siguientes constantes:

- *cpuThreshold*: 60 %
- *weightFactor*: 1

Con este umbral de uso aseguro que el 40% de la capacidad de computación del Cloud privado se reserva para las hipotéticas salas especiales que he mencionado antes. Por otra parte he configurado un factor de peso neutral por lo que no existe ninguna preferencia a la hora de elegir el Cloud público. Cuando sea necesario alojar allí un OTM siempre será seleccionado el proveedor que esté consumiendo menos CPU en ese momento.

Sin embargo este factor sería interesante para diseñar algoritmos que establecieran prioridades teniendo en cuenta criterios avanzados como el precio, la localización geográfica, etc. Además

Figura 4.6 : Uso de CPU en los Agents

podríamos pensar en algoritmos con retroalimentación que modificaran el valor de este factor en tiempo real teniendo en cuenta valores como la fluctuación en los precios.

Resultados

He creado seis salas de videoconferencia y conectado clientes a ellas cada 35 segundos de forma aleatoria empezando en el segundo 110. Esta dinámica varía solamente en el segundo 1320 cuando desconecto a cinco clientes de sus respectivas salas. El resultado del experimento puede observarse en la Figura 4.6.

Como puede observarse, por un lado el uso de CPU en la máquina virtual de Opensatck crece a medida que nuevos clientes se conectan al servicio. Por otra parte la actividad en las máquinas de Google Cloud y Amazon EC2 es nula durante los primeros compases. El primer momento clave ocurre en el segundo número 875 cuando el umbral del 60% establecido se sobrepasa. A partir de este momento el uso de CPU en Google Cloud y Amazon EC2 comienzan a crecer de forma homogénea mientras que el de Openstack se mantiene prácticamente constante sufriendo algunas variaciones mínimas. Esto es debido a que, a pesar de no crear nuevos OTMs en este Agent, los OTMs ya asignados pueden recibir la suscripción de nuevos clientes lo que aumenta de forma mínima su consumo de recursos.

El crecimiento de Google Cloud y Amazon EC2 se detiene en el segundo 1330 como

4.4. ESTIMACIÓN DE CONSUMO DE RECURSOS EN MCUS DISTRIBUIDAS

consecuencia de la desconexión de 5 participantes explicada antes. A partir de ese momento el uso en Openstack deja de sobrepasar el límite por lo que se asignan nuevas conexiones al Agent correspondiente. Finalmente en el segundo número *1410* el umbral es excedido de nuevo por lo que se repite el comportamiento anterior. El uso de CPU en Openstack vuelve a mantenerse constante y el de los proveedores públicos vuelve a crecer de forma conjunta hasta el final del experimento.

4.4 Estimación de consumo de recursos en MCUs distribuidas

En la sección anterior he propuesto un mecanismo que extiende el modelo dOTMs para habilitar en el controlador la configuración de políticas personalizadas para tomar decisiones inteligentes de asignación de recursos. Estas decisiones pueden tomarse de acuerdo al estado de cada servidor agente, reportado de forma periódica. El estado reportado consiste en el uso de recursos en términos de CPU, memoria y ancho de banda consumido. Estos parámetros nos dan una idea de cómo de saturados están los servidores pero no nos dan información suficiente para saber de forma fiable si pueden alojar nuevas conexiones.

Por ejemplo, una máquina de un solo núcleo de computación con un 5% de capacidad de CPU libre podría no ser capaz de alojar un OTM que una máquina con doble núcleo y el mismo espacio libre podría. Por lo tanto, una vez que tenemos la información de uso de recursos de cada agente es necesario ir un paso más allá y ser capaces de determinar qué cantidad de recursos consumiría una hipotética nueva conexión en caso de ser asignada en un agente. Hay que diseñar una manera de estimar la capacidad real de cada servidor en términos de su capacidad de alojar nuevos OTMs o conexiones.

Como explicaba en la Sección 2.3, algunos modelos de programación de tareas como Mesos basan sus decisiones en factores como los que estoy comentando aquí, la cantidad de recursos libres en los servidores distribuidos y la capacidad que cada nueva asignación necesita. Sin embargo esta caracterización de las asignaciones viene dada por los consumidores, que solicitan un espacio determinado. En nuestro caso es el programador el que debe realizar esta estimación basándose en los datos que le proporcionan los servidores.

Continuando con el repaso de las soluciones disponibles analizadas en dicha sección, el modelo propuesto en [Jung 2011] está muy en la línea del problema que se me plantea en este punto. Para tomar las decisiones de asignación de recursos el primer factor que tienen en cuenta

es si un servidor concreto tiene capacidad suficiente para alojarla.

En el caso de los OTMs, calcular la carga de trabajo no es una tarea trivial. Como ya hemos visto los OTMs son intensivos en uso de CPU por lo que este valor podría ser el de referencia a la hora de determinar su carga. Sin embargo, el consumo de CPU de un OTM puede variar en función del número de suscriptores que tenga y por lo tanto es complicado encontrar una relación entre la cantidad de OTMs gestionados en un servidor y el consumo de CPU realizado. Pero sí es posible buscar una relación por OTM y número de suscriptores. En el caso de la videoconferencia, en el que todos los participantes se suscriben al resto, teniendo esta relación podríamos estimar el consumo de un nuevo OTM en base al número de participantes conectados a la sala.

Basándome en esta idea, en esta sección propongo una nueva métrica que, dada una solicitud de asignación de una conexión, permite estimar la cantidad de recursos que necesitaría para ser alojada en un servidor determinado. De esta manera, cuando tenemos que tomar la decisión de dónde asignar dicha conexión tendremos información suficiente para asegurar que los recursos disponibles se explotan de la manera más eficiente posible.

4.4.1 Métrica para estimar el uso de recursos

En el modelo de registro y reporte de agentes propuesto anteriormente, el controlador sabe el uso de recursos realizado en cada uno de ellos. Con esto tiene una idea de cuál tiene más disponibilidad para alojar un nuevo OTM pero una métrica basada en la CPU es muy basta. Resulta evidente que un cierto porcentaje de CPU disponible puede servir para alojar diferente número de OTMs dependiendo de las características de cada agente (número de núcleos de procesamiento, tipo de procesador, etc.). Además, dado un OTM, puede consumir más o menos recursos dependiendo del número de suscriptores que tenga. En el caso de la videoconferencia dependiendo del número de participantes de la sesión en la que se encuentra. Por lo tanto es necesario definir una métrica que nos permita estimar cuántos OTMs puede alojar un agente independientemente de sus características y teniendo en cuenta el número de participantes de la sala.

En el modelo de OTMs, cada uno de los agentes disponibles está monitorizando el consumo de CPU para cada uno de los OTMs que gestiona. Esto es posible porque cada OTM está ejecutándose en un proceso diferente del sistema. Si un agente está gestionando n OTMs:

4.4. ESTIMACIÓN DE CONSUMO DE RECURSOS EN MCUS DISTRIBUIDAS

$$U_i, i = 0, 1, \dots, n \quad (4.1)$$

U representa el uso de CPU por cada OTM. El controlador recibe de forma periódica el reporte de todos los agentes disponibles. En su condición de capa de control, tiene conocimiento de cuántos suscriptores están gestionados por cada OTM, independientemente del agente en el que este OTM se encuentre. Por lo tanto, para cada agente es capaz de calcular una tupla con el número de suscriptores y el uso de CPU correspondiente:

$$\{S, U_i\}_{i=0}^n \quad (4.2)$$

donde S es el número de suscriptores correspondientes a dada medida recibida. Así, dado que el controlador recibe m medidas correspondientes a un número determinado de OTMs con el mismo número de suscriptores, siendo s el máximo número de suscriptores recibidos durante los reportes, recolecta una serie de conjuntos de datos con la forma:

$$\{x_j, y_{j1}, \dots, y_{jm}\}_{j=1}^s \quad (4.3)$$

donde x_j representa el número de suscriptores y y_{jm} el uso de CPU por j suscriptores recibido en la medida m . Como se explica en la referencia del modelo OTM [Rodríguez 2014], el uso de CPU en un OTM crece de forma lineal con el número de suscriptores. Consecuentemente, se puede estimar la relación entre el número de suscriptores de un OTM y su uso de CPU como la regresión lineal resultante de los conjuntos de datos del vector 4.3:

$$y = \alpha + \beta x \quad (4.4)$$

Con esta expresión, dado un OTM con x suscriptores, se puede estimar su consumo de CPU y . Es importante remarcar que esta ecuación corresponde a un agente específico ya que se ha calculado utilizando las medidas reportadas por él. Existirá por tanto una ecuación diferente por cada agente disponible y dicha ecuación dependerá de las características y rendimiento del agente.

Como he introducido antes, en una sesión de videoconferencia cada participante publica su flujo multimedia de audio/vídeo en una sala y se suscribe al resto de flujos presentes en dicha sala (correspondientes al resto de participantes). Esto significa que cuando un nuevo participante se

conecta a una sala con p participantes el sistema debe asignar a un agente un nuevo OTM que tendrá p suscriptores, uno por cada participante ya conectado. En términos de CPU y según la expresión 4.4, el consumo de CPU causado al añadir este OTM a un determinado agente sería:

$$Q = \alpha + \beta p \quad (4.5)$$

Por otra parte, los OTMs de los participantes ya existentes en la sala incrementarán su número de suscriptores en una unidad cada uno. Esto ocurre porque el nuevo participante también se suscribirá a sus flujos multimedia para poder recibirlos. Esto producirá un incremento de CPU según la siguiente ecuación:

$$R_k = \alpha + \beta p - [\alpha + \beta(p-1)], k = 1, 2 \dots o; o \leq n \quad (4.6)$$

$$R_k = \beta, k = 1, 2 \dots o; o \leq n \quad (4.7)$$

siendo k el OTM y o el número total de OTMs de esa sala que están alojados en el agente del que hablamos. Como siempre hablamos de OTMs distribuidos, hemos de recordar que una sala puede estar también distribuida entre diferentes agentes. Por lo tanto cada agente tiene que tener en cuenta solamente los OTMs que está gestionando. Teniendo en cuenta estas consideraciones, el consumo de CPU producido por los OTMs ya existentes en un agente será:

$$R = \sum_{k=1}^o R_k \quad (4.8)$$

$$R = \beta o \quad (4.9)$$

Y de acuerdo a las expresiones 4.5 y 4.9 el uso total de CPU producido al añadir el OTM será:

$$\Delta CPU = R + Q \quad (4.10)$$

$$\Delta CPU = \alpha + \beta(p + o) \quad (4.11)$$

Como podemos ver, la expresión 4.11 estima el incremento de CPU producido en un agente

4.4. ESTIMACIÓN DE CONSUMO DE RECURSOS EN MCUS DISTRIBUIDAS

de coeficientes α y β en su regresión lineal cuando un nuevo participante se une a una sala con p participantes y ya se encuentra alojando o OTMs de la misma sala. Los agentes reportan de manera periódica su actual consumo de CPU, CPU_{use} . Por ello, con este dato podemos estimar cuántos OTMs podría gestionar cada uno de ellos. Estableciendo un umbral máximo de uso de CPU TH evitamos posibles saturaciones en el sistema. Y finalmente, el número de OTMs que un agente puede alojar:

$$N_{otms} = \frac{TH - CPU_{use}}{\Delta CPU} \quad (4.12)$$

$$N_{otms} = \frac{TH - CPU_{use}}{\alpha + \beta(p + o)} \quad (4.13)$$

Gracias a esta nueva métrica, cuando un participante se une a una sala de videoconferencia y es necesario asignar la gestión de un OTM a un agente, podemos estimar el número de OTMs de ese tipo que podría alojar cada uno de los agentes disponibles. Y con ello tomar la decisión correspondiente.

