J Netw Syst Manage (2017) 25:699–734
DOI 10.1007/s10922-017-9431-2

Elastic Resource Allocation Algorithms for Collaboration Applications

Rafael Xavier[1] · Lisandro Zambenedetti Granville[2] · Bruno Volckaert[1] · Filip De Turck[1]

Received: 7 June 2017 / Revised: 23 August 2017 / Accepted: 15 September 2017 /
Published online: 27 September 2017

Abstract Cloud technologies can provide elasticity to real-time audio and video (A/V) collaboration applications. However, cloud-based collaboration solutions generally operate on a best-effort basis, with neither delivery nor quality guarantees, and high-quality business focused solutions rely on dedicated infrastructure and hardware-based components. This article describes our 2-year of research in the EMD project, which targets to migrate a hardware-based and business focused A/V collaboration solution to a software-based platform hosted in the cloud, providing higher levels of elasticity and reliability. Our focus during this period was an educational collaboration scenario with teachers and students (locally present in the classroom or remotely following the classes). A model of collaboration streaming (e.g. network topology, codecs, stream, streaming workflow, software components) is defined as base for software deployment and preemptive VM allocation techniques. These heuristics are evaluated using a version of the CloudSim simulator extended to generate and simulate realistic collaboration scenarios, to manage network congestion and to monitor a.o. cost and session delay metrics. Our results show that the algorithms reduce costs when compared to previously designed

✉ Rafael Xavier
rafael.xavier@ugent.be

Lisandro Zambenedetti Granville
lisandro.granville@inf.ufrgs.br

Bruno Volckaert
bruno.volckaert@ugent.be

Filip De Turck
filip.deturck@ugent.be

[1] IDLab, Department of Information Technology, Ghent University - imec, Technologiepark-Zwijnaarde 15, 9052 Ghent, Belgium

[2] Institute of Informatics Federal University of Rio Grande do Sul, Av. Bento Gonalves, Porto Alegre 9500, Brazil

approaches, having an effectiveness of 99% in meeting A/V collaboration setup deadlines, which is a stringent requirement for this collaboration application.

Keywords Cloud · Elasticity · Multicast · Media · Education

1 Introduction

Collaboration applications, usually offered as in-house conference solutions, are now available in the cloud (e.g. Google Hangouts [1], Facebook Messenger [2], WhatsApp [3]), and are widely adopted worldwide. Although they are accessible for most of the people and inherently scalable, these solutions rely on best-effort mechanisms to provide their services, a risk that is not acceptable by some business-oriented industries and services (e.g. medical audio/video collaboration services, industrial and security control centers).

These risks are partially mitigated in frameworks such as Skype for business [4], Cisco Unified Communications [5], IBM SameTime [6], and Barco collaborative learning solutions [7], which are business-oriented solutions. However, besides performing with the expected quality, they rely on dedicated infrastructure, whose usual dependency on manual maintenance or locally installed hardware-based components compromises automated scaling and elasticity inherent to cloud applications.

The elastic media distribution (EMD) [8] project aimed to solve this by providing an elastic and reliable cloud-based audio/video collaboration platform focused (at first) on educational scenarios, as illustrated in Fig. 1. A teacher presents a class to local students, audio and video (A/V) of the class is streamed to remote and mobile students, and A/V of students is sent back for interaction application purposes. Platform software components are used to compose class material presentation and to provide audiovisual interaction/collaboration services with geographically distributed students who can dynamically join and leave a teaching session. In this context, software components (the green orbs) must be dynamically provisioned over the available infrastructure to provide support for collaboration flows. Provisioning decisions need to take into account current requirements e.g. session users, data center resource availability, resource proximity to endpoints, service level agreements (SLAs) and budget. The arrows in Fig. 1 represent data transfers between endpoints. In this case, all video from student cameras are being transmitted to the teacher classroom (the teacher is given an overview of enrolled students who live-stream their webcam to promote interaction) and the teacher, in turn, transmits classroom data to all students (student in focus, students composition, teacher/classroom camera, and course presentation).

As the platform implements collaboration workflows placed along a distributed infrastructure, the components must be modelled and allocated intelligently. Therefore, this article first describes components and their interconnection model used in the development of cloud-based software component allocation techniques. Considering that multiple data streams flow between endpoints and data centers,

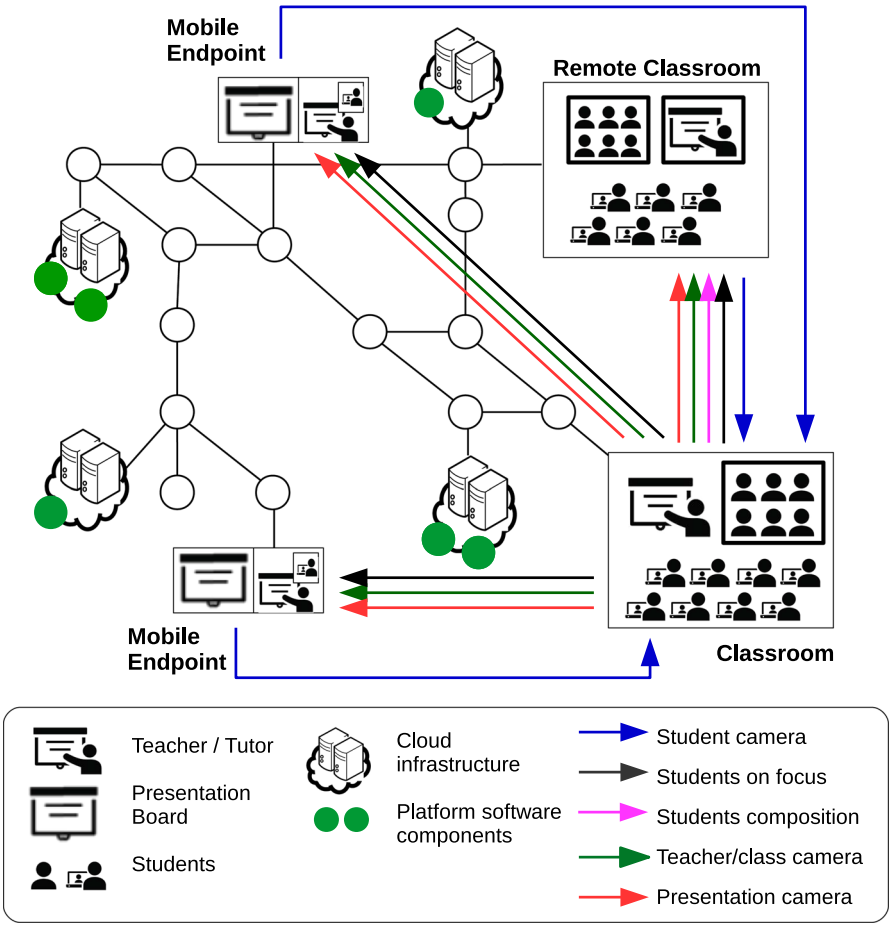

Fig. 1 Example of the distributed educational scenario studied in this article, including topology (interconnected white orbs), endpoints, cloud infrastructure, platform software components (green orbs), and A/V transmissions between users (arrows) (Color figure online)

many times spreading identical content over multiple endpoints, techniques to multicast data are needed. However, existing multicast technologies cannot be purely used for this purpose, since network streams must be transformed between source and destinations by, e.g. changing the A/V format, upscaling or downscaling it. In addition, network-level multicast is often blocked in client networks [9, 10]. Therefore, cloud-based resource provisioning algorithms proposed in this article aim to allocate software components over distributed infrastructure (cloud data centers), multicast and transform data flows (at application level, using specialized software components), and reduce resource usage and costs under a strict service level agreement (SLA) restriction (e.g. collaboration session joining time under 2 s).

Once the software component, workflow model, and provisioning techniques are defined (in which data center they will be put), cloud data centers must provide the

infrastructure to support them. Because of our stringent requirements imposed on the time required to join a collaborative session, most software components required for the cooperative application must be allocated proactively and hence one must be careful to not overspend on allocated resources when the system has no need for them. Thus, the proposed set of algorithms is completed by virtual machine (VM) allocation algorithms to intelligently and automatically decide on the trade-off between response time and resource usage cost to orchestrate the infrastructure setup and provision stand-by virtual machines (VMs).

The resource provisioning algorithms presented in this article, focused on workflow-embedded software components and optimal VM reservation methods, were presented in previous publications [11–14], but have been modified in this article to include an important new software component, the Compositor. As seen in Fig. 1, endpoint screens can host multiple picture-in-picture videos (streamed data). This is handled by the Compositor, which receives multiple data streams as input and composes them within a single screen. The addition of the compositor also changed the evaluation procedure, from the simulation of a pattern of individually scheduled streams to an endpoint-based approach, simulating when endpoints join or leave, as well as generating multiple streams for them based on a set of endpoint templates. Finally, results are provided about the impact of these changes on the algorithms performance by using an extended version of the CloudSim simulator [15], for the most prominent algorithms, tools and results achieved during the 2 years of research on the EMD project.

This article is organized as follows. In Sect. 2, we describe related work dealing with cloud resource allocation and stream distribution. Following this, definitions of components and workflows are detailed in Sect. 3, with component allocation techniques listed in Sect. 4 and VM reservation algorithms listed in Sect. 5. In Sect. 6, we present how the simulation scenarios were generated and how evaluations took place, detailing the used measurement metrics and the obtained results. In Sect. 7, we list our achievements and vision for the next years in this research field, pointing to future work. Finally, in Sect. 8, we describe our conclusions.

2 Related Work

The elastic and distributed nature of the evaluated problem, with teachers and students dynamically joining and leaving multiple educational collaboration sessions, requires a highly performant solution. Cloud infrastructure, along with service workflows orchestrated by custom resource allocation algorithms, can be employed to fulfill these requirements.

2.1 Cloud Infrastructure

The advent of cloud computing led to several cloud resource allocation research efforts [16], investigating scenarios ranging from a single cloud data center to multi-cloud deployments. However, several works do not include constraints related to the

user location relative to the cloud infrastructure [17–20]. In addition to similarities in their research when referring to elasticity and scalability, the scenarios investigated in this article essentially differ in that we investigate resource topologies composed of heterogeneous clouds with fixed-position components and aspects with regards to how (e.g. type of device, capabilities) and from what location end-users connect to the cloud infrastructure. For example, situations like a computer with a camera in-use in a scenario must be taken into account, cannot just be replaced by a generic cloud resource due to the need for that camera. Additionally, a low response time must be achieved, since project requirements state that 99% of the user requests must be answered under 2 s. This system response time is highly dependent on the performance of the interconnecting network and on the performance and accuracy of VM reservation algorithms (based on predictions of VM needs).

2.2 Network Architecture

The network infrastructure, in terms of bandwidth and performance, must be capable of providing the necessary sustained A/V quality. In network delay aware approaches, network status information is taken into account during the proposed algorithm evaluation. Virtual network embedding (VNE) [21] is a network approach to tackle cloud resource allocation problems. The idea is to supply a network layer substrate through mapping VMs into network nodes, and seeing VM interconnections as network links. Besides being useful to the present network requirements, such approach focuses on VM interconnections, and not on the connection from external clients to their required VMs. Furthermore, in our use cases, it is necessary to take into account physical/location limitations, as stated before.

Network functions virtualization (NFV) [22] entails the allocation, configuration, and chaining together multiple network functions to offer complex network services. To achieve the benefits promised by the NFV concept, an architecture to support this approach and a set of allocation algorithms employable to manage its virtual network resources are presented and evaluated by Clayman et al [23]. Moens et al. [24] refer to the NFV resource allocation problem as virtual network function placement (VNF-P) and propose a formal network-aware model to allocate resources for service chains which are provided using both virtual and physical network functions.

Although similarities exist between NFV-based management systems and our proposed approach, the former focus on generic network functions being allocated in a single administrative domain, while our approach focusses on specialized components in order to multicast, transform, and mix streaming media across multiple domains.

2.3 Multicast Approaches

The studied educational scenario comprises multiple sessions among teachers and students. Transmissions in this case are one-to-many (e.g. teachers presenting a class to several Students), which led to the investigation of multicast technologies.

