A Cloud-based Multimedia Function

J-Ch. Grégoire and M. Amziab
INRS-EMT, CANADA
{gregoire,amziab}@emt.inrs.ca

***Abstract*—Two dominant trends of the Internet are the increasing importance of multimedia traffic, not only in the form of streaming videos but also for interactive communications, and the use of cloud technology to deploy services. In this paper, we look at the intersection of these trends and expose a number of considerations to help with the deployment of multimedia functions for interactive, mobile-adaptive and time-constrained applications in the cloud. We show how virtual servers can be CPU or bandwidth constrained and how to use them effectively.**

Keywords–Multimedia processing, Edge Cloud, IMS, mobile media services.

I. INTRODUCTION

Deployments of multimedia functions in a cloud are of interest for a number of reasons. First, it is a way to optimize services offered by operators, through economies of scale. Second, for many applications, it is a way to avoid end-to-end connectivity issues posed by middleboxes (e.g., network address translation boxes). More generally, it is a way to leverage third party service offerings, through outsourcing.

On the other hand, there are multiple ways to implement and exploit cloud technology, designated through different variants of platform, software or infrastructure as a service (PaaS, SaaS, IaaS, resp.), and we can wonder what is the best way to do such deployments to take full advantage of the scalability and flexibility offered by the cloud.

In this paper, we look at media from the perspective of a service infrastructure such as the IP Multimedia Subsystem (IMS), that is, a mobile-supporting media environment where control and processing are split, and control will be related to some signalling infrastructure. In the IMS world, one talks of a media controller and a media processor [1], [2], but we must point out that, although we adopt this decomposition and terminology, this work is in no way specifically tied to IMS. Doing such a split is interesting for a number of reasons but scalability comes naturally to mind, as the demands of media processing, especially video, will dwarf those of control processing.

The main challenge for the cloud deployment problem of a media function then becomes the dual issue of server placement, so as to avoid problems related with latency and general control of quality of service (QoS), especially in the presence of mobility, and of spreading the processing load across a number of processors, which is essentially a scheduling problem. This must be repeated for multiple instances of controllers, possibly for different customers, each requiring the services of multiple processors. However, before we can address the design of such a scheduler, we need to study the requirements of the processors themselves. Such a problem has been widely investigated in the community, but not in such a case as propose here.

In this paper, we study the performance constraints of a number of video codecs used for interactive communications (e.g., video conference), a particularly demanding application, and present a control architecture to support their efficient operations in an Edge cloud environment. Our focus on video processing is meant to expose the needs of the most demanding services, but our long term goal is to support a general set of media types, including voice and audio.

This paper is structured as follows: Section II sets the background on this work. Section III describes the experimental context and Section IV presents the results of our evaluation of the performance constraints. In Section V, we present a sketch for a scheduling function for media operations within the cloud. Section VI has a discussion of our work and we conclude in Section VII.

II. BACKGROUND

Moving a service to a cloud presents a number of benefits, including lower infrastructure costs and scalability of offer through on-demand activation of servers. Indeed, adapting to demand has been an important sales point for cloud-based services. Offers have typically been confined to computation and storage, and media restricted to streaming.

Much has been written on the various guises of cloud infrastructures and service offerings [3]. More recently, there has been a specific interest in the use of clouds for multimedia services [4], [5], typically Video on Demand (VoD), a growing commercial segment with commercial offerings such as Netflix, which incidentally largely relies on a combination of Amazon's Elastic Compute Cloud (EC2) service and Content Delivery Networks (CDN) providers such as Akamai for content delivery. Companies with large cloud infrastructures, such as Google, Apple, and Amazon itself, offer competing services. Gaming has been another topic of interest.

Media streaming, such as VoD, has to be responsive but is non-interactive and supports a large amount of buffering. The constraints on the server side are of a storage and bandwidth nature: deliver content from storage—or memory caches if the content is in high demand—through a network interface. Furthermore, CDNs can be used in conjunction with cloud storage to scale delivery to a large number of customers.

Media services expand beyond VoD, however, and many are interactive, which implies reaction times in the couple of hundreds of milliseconds in the worst case, and very little margin for buffering. Streaming of real time programmes (i.e., live TV) is another example. There have been several studies of the use of cloud to support mobile services, also in the context of IMS [6], [7], which presents a clear distinction between control and processing supporting distribution, including our own work on the Edge Cloud [8].