4.4.2 Evaluación de resultados

Una vez planteada la métrica de estimación de uso de recursos, en esta sección voy a evaluar si ésta ofrece ventajas en comparación con una solución en la que para tomar las decisiones de asignación únicamente se tiene en cuenta el nivel de CPU usado en los agentes. Para ello he diseñado un experimento consistente en conectar de forma consecutiva un número determinado de clientes a un conjunto de salas de videoconferencia. Todos los clientes están gestionados por el mismo agente pero establezco un umbral de consumo de CPU para asegurar que no se satura. Cuando el límite se sobrepasa empiezo a asignar OTMs a un segundo agente.

Para realizar el experimento he implementado en Licode un módulo que gestiona la métrica propuesta. Este módulo forma parte de ECCH y dispone de los métodos detallados en la Tabla 4.3. De esta manera cada vez que ECCH recibe un reporte de los agentes, utiliza el método *updateRegresion* para actualizar la regresión correspondiente al agente con un nuevo valor. Por otra parte, cuando se necesita un nuevo OTM, la política de decisión del ECCH tiene acceso al método *getEstimatedUsage* mediante el que obtiene para cada agente registrado una estimación de la CPU que supondría alojarlo en ellos. Con esta estimación y el consumo en tiempo real

Tabla 4.3 : Interfaz del módulo de gestión de regresiones lineales

<i>updateRegresion</i>	
Introduce una nueva muestra CPU - suscriptores en la regresión lineal de un agente y recalcula la regresión.	
<i>Parameters</i>	agent id, nSubscribers, usage
<i>Returns</i>	-
<i>getEstimatedUsage</i>	
Devuelve el uso de CPU estimado para un nuevo OTM con un número determinado de suscriptores en un agente concreto.	
<i>Parameters</i>	agent id, nSubscribers
<i>Returns</i>	cpu use

reportado, es capaz de calcular el número de OTMs que podría alojar cada agente según la métrica propuesta.

En mi implementación los agentes monitorizan su consumo de CPU utilizando el módulo `NPM use`ⁱ que parsea los datos obtenidos mediante *Linux proc File System*.

Para este experimento he desplegado los componentes de Licode en máquinas virtuales de Amazon Web Services EC2. Para los agentes he utilizado instancias de tipo *m3.medium* con 1 CPU virtual y 3.75 GiB de memoria. He desplegado el controlador en una máquina aparte y Nuve, la aplicación Web y el servidor de AMQP en otra. Las características de éstas dos últimas instancias no son relevantes para el experimento. Todas ellas tienen el sistema operativo Ubuntu 14.04 LTS y los clientes utilizan de nuevo el navegador Chromium versión 45. De igual manera que en los anteriores experimentos la baja capacidad de los agentes no es un problema. Al contrario, nos conviene saturarlos rápidamente para hacer el experimento más ágil y descriptivo.

En la Figura 4.7 se puede observar la evolución de uso de CPU en los agentes en ambas configuraciones, sin y con métrica, cuando los usuarios se conectan de forma aleatoria a cinco salas diferentes. En ambos casos *Agent 1* representa el agente principal en el que alojamos todas las nuevas conexiones. En el caso en el que no uso la métrica sigo asignando OTMs a este agente siempre y cuando el uso de CPU no sobrepase el umbral del 80 % establecido. En el caso de usar la métrica, cuando un nuevo cliente se conecta y es necesario asignar un nuevo OTM, utilizo la ecuación 4.13 para, a partir de los datos disponibles en la regresión lineal del agente, calcular el

ⁱ<https://www.npmjs.com/package/use>

4.4. ESTIMACIÓN DE CONSUMO DE RECURSOS EN MCUS DISTRIBUIDAS

(a) Uso de CPU sin métrica

(b) Uso de CPU con métrica

Figura 4.7 : Uso de CPU en los experimentos

número de OTMs que podría alojar para el cliente que se conecta. Si ese número es mayor que 0 procedo con la asignación en ese agente. En caso contrario asigno su creación al segundo. Para poder realizar una comparativa entre los dos escenarios, he utilizado el mismo orden de conexión de clientes para ambos. En la Figura 4.8 se puede observar la regresión lineal del agente principal una vez terminado el experimento.

Podemos observar como al principio, para los primeros 20 participantes, el comportamiento es muy similar en ambos casos. El uso de CPU del agente principal crece de manera constante

Figura 4.8 : Agent 1 OTMs Linear Regression

hasta que se aproxima al umbral establecido. En el caso en el que no se utiliza la métrica, el participante número 21 puede alojarse en el agente principal ya que el consumo se encuentra por debajo del límite. Lo mismo ocurre para el 22. Como consecuencia el umbral acaba sobrepasándose considerablemente, alcanzado el consumo de CPU prácticamente el 100 % con la consecuente saturación del sistema. Esto podría producir pérdidas de calidad en las comunicaciones ya establecidas. El resto de participantes a partir de ese momento se asignan al *Agent 2*, cuyo consumo de CPU comienza a incrementarse.

Por otra parte, en el segundo caso (con métrica) el agente es capaz de predecir que la conexión de participante 21 va a hacer que se sobrepase el límite de uso. La sala a la que el cliente va a conectarse tiene una ocupación de 4 participantes y como hasta entonces solamente se ha utilizado un agente, los 4 OTMs correspondientes están alojados en él. Así, los parámetros p and o de la expresión 4.11 corresponden a 4. Según la ecuación 4.4 el incremento de CPU corresponde al causado por 8 suscriptores. Comprobando la regresión lineal del agente (Figura 4.8) ese valor está en torno a 4.5%. En ese momento el uso de CPU por parte de *Agent 1* asciende al 78%, un 2% por debajo del umbral. Por lo tanto no es capaz de alojar este nuevo OTM y se éste será asignado a *Agent 2*.

A partir de ese momento *Agent 1* continúa denegando nuevas conexiones hasta que ECCH detecta que un nuevo OTM es lo suficientemente pequeño como para cumplir los requisitos, lo que ocurre para el participante número 25. El umbral se sobrepasa en algunos momentos del experimento debido a la fluctuación en las medidas pero la variación es realmente pequeña en

4.5. EVALUACIÓN DE ESTRATEGIAS DE PROGRAMACIÓN DE RECURSOS

comparación con lo que veíamos en la anterior configuración. De esta forma aseguramos que el sistema no va a saturarse y conseguimos la calidad esperada en las comunicaciones. Evidentemente el gráfico de uso del segundo agente experimenta un mayor incremento que antes ya que está gestionando más OTMs. El resultado es que se ha conseguido balancear la carga haciendo un uso más eficiente de los recursos.

4.5 Evaluación de estrategias de programación de recursos

En la sección anterior he propuesto una métrica que permite estimar cuántos OTMs puede alojar un agente sabiendo el número de participantes conectados a la sala que corresponde. Gracias a esta métrica es posible diseñar estrategias de programación de recursos avanzadas y de manera independiente al las capacidades de computación de los servidores distribuidos en los que se encuentran los agentes. También aseguro que los servidores no se saturan y que siempre se respetan los umbrales de uso establecidos.

Usando esta métrica, en esta sección propongo tres estrategias de programación para asignar OTMs en los agentes disponibles en un sistema determinado. Cada una de estas estrategias está diseñada pensando en un diferente patrón o filosofía de uso de recursos. Una vez diseñadas las implemento en el sistema Licode como parte de las políticas de decisión de ECCH y preparo un escenario para comparar su rendimiento. Frente a los resultados de un experimento en este escenario analizo los pros y los contras de cada uno de ellos y abro una discusión en torno a los tipos de casos de uso en los que cada uno es aplicable. Para realizar la comparativa propongo una serie de coeficientes de rendimiento que permiten determinar la idoneidad de las políticas en cualquier caso de uso.

4.5.1 Diseño de políticas de asignación

En la Tabla 4.4 podemos ver una descripción de los métodos que voy a utilizar en los algoritmos de decisión. Todos los algoritmos requieren como entrada los parámetros que indican el umbral de uso establecido en los agentes, la sala a la que el participante va a conectarse y la lista de agentes disponibles en el sistema. Tras su ejecución devuelven el agente seleccionado para realizar la asignación o el código *error* cuando ningún agente es seleccionado.

Tabla 4.4 : Descripción de los métodos usados en los algoritmos

<i>getParticipants</i>	
Obtiene el número de participantes conectados a una sala.	
<i>Parameters</i>	room
<i>Returns</i>	nParticipants
<i>getCurrentCPU</i>	
Obtiene el uso de CPU actual por un agente.	
<i>Parameters</i>	agent
<i>Returns</i>	CPU use
<i>getAgentRegression</i>	
Obtiene los coeficientes de la regresión lineal de un agente.	
<i>Parameters</i>	agent
<i>Returns</i>	regression coefficients (α and β)
<i>getRoomOTms</i>	
Obtiene el número de OTMs de una sala alojados en un agente.	
<i>Parameters</i>	agent, room
<i>Returns</i>	nOTMs
<i>metric</i>	
Calcula el número de OTMs estimado que un agente puede alojar.	
<i>Parameters</i>	regression coefficients, participants, otms, threshold, cpu
<i>Returns</i>	number of OTMs

Explotación completa de Agents

En un escenario en el que queremos utilizar nuevos servidores sólo cuando es estrictamente necesario, hemos de explotar cada agente hasta su máxima capacidad. De esta forma, en un primer momento el sistema utiliza un único agente en el que son alojados todos los OTMs. Sólo cuando la capacidad del agente alcanza su límite se añade un nuevo agente al conjunto. Cuando se encuentra disponible más de un agente se intenta alojar un OTM siguiendo siempre el mismo orden de asignación. De esta manera hay más posibilidades de liberar espacio en el último de los agentes y ser capaces de apagarlo.

Para lograr este patrón de uso es necesario diseñar una estrategia de asignación que elija siempre el mismo agente mientras tenga recursos disponibles, estableciendo un umbral de uso

4.5. EVALUACIÓN DE ESTRATEGIAS DE PROGRAMACIÓN DE RECURSOS

para evitar la saturación del servidor. Para asignar un nuevo OTM se utiliza por tanto el mismo agente siempre que tenga espacio suficiente según ese umbral y la métrica propuesta. En otras palabras, cuando es necesario asignar un nuevo OTM, calculo el número de OTMs que el primer agente puede alojar. Si ese número es mayor que cero procedo a su asignación y en caso contrario utilizo el siguiente. Este proceso es repetido para cada nuevo OTM requerido teniendo en cuenta que la asignación de un determinado OTM podría ser denegada pero la de uno posterior admitida por necesitar una cantidad menor de recursos. El Algoritmo 3 muestra el detalle de esta estrategia.

Algorithm 3 Explotación completa de Agents

Require: *threshold, room, agents*

```
1:  $p \leftarrow \text{getParticipants}(\text{room})$ 
2: for all agents do
3:    $\text{cpu} \leftarrow \text{getCurrentCPU}(\text{agent})$ 
4:    $\text{reg} \leftarrow \text{getAgentRegression}(\text{agent})$ 
5:    $o \leftarrow \text{getRoomOtms}(\text{agent}, \text{room})$ 
6:    $n\text{Otms} \leftarrow \text{metric}(\text{reg}, p, o, \text{threshold}, \text{cpu})$ 
7:   if  $n\text{Otms} > 0$  then
8:     return agent
9:   end if
10: end for
11: return error
```

Distribución homogénea

Si existen una serie de recursos disponibles establecidos es muy probable que se quiera distribuir la carga de manera homogénea entre ellos. De esta forma mejoraremos el rendimiento de cada servidor y evitaremos su saturación. En este caso la política de decisión debe seleccionar siempre el agente que disponga de más recursos. En términos de la métrica, dado un nuevo OTM que asignar, se utilizará siempre el agente que más OTMs de ese tipo es capaz de alojar. Por supuesto teniendo en cuenta el umbral de uso establecido. Puede observarse el detalle de esta estrategia en el Algoritmo 4.