Internet protocol (IP) Multicast technology reduces resource usage by distributing data in a one-to-many fashion, reaching several destinations through a unique network address [25–27]. However, design problems made it very difficult to achieve Internet-wide support [9, 10] and motivated researchers to climb from the IP layer to application layer overlay networks, implementing application layer multicast (ALM) [10].

Several ALM approaches were proposed before [28–32]. Although efficient in optimizing network resource usage, the existent technologies are unable to automatically and preemptively allocate components such as upscalers and downscalers to transform data between source and sink. This issue is exemplified in Fig. 2. Figure 2a shows a single-resolution streaming distribution, implementable through traditional multicast approaches to deliver 720p data to all nodes, while Fig. 2b illustrates a multicast scenario where components must be placed along the network to transform data between different formats. In this example, a 720p video is converted during transmission to be delivered in downscaled 480p and upscaled 1080p.

Considering that different streaming formats consume different amounts of bandwidth, an efficient placement of these intermediate transformer components can also be used to reduce network usage. Multicast technologies are therefore complementary to our approach, as they can be used to implement software components able to distribute content in an one-to-many fashion, but they cannot be used to implement more advanced end-to-end streaming workflows, as depicted in Fig. 2b.

2.4 Resource Allocation for Elastic Collaboration Tools

The resource allocation algorithms are responsible for the orchestration of the collaboration platform needs. Platform components and VM placement decisions (VMs instantiated and placed among the different cloud data centers) are made under strict SLA requirements i.e. users must be able to set up or join an A/V collaboration session rapidly.

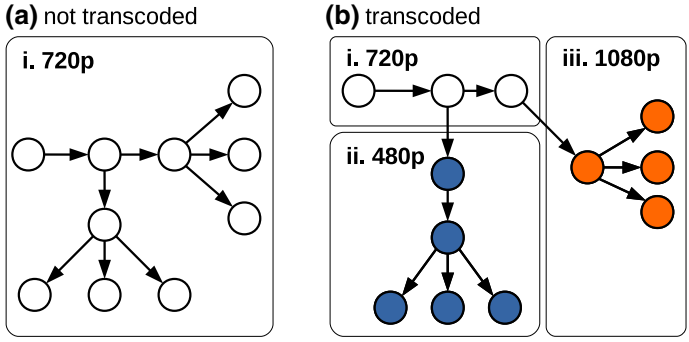

Fig. 2 Examples of streams being distributed using a single video format and multiple transcoded formats

Preliminary approaches that deal with A/V collaboration have been proposed. A similar educational A/V collaboration scenario has been studied, proposing software component allocation algorithms to support application-level multicast streaming [13, 14]. The same scenario was studied to model techniques of preemptive VM allocation, as these are necessary to host software components [11, 12] while proposing technologies to preemptively allocate VMs in pools distributed along the cloud network. All these initiatives employed a scenario generator able to provide us with realistic requirements for educational collaboration use case scenarios. However, these studies did not focus on the compositor component, which is key to implement endpoints able to flexibly receive, compose, and display multiple data streams simultaneously in a single screen. This article extends the scenario, component model, algorithms, and the streaming pattern generator to include the compositor component, and evaluates the impact of such change over all combinations of the proposed service component and VM allocation techniques.

2.5 Simulation Frameworks

For the evaluation of the proposed scenarios, a simulator capable of simulating a multi-cloud infrastructure with support for network interconnections, is necessary. This simulator must be able to model VM placement and elastic media service workflows. NSGrid [33], NS-2 [34], and NS-3 [35] are robust solutions to simulate Grid computing and network environments, but they are not ready to represent the required cloud entities or workflow-based data transfers. GreenCloud [36] and DCSim [37] focus on green computing and cloud data centers, respectively, but do not focus on the network layer simulation.

Another option, CloudSim [15], is not, in its original form, able to completely simulate the network layer (more specifically network contention and bandwidth usage). However, it was extended to better simulate the network layer and support evaluating cloud resource allocation algorithms [11]. We use CloudSim and these extensions, but with modifications that enable the generation of compositor-enabled scenarios and collaboration patterns.

3 Requirements, Scenario and Stream Concepts

Dynamic and heterogeneous streaming transmission requests are expected when endpoints join the collaboration. The modelling of this scenario and of the platform that supports these A/V collaborations experience is in line with project requirements and relies on use case specific concepts such as employed A/V codecs and a number of software components used in encoding/transcoding/decoding the collaboration videos. These concepts are defined in this section.

3.1 EMD Project Requirements

The main goal of the EMD project is to investigate how professional (hardware-based) A/V systems can be migrated to scalable software-based cloud systems,

focusing on educational collaboration (at first). In EMD, out of scope of this article, also network solutions were studied that allow deploying an elastic media distribution platform across networks with different characteristics [38]. The combination of technologies should provide an online A/V collaboration platform capable of combining high quality with low delays, while ensuring high security and usability. In the scope of this article, the main requirements are:

Session join delay: a user joining an A/V collaboration session must take no longer than 2 s.

Set of available components, codecs and video resolutions: the set of codecs and video formats listed in Sect. 3.2 was provided by project partners, along with their respective bandwidth needs. As an important goal is to migrate from a hardware-based solution to a software component-based one, the set of components available to set up these A/V collaborations, their functionalities and resource consumption characteristics were also described.

Cost: cost must be kept in check (and resource allocation algorithms should attempt to minimise cost) but is secondary to meeting SLAs in our scenarios.

Cloud-based: cloud technology was used for elasticity (one of the drivers in migrating from hardware-based solutions to software based solutions). The pay-as-you-go business model, offered by the major cloud infrastructure vendors, drives the cost measurement.

3.2 Codecs and Data Formats

Video *codecs* are compression or decompression software or hardware that implements video standards. In this article, we define $C \leftarrow (c_1, \ldots, c_n)$ as the set of available codecs, which implement n different industry standards such as H.265 high efficiency video coding (HEVC), H.264 advanced video coding (AVC) and RAW.

Next to the codec choice, *data format* is defined as the combination of a codec with a screen resolution. Being w the screen width, h the screen height, and c the chosen codec, a data format d is declared as $d \leftarrow (c, w, h)$, and the set D of the n available data formats is defined as $D \leftarrow \{d_{c_1,w_1,h_1}, \ldots, d_{c_n,w_n,h_n}\}$. The maximum bandwidth bw used when transmitting a data format d is calculated as $bw_d \leftarrow \beta_c \times w \times h$, where $w \times h$ represents the video resolution and β_c a compression factor related to the used codec c.

Codecs and data formats are generic concepts. When applied to the dynamically generated A/V information provided by users or platform components, they standardize the generated data to content that the platform can process, stream and display.

3.3 Content and Stream Definition

Stream data is defined as the content provided by a platform endpoint or service and compressed using a data format. For instance, K_d is the stream data representation of a sourced content K using the data format d. Delving deeper into the stream data

flow, sources and destinations are endpoints or platform services placed in the cloud. Thus, *streams* are defined as transfers of a stream data K_d between a source S and a destination D (stream $s \leftarrow (S, D, K, d)$) and *stream workflow* is the set of all active streams being transmitted through the platform.

The stream workflow structure changes dynamically as the same content source can be used by several destinations, and is constantly adapting (in terms of network paths and software components in use) when new endpoints join or leave the collaborative session. These workflows are formed by multiple interconnected software subcomponents, discussed next.

3.4 Collaborative Streaming Workflow Subcomponents

Subcomponents are platform services responsible for ingesting, receiving, or processing streaming data. They receive an input data set and provide an output data set accordingly to their functional design (change of data and resolution, generation of new content, replication or consolidation of flows). The available subcomponents, detailed below, are show in Fig. 3.

A subcomponent S_k is defined by two sets of stream data entries: input $I_k \leftarrow \{i_1, \ldots, i_n\}$ and output $O_k \leftarrow \{o_1, \ldots, o_m\}$ and by an internal functionality $f_k(x) = y$ that describes how this specific component processes each input $x \in I_k$ in order to

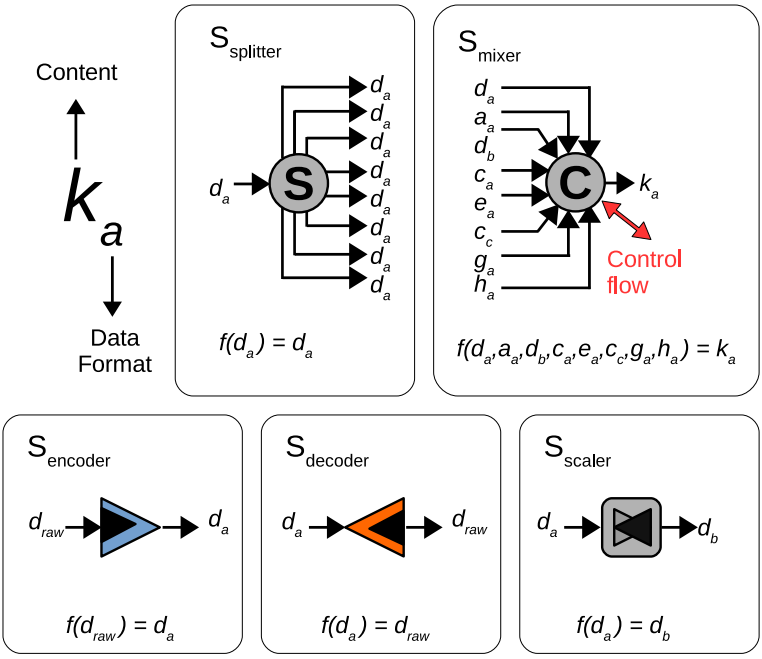

Fig. 3 Representation of a stream data k_a, composed by content k using the data format a, and of the available subcomponents: $S_{splitter}$ to distribute one received data to multiple outputs; $S_{composer}$ that receives several distinct inputs and composes them as one single output; S_{scaler}, which changes resolution or codec of a received data; $S_{encoder}$ and $S_{decoder}$ to encode and decode raw data, respectively

employ the projected functional behaviour ($y \in O_k$). Considering these definitions, a subcomponent S_k is described in Eq. 1

$$S_k \leftarrow (I_k, O_k, f_k). \tag{1}$$

As components process data (accordingly to the F_k internal functionality), they introduce a processing delay between receiving and providing the result data. This *incurred delay* is represented by id_{s_k} for a component s_k.

$S_{splitter}$: responsible for replicating one input into multiple outputs while preserving the received stream data (codec and resolution). Considering t content encoded using a data format $d \leftarrow (c, w, h)$ that uses a codec c to compress a $w \times h$ resolution video, $I_k \leftarrow \{t_d\}$ the input set with one entry, $O_k \leftarrow \{t_{d_1}, \ldots, t_{d_n}\}$ the output set containing n entries and f_k the internal function responsible for replicating the input content along all available outputs.

S_{mixer}: the S_{mixer} subcomponent is responsible for receiving multiple different inputs and joining them as a single and new output to be shown on a single screen (used later on to define the Compositor component). Different from other subcomponents, the internal function f expects interaction: a control flow must be received from an external entity, like a platform component or a user interface, in order to manage how the inputs will be selected and shown within the output screen area (possibly with image overlapping). A S_{mixer} subcomponent S_k is defined by an input $I_k \leftarrow \{a_1, b_2, c_4, c_1, b_1\}$, a new generated stream data t_n as the output $O_K \leftarrow \{t_n\}$ and a composition function f_k coordinated by the control flow.

S_{scaler}: responsible for transforming the resolution of received input data. A S_{scaler} S_k is defined considering a content c, two data formats $x \leftarrow (d_1, w_1, h_1)$ and $y \leftarrow (d_1, w_2, h_2)$, an input set $I_k \leftarrow \{c_x\}$, an output set $O_k \leftarrow \{c_y\}$ and a scaling function f_k.

$S_{encoder}$: this subcomponent is responsible for converting raw stream data (without compression) to a compressed format. It is defined considering a video codec c, two data formats $i \leftarrow (raw, w, h)$ and $o \leftarrow (c, w, h)$ two data formats, input data set $I_k \leftarrow \{i\}$ and output data set $O_k \leftarrow \{o\}$.