IMS also illustrates the need for different varieties of media processing. Beyond the streaming services already described, we find services strongly related to telephony, such as DMTF (tone) decoding, voice mail or also interactive voice response (IVR), but also more generic services such as transcoding or conference bridges. Beyond IMS, IP/TV is another example of media processing, especially in contexts, such as mobility, where a uniform multicast model can be difficult to deploy at the network level and needs to be provided as an adaptive application. There have been so far little effort reported to study the effects of the deployment of interactive media services in the cloud. We can note that some commercial offerings, in the form of virtual media servers, are required to be run alone on a hardware platform and have strong limitations (e.g., Microsoft Media Platform).

Unlike streaming services, which can be accessed and controlled from a web page, interactive services tend to be related to a signalling protocol, typically SIP. Also, for scalability reasons, the media function is separated into control and processing. From this perspective, we argue that it makes sense to study how the media processing function can be deployed in the cloud, to take advantage of the latter's flexibility and scalability. In the following sections, we study first the cost of hosting a media function in the cloud and second, how can it be properly orchestrated.

III. Media Function Performance Characterization

We look here at the characterization of the performance cost of running a media processing function on a generic processor. We have created a simple testbed to isolate the contribution of video flows on computer resources along three parameters: CPU, memory and bandwidth consumption; we have also measured latency. The purpose of the characterization is to identify the key parameters to be used by a scheduling function, which we will explore in the next section.

The goal, quite straightforward, is to study how it is possible to multiplex different functions onto single processors. To illustrate our purpose we consider only one example–transcoding–a function that is however quite certainly CPU demanding and subject to latency.

Video processing: Our experimental environment is based on the use of the GStreamer framework. Such a framework, extensible through plugins and composition is an ideal vehicle for custom tests. It is also quite efficient, in spite of its flexibility, as has been demonstrated in performance evaluations [9].

A GStreamer application is a pipeline of different modules which contribute one specific element of the audio/video transmission and processing chain, including coding/decoding, mixing, filtering, scaling, effects, etc.

To illustrate how processing pipelines can be specified in succinct term, we present in Fig. 1 an example of a simple GStreamer pipeline, and the matching code is presented below. The pipeline starts with a live video capture from a camera (Microsoft LifeCam Studio, 1080p), although a network or

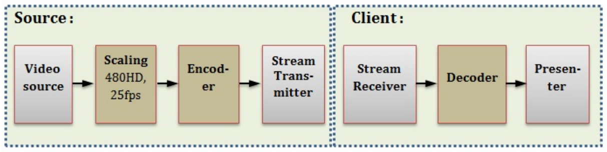


Figure 1: Simple GStreamer Pipeline

disk stream is also a possibility. This stream is scaled to a smaller video size, coded, transmitted, received, decoded and presented on a screen.

Latency measures: We use GStreamer extensively to generate our media streams, with different codecs. But also quite important for our study is the possibility of using it to insert data in a media stream, which can be identified at the receiver and used for latency measurements. For this purpose, we use the Zbar module, a generic part of the GStreamer framework, which allows the detection of the presence of a barcode in a picture. The Zbar module reads frames from a video stream, detects barcodes and sends them as element messages to the GStreamer bus, from where we can retrieve the detected barcode data and the timestamp of the frame that triggered the message.

The use of the Zbar module is key to measuring latency. A barcode picture is integrated in the stream and marked with a timestamp as part of normal processing for transmission with RTP transport. The module detects that barcode at the receiving end and retrieves the corresponding timestamp. The latency is the difference between the timestamps retrieved from the stream and the current time at the receiver. We note also that this technique is robust in the presence of video transcoding.

For such a measure to be useful, the time on both machines must be synchronized. In our studies, all computers run the NTP protocol with a server polling interval of 10 seconds. The latency reported is the average of 10 samples.

Codecs: We have used three codecs suitable for use for video conferencing over the Internet. They were meant to be representative of the most popular techniques, but not necessarily to present an exhaustive choice of possibilities. Most specifically, we have used motion JPEG (MJPEG), MPEG2, et MPEG4-AVC (H.264), all available within the GStreamer framework. The key rate was fixed at a standard 25fps, and the video format was 480p, which, with the arrival of a new generation of mobile terminals, is slowly becoming the standard low end of video resolution. In the case of H.264, we have configured the implementation we used (x264) for an interactive application and not for its default streaming operation, which uses a large amount of buffering to achieve high video compression rates.