Algorithm 4 Distribución homogénea**Require:** *threshold, room, agents*

```

1:  $p \leftarrow \text{getParticipants}(\text{room})$ 
2:  $\text{selectedAgent} \leftarrow \text{null}$ 
3:  $\text{selectedCapacity} \leftarrow 0$ 
4: for all agents do
5:    $\text{cpu} \leftarrow \text{getCurrentCPU}(\text{agent})$ 
6:    $\text{reg} \leftarrow \text{getAgentRegression}(\text{agent})$ 
7:    $o \leftarrow \text{getRoomOtms}(\text{agent}, \text{room})$ 
8:    $n\text{Otms} \leftarrow \text{metric}(\text{reg}, p, o, \text{threshold}, \text{cpu})$ 
9:   if  $n\text{Otms} > \text{selectedCapacity}$  then
10:     $\text{selectedAgent} \leftarrow \text{agent}$ 
11:     $\text{selectedCapacity} \leftarrow n\text{Otms}$ 
12:   end if
13: end for
14: if  $\text{selectedAgent} \neq \text{null}$  then
15:   return  $\text{selectedAgent}$ 
16: end if
17: return error

```

Agrupación por salas

Como sabemos, los servicios de videoconferencia agrupan usuarios en salas para el intercambio de sus flujos multimedia. Si pensamos en los casos de uso más típicos (reuniones, clases, conferencias, etc.) los usuarios de la misma sala se conectan y desconectan de las salas al mismo tiempo o con diferencias de tiempo muy cortas. Teniendo esto en cuenta, podría ser interesante agrupar conexiones de usuarios de la misma sala en los mismos agentes. Así aumentamos la probabilidad de poder apagar un servidor agente que se quede libre como resultado del final de una de estas reuniones o clases.

Esta estrategia casa con la idea de incentivos de Mesos, expuesta en la Sección 2.3.2. Siempre que sea posible intento agrupar OTMs de usuarios conectados a la misma sala en el mismo agente. A igualdad de condiciones el OTM será asignado en el agente que más OTMs de ese tipo sea capaz de alojar de acuerdo a la métrica. Pueden verse los detalles de esta política de asignación en el Algoritmo 5.

4.5. EVALUACIÓN DE ESTRATEGIAS DE PROGRAMACIÓN DE RECURSOS

Algorithm 5 Agrupación por salas

Require: *threshold, room, agents*

```
1:  $p \leftarrow \text{getParticipants}(\text{room})$ 
2:  $\text{selectedAgent} \leftarrow \text{null}$ 
3:  $\text{selectedCapacity} \leftarrow 0$ 
4:  $\text{selectedRoomOtms} \leftarrow -1$ 
5: for all agents do
6:    $\text{cpu} \leftarrow \text{getCurrentCPU}(\text{agent})$ 
7:    $\text{reg} \leftarrow \text{getAgentRegression}(\text{agent})$ 
8:    $o \leftarrow \text{getRoomOtms}(\text{agent}, \text{room})$ 
9:    $n\text{Otms} \leftarrow \text{metric}(\text{reg}, p, o, \text{threshold}, \text{cpu})$ 
10:  if  $o > \text{selectedRoomOtms}$  and  $(n\text{Otms} > 0)$  then
11:     $\text{selectedAgent} \leftarrow \text{agent}$ 
12:     $\text{selectedCapacity} \leftarrow n\text{Otms}$ 
13:     $\text{selectedRoomOtms} \leftarrow o$ 
14:  else if  $o == \text{selectedRoomOtms}$  then
15:    if  $n\text{Otms} > \text{selectedRoomOtms}$  and  $n\text{Otms} > 0$  then
16:       $\text{selectedAgent} \leftarrow \text{agent}$ 
17:       $\text{selectedCapacity} \leftarrow n\text{Otms}$ 
18:       $\text{selectedRoomOtms} \leftarrow o$ 
19:    end if
20:  end if
21: end for
22: if  $\text{selectedAgent} \neq \text{null}$  then
23:   return  $\text{selectedAgent}$ 
24: end if
25: return error
```

4.5.2 Comparativa y discusión

Para probar el comportamiento de las estrategias diseñadas utilizo la implementación de la métrica en Licode explicada anteriormente. He desplegado los distintos componentes en máquinas virtuales de Amazon Web Services EC2 con las mismas características que en el experimento anterior. En este caso despliego tres agentes en máquinas *m3.medium*, siendo este número de agentes es suficiente para extraer las conclusiones que pretendo. Cuando un algoritmo descarta la asignación de un OTM considero que la asignación ha sido desatendida. Para los clientes utilizo de nuevo las mismas máquinas que anteriormente pero en este caso es necesario conectar un número más alto de ellos, ya que tenemos más capacidad de computación. Este número asciende a 50 participantes y he creado para ellos 6 salas diferentes de videoconferencia a las que se conectarán en un orden aleatorio pero único para todos los experimentos realizados.

ASIGNACIÓN DE RECURSOS EN SISTEMAS DE VIDEOCONFERENCIA DISTRIBUIDOS

En este caso he fijado el límite de uso en el 60 % de la capacidad de procesamiento total de los agentes. Pretendo que el límite se alcance antes y no hacer los experimentos innecesariamente largos.

Resultado de los experimentos

La Figura 4.9 muestra la evolución de uso de CPU en los tres agentes al usar cada una de las tres estrategias de programación diseñadas. Los resultados reflejan el comportamiento esperado. En el primer experimento las curvas de uso de CPU de los agentes comienzan a crecer solamente cuando el anterior ha alcanzado su límite y no puede alojar nuevos OTMs. Puede observarse cómo el primer agente sobrepasa el límite continuamente, discutiré esta característica más adelante. En el segundo experimento las tres líneas avanzan siguiendo la misma progresión debido a la programada distribución homogénea. Por último, en la estrategia de agrupación por salas, la evolución depende del orden de conexión de los clientes y por lo tanto no responde a ningún tipo de patrón.

Por otra parte, en la Figura 4.10 podemos observar las regresiones lineales de cada agente en todos los experimentos, en concreto en las Figuras 4.10(a), 4.10(c) y 4.10(e). Todas ellas son muy similares por lo que no ofrecen información relevante para discutir. Finalmente proporciono una serie de gráficos representando la distribución de las salas de videoconferencia entre los agentes, en las Figuras 4.10(b), 4.10(d) y 4.10(f). En ellas se puede observar el porcentaje de cada sala presente en cada agente. Una sala está presente totalmente en un agente si todos los OTMs de dicha sala están alojados en ese agente. Como introducía antes, la forma en que las salas se distribuyen en los agentes puede ser interesante en algunos casos de uso.

Herramientas

Para evaluar y comparar los resultados obtenidos en los experimentos, defino los siguientes coeficientes de rendimiento:

Saturación del sistema: defino el coeficiente de saturación del sistema como el porcentaje de tiempo que el sistema está saturado durante el experimento. Se calcula como la media entre los coeficientes de saturación de los agentes. Un agente está saturado si su nivel de uso de CPU excede el umbral establecido. Así, si un agente i está saturado durante el $S_i\%$ de la duración del

4.5. EVALUACIÓN DE ESTRATEGIAS DE PROGRAMACIÓN DE RECURSOS

(a) Explotación completa de Agents

(b) Distribución homogénea

(c) Agrupación por salas

Figura 4.9 : Uso de CPU en las estrategias diseñadas

ASIGNACIÓN DE RECURSOS EN SISTEMAS DE VIDEOCONFERENCIA DISTRIBUIDOS

(a) Regresiones en explotación completa de Agents

(b) Fragmentación en explotación completa de Agents

(c) Regresiones en distribución homogénea

(d) Fragmentación en distribución homogénea

(e) Regresión en agrupación por salas

(f) Fragmentación en agrupación por salas

Figura 4.10 : Regresiones lineales y nivel de fragmentación en las estrategias diseñadas

experimento:

$$C_1 = \frac{\sum_{i=1}^N S_i}{N} \quad (4.14)$$

donde N es el número de agentes disponibles.

4.5. EVALUACIÓN DE ESTRATEGIAS DE PROGRAMACIÓN DE RECURSOS

OTMs atendidos: este coeficiente representa el porcentaje de las peticiones de asignación de OTMs que han sido atendidas por los agentes disponibles. En otras palabras, indica el porcentaje de participantes que se han conectado correctamente a una sesión de acuerdo a los recursos disponibles.

$$C_2 = \frac{Attended_{otms}}{Total_{otms}} \times 100 \quad (4.15)$$

Eficiencia del sistema: para tener una idea de la eficiencia del sistema, defino un coeficiente que establece una relación entre el número de OTMs atendidos y el tiempo de no-saturación. Sistemas que atienden un gran número de OTMs pero saturan el uso de CPU no se consideran eficientes.

$$C_3 = \frac{(100 - C_1) \times C_2}{100} \quad (4.16)$$

Uso de recursos del sistema: este coeficiente representa la cantidad de recursos utilizados por los agentes durante el experimento respecto al total de los recursos disponibles. Dada una gráfica de uso de CPU para cada agente, el uso de recursos corresponde al área debajo de la línea de uso de CPU del agente, V_i . La cantidad total de recursos disponibles corresponde al área que queda debajo de la línea de umbral de uso, V_T .

$$V_i = \int_0^P CPU_i dp \quad (4.17)$$

$$V_T = \int_0^P TH dp \quad (4.18)$$

Una vez calculadas estas dos áreas, obtengo el coeficiente para cada agente i , llamado C_{4i} . Y finalmente calculo la relación para el sistema completo a partir de la media entre todos ellos.

$$C_{4i} = \frac{V_i}{VS_T} \quad (4.19)$$

$$C_4 = \frac{\sum_{i=1}^N C_{4i}}{N} \quad (4.20)$$

siendo N el número de agentes disponibles y P el número de participantes. C_{4i} representa el

porcentaje de uso de recursos para cada agente. C_4 la media entre todos ellos.

Desviación en el uso de recursos: el coeficiente anterior nos da una idea del uso medio del sistema. Pero es también interesante saber si ese uso se distribuye de forma homogénea entre los agentes o hay algunos de ellos consumiendo menos recursos que otros. Defino un nuevo coeficiente que indica la desviación media de los agentes respecto al uso medio del sistema.

$$C_5 = \frac{\sum_{i=1}^N |C_{4i} - C_4|}{N} \quad (4.21)$$

donde N es el número de agentes disponibles, C_{4i} el porcentaje de uso para el agente i y C_4 la media entre ellos.

Fragmentación: este coeficiente ayuda a cuantificar la fragmentación de las salas entre los agentes. Si una sala j tiene OTMs alojados en x_j agentes, habiendo R salas y A agentes disponibles:

$$C_6 = \frac{\sum_{j=1}^R x_j}{R \times A} \times 100 \quad (4.22)$$

representa el porcentaje de fragmentación de salas entre los agentes. Es decir, suponiendo 100 disponibles, una sala tendría "representación" en C_6 de ellos.