$S_{decoder}$: this subcomponent does the opposite as the $S_{encoder}$: decodes an encoded stream to a raw format. Compared to the $S_{encoder}$ definition, the input set is defined as $I_k \leftarrow \{o\}$, output set as $O_k \leftarrow \{i\}$, and a decoding function f_k is employed instead of an encoding one.

Each subcomponent has a specific internal function, and to deliver a fully functional end-to-end data transmission (e.g. decoding, transcoding and encoding data again to deliver data transformation functionality), they must be interconnected.

3.5 Subcomponent Interconnection

Subcomponents are interconnected by the platform to compose streaming workflows. To allow a component to connect with another component, component input and output sets must share the same content and data format. For instance, considering two subcomponents S_x and S_y, being $a \leftarrow (c_a, w_a, h_a)$ an output of the

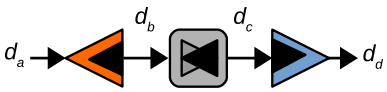

Fig. 4 An example of interconnection between subcomponents. In this example, a same data source d has a compressed version d_a, which is decoded to a raw one d_b, to be then scaled to d_c and encoded as d_d, forming a subcomponent chain capable of streaming transcoding

former and $b \leftarrow (c_b, w_b, h_b)$ an input of the latter, a can only be connected to b if $c_a = c_b$, $w_a = w_b$, and $h_a = h_b$. Figure 4 depicts an example of subcomponents interconnections.

An important consequence of interconnections is the incurred processing delay, as chains of subcomponents process stream data sequentially. Thus, for a chain composed of n subcomponents, Eq. 2 defines the resultant incurred delay of the chain as the sum of all subcomponents' incurred delay. Network delay is not taken into account as chained subcomponents are always placed in the same data centre

$$\sum_{i=1}^{n} (id_{S_i}). \tag{2}$$

Subcomponents, when interconnected, form so-called platform components. While subcomponents are always placed in the same data centre, components can be placed in a fully distributed manner, over the cloud infrastructure, implementing end-to-end transmissions among platform users at a higher abstraction level.

3.6 Collaborative Streaming Workflow Components

Components are high-level functionalities that are composed of basic service-based building blocks (the subcomponents). As illustrated by Fig. 4, a chain of subcomponents can implement streaming transcoding when interconnected. All available components, with definition and examples of decomposition, are shown on Fig. 5.

Encoder: employs one $S_{encoder}$ subcomponent to convert raw streaming data formats into compressed formats. Although made up of a single component, the component-subcomponent logic is kept to avoid mixing component and subcomponent layers and to preserve the flexibility of varying component definitions among different domains.

Decoder: inverting the Encoder's logic, employs one $S_{decoder}$ subcomponent to convert compressed streaming data formats to raw.

Transcoder: implements the transformation between two data formats by changing codec, resolution or both. Uses one $S_{decoder}$ and one $S_{encoder}$ to decompress and re-compress the content, with an optional S_{scaler} to change the resolution (upscaling or downscaling). Can imply data or quality loss when employing lossy, compression or scaling algorithms instead of lossless ones (redundant information removal techniques without loss of any information).

Tee: $S_{splitter}$ subcomponents are used to distribute the received data format to multiple outputs and compose the Tee component. As a $S_{splitter}$ is composed of one

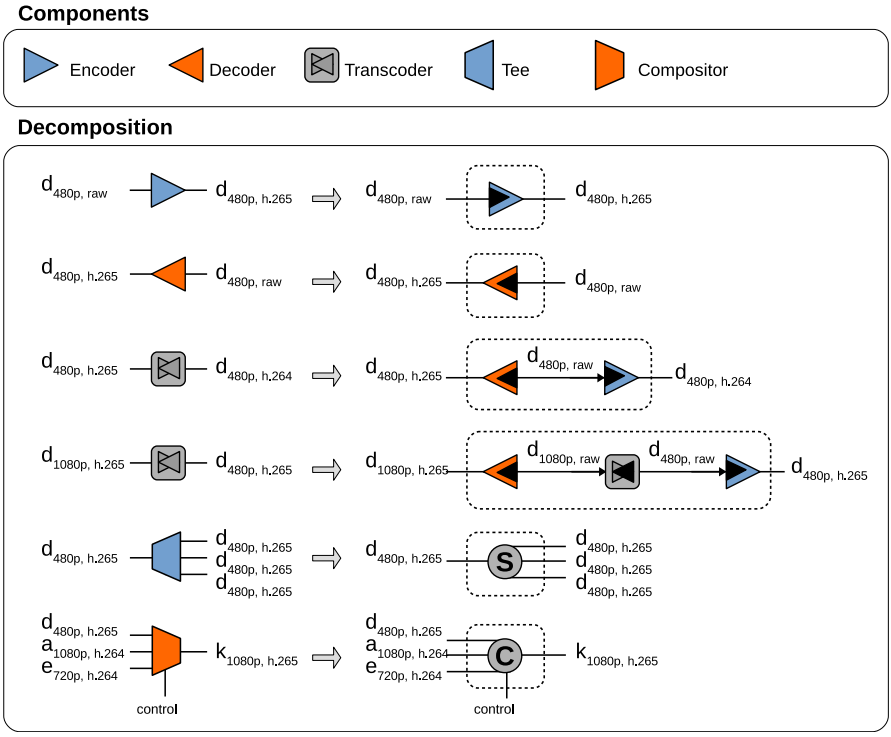

Fig. 5 Available components and an exemplification of their decomposition into subcomponents chains (using the HDTV 16:9 resolution notation as 480p, 1080p, among others, instead of *width* × *height*)

input and a fixed number of outputs (e.g. 4, 8 outputs), a cascade of $S_{splitter}$ components can expand this number and overcome the output number limitation (for instance, two $S_{splitter}$ components, with 4 outputs each, will result in 7 outputs when chained, as one output is used for interconnection). However, as subcomponents are chained, the incurred processing delay increases.

Compositor: the received inputs are (manually) organized in the output stream screen area, sometimes being resized or overlapping other input content. Differently from the Tee, in our model Compositors provide only one output and cannot be chained to expand the number of inputs.

Components are chained, composing workflows, and placed along the cloud infrastructure. The characteristics of the network topology that supports this infrastructure must be taken into account when placement decisions are made.

3.7 Network Topology

The network topology is composed of multiple interconnected nodes, as illustrated in Fig. 6. Endpoints and data centres are spread over the topology and are represented in this example by *a* to *f* and 1–6, respectively. All data centres are at all times accessible by all endpoints throughout the routed network. Considering the

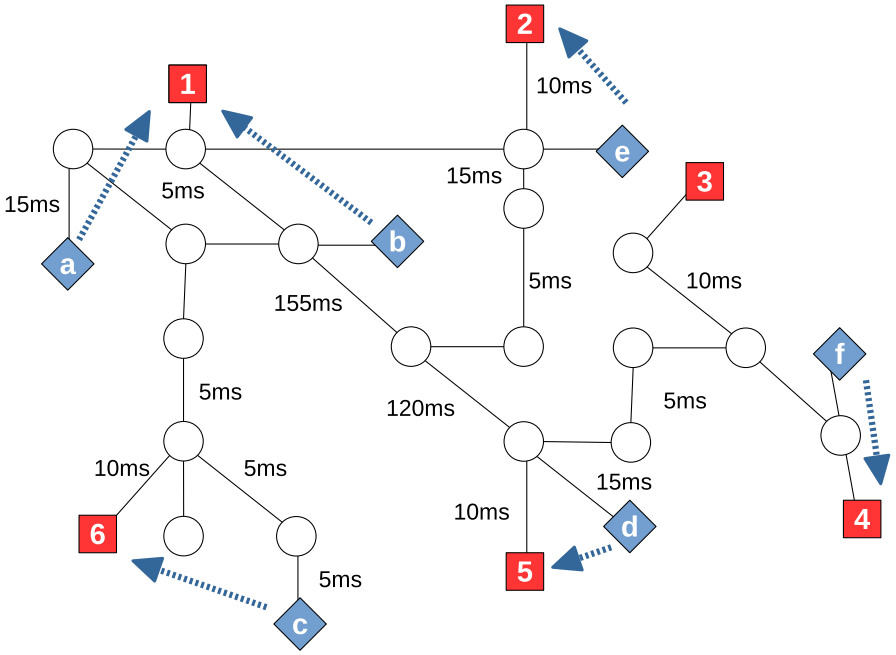

Fig. 6 Example of network topology: depicts several interconnected routers; six data centres, numbered from *1* to *6*; and six connected endpoints (from *a* to *f*). The blue arrows point to the elected datacenter for each endpoint, the respective best choices for component allocation due to lower latency (Color figure online)

summed network delay as distance, each endpoint has one data centre elected as the best candidate for hosting its components. A table sorting networking delay information is kept up to date by the platform with all distances among each endpoint/data centre pair to facilitate the service provisioning decisions. Additionally, as a component can be placed in a cloud data centre, but also within non-scalable pools of computational resources (non-cloud ones e.g. endpoints with some resources available for hosting an encoder/decoder/transcoder), this article names the possible placement spots as resource pools (RPs).

3.8 Component Chaining and Placement

When joining a collaboration session, endpoints request data stream transmissions. These transmissions are converted in collaboration workflows, a chain composed of multiple collaborative streaming workflow components, and deployed along the topology.

3.8.1 Workflow Composition

An example of a workflow (chained components) is depicted on Fig. 7. Encoders and Decoders are always placed at the workflows edges, but Tee's, Compositors, and Transcoders can be deployed wherever they are needed. This flexible

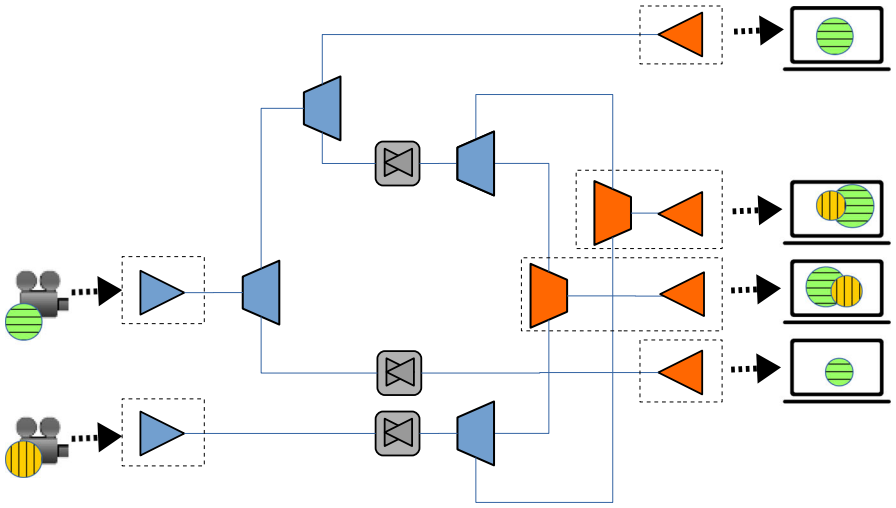

Fig. 7 Stream workflow example with two distinct A/V sources (two camera/encoder pairs), Tee's splitting the transmission in different paths, Transcoders, Decoders and two Compositors mixing both A/V sources over a single screen

component placement must be regulated by resource allocation algorithms taking into account optimization-driven design choices in compliance with service interconnection limitations (inputs and outputs placed for different contents or using different data formats cannot be interconnected), as explained in Sect. 3.5.

A chain of components, implemented by chained subcomponents, process the streamed data sequentially. Thus, for a chain composed between two components x and y, and formed by n interconnected components, with the network delay between two components a and b defined as $nd_{a,b}$, the incurred delay between x and y can be calculated as follows:

$$id_{S_1} + \sum_{i=2}^{n} \left(id_{S_i} + nd_{S_{i-1},S_i} \right) \tag{3}$$

Additionally, complexity is increased as workflows are not simply static, but dynamically adapted when dealing with new demands. Consider that the screens shown in Fig. 7 joined the workflow at different moments, from the highest endpoint joining initially up to the lowest endpoint joining last, Tee's were dynamically introduced to split the streams when needed.