IV. Experimental Results

All tests were performed on a computer with an AMD Phenom(tm) II processor at 2.7 GHz, with 16 GiB of memory, running a XEN-enabled bare-bone Debian linux distribution. All media processing instances were running single-threaded, to avoid conflicts between different levels of scheduling, in separate processes but without virtual machines. The communication link speed was limited to 100 Mbps, a realistic perspective if we consider that this machine would be part of a cloud and have many siblings performing the same operations.

We begin with the performance limits imposed by the computing platform itself, and then consider the impact on latency.

Physical setup: The end-to-end view of our basic system is presented in Fig. 2. More specifically, it shows the sender side, relay and receivers side. On the sender side, a GStreamer pipeline encodes the video stream and transmits it over UDP. The streams received at the relay are transcoded and retransmitted to the receivers. Each receiver decodes and renders the stream to the screen.

The number of receivers is limited by the number of instances executed in the relay. Once the sender transmits the stream, on the relay side, we continue adding instances, all the while measuring the resource consumed, until we reach the point where the relay has exhausted its capacity to do further useful work. After the generation of each instance we wait 10 seconds before making a measurement to avoid transient effects. Indeed, during the evaluation of the CPU metric we found that, after the generation of an instance, the results were not correlated until the processor had stabilized, which required 5 seconds on the average. Also, to reduce interference, all background tasks were disabled during measurements.

On the relay side, the CPU and memory utilization were measured based on the standard process statistics report that contain information related to overall system status. We have used the libpcap library to capture the network traffic into a file, which was later analysed using the Wireshark graphic analysis tool to extract the bandwidth information. This analysis was performed offline to avoid interfering with the experiments. The latency between the sender and the receiver side was measured by the barcodes frames as explained before.

Platform limits: Fig. 3 presents the results of our performance tests for two types of videos: a talking head-type video stream, typical of video-conference applications, and an action video stream with many changes of background. For all figures, the x-axis presents the number of instances of a video operation and the y-axis the percentage of a CPU or the amount of bandwidth used for each type of video (memory is not presented because of space considerations). We are considering only homogeneous instances: only one type of video for all instances. The results are presented for three codecs denoted as jpeg (MJEPG), Mpeg2 (MPEG2, which gives equivalent results as H.263 and MPEG4-part 2) and x264 (H.264).

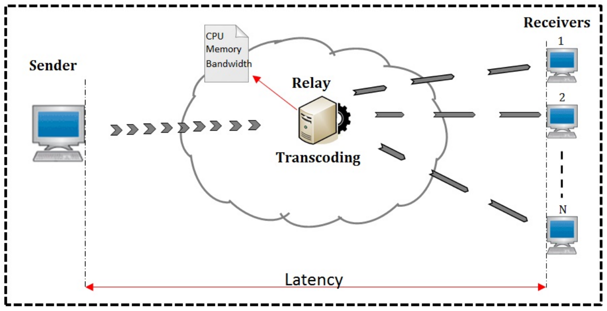


Figure 2: Experimental setup

We see that, as could be predicted, H.264 coding is the most demanding in terms of CPU, with the best compression results for dynamic video content. At the other extreme, MJPEG is low in CPU demand, while requiring higher bandwidth. Mpeg2 trails x264 closely and presents little noticeable advantage over it. In the case of x264, the limiting factor will be the CPU consumed—between 25 and 30 instances—while for jpeg, it is the bandwidth—about 18 for a 100 Mbps link.

These results clearly establish the soundness of using a standard computational platform to perform video processing, as the number of simultaneous instances that can be supported is rather large and lends itself to a mix of operations.

Latency: We next consider the impact of running multiple instances on latency. As explained above, these measures were done by the insertion of barcodes with a timestamp in the video stream and their extraction at the reception. A null operation was performed to eliminate the delay due to this technique and we consider only the increase in latency in our results.

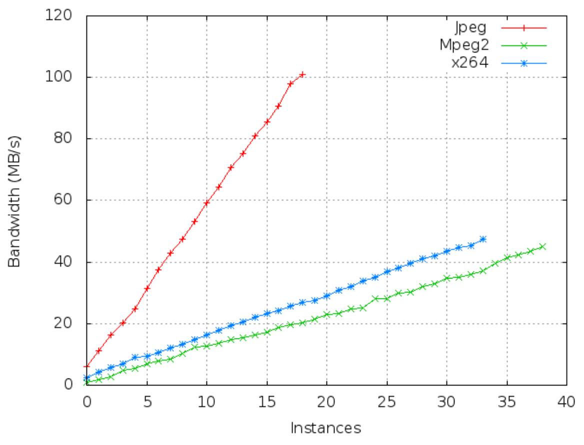
Fig. 4 again shows the results for two different kinds of content, static and dynamic. At first glance we see that we do not have the linear behaviour that we had observed with the performance indicators. At about 10 instances, in both cases and for two codecs, we see that we lose the linear behaviour, and latency increases dramatically for the MJPEG codec. A correlation with the use of bandwidth points to a likely answer for this behaviour, which would denote a greater level of contention at the network interface. The x264 module, on the other hand, behaves rather linearly, with similar results for both types of video. Furthermore, the results are quite acceptable for interactive communications, remaining inferior to 100 ms.