Discusión

En la tabla 4.5 pueden observarse los resultados de calcular los coeficientes enunciados anteriormente para los tres experimentos realizados.

Tabla 4.5 : Coeficientes de rendimiento para las estrategias evaluadas.

		Explot. de Agents	Distrib. homogénea	Agrup. por salas
C_1	Saturación del sistema	27.2 %	10.8 %	14.1 %
C_2	OTMs atendidos	92.0 %	82.4 %	88.0 %
C_3	Eficiencia del sistema	67.0 %	73.4 %	75.6 %
C_4	Uso de recursos del sistema	56.9 %	56.8 %	56.0 %
C_5	Desviación en el uso recursos	22.4 %	2.2 %	6.6 %
C_6	Fragmentación	94.4 %	94.4 %	61.1 %

4.5. EVALUACIÓN DE ESTRATEGIAS DE PROGRAMACIÓN DE RECURSOS

Como podemos ver, cada estrategia tiene resultados diferentes de rendimiento. La estrategia *Explotación completa de Agents* es la que tiene un mayor coeficiente de *Saturación del sistema*, con un 27.2 %, prácticamente doblando al del resto de estrategias. Este resultado se debe principalmente a la saturación del primer agente una vez que ha sido ocupado del todo. Cuando se asignan nuevos OTMs a otros agentes, el consumo por los OTMs que han sido alojados previamente en este primer agente sigue creciendo debido a que tienen que añadir suscriptores de los nuevos participantes en las salas. Aunque este incremento es muy pequeño en comparación con la gestión de un nuevo OTM, el agregado a lo largo del tiempo es significativo produciendo esta saturación. Una solución inmediata para esta circunstancia es la de reducir el umbral de uso para dejar un margen de actuación (en este caso ya lo tenemos) pero entonces reducimos también el número de OTMs que pueden alojar otros agentes. Podemos ver cómo esta estrategia casa perfectamente con el escenario sugerido en el que nuevas máquinas virtuales agentes son añadidas al conjunto de recursos del sistema únicamente cuando la anterior alcanza el límite de uso establecido.

Por otra parte, el algoritmo de *Distribución homogénea* funciona perfectamente para un caso en el que es una prioridad respetar el umbral de uso, con un nivel de *Saturación del sistema* del 10.8 %. En este caso podríamos pensar en establecer valores de umbral más altos ya que es más difícil que llegue a alcanzarse.

Sin embargo, es evidente que cuanto más alto es el nivel de saturación más alto es el porcentaje de *OTMs atendidos*, con un 92 % en el caso de la primera estrategia, respecto al 82.4 % y el 88 % de las otras dos. El coeficiente de *Eficiencia del sistema* nos ayuda a entender cuál de las estrategias diseñadas tiene una mejor eficiencia global teniendo en cuenta los ratios de saturación y de atención de OTMs. Así, la estrategia de *Agrupación por salas* resulta ser la más eficiente, lo que confirma que el escenario que he sugerido para este caso es el ideal, donde se intentan agrupar de forma inteligente los OTMs para tener un rendimiento óptimo.

Todos los algoritmos tienen un coeficiente de *Uso de recursos del sistema* similar, en torno al 56-57 %, mientras que la distribución de este uso entre los agentes es completamente distinta, como muestra el coeficiente de *Desviación en el uso de recursos*. En el caso de *Explotación completa de Agents* esta diferencia es especialmente alta, con una variación media del 22.4 %, mientras que la *Distribución homogénea* alcanza su objetivo y resulta mínima, con un 2.2 %. Con esto se enfatiza la idea de que la estrategia de *Explotación completa de Agents* está pensada para

arrancar agentes gradualmente.

Finalmente, el coeficiente de *Fragmentation*, muestra cómo en las dos primeras estrategias una sala está presente en prácticamente todos los agentes arrancados. Como era de esperar, en un escenario en que queremos reducir al máximo esta fragmentación, la estrategia de *Agrupación por salas* es la idónea, reduciendo este coeficiente casi el 30 %, desde el 94.4 % hasta el 61.1 %.

En resumen, con este experimento he comprobado que las estrategias propuestas cuadran con el escenario para el que habían sido diseñadas, cada una de ellas con sus ventajas y limitaciones. Los resultados muestran que la estrategia de *Agrupación por salas* quizá sea la más completa, con mejores resultados en eficiencia y fragmentación. En escenarios donde no tenemos un conjunto fijo de recursos y los agentes se van arrancando según se necesita, sería conveniente optar por una solución como la de la *Explotación completa de Agents*. Finalmente, si tenemos ese conjunto fijo de recursos y queremos utilizarlos de la manera más eficiente posible la mejor opción es la de distribuir la carga con el algoritmo de *Distribución homogénea*.

4.6 Conclusiones

En este capítulo he abordado el problema de cómo programar recursos en sistemas de videoconferencia distribuidos. Estos sistemas tienen ciertas características que los hacen especiales respecto a otros sistemas distribuidos en los que ya existen soluciones de asignación. El primer objetivo ha sido por tanto analizar esas características sobre el modelo de MCUs distribuidas y determinar cuáles de las soluciones descritas en el estado del arte (Sección 2.3) podrían ser aplicadas o valer como punto de partida para mi escenario concreto.

Una vez claras las condiciones de partida, he propuesto un modelo que habilita la programación inteligente de recursos utilizando algoritmos de decisión personalizados. Para ello es necesario establecer un mecanismo de registro de las MCUs distribuidas así como de reporte de su estado en tiempo real para tener parámetros de decisión en dichos algoritmos. En el modelo de MCUs distribuidas, la unidad de escalabilidad pasa a ser el OTM, un componente software que realiza la función de repartición de flujos multimedia de un cliente a varios. Y el consumo de recursos por esta unidad de procesamiento depende de varios factores que hacen que el cálculo de espacio disponible en los servidores no sea trivial. Para solucionar esta circunstancia propongo una métrica de estimación de uso de recursos que facilitará la toma de decisiones de asignación.

Con estas herramientas tenemos las piezas suficientes para empezar a plantear el problema de

4.6. CONCLUSIONES

cómo realizar esta asignación de recursos entre los servidores distribuidos. Para ello he identificado tres escenarios con requisitos de escalabilidad diferentes y he diseñado políticas de asignación para ellos probándolas en un despliegue real y comparando los resultados de su rendimiento. Para analizar esos resultados propongo una serie de coeficientes que permiten sacar conclusiones y realizar un estudio detallado de la idoneidad de cada estrategia.

He comprobado que cada escenario concreto tiene sus propios requisitos y las políticas de decisión deben adaptarse a sus características y condiciones. Como conclusión general de este capítulo podría decir que gracias al mecanismo de registro y reporte, la métrica de estimación de uso y las herramientas de análisis de políticas de asignación que he propuesto, tenemos todas las piezas necesarias para resolver el problema de la asignación de recursos en sistemas de videoconferencia distribuidos, el objetivo principal planteado al inicio.

Es importante destacar que el modelo de escalabilidad de MCUs propuesto en la sección anterior sigue siendo válido para una arquitectura distribuida. Ahora simplemente tenemos un nivel más de escalabilidad. Así, el módulo Cloud Handler seguirá pudiendo escalar basándose en salas (su unidad de escalabilidad) arrancando y parando controladores y balanceando la carga entre ellos de acuerdo a sus políticas. Los controladores a su vez repartirán la carga entre los agentes disponibles asignando OTMs (su unidad de escalabilidad) de acuerdo a sus propias políticas. Tenemos una solución flexible y granular que permite realizar un despliegue de recursos de videoconferencia distribuidos y repartir la carga entre ellos de manera eficiente y de acuerdo a unas políticas de uso adaptables a los requisitos de cada escenario.

Capítulo 5

Validación y Resultados

En este capítulo describo del trabajo realizado durante estos años en relación con mi investigación y que sirve como validación de los resultados obtenidos. Esta validación se divide en dos grandes grupos. En primer lugar se encuentra el trabajo realizado en proyectos de investigación tanto nacionales como europeos. En la Sección 1.4 he explicado el objetivo de cada uno de estos proyectos por lo que aquí me limitaré a detallar la labor realizada en cuanto al uso concreto de mis propuestas. El segundo grupo corresponde a los trabajos de divulgación. Éstos consisten principalmente en la publicación de artículos en congresos y revistas aunque también citaré mi trabajo docente impartiendo cursos y seminarios relacionados.

5.1 Licode

Licode es sin duda el proyecto conductor de todo mi trabajo de investigación. Como se ha comprobado a lo largo de esta tesis, Licode implementa todos los resultados que he propuesto y sirve como banco de pruebas para validarlos. Mis tareas de implementación en este proyecto se centran por lo tanto en los diferentes componentes que he ido proponiendo aunque también he colaborado en otras partes del software.

Licode es un proyecto de software libre que tiene una comunidad de más de seiscientos desarrolladores por lo que su impacto es bastante notable. Su popularidad hace que sean muchas las empresas que se interesan por él y desean utilizarlo como proveedor de comunicaciones en sus servicios. Y en muchos casos necesitan del soporte directo de nuestro equipo para realizar despliegues o implementar ciertas funcionalidades adicionales al núcleo del producto.

Además, el esfuerzo invertido en Licode se ha visto reconocido de con diversos premios durante estos años:

- Primer premio en la competición de Innovación Abierta Open Middleware actúa_COM
- Premio especial de la comunidad del VII Concurso universitario de software libre

- Una de las 10 mejores ideas de negocio en Actúa UPM.
- Una de las 3 mejores ideas en el Concurso Actúa Loop.

5.2 Proyectos Nacionales

5.2.1 LTextreme

Uno de los objetivos de LTextreme es probar el uso de algoritmos de adaptación a la calidad de servicio sobre redes LTE. Para ello, participé en la tarea de la combinación de Licode con el protocolo de medida y adaptación de calidad de servicio Q4S sobre una maqueta LTE.

Figura 5.1 : Arquitectura del demostrador LTextreme

En este escenario, los clientes se conectan con diversos dispositivos con capacidades WebRTC y participan en sesiones de videoconferencia WebRTC a través de LTE. Q4S monitoriza constantemente la calidad de servicio y lanza alarmas a Licode para que pueda, por un lado, adaptar la calidad de servicio bajo demanda y por otro actuar sobre elementos de la red para intentar garantizar una calidad de servicio mínima. La arquitectura del prototipo puede verse en la Figura 5.1.

Para permitir esta adaptación al protocolo Q4S fue necesario modificar el núcleo de la MCU de Licode para limitar la adaptabilidad hacia abajo de la calidad, permitiendo a Q4S actuar sobre la red en casos especialmente críticos manteniendo un mínimo de calidad. La arquitectura desplegada en estos prototipos es la que corresponde al Capítulo 3 de esta tesis, con la citada

5.2. PROYECTOS NACIONALES

modificación en el núcleo. El prototipo estaba por lo tanto preparado para escalar al aumento de la demanda de usuarios. La experiencia sirvió para validar el modelo en un escenario con ciertas condiciones diferenciadoras respecto a otros escenarios probados con anterioridad.

5.2.2 MCS5

Como explicaba en la introducción, el proyecto MCS5 desarrollado para Telefónica tiene como objetivo la creación de un show-center virtual basado en la tecnología HTML5 y en el servicio colaborativo MashMeTV. El centro virtual contiene salas de demostración para los diversos servicios que se describen a continuación:

- Salas virtuales MashMeTV adaptadas al estilo de Telefónica.
- Salas de televisión social para acceso a contenidos específicos.
- Salas de video-vigilancia distribuida: "Vigile su hogar".
- Salas de teleasistencia.
- Salas de juegos.