The first step when placing components (before instantiating them within the cloud infrastructure) is the choice of where they will be connected to the existing stream workflow. An initial transmission/workflow must be set up if no compatible stream is found. Otherwise, an existing transmission must be replicated by a Tee if applicable, transcoded if needed, and then connected to the destination. Once defined, the workflow must be provisioned in the cloud infrastructure. The decisions

of which components should be used, where they should run, how to have stand-by VMs ready to run them, and how to interconnect them to come to the desired workflow are accomplished by the resource allocation algorithms detailed next.

4 Intelligent Component Allocation

The first step in allocating a stream is to define its component-based workflow and on which resource pool (RP) each chosen component will be placed. Figure 8a shows the first of three previous proposals [12], a baseline approach which employs an individual encoder, transcoder, and decoder sequence for each established stream, even if the same source is transmitted to multiple destinations. In a second approach, as multiple Encoders processing the same input data can be costly and overload the network, the number of encoders is reduced from 4 to 1, but an additional Tee is introduced (Fig. 8b).

The third and last component allocation technique (Fig. 8c) allows flexibly positioned Tee's and Transcoders over several locations. Such flexibility makes it possible to allocate Tee's in the destination RP, reducing the used bandwidth when multiple destinations are in the same RP, but increases the number of chained

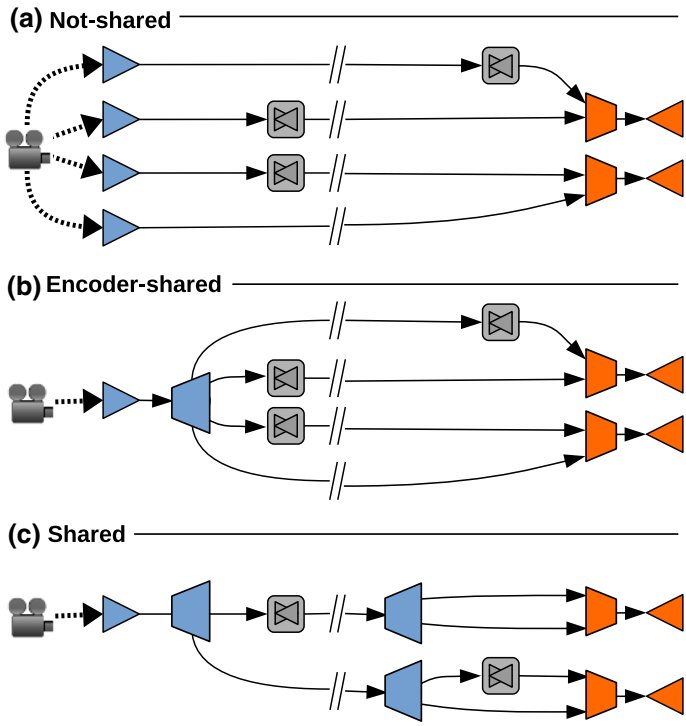

Fig. 8 Illustration of not-shared, encoder-shared and shared strategies applied to a transmission delivered to four decoders. The line interruptions represent cross-RP interconnections

components and requires the incurred processing delay to be measured and controlled. These three approaches are listed as algorithms using the auxiliary procedures and definitions described next.

4.1 Auxiliary Procedures and Definitions

Before listing the algorithms, some common procedures and parameters must be defined. Table 1 lists parameters and variables employed by all proposed algorithms. Two crucial functions are described in the table: $L(t, s, r)$ and $LN(t, s, r)$, which return subsets of the component set currently in use as base to search for optimized locations to allocate new Tee's. To be included in the $L(t, s, r)$ return subset, the component must be running at RP r, match the component type t (e.g. encoder, decoder, transcoder, compositor, or tee) and be processing the same data format carried through stream s. In contrast, $LN(t, s, r)$ differs only with regards to the data format: components must not be processing s stream's format. This $L()$ and $LN()$ data format dichotomy allows to first attempt allocations that avoid format conversions, which would require extra Transcoders

Table 1 Common functions for the proposed component allocation algorithms

Structure	Description
$src(s)$	Returns the source RP of a stream s
$dst(s)$	Returns the destination RP of a stream s
$new\,Encoder(y)$	Creates an encoder instance at RP y
$new\,Decoder(y)$	Creates a decoder instance at RP y
$new\,Compositor(y)$	Creates compositor at RP y
$new\,Transcoder(y)$	Creates a Transcoder at RP y
$new\,Tee(y, x)$	Creates tee with x outputs at RP y
$rp(c)$	Returns the RP which hosts component c
$inBw(c)$	Returns a bandwidth consumption approximation for the c component input based on its employed stream data format (Table 3)
$outBw(c)$	Analogous to $inBw(c)$, but for the output data format
$L(t, s, r)$	Return a list of components that are running at RP r, match the component type t (e.g. encoder, decoder, transcoder and tee) and are processing the same data format carried through stream s. If the return subset is empty, returns false
$LN(t, s, r)$	Differs from $L()$ only with regards to the data format: components must not be processing s stream's format
$getComp(s)$	Returns the compositor associated to the destination endpoint of an stream s, or false
$linkComp(C_0, \ldots, C_n)$	For components from C_0 to C_n, concatenate them as a ordered chain (e.g. $C_1 \rightarrow C_2 \rightarrow C_3$)

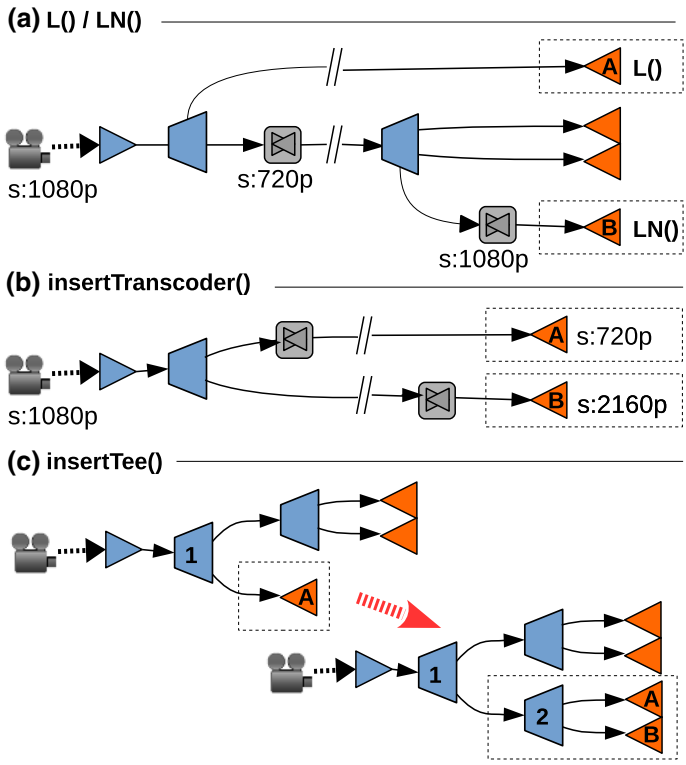

Fig. 9 Examples of **a** *L*() and *LN*() usage, picking locations for new decoders *A* and *B*; **b** *insertTranscoder*(), placing a Transcoder closer to the source or the destination to optimize bandwidth usage; **c** *insertTee*(), introducing a new Tee in a balanced way to accommodate a Decoder *B* (avoiding the additional processing delay of using more in-line components then necessary)

and consequently extra resources. Figure 9a depicts examples of *L*() and *LN*() usage to deliver a 1080p data stream: the former connects the decoder *A* near the encoder due to the need for picking up a 1080p data format, using more inter data center bandwidth, while the latter is more flexible and can choose for *B* a location closer to the destination. However, the location chosen for *B* provides 720p instead of 1080p, requiring an additional transcoding component.

Other functionalities are available to all algorithms: procedures to insert Transcoders and Tee's within the component chain are frequently employed by the algorithms, as shown in Procedures 1 and 2, respectively. The first, $insertTranscoder(s, i, o)$, receives the stream being allocated (s) and two components, an input component i and an output component o, between which a Transcoder shall be inserted if input and output data formats are not the same.

```
Data: s ← stream request being allocated
Data: i ← the input component, located before the transcoder
Data: o ← the output component, located after the transcoder
Procedure insertTranscoder(s,i,o)
    if outBw(i) > inBw(o) then
        transcoder ← new Transcoder(rp(i))
        linkComp(i, transcoder, o)
    else if outBw(i) < inBw(o) then
        transcoder ← new Transcoder(rp(o))
        linkComp(i, transcoder, o)
    else
        linkComp(i, o)
```

Procedure 1: Allocate a new Transcoder between two different components when needed; then, update the stream instance s with the new components and links between them.

Continuing, *insertTee*(s, c), shown in Procedure 2, searches for an available Tee (with at least one available unused output) that matches the s stream and can accommodate a new component c. If no available Tee is in place, a new one is inserted. If a Tee is available but fully used, a new $S_{splitter}$ is inserted to expand the capacity and provide a new allocatable output. Both procedures, *insertTranscoder*() and *insertTee*(), are exemplified in Fig. 9b, c, respectively.

```
Data: s ← stream being allocated
Data: T_s ← list of active Tee's related to stream s
Data: nearest(T_s): returns the Tee contained in T_s that is positioned nearest to the
      s stream Encoder.
Data: nearestAvailable(T): same as nearest(), but only for Tee's with at least one
      unused output.
Data: c ← new component being allocated on a Tee output
Procedure insertTee(s,c)
    tee ← nearestAvailable(T_s)
    if not tee then
        next ← nearest(T_s)
        prev ← prev(next)
        tee ← new Tee(rp(next))
        linkComp(prev, tee, next)
    linkComp(tee, c)
```

Procedure 2: Choose a Tee from an existing chained Tee set. If no tee is available, a new one is instantiated.

The third auxiliary procedure is *scheduleStream*(), which translates the component workflow to the subcomponent-based version ready for deployment on VMs and schedules it for placement. Such a task is illustrated in Fig. 5, which defines the decomposition of each component during the decomposition Component → Subcomponent.

4.2 Component Allocation Algorithms

The not-shared algorithm (NS), listed in Algorithm 3, implements the not-shared strategy presented in Fig. 8a. The main idea is to employ only Encoders, Transcoders, and Decoders, which means that no stream splitting will take place.

```
Data: s ← stream request being allocated
Procedure NS(s)
    encoder ← new Encoder(src(s))
    comp ← gc(s)
    if comp = false then
        comp ← new Compositor(dst(s))
        decoder ← new Decoder(dst(s))
        linkComp(comp, decoder)
    insertTranscoder(s, encoder, comp)
    scheduleStream(s)
```

Algorithm 3: The Not-Shared Algorithm (NS) receives a stream s to be allocated using only Encoders, Decoders and Transcoders.

The encoder-shared algorithm (ES), implementing the encoder-shared strategy described in Fig. 8b, is listed in Algorithm 4. To enforce the one-to-many multicast approach, Tee's are allocated: first, the algorithm looks for Tee's already processing the same A/V data in a specific video format. If not available, a Tee is allocated after the encoder.

```
Data: s ← stream request being allocated
Procedure ES(s)
    comp ← gc(s)
    if comp = false then
        comp ← new Compositor(dst(s))
        decoder ← new Decoder(dst(s))
        linkComp(comp, decoder)
    if L("Tee", s, src(s)) then
        tee ← insertTee(s, comp)
    else if LN("Tee", s, src(s)) then
        tee ← insertTee(s, comp)
        insertTranscoder(s, tee, comp)
    else if L("Encoder", s, src(s)) then
        tee ← insertTee(s, comp)
    else if LN("Encoder", s, src(s)) then
        tee ← insertTee(s, comp)
        insertTranscoder(s, tee, comp)
    else
        enc ← newEncoder(src(s))
        insertTranscoder(s, enc, comp)
    scheduleStream(s)
```

Algorithm 4: The Encoder-Shared Algorithm (ES) Algorithm allocates a stream s using Tee's placed at the source RP that multicast the Encoder's output.