Other factors: We have also analysed jitter and video quality degradation. Jitter does increase with the number of instances but we could not measure it with sufficient precision to have statistically significant results. Similarly, no statistically significant degradation has been observed in the video content, on the basis of a PSNR-based comparison of original vs. received content.

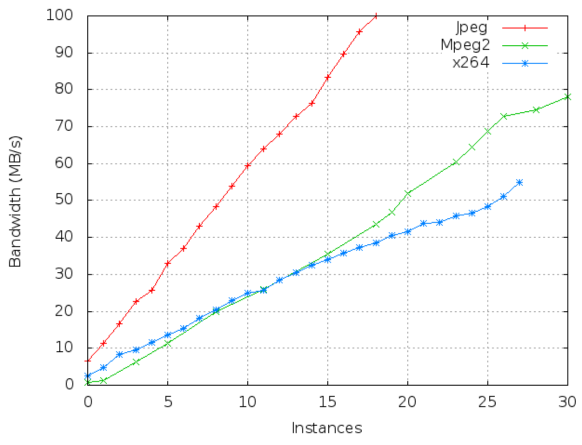
V. Edge Cloud Deployment

Through the assessment of the performance of a media function, our experiments have established that it is possible to run several instances of demanding media functions on a single computer. We now analyse how such a deployment could be orchestrated under the supervision of cloud management, and on demand by a control function.

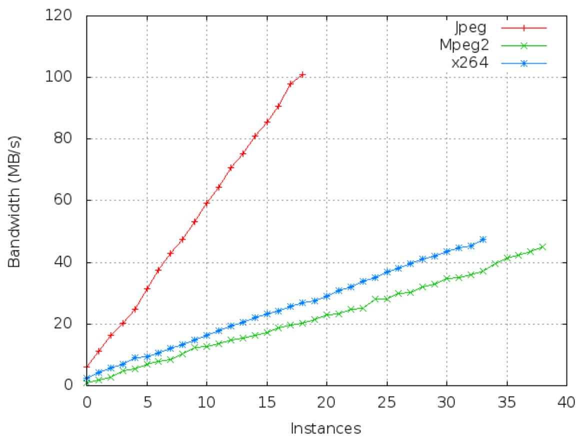
Edge Cloud: First, we consider that processing is deployed on Edge Clouds, that is, broadly speaking, a Cloud within an access provider's domain, close to users. There are several benefits to this model as it provides more flexibility to integrate user feedback based on the nature of the access link, be it for cellular phone services or over-the-top services. It also supports scalability through the availability of such infrastructures across many sites. While not acknowledged as such, the Edge Cloud is a reality as most ISPs have taken to deploy cloud infrastructures of their own, to take advantage of this booming market. Furthermore, it can be closely related to the way CDN is deployed, although CDN is not meant to deliver computing power and is traditionally quite limited in that respect, and typically restricted to the support of dynamic web content.