Estos servicios son accesibles en Internet desde navegadores Web que soporten HTML5. El acceso se puede realizar desde un entorno multi-pantalla incluyendo ordenadores de sobremesa, portátiles, smartphones, tabletas y la propia televisión. Y está diseñado para ser accesible desde todos estos terminales, con un interfaz adaptado al tamaño de cada pantalla, y a la interacción humano-máquina (pantalla táctil, mando a distancia o teclado y ratón).

En este proyecto participé en el despliegue de Licode para proveer salas de videoconferencia al servicio MashMeTV, parte de cliente de esta plataforma. En la Figura 5.2 puede verse la arquitectura desplegada que corresponde de nuevo al modelo de escalabilidad propuesto en el Capítulo 3. En este caso podemos ver cómo además de escalar la MCU (Erizo y ErizoController) proponemos un modelo en el que en caso de ser necesario se pueden arrancar múltiples servidores de Nuve. Para ello se despliega un balanceador de carga que distribuirá las peticiones entre ellos. La base de datos es compartida por lo que no se añade complejidad en la implementación.

Figura 5.2 : Arquitectura de MCS5

5.2.3 EasyVideo

El proyecto *EasyVideo* proporciona una herramienta de colaboración interna para los empleados del Grupo Santander. Ofrece un servicio de videoconferencia con funcionalidades como compartición de ficheros, chat, visualización de escritorio remoto, moderación de participantes, etc. Es uno de los primeros proyectos en los que desplegué la arquitectura de MCU distribuida basada en OTMs y utilicé las políticas de decisión propuestas en mi modelo en la Sección 4. Esto se debe a que el despliegue se realiza en dos infraestructuras, una privada para acceso exclusivo de empleados del banco y una pública en la que pueden realizarse reuniones con usuarios externos. Así, en función del tipo de reunión se asignarán recursos a una u otra infraestructura.

Esta empresa tiene grupos en varios continentes por lo que resulta interesante replicar este despliegue en las diferentes localizaciones geográficas para permitir a los usuarios conectarse a la infraestructura más cercana y así ahorrar costes en las comunicaciones y mejorar la latencia de

5.3. PROYECTOS EUROPEOS

red. En el momento en que se escriben estas líneas nos encontramos en proceso de realizar estos despliegues distribuidos geográficamente en los que se utilizarán políticas de asignación avanzadas basadas en el rendimiento del sistema pero también en criterios de localización de los usuarios.

5.3 Proyectos Europeos

5.3.1 FIWARE, FI-XIFI y FICORE

Se trata de tres proyectos de financiación europea muy relacionados entre sí. FICORE es la continuación de FIWARE y el propósito de ambos es el de la creación de una nueva infraestructura de servicios basada en componentes básicos genéricos y reutilizables denominados Capacitadores Generales para la Internet del Futuro. El objetivo de FI-XIFI es el de desplegar esta infraestructura de servicios de manera federada entre múltiples nodos de computación en Europa.

La tarea principal de nuestro grupo de investigación en este proyecto es en los grupos de trabajo de Seguridad y de Cloud. En el primero de ellos me encargo de coordinar las tareas de diseño e implementación de dos componentes de autenticación y autorización para el acceso a recursos Cloud. Estos componentes se basan en la plataforma Openstack y permiten securizar la interacción de los usuarios y dispositivos que intervienen en todo el ecosistema de FIWARE. Por su parte, en el capítulo de Cloud me encargo así mismo de coordinar e implementar un cliente Web para la gestión de recursos Cloud también basada en los componentes de Openstack.

Obviamente la experiencia adquirida en este proyecto ayuda de forma directa a mejorar los modelos de diseño que realizo en el campo de la videoconferencia ya que se trata siempre de un trabajo orientado a sistemas Cloud. Pero a parte de estas tareas despliego una versión de Licode que se utiliza en algunas de las sesiones de coordinación y diseminación que tienen lugar durante el proyecto. Sirve por tanto este despliegue como banco de pruebas de cara a extraer conclusiones y datos de uso que serán fundamentales para el desarrollo de mis propuestas.

5.4 Otros proyectos

Aparte de estos proyectos de índole nacional y europeo durante estos años de investigación he validado mis propuestas en algunos proyectos de menor envergadura con empresas privadas. En cada uno de estos proyectos ha sido necesario desplegar Licode en la versión correspondiente a

cada momento. Por lo tanto en los primeros se utilizaba la versión de MCU centralizada propuesta al principio y en los más recientes la versión distribuida con los mecanismos de reporte y decisión de asignación de recursos. Aunque en algunos de ellos, aun desplegándose esta última versión, las necesidades de demanda no exigían utilizar múltiples servidores por tratarse de pequeños prototipos orientados a grupos pequeños de usuarios.

En cualquier caso la experiencia en todos estos proyectos ha servido para ir evolucionando el proyecto Licode obteniendo la retroalimentación de cada escenario concreto. El último de estos proyectos, en el que me encuentro inmerso en la actualidad, pretende dar servicio a varias decenas de miles de usuarios, agrupados en pequeñas salas de videoconferencia. Este caso supone un reto importante para probar las limitaciones de la arquitectura distribuida. Pero puesto que se encuentra en desarrollo no dispongo aún de conclusiones relevantes que incluir en este documento.

5.5 Divulgación de resultados

5.5.1 Publicaciones

Revistas internacionales indexadas

- Pedro Rodríguez, Álvaro Alonso, Joaquín Salvachúa and Javier Cerviño. *Materialising a new architecture for a distributed MCU in the Cloud*. Computer Standards & Interfaces, Volumen 44, páginas 234–242. 2016.

Revistas internacionales no indexadas

- Pedro Rodríguez, Álvaro Alonso, Joaquín Salvachúa, Enrique Barra and Javier Cerviño. *Cross-device Videoconferencing based on Adaptive Multimedia Streams*. JMPT, vol.4, no.1, páginas 14–24, 2013.

Congresos internacionales

- Álvaro Alonso, Ignacio Aguado, Joaquín Salvachúa and Pedro Rodríguez. *A Metric to Estimate Resource Use in Cloud-based Videoconferencing Distributed Systems*. En The 4th International Conference on Future Internet of Things and Cloud (FiCloud-2016), páginas TBA. IEEE, 2016.

5.5. DIVULGACIÓN DE RESULTADOS

- Álvaro Alonso, Pedro Rodríguez, Ignacio Aguado and Joaquín Salvachúa. *Enabling Resource Scheduling in Cloud Distributed Videoconferencing Systems*. En The Seventh International Conference on Cloud Computing, GRIDs and Virtualization (CLOUD COMPUTING 2016), páginas 51 -56 2016.
- Álvaro Alonso, Pedro Rodríguez, Joaquín Salvachúa and Javier Cerviño. *Dynamic media stream mobility with TURN*. En The 2nd International Conference on Future Internet of Things and Cloud (FiCloud-2014), páginas 15–22. IEEE, 2014.
- Pedro Rodríguez, Álvaro Alonso, Joaquín Salvachúa and Javier Cerviño. *dOTM: A mechanism for distributing centralized multi-party video conferencing in the cloud*. En The 2nd International Conference on Future Internet of Things and Cloud (FiCloud-2014), páginas 61–67. IEEE, 2014.
- Álvaro Alonso, Pedro Rodríguez, Joaquín Salvachúa and Javier Cerviño. *Deploying a multipoint control unit in the cloud: Opportunities and challenges..* En The Fourth International Conference on Cloud Computing, GRIDs and Virtualization (CLOUD COMPUTING 2013), páginas 173-178. 2013.
- Pedro Rodríguez, Álvaro Alonso, Joaquín Salvachúa, Enrique Barra and Javier Cerviño. *Adaptive cross-device videoconferencing solution for wireless networks based on QoS monitoring*. En World Congress on Computer and Information Technology (WCCIT), páginas 1–6. IEEE, 2013.

5.5.2 Docencia y seminarios

- *Introducción al Cloud Computing*. Proceso Formativo Cátedra Ce-Consulting. 2016.
- *Introducción a Amazon Web Services*. Máster Orange UPM. 2016.
- *Fundamentos de sistemas de Información para Big Data*. Máster Universitario en Ingeniería de Redes y Servicios Telemáticos. 2015 - 2016.
- *Centros de datos y provisión de servicios*. Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación. 2014 - 2016.

VALIDACIÓN Y RESULTADOS

- *Cómo desarrollar en 20 min una App de Videoconferencia avanzada.* TADHack Madrid 2015.
- *Escalabilidad de MCUs distribuidas.* MeetUp WebRTC Madrid. 2015.
- *Introducción a WebRTC y Licode.* Escuela Superior de Diseño de Madrid. 2013.
- *Comunicaciones en tiempo real desde tu navegador.* MeetUp MadridJS. 2012.

Capítulo 6

Conclusiones

Llegados a este punto es hora de tratar de extraer conclusiones acerca de las propuestas que he ido enunciando a lo largo de este trabajo. Comenzaba el texto realizando un análisis de la situación actual de los servicios y aplicaciones de videoconferencia y de su evolución durante los últimos años.

El auge de nuevas tecnologías que habilitan el uso de estas aplicaciones en dispositivos personales como ordenadores portátiles, tabletas o teléfonos móviles hace que el número de usuarios que los demandan sea cada vez mayor. De esta forma, servicios que dada su complejidad y gran consumo de recursos habían sido históricamente utilizados en entornos muy concretos y en la mayoría de los casos con infraestructuras hardware dedicadas, ahora están presentes en la vida cotidiana de muchos usuarios.

Surge un reto por tanto para los proveedores de estos servicios, que tratan de atender esta demanda creciente y variable en un entorno que no responde a los parámetros de configuración tradicionales válidos para otro tipo de sistemas. El Cloud ofrece una serie de ventajas que permiten adaptar los recursos que se ofrecen en el despliegue de forma dinámica ahorrando costes y mejorando la eficiencia general del servicio. Sin embargo existen una serie de retos que hacen que este tipo de despliegues no sean triviales.

Y son estos retos los que motivan las preguntas que planteaba en la introducción y sirven como punto de partida para comenzar a diseñar las contribuciones de esta tesis. Ahora que hemos recorrido el camino necesario intentaré darles respuesta basándome en los contenidos aprendidos en este texto.

¿Cómo debe diseñarse un sistema de multiconferencia Cloud para escalar de forma dinámica a la variación en la demanda?

El primer paso para contestar a esta pregunta es el de realizar un análisis de las ventajas que plantea el despliegue de un sistema de videoconferencia en una infraestructura Cloud. En

concreto hablamos del servidor central de gestión de flujos multimedia, la MCU. Pero estas ventajas implican también una serie de retos que hay que resolver y en base a los cuales obtengo respuesta a esta primera cuestión.

El estudio de estas ventajas y retos lo he realizado en la Sección 3.2. Como hemos visto, un despliegue de MCUs en infraestructuras Cloud ofrece una flexibilidad de cara a la escalabilidad del sistema que es sin duda fundamental para lograr adaptar los recursos disponibles a la demanda variable por parte de los usuarios. Gracias a esta flexibilidad es posible además utilizar diferentes tipos de servidores dependiendo de la función que realice cada MCU y sus requisitos de rendimiento concretos. Por último, el despliegue de MCUs en diferentes zonas geográficas gracias a la disponibilidad de la mayoría de proveedores Cloud públicos, ofrece la posibilidad de mejorar las condiciones de latencia y por tanto la calidad de las comunicaciones.