The shared algorithm (SH), listed in Algorithm 5, implements the Shared strategy described in Fig. 8c. The logic is similar to the ES algorithm, except that more steps are employed to first try to allocate Tee's and Transcoders closer to the destination (in terms of network delay). This can lead to reduced bandwidth usage, but also to more chained Tee's and consequently longer processing delays.

```
Data: s ← stream request being allocated
Procedure SH(s)
    comp ← gc(s)
    if comp = false then
        comp ← new Compositor(dst(s))
        decoder ← new Decoder(dst(s))
        linkComp(comp, decoder)
    if L("Tee", s, dst(s)) then
        tee ← insertTee(s, comp)
    else if L("Decoder", s, dst(s)) then
        tee ← insertTee(s, comp)
    else if LN("Tee", s, dst(s)) then
        tee ← insertTee(s, comp)
        insertTranscoder(s, tee, comp)
    else if LN("Decoder", s, dst(s)) then
        tee ← insertTee(s, comp)
        insertTranscoder(s, tee, comp)
    else if L("Tee", s, src(s)) then
        tee ← insertTee(s, comp)
    else if LN("Tee", s, src(s)) then
        tee ← insertTee(s, comp)
        insertTranscoder(s, tee, comp)
    else if L("Encoder", s, src(s)) then
        tee ← insertTee(s, comp)
    else if LN("Encoder", s, src(s)) then
        tee ← insertTee(s, comp)
        insertTranscoder(s, tee, comp)
    else
        encoder ← newEncoder(src(s))
        insertTranscoder(s, encoder, comp)
    scheduleStream(s)
```

Algorithm 5: Shared Algorithm (SH) works similarly to the ES algorithm, except that Tee's can also be placed at the sink RP.

Component allocation algorithms decide where to place software components. After deciding where to instantiate a component, the chosen location must be able to host it in time, with no unacceptable delays when provisioning the required VMs. This task is accomplished by the VM provisioning algorithms.

5 VM Provisioning Algorithms

Supporting the component placement decisions, these heuristics aim to efficiently provision VMs to support the designed workflows under acceptable costs and under the employed SLA constraints. As detailed earlier in this article, a mandatory SLA

requirement for this application is the 2 s deadline for a user to join a professional media collaboration meeting, especially customers who historically have used dedicated rapid-response hardware solutions. Such behavior is considered essential in keeping the consumer satisfied. However, the cold start of a new VM is around 44–810 s [39], and several VMs can be reserved when deploying a workflow. In this scenario, when dealing with several users demanding resources periodically/seasonally or for longer periods, VM provisioning must be performed preemptively in order to not fail this constraint.

5.1 VM Pools

Each component placed runs in a VM hosted in the cloud. These VMs are requested from the destination RP after deciding on the component placement location. Thus, when requested, the RP picks one available VM in compliance with the required template and reserves it. To host these available VMs, every RP has a pool of (locally managed) VMs running and standby.

If a VM is available, it is immediately delivered. However, if unavailable, the requested RP starts the instantiation process of a new VM, holding the request until the VM is up and running and available in the VM pool. Thus, the algorithms presented in this section aim to predict the near-future VM demand, populate the VM pools in advance and avoid compromising SLA due to long VM instantiation times, all while trying to keep budget in check.

5.2 Request Rate-Based Algorithm (RATE)

To provision VMs beforehand, the request rate algorithm parametrizes and triggers the VM pre-allocation. The idea of this algorithm is to detect a VM request rate increase, based on the growth angle to predict the amount of VMs to keep available (Algorithm 6 shows the algorithm pseudo-code using Table 2 definitions). To implement it, a polling mechanism runs periodically and triggers a maintenance routine. On each polling cycle, it observes the amount of VM requests received in a past time window $now() - P_w$. The difference is used to calculate the growth angle between both moments. Finally, this growth angle is used to predict the VM demand for the next time window $now() + P_w$.

Data: $P_w \leftarrow$ time window to analyze for VM requests
Data: $P_i \leftarrow$ adjustment multiplier to the growth angle
Data: $P_m \leftarrow$ maximum number of VMs per user
Data: $P_u \leftarrow$ current user count related to this VM pool
Data: $P_l \leftarrow$ minimum VM lifetime
Data: $P_k \leftarrow$ minimum VM amount to keep for each image
Procedure `vmPoolPolling()`

```
for each image i ∈ I do
    count ← 0
    for each request r ∈ R do
        if i == image(r) then
            if (now() − P_w) < time(r) ≤ now() then
                count ← count + 1
    prediction ← tan(asin(count / ∛(count² + P_w²)) × P_i) × P_w
    if prediction > P_m × P_u then
        prediction ← P_m × P_u
    diff ← avail(i) − prediction
    if diff + avail(i) < P_k then
        diff ← P_k − avail(i)
    if diff > 0 then
        add(i, diff)
    else
        rem(i, diff, P_l)
```

Algorithm 6: The RATE Algorithm, which manages VM pools based on the incoming VM request rate.

As a VM can take several s/min to be ready for use, to reach a VM initialization time befitting the SLA constraints a stronger growth may be needed, which can be calibrated by the P_i angle multiplier parameter. Another parameter ($P_m \times P_u$) limits

Table 2 Common functions for the proposed VM allocation algorithms

Structure	Description
I	A set of VM images available to create new VMs
R	List of all VM requests received
V	List of all already deployed, running and ready to be used VMs
t	Interval in milliseconds to parametrize polling cycles
$image(r)$	Returns the image requested by a VM request r
$time(r)$	Returns the time when a request r was received
$now()$	Current clock in milliseconds
$avail(i)$	Available amount of VMs deployed over a image i
$add(i, c)$	Create c new VMs using a VM image i
$rem(i, c, l)$	Remove c available VMs that uses the image i and are older than a minimum lifetime l
initPolling()	Called when the system initializes, this procedure controls the polling mechanism by calling the procedure *vmPoolPollingcall*() every t milliseconds
vmRequest(i)	Return the next available VM deployed using a image i

the maximum amount of available VMs to avoid huge reservation spikes in case of a too steep angle. As a final algorithm parameter, P_k provides a configurable minimal amount of VMs to remain in the VM pool to attend initial requests, since this algorithm will not work properly when previously monitored information is still unavailable.

One potential behavior of this approach are spikes in number of unused and later removed VMs in the VM pool. One technique that partially remedies this situation is a history-based prediction of VM requirement patterns.

5.3 History-Based Algorithm (HISTORY)

This algorithm acts similarly to RATE, but predicts the necessary amount of VMs in a different way: by analyzing the (partial) VM request history. This algorithm, whose pseudo-code is listed in Algorithm 7 (using Table 2 definitions) looks back a parameterized number of weeks P_r to predict next week's behavior.

```
Data: P_s ← time window to analyze for VM requests
Data: P_r ← week count to look back
Data: P_a ← prediction adjustment
Data: P_p ← one week in milliseconds (seasonal period)
Data: P_l ← minimum VM lifetime
Data: P_k ← minimum VM amount to keep for each image
Procedure vmPoolPolling()
    for each image i ∈ I do
        prediction ← 0
        for w = 1 to P_r do
            count ← 0
            for each request r ∈ R do
                if i == image(i) then
                    min ← now() − P_p × w − P_s
                    max ← now() − P_p × w + P_s
                    if min < time(r) ≤ max then
                        count ← count + 1
            if count > prediction then
                prediction ← count
        diff ← avail(i) − prediction × P_a
        if diff + avail(i) < P_k then
            diff ← P_k − avail(i)
        if diff > 0 then
            add(i, diff)
        else
            rem(i, diff, P_l)
```

Algorithm 7: The HISTORY Algorithm, which analyzes the VM request history to keep VM pools properly populated.

Besides being able to deal with preexisting seasonal meeting patterns, the HISTORY algorithm shows predisposition to bad performance when dealing with single non-recurring requests. On the other hand, due to difficulties to handle initial requests (before having sufficient data to predict the usage growth), the RATE

approach requires a higher number of VMs to be kept constantly available, which raises the cost. Taking these limitations into account, a combined history and rate based approach is designed.

5.4 History and Rate-Based Algorithm (HISTORY+RATE)

This approach is similar to the already presented Algorithms 6 and 7, but combining both *vmPoolPolling*() routines in a single one, acting based on the higher predicted amount of VMs to be available. The combined routine, employing the same RATE and HISTORY parameters and Table 2 definitions, is shown in Algorithm 8.

```
Procedure vmPoolPolling()
    for each image i ∈ I do
        history ← 0
        for w = 1 to P_r do
            count ← 0
            for each request r ∈ R do
                if i == image(i) then
                    min ← now() − P_p × w − P_s
                    max ← now() − P_p × w + P_s
                    if min < time(r) ≤ max then
                        count ← count + 1
            if count > history then
                history ← count
        count ← 0
        for each request r ∈ R do
            if i == image(r) then
                if (now() − P_w) < time(r) ≤ now() then
                    count ← count + 1
        rate ← tan(asin(count / ∛(count^2 + P_w^2)) × P_i) × P_w
        if rate > P_m × P_u then
            rate ← P_m × P_u
        if rate > history then
            diff ← avail(i) − rate
        else
            diff ← avail(i) − history
        if diff + avail(i) < P_k then
            diff ← P_k − avail(i)
        if diff > 0 then
            add(i, diff)
        else
            rem(i, diff, P_l)
```

Algorithm 8: The HISTORY+RATE Algorithm, a combination of RATE and HISTORY approaches.

6 Results

We will describe now the simulation setup, the different quality metrics used to measure the algorithms' performance and the obtained results.

6.1 Scenario and Simulation Setup

Our collaboration scenario generation tool, developed taking into account industrial partners' feedback [11], was afterwards extended to use a normal distribution to generate stream requests and construct a realistic representation of the scenario under investigation [14]. However, this version handled each stream individually, while the compositor component, added in this article, requires an endpoint-wise stream management approach. Therefore, the session generation method was changed in the presented work to fit an endpoint-based approach: instead of scheduling streams, endpoints are programmed by the generator to join and leave sessions in normally distributed time frames. As templates define which streams are sourced or received by each kind of endpoint, a joining endpoint triggers its streams. These streams are all terminated when it ends the participation and leaves the session. The generator configuration is listed below:

Topology: for the experiments in this article, the generated and evaluated topology is composed of 100 network nodes and 5 distributed RPs (compliant with Fig. 6).

Data formats: the set of available data formats is shown in Table 3, each one with a predicted bandwidth utilization. The format choice, usually done through negotiation between source and sink, is done in this study by statically defining data formats (supplied as input to the simulator). Additionally, even though a lot of cameras can provide compressed video, we adopted RAW as basic input Audio/Video.

Subcomponents: the set of available subcomponents is shown in Table 4. Workflows are defined by matching components from this table, adding intermediate subcomponents if needed (e.g. if a stream is sent from a H.264/AVC 480p source to H.264/AVC 1080p destination, a H.264/AVC 480p → H.264/AVC 1080p S_{scaler} must be available to be employed inside a new Transcoder).

Table 3 Available data formats

Format	Bandwidth (Mbit/s)	Format	Bandwidth (Mbit/s)
240p H.264/AVC	0.5	240p RAW	64
360p H.264/AVC	1	360p RAW	127
480p H.264/AVC	2	480p RAW	254
720p H.264/AVC	6	720p H.264/AVC	552
1080p H.264/AVC	10	1080p H.264/AVC	1240
2160p H.264/AVC	30	2160p H.264/AVC	4980

Table 4 Component and subcomponent set (H.264/AVC)

Name	Input	Output	Delay (ms)	Name	Input	Output	Delay (ms)
$S_{encoder}$	RAW	240p	30	S_{scaler}	480p	240p	20
$S_{encoder}$	RAW	360p	30	S_{scaler}	480p	360p	20
$S_{encoder}$	RAW	480p	30	S_{scaler}	480p	720p	20
$S_{encoder}$	RAW	720p	40	S_{scaler}	480p	1080p	20
$S_{encoder}$	RAW	1080p	50	S_{scaler}	480p	2016p	20
$S_{encoder}$	RAW	1080p	50	S_{scaler}	720p	240p	20
$S_{decoder}$	240p	RAW	20	S_{scaler}	720p	360p	20
$S_{decoder}$	360p	RAW	20	S_{scaler}	720p	480p	20
$S_{decoder}$	480p	RAW	20	S_{scaler}	720p	1080p	20
$S_{decoder}$	720p	RAW	20	S_{scaler}	720p	2016p	20
$S_{decoder}$	1080p	RAW	20	S_{scaler}	1080p	240p	20
$S_{decoder}$	2016p	RAW	20	S_{scaler}	1080p	360p	20
S_{scaler}	240p	360p	20	S_{scaler}	1080p	480p	20
S_{scaler}	240p	480p	20	S_{scaler}	1080p	720p	20
S_{scaler}	240p	720p	20	S_{scaler}	1080p	2016p	20
S_{scaler}	240p	1080p	20	S_{scaler}	2016p	240p	20
S_{scaler}	240p	2016p	20	S_{scaler}	2016p	360p	20
S_{scaler}	360p	240p	20	S_{scaler}	2016p	480p	20
S_{scaler}	360p	480p	20	S_{scaler}	2016p	720p	20
S_{scaler}	360p	720p	20	S_{scaler}	2016p	1080p	20
S_{scaler}	360p	1080p	20	$S_{splitter}$	RAW	RAW	20
S_{scaler}	360p	2016p	20	S_{mixer}	RAW	RAW	20

Usage pattern: the simulation was parameterized with cyclic sessions within one uninterrupted month. A session starts at 09:00 of Tuesdays and Thursdays, lasting 8 h on average.

User parameterization: a single teacher lesson comprising one source locations and up to 100 normally distributed endpoints, simulating local endpoints (same locations as the Teacher), but also a dynamically scaling (remote) classroom and mobile users positioned along the edge. Considering that an endpoint can be used by more than one student, the total number of users will depend on the user/endpoint ratio but this has no impact on the presented results.

With regards to costs calculation, we chose as base the Amazon Elastic Compute Cloud (EC2) c4.8xlarge image [40], which offers 60 GB of memory and 36 virtual central processing units (VCPUs) (equivalent to 23.882 millions instructions per second (MIPS) [41]). The $1.675 per h cost was extrapolated to MB and millions instructions per second (MIPS) per hour values and used when calculating costs of differently sized VMs. Additionally, as components mostly operate utilizing random access memory (RAM) memory, hard disk storage usage was considered constant and was not taken into account when calculating cost.

Simulation infrastructure: this article shows results obtained using a Ubuntu 14.04 server hosted by a VMWare ESXi [42] VM with 8 virtual Intel Xeon E5-2630

Central Processing Unit (CPU) cores (2.3 GHz), 24 GB of RAM and 150 GB of allocated hard disk space.

6.2 Metric Set

Total VM cost ($): sum of costs for all employed VMs. For a simulated scenario composed of R resource pools, with V_r the full set of VMs running in a RP r, $C_{r,v}$ the cost of a VM v when running in a RP r, and with the time of start and shutdown of a VM v being given by S_v and E_v, respectively, the measured cost is calculated as:

$$\sum_{r=1}^{R}\sum_{v=1}^{V_r}(C_{r,v} \times (E_v - S_v)) \tag{4}$$

Successful join rate (%): industry partners agreed a 2 s collaboration meeting join deadline as critical to maintain a good user experience. This metric measures the compliance with this restriction. For a set S of streams, and with E_s and S_s the respective end and start time moments of a stream s, the adherence rate is provided by Eq. 5

$$\frac{1}{|S|}\sum_{s=1}^{S}\begin{cases}1, & if \quad (E_s - S_s) \leq 2s\\ 0, & if \quad (E_s - S_s) > 2s\end{cases}. \tag{5}$$

Incurred delay (ms): as each of the sequentially chained components can take a different time to process the streamed data, the impact of this aggregated transmission delay must be measured and evaluated. This task is accomplished by this metric, which sums up the incurred delay over all components of each stream (from encoder to decoder). For a set S of streams, being C_s the stream s component set and D_c the delay imposed by a component c, the measured value is provided by Eq. 6

$$\frac{1}{|S|}\sum_{s=1}^{S}\left(\frac{1}{|C_s|}\sum_{c=1}^{C_s}D_c\right). \tag{6}$$

Source RP bandwidth utilization (Mbit/s): with the teacher location as source RP, its uplink is a good spot to collect network bandwidth utilization (as it will exchange data with all students). This value allows comparing the bandwidth utilization incurred by the different proposed algorithms.

Algorithm runtime (ms): processing time between the request to join performed by a new endpoint and a decision of how to compose the stream workflow and where to allocate the chosen software components. Does not include component instantiation times.

6.3 Results

Considering the two different algorithm types proposed to manage components and VMs allocation, nine combinations of algorithms were evaluated: *SH–HR* as the component placement algorithm SH running on VMs managed by the

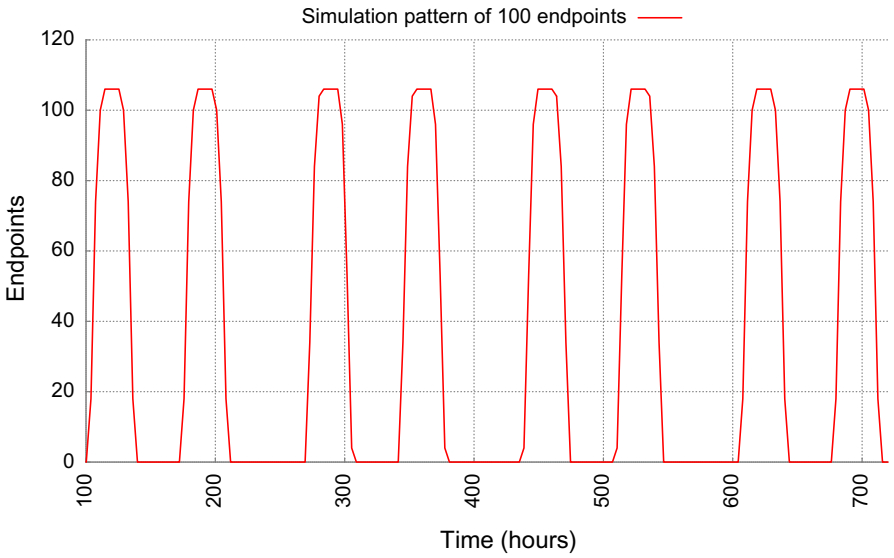

Fig. 10 Maximum volume of simulated endpoints joining and leaving the collaboration session (up to 100 endpoints)

HISTORY+RATE VM placement algorithm; *SH-HISTORY* as the component placement algorithm SH running on VMs managed by the HISTORY VM placement algorithm and *SH-RATE* as the component placement algorithm SH running on VMs managed by the RATE VM placement algorithm. In the same way, the component placement algorithms ES and NS were joined to HISTORY+RATE, HISTORY, and RATE, respectively providing the *ES–HR*, *ES-HISTORY*, *ES-RATE*, *NS–HR*, *NS-HISTORY* as *NS-RATE* combinations.

The simulated collaboration pattern is shown on Fig. 10 for the endpoints joining and leaving. These endpoints generate the stream volume depicted on Fig. 11, showing usage spikes every Monday and Thursday of a whole month, as modelled using the scenario generator. Compliance of the resource provisioning methods with the 2 s deadline in joining a collaborative meeting is shown in Fig. 12. SH algorithm shows a better performance when compared to ES and NS, as less components are instantiated during the session (the Tee usage optimizes the stream delivery). The need for less new components makes the SH-based algorithms a better option when distributing the same content to multiple destinations.

One of the factors that can potentially compromise our 2 s SLA compliance is the algorithm runtime, since component placement must wait for an algorithm decision to be taken. Figure 13 shows the time spent when running the algorithm, with a linear grow until reaching a maximum of 195 ms (ES–HR). A second factor is the delay incurred when extra in-line components are placed. Figure 14 shows this incurred delay average for each algorithm. While NS employs fewer sequential components, incurring a lower aggregated delay, SH and ES present a higher aggregated delay, a trade-off to be closely managed in order to guarantee adherence to industry requirements.

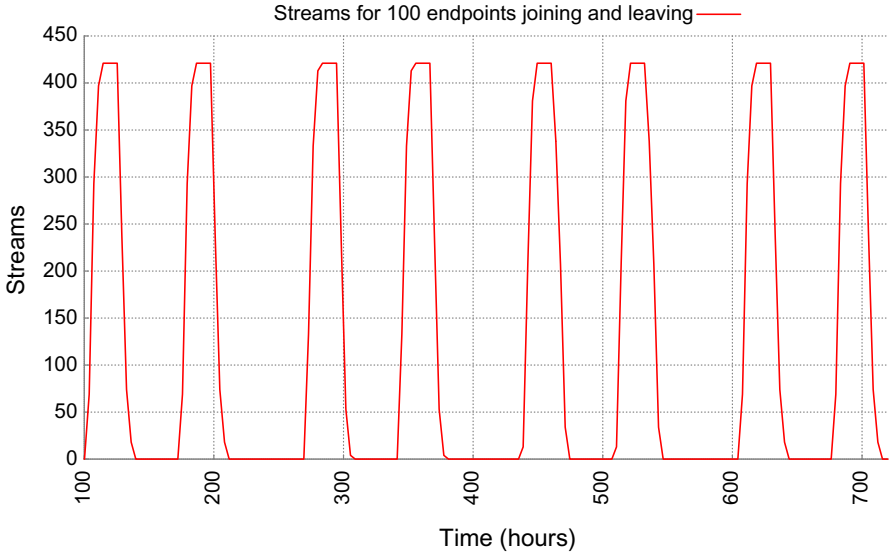

Fig. 11 Stream pattern when simulating 100 endpoints

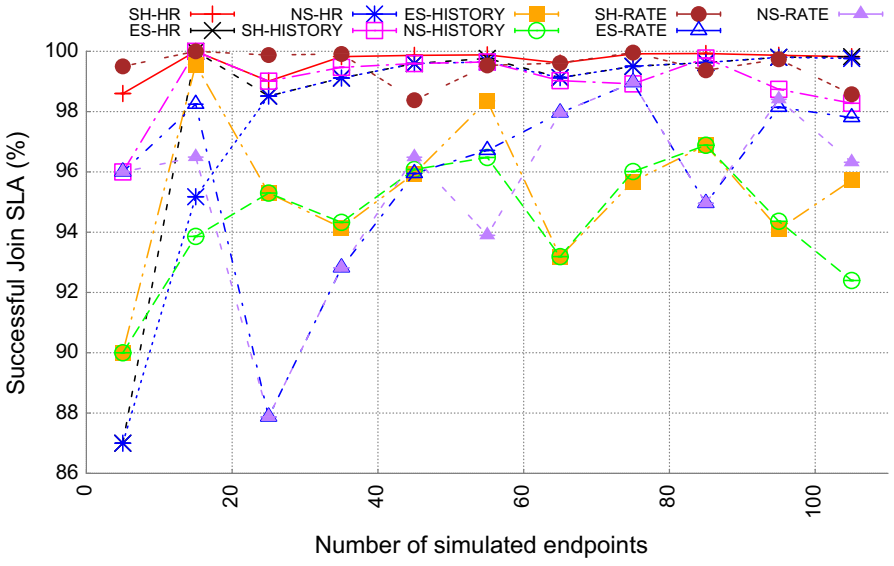

Fig. 12 Percentage of endpoints that join collaborative sessions in acceptable time, in compliance with the 2 s join time deadline

With regards to the user experience, bandwidth congestion, also a possible cause of delays, must be controlled. Figure 15 illustrates the bandwidth usage reduction induced by the SH multicast approach, 92.5 and 80% when compared to NS and ES-