Pero aprovechar estas ventajas no es una tarea trivial y es necesario afrontar una serie de retos impuestos por las características particulares de los sistemas de tiempo real. El consumo de recursos físicos por una MCU gestionando una videoconferencia normalmente no puede estimarse en función del número de participantes o usuarios conectados. Se trata de un problema más complejo en el que es imprescindible tener en cuenta el consumo efectivo de cada servidor en cada momento. Si conseguimos caracterizar un sistema para poder extrapolar el consumo de recursos en base al tipo de sesiones y el número de usuarios podremos realizar de manera más eficiente la escalabilidad.

Por otra parte es importante tener en cuenta que las conexiones de usuarios a estas MCUs suelen ser permanentes ya que deben mantenerse abiertas durante todo el tiempo que dura una sesión de videoconferencia. Esto es un aspecto que influye directamente en la forma en que el sistema escala hacia abajo ya que es muy frecuente que frente a una bajada en la demanda de usuarios tengamos conexiones repartidas entre múltiples servidores haciendo que desaprovechemos una cantidad de recursos que de estar concentrados en un único servidor podríamos liberar. Surge la necesidad por lo tanto de realizar movilidad de flujos entre servidores pero, en un entorno de tiempo real donde la experiencia de usuario es crucial, la manera en que se realiza debe ser transparente para los participantes.

Este análisis sirve como contexto para saber cómo debe realizarse el diseño de un sistema de videoconferencia orientado a su despliegue y escalabilidad en el Cloud, que llevo a cabo en la Sección 3.3. Este diseño, validado en proyectos como MCS5 o LTextreme, se basa en la idea de

videoconferencia como servicio extraída de otros trabajos de investigación de mi grupo y cuyo componente clave es Nuve. A partir de este modelo propongo unas bases de diseño que son fundamentales a la hora de diseñar mi arquitectura:

- **Control centralizado:** para poder gestionar los recursos disponibles y escalar las MCUs en consecuencia, es necesario un módulo central de control que mantenga la información relativa a dichos recursos.
- **Balanceo de carga:** en base a los recursos disponibles en cada MCU, este componente central asigna una nueva conexión o participante a una u otra balanceando la carga en función de unos parámetros establecidos.
- **Actuador de escalabilidad:** en función de las necesidades de cada momento, este componente será el encargado de añadir o liberar recursos del sistema estableciendo comunicación con el proveedor Cloud correspondiente.
- **Monitorización de recursos disponibles:** para poder realizar el balanceo de carga así como para poder tomar la decisión de añadir o liberar recursos, es imprescindible tener constancia de los recursos disponibles y utilizados en cada momento. Esto es algo que se reporta desde cada una de las MCUs de manera periódica y se agrega en el componente central de control.

Este componente central se llama Cloud Handler y sirve como complemento a Nuve encargándose de las cuestiones relacionadas con la gestión de recursos y la escalabilidad de las MCUs. Consigo así un modelo en el que se puede adaptar de manera dinámica la cantidad de recursos disponibles a la demanda de usuarios.

¿Cómo y en base a qué criterios debe realizarse dicha escalabilidad?

La respuesta a esta pregunta se encuentra también en el Capítulo 3, entre las Secciones 3.3 y 3.4, donde propongo soluciones para escalar un sistema de videoconferencia hacia arriba y hacia abajo.

Es el módulo Cloud Hanlder el encargado de tomar la decisión de cuándo debe añadir o liberar recursos del sistema y lo hace en base a la información que recibe de cada MCU disponible. No hay una respuesta genérica a la pregunta de cómo debe realizarse la escalabilidad ya que ésta depende de las necesidades concretas de cada escenario. Sin embargo sí que hay unos criterios a tener en cuenta a la hora de tomar estas decisiones. Gracias al modelo propuesto proporciono las

herramientas necesarias para que estas decisiones puedan tomarse y para que pueda actuarse en consecuencia.

En este punto de la investigación y teniendo un paradigma de MCUs monolíticas en las que una misma sala de videoconferencia debe ser controlada por la misma MCU, las decisiones de escalabilidad deben tomarse en base a esta unidad, la sala. He propuesto por tanto un mecanismo de reporte en el que cada MCU comunica al componente de control su estado de disponibilidad en función del número de salas que está gestionando. Para ello debe haberse realizado una caracterización previa de una MCU modelo que nos permita extrapolar los resultados a otras MCUs con capacidades similares o proporcionales.

En base a este estado de disponibilidad, Cloud Handler establece una lista de prioridades para tomar la decisión de asignación de cada sala en un modo básico en el que siempre se utiliza la MCU que más disponibilidad tiene. Sin embargo habilito también la posibilidad de utilizar algoritmos personalizados que tomen estas decisiones en función de este estado de disponibilidad y de otros parámetros fijos de cada MCU, como su localización, el tipo de servidor, etc. De esta forma cada proveedor de videoconferencia podría tomar las decisiones atendiendo a las necesidades concretas de su escenario o servicio.

Aparte de las decisiones de asignación de salas, este modelo permite tomar la decisión de cuándo arrancar más MCUs o cuándo apagar alguna de las existentes basándose en los mismos reportes pero en este caso haciendo un balance de su agregado. El problema es que al tener excedente de recursos y necesitar apagar una de las máquinas disponibles, estos recursos no siempre estarán agrupados en las mismas MCUs. Surge la necesidad por tanto de agrupar las salas en un mismo subconjunto de MCUs, con la consecuente movilidad de las conexiones pertinentes.

En el caso de un sistema de tiempo real en el que la continuidad de la señal es fundamental para la buena experiencia de usuario, la movilidad de estas conexiones no es una tarea trivial. Existen algunas soluciones en la literatura pero no son aplicables a nuestro caso concreto, como demuestro en la Sección 3.4. Para solventar esta carencia propongo una extensión del protocolo MICE que permite realizar la movilidad sin pérdidas de paquetes.

¿Cómo debe diseñarse un sistema de videoconferencia distribuido para permitir la asignación inteligente de recursos?

Para responder esta pregunta realizo un análisis de un modelo de videoconferencia distribuido propuesto por un compañero de investigación y en cuyo diseño también participé. Este modelo, llamado dOTM, separa las diferentes capas que componen una MCU (control, señalización, media) para permitir su distribución entre múltiples servidores. La consecuencia más relevante de esta distribución es que se elimina la limitación de tener que gestionar una misma sala en el mismo servidor. Sin embargo, este modelo no contempla la asignación de recursos entre los servidores distribuidos.

Una vez realizado el análisis, en la Sección 4.3 propongo una extensión que habilita dicha programación de recursos. Para ello es necesario disponer de un mecanismo de reporte similar al que proponía anteriormente pero más avanzado ya que en este caso se añade una capa más de escalabilidad. Una nueva capa de control centralizada toma las decisiones en función de los datos de reporte permitiendo de nuevo la configuración de algoritmos de decisión personalizados.

Gracias a esta extensión, el modelo dOTM permite ahora desplegar OTMs (la nueva unidad de escalabilidad resultado de dividir en partes una MCU) de forma inteligente en los diferentes servidores distribuidos. Valido el modelo mediante una serie de experimentos que demuestran cómo puede reaccionarse de forma dinámica al estado de cada servidor y teniendo en cuenta también una serie de características fijas configurables a la hora de arrancarlos.

También hice un primer despliegue de este modelo en el proyecto EasyVideo para el Grupo Santander. En función del tipo de participante se elegiría una MCU desplegada en la infraestructura privada del banco o una MCU desplegada en una infraestructura pública. Obviamente utilizar los recursos propios del banco es más rentable ya que no supone un gasto adicional. Pero en situaciones en las que los usuarios son externos y no tienen acceso a las redes privadas de dicha infraestructura, un segundo conjunto de recursos serían utilizados bajo demanda.

¿Cómo y en base a qué criterios debe realizarse dicha asignación de recursos?

La respuesta a esta cuestión se encuentra en el trabajo realizado en las Secciones 4.4 y 4.5. Una vez diseñado el mecanismo de registro y reporte que permite la asignación inteligente de recursos

a los servidores de videoconferencia distribuidos, surge la evidente pregunta de cómo realizar esa asignación. Esto me lleva a repasar los criterios seguidos a la hora de realizar la distribución de carga en las MCUs monolíticas. Sin embargo ahora el problema es más complejo ya que la unidad de escalabilidad se ha reducido y no resulta tan fácil realizar la caracterización. Además, la experiencia me ha hecho querer ir un poco más allá y no basarme en caracterizaciones previas sino en una caracterización dinámica y actualizada en tiempo real a partir de los reportes recibidos.

Con esta idea en mente diseñé una métrica que permite estimar el uso de recursos por cada unidad OTM teniendo en cuenta el histórico de consumo en cada uno de los servidores distribuidos. Gracias a esta métrica tenemos una idea muy precisa de la cantidad de conexiones que un servidor determinado puede llegar a alojar y esta cantidad es diferente para cada tipo de conexión, en este caso en función del número de participantes de la sala a la que se conecta.

Esta métrica es por tanto una herramienta que permite realizar la asignación de recursos de una forma precisa y eficiente aprovechando al máximo las capacidades de cada servidor sin saturarlo de tal forma que el rendimiento será siempre óptimo. Para probar esta afirmación realicé un experimento que compara el mismo caso de programación de recursos en dos modalidades. La primera de ellas asignando los recursos en base al consumo de CPU de cada servidor y la segunda utilizando la métrica propuesta. Los resultados dejan claro el aporte significativo de esta solución.

El siguiente paso ha sido el de realizar una comparativa de algoritmos de decisión pensando en los casos de uso más típicos desde el punto de vista de un proveedor de videoconferencia como servicio. Probando el rendimiento de cada uno de estos algoritmos con la implementación de la métrica en Licode, llegué a la conclusión de que cada uno de ellos cubre las expectativas para las que ha sido diseñado. Por tanto y como cabía esperar cada escenario tendrá sus propias necesidades y aplicará sus propios algoritmos de decisión de acuerdo a ellas. Propongo en este punto una serie de coeficientes que ayudan a valorar la efectividad y eficiencia de estos algoritmos pudiendo sacar conclusiones sobre su idoneidad en cada caso de uso concreto.

6.1 Contribuciones

Una vez comprobado que el contenido propuesto durante este trabajo es válido para responder a las preguntas planteadas al inicio y que se cumplen por tanto los objetivos propuestos, en esta sección sintetizo las dos grandes contribuciones que esta tesis aporta al panorama de investigación en el terreno de la videoconferencia. Estas dos contribuciones coinciden con el trabajo desarrollado en

6.1. CONTRIBUCIONES

los dos capítulos centrales de este documento, los Capítulos 3 y 4:

- *Escalabilidad dinámica de sistemas de videoconferencia Cloud.*

En primer lugar he realizado un análisis de las oportunidades y retos que supone el despliegue de sistemas de videoconferencia en infraestructuras Cloud. Gracias a este análisis identifiqué qué características tiene que tener una arquitectura diseñada para su escalabilidad en este tipo de infraestructuras y qué condiciones hay que tener en cuenta a la hora de diseñarla.

Una vez claro el contexto procedo a diseñar una arquitectura que permita adaptar de forma dinámica los recursos de computación disponibles (en concreto la cantidad de MCUs) a una demanda de usuarios variable. Implemento esta arquitectura en el proyecto de código libre Licode y realizo un ejemplo de caracterización de una MCU que sirve para establecer los parámetros de escalabilidad en el modelo propuesto. Esta implementación se utiliza en proyectos como FIWARE o MCS5.