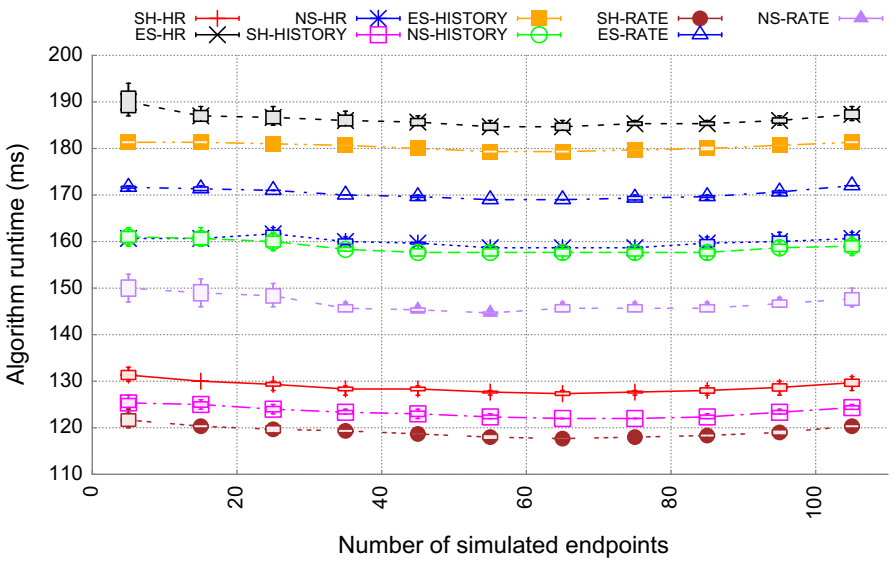

Fig. 13 Algorithms average runtime. An important part of the joining time, since the allocation can only occur after a placement decision is made

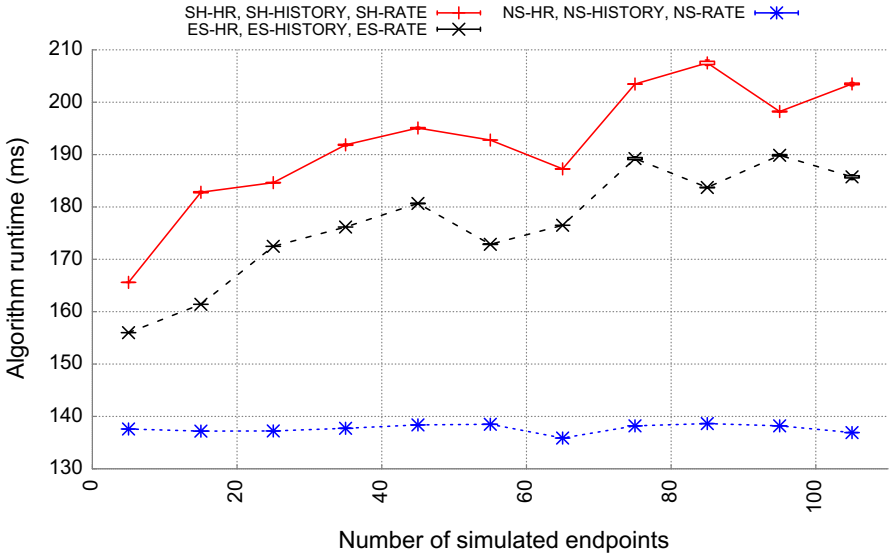

Fig. 14 Processing and network delay accumulated by multiple in-lined components: this delay is part of the joining time, as an endpoint is considered joined when the data is shown to the user for the first time

based algorithms, respectively. Although knowing that such drastic reduction is dependent on the scenario, with multiple destinations for a single source, we consider this promising because this type of scenario occurs commonly in educational collaboration use cases.

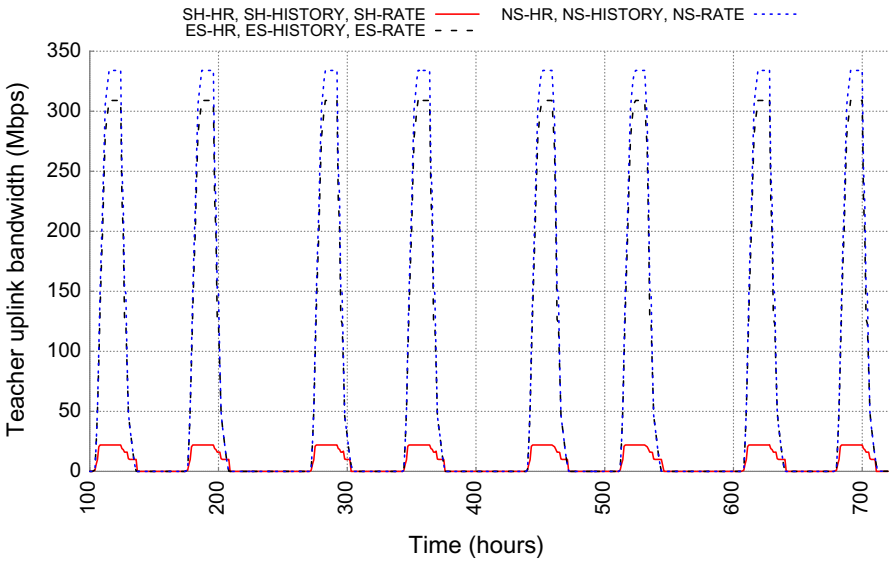

Fig. 15 Bandwidth measured in the teacher data center uplink, topology spot chosen to measure the data sent or received over Internet links to remote endpoints or data centers. Inter data center bandwidth usage measurements are not part of this graph

Looking at costs, Fig. 16 demonstrates the Total VM Cost measured for each algorithm. SH-based algorithms take advantage of splitting streams and employ fewer VMs in total, reducing total VM costs up to 66 and 33% when compared to algorithms that use NS and ES, respectively. This result, compared with the SLA compliance levels depicted by Fig. 12, raises SH as the better choice among SH, ES, and NS: acceptable time to join a collaboration session accompanied by lower total VM cost.

Even if not individually measured during the simulation, it is important to mention two other cost measure points: the costs related to bandwidth usage, whose reduction is shown in Fig. 15, and the cost related to larger incurred delays, as keeping VMs running longer than needed (stand by) increases the total VM cost. With regards to the VM provisioning approaches, Fig. 17 shows a different perspective for this growth in cost: the amount of VMs kept in stand-by on each VM pool. While algorithms based on ES and NS keep more VMs in the pool, it is more expensive due to the pay as you go data center business model, SH-based algorithms behave more efficiently.

Considering the presented results, the SH–HR method turns out having the better cost-benefit balance, significantly reducing cost and bandwidth usage when compared to the other proposed options for component or VM placement.

7 Evolution and Future Work

Summarizing the final results of the work in the EMD project on cloud-based resource provisioning for audio/video collaboration applications, this article has focused on virtualization and scalability requirements, developing centrally

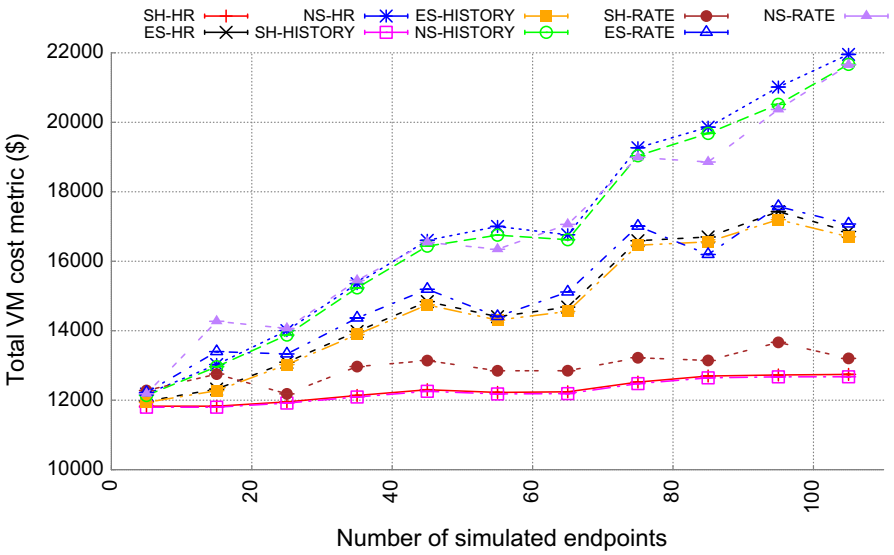

Fig. 16 View of the total VM cost per algorithm for the evaluated simulations

managed VM and software component placement techniques under strict SLA constraints. These needs originated from the virtualization of a static hardware-based collaboration platform, in order to come to an elastic cloud-hosted one.

Over the course of the research project, hardware components such as *Encoders*, *Decoders*, *Transcoders*, *Tee's*, and *Compositors*, all described in Sect. 3, were used as software components to be hosted by cloud VMs. Once hosted in the cloud, they were interconnected to implement streaming and interaction services among end users.

The hardware to software transition, when combined with the developed resource allocation techniques, enabled the platform to evolve from the previous statically dimensioned and hardware-based model to the automated deployment of the new software-based cloud platform. The research carried out during this hardware to software evolution raised possible future work avenues.

When dealing with collaboration users distributed over the Internet, several of them joining the platform from unanticipated locations and expecting to instantly be part of a collaborative session by the click of a button, a preemptive deployment of VMs was chosen. As VM instantiation delays can last several seconds (between 44 and 810 s depending on the operational system and the size of the VM) [39] and a single session requiring an orchestration components hosted in different cloud data centers, preemptive deployment of stand-by VMs along all the available cloud data centers were developed.

Our approach can be optimized by employing more agile container technology [43], as a replacement of the approach purely based on VMs. Containers can be started in a fraction of VMs cold start times [44], as the hosting server operational system is ready. This minimizes the need for standby VMs and encourages the

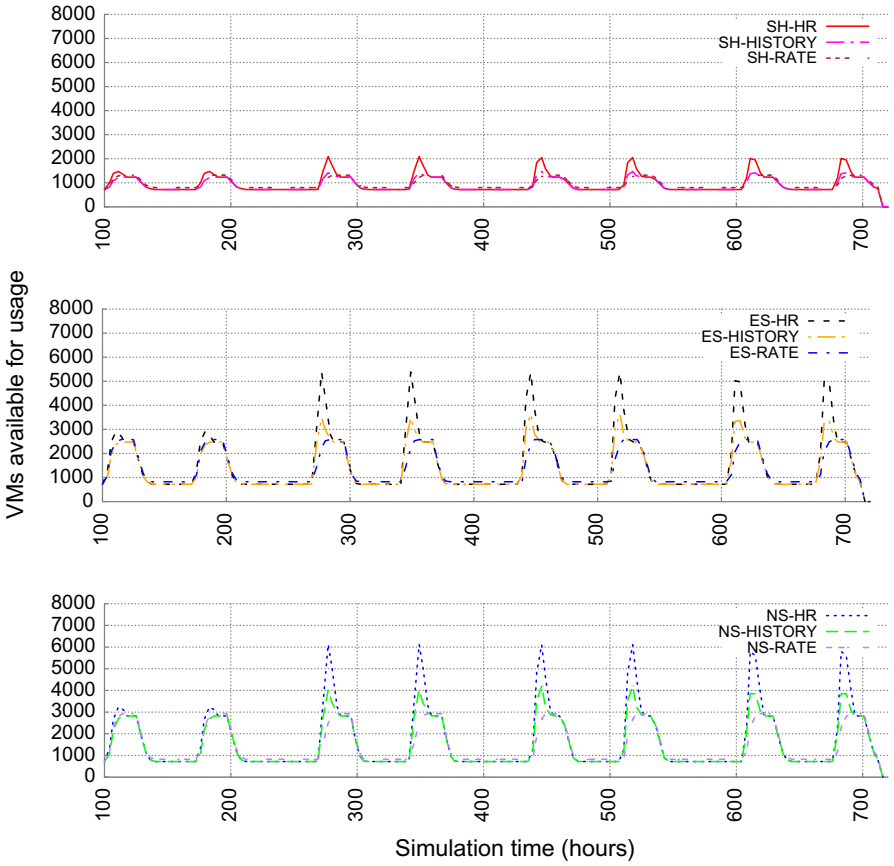

Fig. 17 Volume of VMs available in all VM pools for all types of VM images

adoption of cloud offer business models where one pays as he/she instantiates containers. This solution is expected to reduce VM costs while keeping an acceptable user experience.

Additionally, increased modularisation and automatic scaling with no centralised manager is becoming increasingly more important to deal with the amount of data going through this kind of Internet application. The evolution to a distributed management approach instead of the current centralized one, associated with stateless and container-based micro services, should allow us to attain higher scalability and resilience levels.

Analysing the software components used during this work, from a micro services perspective, all of them are standardized, composed of a full software stack and independent from the others. These highly cohesive modules are good candidates to be interconnected and compose loosely coupled system, which is an important factor to take into account when building microservices.

8 Conclusion

In this article, we have presented cloud provisioning algorithms and models tuned for professional collaborative media applications, and presented evaluation results of the proposed algorithms applied to a distributed educational collaboration scenario. The CloudSim simulator has been extended for this purpose, implementing the collaborative media application model, collecting valuable statistics, and comparing them with previously proposed approaches.

The SH–HR algorithm combination has been shown to provide the highest cost-benefit when compared to the other presented alternatives. It sets up collaboration meetings in acceptable time (under a stringent SLA, stating 2 s to join a meeting), execution time and delay, and under reduced VMs cost and bandwidth usage levels.

The results and models presented in this article are outputs of in the elastic media distribution (EMD) project, where industry, assisted by academic research groups, investigated the migration from highly reliable (but statically dimensioned and costly) A/V collaboration hardware solutions to equally reliable (but much more elastic/scalable) software/cloud-based solutions. We also delved briefly into our vision with regards to future trends and are investigating a migration to fully decentralized management (as opposed to the centralized approach taken here) and orchestration of container-hosted micro services as opposed to VM-based service instantiation.

Acknowledgements The research described in this article is partially funded by the iMinds Elastic Media Distribution (EMD) research project.

References

1. Google (2017) Google Hangout. https://hangouts.google.com/. Accessed: 2017-05
2. Facebook (2017) Facebook Messenger. https://www.messenger.com/. Accessed: 2017-05
3. WhatsApp (2017) WhatsApp Messenger and Collaboration Tool. https://www.whatsapp.com. Accessed: 2017-05
4. Microsoft (2017) Enterprise capabilities with Skype for Business in Office 365. http://products.office.com/en-us/skype-for-business/online-meetings. Accessed: 2017-05
5. Cisco (2017) Cisco Hosted Collaboration Solution (HCS). http://www.cisco.com/web/solutions/hcs/index.html. Accessed: 2017-05
6. IBM (2017) IBM Sametime: enterprise instant messaging, online presence indicators and community collaboration. http://www-03.ibm.com/software/products/en/ibmsame. Accessed: 2017-05
7. NV B.: Solutions for higher education and training (2017). https://www.barco.com/en/Products/Collaborative-Learning/Solutions-for-higher-education-and-training. Accessed: 2017-05
8. iMinds (2017) EMD project: elastic media distribution for online collaboration. http://www.iminds.be/en//projects/2015/03/11/emd. Accessed: 2017-05
9. Diot, C., Levine, B.N., Lyles, B., Kassem, H., Balensiefen, D.: Deployment issues for the IP multicast service and architecture. IEEE Netw. **14**(1), 78–88 (2000)
10. Hosseini, M., Ahmed, D.T., Shirmohammadi, S., Georganas, N.D.: A survey of application-layer multicast protocols. IEEE Commun. Surv. Tutor. **9**(3), 58–74 (2007)
11. Xavier, R., Moens, H., Volckaert, B., De Turck, F.: Design and evaluation of elastic media resource allocation algorithms using CloudSim extensions. In: 2015 11th International Conference on Network and Service Management (CNSM), pp. 318–326 (2015)

12. Xavier, R., Moens, H., Volckaert, B., De Turck, F.: Adaptive virtual machine allocation algorithms for cloud-hosted elastic media services. In: Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP. IEEE, pp. 564–570 (2016)
13. Xavier, R., Moens, H., Volckaert, B., De Turck, F.: Resource allocation algorithms for multicast streaming in elastic cloud-based media collaboration services. In: 2016 IEEE 9th International Conference on Cloud Computing (CLOUD). IEEE, pp. 947–950 (2016)
14. Xavier, R., Moens, H., Slowack, J., Sandra, W., Delputte, S., Volckaert, B., De Turck, F.: Cloud resource allocation algorithms for elastic media collaboration flows. In: 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). IEEE, pp. 440–447 (2016)
15. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Exp. **41**(1), 23–50 (2011)
16. Jennings, B., Stadler, R.: Resource management in clouds: survey and research challenges. J. Netw. Syst. Manag. **23**(3), 567–619 (2015)
17. Koslovski, G., Soudan, S., Goncalves, P., Vicat-Blanc, P.: Locating virtual infrastructures: users and InP perspectives. In: 2011 IFIP/IEEE International Symposium on Integrated Network Management (IM), pp. 153–160 (2011)
18. Alicherry, M., Lakshman, T.: Network aware resource allocation in distributed clouds. In: IEEE INFOCOM, pp. 963–971 (2012)
19. Steiner, M., Gaglianello, B.G., Gurbani, V., Hilt, V., Roome, W., Scharf, M., Voith, T.: Network-aware service placement in a distributed cloud environment. SIGCOMM Comput. Commun. Rev. **42**(4), 73–74 (2012)
20. Zhu, Y., Liang, Y., Zhang, Q., Wang, X., Palacharla, P., Sekiya, M.: Reliable resource allocation for optically interconnected distributed clouds. In: 2014 IEEE International Conference on Communications (ICC), pp. 3301–3306 (2014)
21. Fischer, A., Botero, J., Till Beck, M., de Meer, H., Hesselbach, X.: Virtual network embedding: a survey. IEEE Commun. Surv. Tutor. **15**(4), 1888–1906 (2013)
22. ETSI Industry Group (2013) Network function virtualisation NFV. http://www.etsi.org/technologies-clusters/technologies/nfv. Accessed: 2017-05
23. Clayman, S., Maini, E., Galis, A., Manzalini, A., Mazzocca, N.: The dynamic placement of virtual network functions. In: Network Operations and Management Symposium (NOMS), 2014 IEEE, pp. 1–9. IEEE (2014)
24. Moens, H., De Turck, F.: VNF-P: a model for efficient placement of virtualized network functions. In: Proceedings of the 10th International Conference on Network and Service Management (CNSM 2014), pp. 418–423 (2014)
25. Holbrook, H.W., Cheriton, D.R.: IP multicast: EXPRESS support for large-scale single-source applications. ACM SIGCOMM Comput. Commun. Rev. **29**, 65–78 (1999)
26. Steve, D.: Host extensions for IP multicasting. RFC1112 (1989)
27. Deering, S.E., Cheriton, D.R.: Multicast routing in datagram internetworks and extended lans. ACM Trans. Comput. Syst. (TOCS) **8**(2), 85–110 (1990)
28. Ruso, T., Chellappan, C., Sivasankar, P.: Ppssm: push/pull smooth video streaming multicast protocol design and implementation for an overlay network. Multimed. Tools Appl. **75**, 1–23 (2015)
29. Castro, M., Druschel, P., Kermarrec, A.M., Nandi, A., Rowstron, A., Singh, A.: SplitStream: high-bandwidth multicast in cooperative environments. ACM SIGOPS Oper. Syst. Rev. **37**, 298–313 (2003)
30. Banerjee, S., Bhattacharjee, B., Kommareddy, C.: Scalable Application Layer Multicast, vol. 32. ACM, New york (2002)
31. Venkataraman, V., Yoshida, K., Francis, P.: Chunkyspread: heterogeneous unstructured tree-based peer-to-peer multicast. In: Proceedings of the 2006 14th IEEE International Conference on Network Protocols, 2006. ICNP'06. IEEE, pp. 2–11 (2006)
32. Tran, D.A., Hua, K.A., Do, T.: Zigzag: an efficient peer-to-peer scheme for media streaming. In: INFOCOM 2003. 22nd Annual Joint Conference of the IEEE Computer and Communications, vol. 2, pp. 1283–1292. IEEE Societies, IEEE (2003)
33. Volckaert, B., Thysebaert, P., De Turck, F., Demeester, P., Dhoedt, B.: Evaluation of grid scheduling strategies through a network-aware grid simulator. In: PDPTA'03: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, vols. 1–4, pp. 31–35 (2003)

34. Issariyakul, T., Hossain, E.: Introduction to Network Simulator NS2, 2nd edn. Springer, Berlin (2011)
35. Riley, G.F., Henderson, T.R.: The NS-3 network simulator modeling and tools for network simulation. In: Wehrle, K., Güneş, M., Gross, J. (eds.) Modeling and Tools for Network Simulation, pp. 15–34. Springer, Berlin (2010). (Chap. 2)
36. Liu, L., Wang, H., Liu, X., Jin, X., He, W.B., Wang, Q.B., Chen, Y.: GreenCloud: a new architecture for green data center. In: Proceedings of the 6th International Conference Industry Session on Autonomic Computing and Communications Industry Session, ICAC-INDST '09, pp. 29–38. ACM, New York (2009)
37. Keller, G., Tighe, M., Lutfiyya, H., Bauer, M.: DCSim: a data centre simulation tool. In: 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), pp. 1090–1091 (2013)
38. Sahhaf, S., Tavernier, W., Colle, D., Pickavet, M.: Resilient availability and bandwidth-aware multipath provisioning for media transfer over the internet. In: 2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM), pp. 134–141. IEEE (2016)
39. Mao, M., Humphrey, M.: A performance study on the VM startup time in the cloud. In: 2012 IEEE 5th International Conference on Cloud Computing (CLOUD), pp. 423–430. IEEE (2012)
40. Amazon Inc. (2017) Amazon elastic compute cloud (EC2) images. http://aws.amazon.com/pt/ec2/instance-types/. Accessed: 2017-05
41. Carles Mateo (2015) Benchmarking the new Amazon C4 instances. http://www.cmips.net/tag/intel-xeon-e5-2666-v3-2-90ghz/. Accessed: 2017-05
42. VMWare (2017) VMWare vSphere ESXi Hypervisor. https://www.vmware.com/products/esxi-and-esx.html/. Accessed: 2017-05
43. Soltesz, S., Pötzl, H., Fiuczynski, M.E., Bavier, A., Peterson, L.: Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. ACM SIGOPS Oper. Syst. Rev. **41**, 275–287 (2007)
44. Seo, K.T., Hwang, H.S., Moon, I.Y., Kwon, O.Y., Kim, B.J.: Performance comparison analysis of linux container and virtual machine for building cloud. Adv. Sci. Technol. Lett. **66**(105–111), 2 (2014)

Rafael Xavier is a Ph.D. student at the Department of Information Technology of the Ghent University, imec, Belgium. He received his M.Sc. degree from the Institute of Informatics of the Federal University of Rio Grande do Sul, Brazil. His research interests include network management, software-defined networking, network functions virtualization, security, and optimization of cloud-based applications.

Lisandro Zambenedetti Granville is a professor at the Institute of Informatics of the Federal University of Rio Grande do Sul, Brazil. He is co-chair of the Network Management Research Group (NMRG) of the IRTF and president of the Brazilian Computer Society (SBC). His topics of interest include network management, software-defined networking, and network functions virtualization.

Bruno Volckaert is a professor of advanced programming and software engineering in the Department of Information Technology (INTEC) at Ghent University and senior researcher at imec. He obtained his Master of Computer Science degree in 2001, after which he investigated network aware Grid service management in his Ph.D. His current research deals with reliable and high performance distributed software systems for a.o. City-of-Things, UAVs, intelligent railway applications and autonomous optimization of cloud-based applications.

Filip De Turck is a professor at the Department of Information Technology of the Ghent University, imec, Belgium. His research interests include telecommunication network and service management, and design of efficient virtualized network systems. He serves as Chair of the IEEE Technical Committee on Network Operations and Management (CNOM), and is in the Editorial Board of several journals on network and service management.