La última parte de esta contribución consiste en una extensión de la especificación MICE que permite realizar movilidad de flujos multimedia entre MCUs sin pérdidas de paquetes. Esto es posible gracias a una modificación en los mensajes STUN y la configuración de las asignaciones TURN propuestas inicialmente en dicha especificación.

- *Programación de recursos en sistemas de videoconferencia distribuidos.*

La primera parte de esta contribución consiste en la extensión de un modelo de MCU distribuida (dOTMs) para permitir la programación de recursos de forma inteligente entre los servidores disponibles. Para ello he diseñado un mecanismo de registro y reporte de las MCUs distribuidas así como un módulo de configuración de algoritmos personalizados para tomar decisiones de asignación en función de estos reportes y de información de contexto de cada MCU. El diseño de esta extensión se basa en un análisis previo de las características específicas de los sistemas de videoconferencia distribuidos.

A continuación propongo una nueva métrica que permite estimar el consumo de recursos que producirá una nueva conexión al ser asignada a una MCU distribuida. Para ello me baso en una regresión lineal que aprende del consumo histórico que han tenido las conexiones anteriores del mismo tipo en cada servidor distribuido.

Por último he realizado una comparativa entre un conjunto de algoritmos de programación de recursos adaptados a las necesidades concretas de varios escenarios. Para ello defino una serie de coeficientes que ayudan a analizar los resultados y valorar la idoneidad de cada algoritmo en función de los requisitos de despliegue.

He implementado todos los mecanismos propuestos en el proyecto Licode y realizado una serie de experimentos desplegando instancias de dicho software en proveedores de Cloud públicos para analizar su validez. También he utilizado esta implementación en proyectos como EasyVideo.

6.2 Trabajos futuros

A lo largo de este texto han ido surgiendo algunas ideas relacionadas con las propuestas que he realizado y que podrían dar pie a nuevos trabajos de investigación. A continuación propongo una serie de líneas que pueden resultar interesantes para estos nuevos trabajos o como continuación de los expuestos aquí.

- *Movilidad de conexiones entre agentes*: en la Sección 3.4 he propuesto un mecanismo de movilidad de flujos multimedia entre MCUs. Sin embargo, la distribución de salas entre varias MCUs conseguida gracias al modelo dOTMs hace que esta movilidad sea menos crítica en los despliegues. No obstante, el modelo de movilidad propuesto puede ser aplicado para trasladar OTMs de un agente a otro reagrupando así las conexiones.
- *Distribución geográfica*: los modelos de programación de recursos propuestos en el Capítulo 4 permiten asignar conexiones a una u otra MCU utilizando políticas personalizables. Una línea de investigación realmente interesante es la estudiar cómo tomar las decisiones de asignación en función de criterios geográficos. Como explicaba en la Sección 3.2.2 ésta es una de las grandes ventajas que ofrece el despliegue de sistemas de videoconferencia en infraestructuras Cloud. En el caso de los OTMs conviene también analizar en qué casos es mejor utilizar agentes cercanos a los clientes que publican o a los que se suscriben a los flujos multimedia. Además, modelos en los que se realiza comunicación entre agentes para el intercambio de estos flujos puede resultar de interés como vemos en [Cerviño 2013].

6.2. TRABAJOS FUTUROS

- *Parámetros avanzados de caracterización de OTMs*: la métrica propuesta en la Sección 4.4 tiene en cuenta el número de suscriptores de un OTM para estimar su consumo de recursos. En sesiones en las que la calidad de transmisión es variable, ésta puede influir en la cantidad de recursos necesarios. Por lo tanto, se abre la posibilidad de estudiar el uso de regresiones con múltiples variables para mejorar la métrica.
- *Algoritmos predictivos*: como comentaba en la Sección 3.2.2, utilizar algoritmos que predigan la necesidad de recursos en el sistema puede mejorar la eficiencia y evitar pérdida del servicio frente a subidas abruptas en la demanda. Esta predicción puede realizarse aprendiendo del uso y actividad del propio sistema u obteniendo datos de sistemas similares.
- *Algoritmos basados en costes*: cuando desplegamos MCUs en servidores de proveedores Cloud públicos, un factor clave a tener en cuenta es el coste económico de los recursos virtuales que utilizamos. Algunos de estos proveedores ofrecen sistemas de precios que fluctúan según la demanda, según la antelación con la que reservan o que incluso se basan en pujas. Algoritmos de decisión que tienen en cuenta estas fluctuaciones y aprenden de su evolución son otra posible línea de trabajo.

CONCLUSIONES

Bibliografía

- [Alonso 2013] A Alonso, P Rodríguez, J Salvachúa and J Cerviño. *Deploying a multipoint control unit in the cloud: Opportunities and challenges*. En CLOUD COMPUTING 2013, The Fourth International Conference on Cloud Computing, GRIDs, and Virtualization, páginas 173–178, 2013.
- [Alonso 2014] Álvaro Alonso, Pedro Rodríguez, Joaquín Salvachúa and Javier Cerviño. *Dynamic media stream mobility with TURN*. En The 2nd International Conference on Future Internet of Things and Cloud (FiCloud-2014), página TBA. IEEE, 2014.
- [Alonso 2016a] Álvaro Alonso, Ignacio Aguado, Joaquín Salvachúa and Pedro Rodríguez. *A Metric to Estimate Resource Use in Cloud-based Videoconferencing Distributed Systems*. En The 4th International Conference on Future Internet of Things and Cloud (FiCloud-2016), página TBA. IEEE, 2016.
- [Alonso 2016b] Álvaro Alonso, Pedro Rodríguez, Ignacio Aguado and Joaquín Salvachúa. *Enabling Resource Scheduling in Cloud Distributed Videoconferencing Systems*. En CLOUD COMPUTING 2016, The Seventh International Conference on Cloud Computing, GRIDs, and Virtualization, página TBA, 2016.
- [Andersen 2007] Per Andersen. *What is web 2.0?: ideas, technologies and implications for education*, volume 1. JISC Bristol, UK, 2007.
- [Anderson 2006] Chris Anderson. *The long tail*. Wereldbibliotheek, 2006.
- [Armbrust 2010] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica *et al.* *A view of cloud computing*. Communications of the ACM, vol. 53, no. 4, páginas 50–58, 2010.

- [Baugher 2004] M. Baugher. *The Secure Real-time Transport Protocol (SRTP)*. RFC 3711, RFC Editor, March 2004.
- [Bergkvist 2012] Adam Bergkvist, Daniel C. Burnett, Cullen Jennings and Anant Narayanan. *WebRTC 1.0: Real-time Communication Between Browsers*. Working draft WD, W3C, August 2012.
- [Bondy 1976] J.A. Bondy and U.S.R. Murty. *A graph theory with applications*. 1976.
- [Caron 2010] E. Caron, F. Desprez and A. Muresan. *Forecasting for Grid and Cloud Computing On-Demand Resources Based on Pattern Matching*. En *Cloud Computing Technology and Science (CloudCom)*, 2010 IEEE Second International Conference on, páginas 456–463, Nov 2010.
- [Cerviño 2008] Javier Cerviño, Pedro Rodríguez, Joaquín Salvachúa, Gabriel Huecas and Fernando Escribano. *Marte 3.0: Una videoconferencia 2.0*. En *VII Jornadas de Ingeniería Telemática JITEL 2008*, páginas 209–216. JITEL, 2008.
- [Cerviño 2011a] Javier Cerviño, Pedro Rodríguez, Irena Trajkovska, Alberto Mozo and Joaquín Salvachúa. *Testing a cloud provider network for hybrid P2P and cloud streaming architectures*. En *Cloud Computing (CLOUD)*, 2011 IEEE International Conference on, páginas 356–363. IEEE, 2011.
- [Cerviño 2011b] Javier Cerviño, Pedro Rodríguez, Irena Trajkovska, Fernando Escribano and Joaquín Salvachúa. *Leasing videoconference resources on hybrid cloud*. En Lizhe Wang, Rajiv Ranjan, Jinjun Chen and Boualem Benatallah, editores, *Cloud computing: methodology, system, and applications*. CRC Press, 2011.
- [Cerviño 2012] Javier Cerviño. *Contribution to multiuser videoconferencing systems based on cloud computing*. PhD thesis, Universidad Politécnica de Madrid, 2012.
- [Cerviño 2013] Javier Cerviño, Pedro Rodríguez, Irena Trajkovska, Fernando Escribano and Joaquín Salvachúa. *A Cost-Effective Methodology Applied to Videoconference Services over Hybrid Clouds*. *Mobile Networks and Applications*, vol. 18, no. 1, páginas 103–109, 2013.

BIBLIOGRAFÍA

- [Cho 2014] Hsin-Hung Cho, Chi-Yuan Chen, Hao-Wen Li, T.K. Shih and Han-Chieh Chao. *A fair cloud resource allocation using data envelopment analysis*. En Heterogeneous Networking for Quality, Reliability, Security and Robustness (QShine), 2014 10th International Conference on, páginas 31–36, Aug 2014.
- [Chou 2015] David C. Chou. *Cloud computing: A value creation model*. Computer Standards & Interfaces, vol. 38, no. 0, páginas 72 – 77, 2015.
- [Clark 1992] William J Clark. *Multipoint multimedia conferencing*. Communications Magazine, IEEE, vol. 30, no. 5, páginas 44–50, 1992.
- [Corici 2008] M.-I. Corici, A.F. Murarasu, S. Arbanowski, T. Magedanz, Suwon Lee and Xiaoyu Liu. *Multimedia Mobility Service Solution*. En Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th, páginas 1–5, 2008.
- [De Miguel 1994] Tomás P De Miguel, Santiago Pavón, Joaquín Salvachúa, Juan Quemada Vives, Pedro Luis Chas Alonso, Javier Fernandez-Amigo, Carlos Acuña, Lidia Rodríguez Yamamoto, Vasco Lagarto and Joao Vastos. *ISABEL Experimental distributed cooperative work application over broadband networks*. En Multimedia: Advanced Teleservices and High-Speed Communication Architectures, páginas 353–362. Springer, 1994.
- [Do 2011] A. V. Do, J. Chen, C. Wang, Y. C. Lee, A. Y. Zomaya and B. B. Zhou. *Profiling Applications for Virtual Machine Placement in Clouds*. En Cloud Computing (CLOUD), 2011 IEEE International Conference on, páginas 660–667, July 2011.
- [Firestone 2007] S. Firestone, T. Ramalingam and S. Fry. *Voice and Video Conferencing Fundamentals*, 2007.
- [Fraternali 2010] Piero Fraternali, Gustavo Rossi and Fernando Sánchez-Figueroa. *Rich internet applications*. Internet Computing, IEEE, vol. 14, no. 3, páginas 9–12, 2010.
- [Galán 2009] Fermín Galán, Americo Sampaio, Luis Rodero-Merino, Irit Loy, Victor Gil and Luis M. Vaquero. *Service Specification in Cloud Environments Based on Extensions to Open Standards*. En Proceedings of the Fourth International ICST Conference on

- COMmunication System softWAre and middlewaRE, COMSWARE '09, páginas 19:1–19:12, New York, NY, USA, 2009. ACM.
- [Ghose 2000] Debasish Ghose and Hyoung Joong Kim. *Scheduling Video Streams in Video-on-Demand Systems: A Survey*. Multimedia Tools and Applications, vol. 11, no. 2, páginas 167–195, 2000.
- [Golubchik 1996] Leana Golubchik, John C. S. Lui and Richard R. Muntz. *Adaptive Piggybacking: A Novel Technique for Data Sharing in Video-On-Demand Storage Servers*. ACM Multimedia Systems Journal, vol. 4, páginas 140–155, 1996.
- [Group 1996] Audio-Video Transport Working Group, H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. RFC 1889 (Proposed Standard), 1996. Obsoleted by RFC 3550.
- [Hagras 2003] T. Hagras and J. Janecek. *A high performance, low complexity algorithm for compile-time job scheduling in homogeneous computing environments*. En Parallel Processing Workshops, 2003. Proceedings. 2003 International Conference on, páginas 149–155, 2003.
- [Handley 1998] M Handley and V Jacobson. *RFC 2327. SDP: session description protocol*, vol. 10, 1998.
- [Hill 2010] Zach Hill, Jie Li, Ming Mao, Arkaitz Ruiz-Alvarez and Marty Humphrey. *Early Observations on the Performance of Windows Azure*. En Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10, páginas 367–376, New York, NY, USA, 2010. ACM.
- [Hindman 2011] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker and Ion Stoica. *Mesos: A Platform for Fine-grained Resource Sharing in the Data Center*. En 8th USENIX Conference on Networked Systems Design and Implementation, NSDI'11, páginas 295–308, Berkeley, CA, USA, 2011. USENIX Association.
- [Hobfeld 2012] Tobias Hobfeld, Raimund Schatz, Martin Varela and Christian Timmerer. *Challenges of QoE management for cloud applications*. Communications Magazine,

BIBLIOGRAFÍA

- IEEE, vol. 50, no. 4, páginas 28–36, 2012.
- [Hong 2009] Sungwoo Hong and Youjip Won. *Incorporating packet semantics in scheduling of real-time multimedia streaming*. Multimedia Tools and Applications, vol. 46, no. 2, páginas 463–492, 2009.
- [Ibarra 1977] Oscar H. Ibarra and Chul E. Kim. *Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors*. J. ACM, vol. 24, no. 2, páginas 280–289, Abril 1977.
- [ITU-T 1990] ITU-T. *ITU-T Recommendation: H.320: Narrow-band visual telephone systems and terminal equipment*, december 1990.
- [Jennings 2013] C. Jennings, T. Hardie and M. Westerlund. *Real-time communications for the web*. IEEE Communications Magazine, vol. 51, no. 4, páginas 20–26, April 2013.
- [Jung 2011] Gihun Jung and Kwang Mong Sim. *Agent-Based Adaptive Resource Allocation on the Cloud Computing Environment*. En Parallel Processing Workshops (ICPPW), 2011 40th International Conference on, páginas 345–351, 2011.
- [Koh 2007] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen and C. Pu. *An Analysis of Performance Interference Effects in Virtual Environments*. En 2007 IEEE International Symposium on Performance Analysis of Systems Software, páginas 200–209, April 2007.
- [Kristiansson 2004] J. Kristiansson and Peter Parnes. *Application-layer mobility support for streaming real-time media*. En Wireless Communications and Networking Conference, 2004. WCNC. 2004 IEEE, volume 1, páginas 268–273 Vol.1, 2004.
- [Lei 1994] Shaw-Min Lei, Ting-Chung Chen and Ming-Ting Sun. *Video bridging based on H.261 standard*. Circuits and Systems for Video Technology, IEEE Transactions on, vol. 4, no. 4, páginas 425–437, 1994.
- [Li 2010] Jiayin Li, Meikang Qiu, Jian-Wei Niu, Yu Chen and Zhong Ming. *Adaptive resource allocation for preemptable jobs in cloud systems*. En Intelligent Systems Design and Applications (ISDA), 2010 10th International Conference on, páginas 31–36, 2010.
- [Loreto 2012] S. Loreto and S. P. Romano. *Real-Time Communications in the Web: Issues, Achievements, and Ongoing Standardization Efforts*. IEEE Internet Computing, vol. 16,

- no. 5, páginas 68–73, Sept 2012.
- [Lu 2010] Wei Lu, Jared Jackson and Roger Barga. *AzureBlast: A Case Study of Developing Science Applications on the Cloud*. En Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10, páginas 413–420, New York, NY, USA, 2010. ACM.
- [Lu 2011] Xin Lu and Zilong Gu. *A load-adaptive cloud resource scheduling model based on ant colony algorithm*. En Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on, páginas 296–300, 2011.
- [Mahy 2010] R. Mahy, P. Matthews and J. Rosenberg. *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)*. RFC 5766, RFC Editor, April 2010.
- [Maltz 1998] D.A. Maltz and P. Bhagwat. *MSOCKS: an architecture for transport layer mobility*. En INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, volume 3, páginas 1037–1045 vol.3, 1998.
- [Mell 2011] Peter M. Mell and Timothy Grance. *SP 800-145. The NIST Definition of Cloud Computing*. Reporte técnico, Gaithersburg, MD, United States, 2011.
- [Mineno 2008] H. Mineno, M. Yoshida and T. Mizuno. *Quick transfer of media stream in FMC environment*. En Innovations in NGN: Future Network and Services, 2008. K-INGN 2008. First ITU-T Kaleidoscope Academic Conference, páginas 401–406, 2008.
- [O'reilly 2007] Tim O'reilly. *What is Web 2.0: Design patterns and business models for the next generation of software*. Communications & strategies, no. 1, página 17, 2007.
- [Ott 2006] Jörg Ott, Stephan Wenger, Noriyuki Sato, Carsten Burmeister and Jose Rey. *Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)", RFC 4585*. RFC 4585, 2006.
- [Qiang 2009] Yan Qiang, Yi Li and Junjie Chen. *The Workload Adaptation in Autonomic DBMSs Based on Layered Queuing Network Model*. En Knowledge Discovery and Data Mining, 2009. WKDD 2009. Second International Workshop on, páginas 781–785, Jan 2009.

BIBLIOGRAFÍA

- [Quemada 1996] Juan Quemada, Tomas de Miguel, Arturo Azcorra, Santiago Pavon, Joaquín Salvachúa, Manuel Petit, David Larrabeiti, Tomas Robles and Gabriel Huecas. *ISABEL: A CSCW Application for the Distribution of Events*. En *Multimedia Telecommunications and Applications*, páginas 137–153. Springer, 1996.
- [Reese 2009] George Reese. *Cloud application architectures: Building applications and infrastructure in the cloud*. O'Reilly Media, Inc., 2009.
- [Rodríguez 2009] Pedro Rodríguez, Daniel Gallego, Javier Cerviño, Fernando Escribano, Juan Quemada and Joaquín Salvachúa. *Vaas: Videoconference as a service*. En *Collaborative Computing: Networking, Applications and Worksharing*, 2009. CollaborateCom 2009. 5th International Conference on, páginas 1–11. IEEE, 2009.
- [Rodríguez 2013] Pedro Rodríguez, Álvaro Alonso, Joaquín Salvachúa, Enrique Barra and Javier Cerviño. *Cross-device Videoconferencing based on Adaptive Multimedia Streams*. JMPT, vol. 4, no. 1, páginas 14–24, 2013.
- [Rodríguez 2013] Pedro Rodríguez, Álvaro Alonso, Joaquín Salvachúa, Enrique Barra and Javier Cerviño. *Adaptive cross-device videoconferencing solution for wireless networks based on QoS monitoring*. En *Computer and Information Technology (WCCIT), 2013 World Congress on*, páginas 1–6. IEEE, 2013.
- [Rodríguez 2014] Pedro Rodríguez, Álvaro Alonso, Joaquín Salvachúa and Javier Cerviño. *dOTM: A mechanism for distributing centralized multi-party video conferencing in the cloud*. En *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*, páginas 61–67. IEEE, 2014.
- [Rodríguez 2016a] Pedro Rodríguez. *Contribución a las arquitecturas flexibles para la multiconferencia web adaptable a la demanda*. PhD thesis, Universidad Politécnica de Madrid, 2016.
- [Rodríguez 2016b] Pedro Rodríguez, Álvaro Alonso, Joaquín Salvachúa and Javier Cerviño. *Materialising a new architecture for a distributed MCU in the Cloud*. *Computer Standards & Interfaces*, vol. 44, páginas 234 – 242, 2016.

- [Rosenberg 2003] J. Rosenberg. *STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)*. RFC 3489, RFC Editor, March 2003.
- [Rosenberg 2008] J. Rosenberg, R. Mahy, P. Matthews and D. Wing. *Session Traversal Utilities for NAT (STUN)*. RFC 5389, RFC Editor, October 2008.
- [Rosenberg 2010] J. Rosenberg. *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols*. RFC 5245, RFC Editor, April 2010.
- [Sha 2004] Lui Sha, Tarek Abdelzaher, Karl-Erik Arzen, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky and Aloysious K. Mok. *Real Time Scheduling Theory: A Historical Perspective*. Real-Time Systems Journal, vol. 28, no. 2/3, páginas 101–155, 2004.
- [Srisuresh 1999] P. Srisuresh. *IP Network Address Translator (NAT) Terminology and Considerations*. RFC 2663, RFC Editor, August 1999.
- [Tickoo 2010] Omesh Tickoo, Ravi Iyer, Ramesh Illikkal and Don Newell. *Modeling Virtual Machine Performance: Challenges and Approaches*. SIGMETRICS Perform. Eval. Rev., vol. 37, no. 3, páginas 55–60, Enero 2010.
- [Turletti 1993] Thierry Turletti. *H. 261 software codec for videoconferencing over the Internet*. 1993.
- [Turletti 1996] Thierry Turletti and Christian Huitema. *Videoconferencing on the Internet*. Networking, IEEE/ACM Transactions on, vol. 4, no. 3, páginas 340–351, 1996.
- [Vaquero 2008] Luis M Vaquero, Luis Roderio-Merino, Juan Caceres and Maik Lindner. *A break in the clouds: towards a cloud definition*. ACM SIGCOMM Computer Communication Review, vol. 39, no. 1, páginas 50–55, 2008.
- [Vaquero 2011] Luis M. Vaquero, Luis Roderio-Merino and Rajkumar Buyya. *Dynamically Scaling Applications in the Cloud*. SIGCOMM Comput. Commun. Rev., vol. 41, no. 1, páginas 45–52, Enero 2011.

BIBLIOGRAFÍA

- [Wang 2010] Lizhe Wang, Gregor Von Laszewski, Andrew Younge, Xi He, Marcel Kunze, Jie Tao and Cheng Fu. *Cloud computing: a perspective study*. New Generation Computing, vol. 28, no. 2, páginas 137–146, 2010.
- [Willebeek-LeMair 1994] MH Willebeek-LeMair, Dilip D Kandlur and Z-Y Shae. *On multipoint control units for videoconferencing*. En Local Computer Networks, 1994. Proceedings., 19th Conference on, páginas 356–364. IEEE, 1994.
- [Wing 2013] D. Wing, P. Patil and T. Reddy. *Mobility with ICE (MICE)*. Internet-Draft draft-wing-mmusic-ice-mobility-05.txt, IETF Secretariat, Septiembre 2013.
- [Wu 2001] Min-You Wu and Wei Shu. *Optimal Scheduling for Parallel CBR Video Servers*. Multimedia Tools and Applications, vol. 14, no. 1, páginas 79–99, 2001.
- [Yu 1995] Philip S. Yu, Joel L. Wolf and Hadas Shachnai. *Design and analysis of a look-ahead scheduling scheme to support pause-resume for video-on-demand applications*. Multimedia Systems, vol. 3, no. 4, páginas 137–149, 1995.