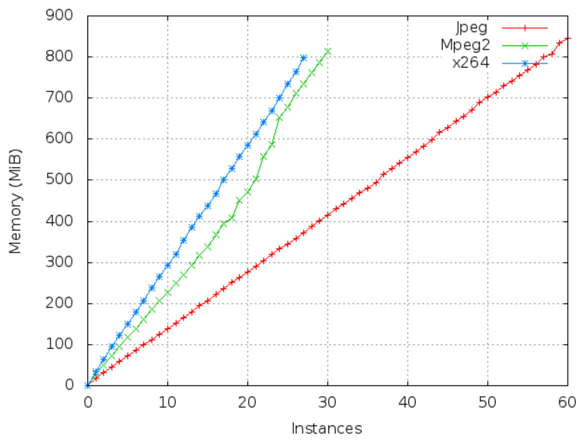
(a) Talking Head Bandwidth



(b) Dynamic Content Bandwidth

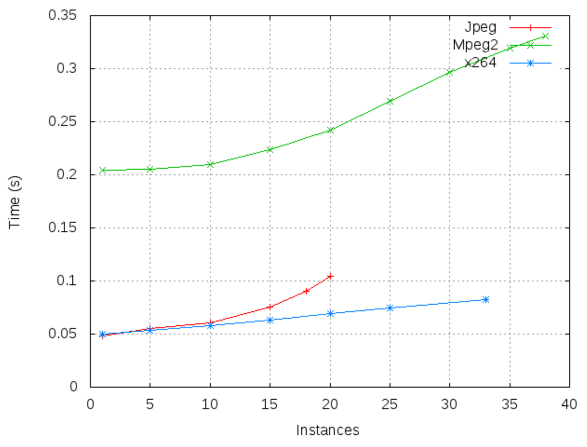


(c) Talking Head Memory

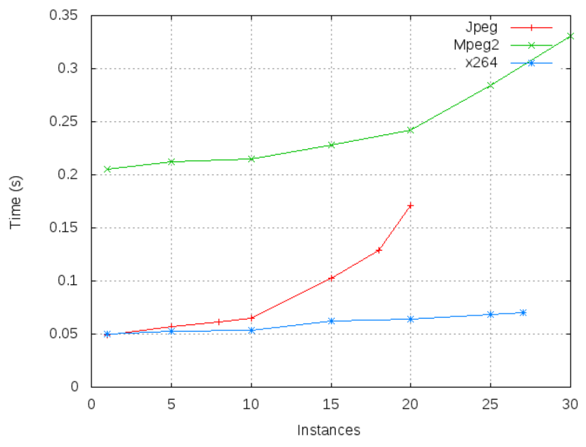


(d) Dynamic Content Memory

Figure 3: Performance of static and dynamic video content



(a) Talking Head



(b) Dynamic Content

Figure 4: Latency

Media Processing: As we have already said, we postulate that media control can be done remotely while processing will be closer to the access network–what we call the network edge. We have already discussed that we consider media functions with tight time constraints. This includes media relay (to bypass firewalls or middleboxes), transcoding, live broadcasting, IVR interactions, etc.

In the line of the experiment described above, we also characterize each form of processing in terms of the resources it requires (CPU, Memory, Bandwidth, Storage).

Computing structure: Remember that virtualization is the tool of choice for deployment in the cloud, from small, language-specific virtual machines (e.g., Python, Java) to a full virtualized computer. This is however not necessary in our case: we only need minimal support from a generic OS to be able to run multiple instances of GStreamer pipelines, encapsulated in their own process, and virtualization is irrelevant. Of course the observation we make about these pipelines generalizes to other implementations of media processes.

We propose then that media processing be organized along the lines of a grid; that is, a pool of machines dedicated to media processing, running on a software-tuned platform. To harmonize with normal cloud operations, this software platform could run on top of the hypervisor used for PaaS operations. This also implies that the size of the pool could be elastic, with more machines assigned to its operations as required. Each software platform has a control module to accept new instances of media processes.

Monitoring is organized in similar fashion, keeping in mind that the ratio of monitoring to media functions can be quite small. Monitoring is in turn connected with media control, which can run on a different cloud and be more centralized.

Scheduling: Scheduling, in this environment, takes several dimensions:

- *static* adjustment of the vocation of machines in the pool based on the number of control functions activated;
- *dynamic* dispatching of activation of a media processor;
- *meta* management of elastic behaviour (size of the pool) based on runtime demands.

We concern ourselves only briefly with meta management here, and do not discuss the static dimension, as they largely depend on contractual terms balanced with a history of the behaviour and consequent demand prediction model [10], [11]. Still, the adjustment of the size of the pool requires some degree of concern to make sure that the resources we need are available when we need them, but are also not kept around beyond the time they are required. Dynamic scheduling, on the other hand, is directly relevant to our study as we must make sure that machines running media functions are well used. In that spirit, observe that, unlike streaming content, we do not know a priori what the duration of the service will be, especially for communications where transcoding/conferencing is involved. Under such conditions, it is difficult to hope to find an optimal scheduler.

Remember that we characterize the different media processes in terms of their CPU, memory and bandwidth needs. Since we have seen CPU load is the dominant factor for video processing, we propose, as a first approximation, a straightforward scheduler where machines are sorted from least loaded to most heavily loaded and a suitable machine is chosen based on that order, in a greedy fashion, also matching the latency constraint of the application. For meta-management, When the least loaded machine's load increases beyond a high water mark, the active pool size is increased; similarly, when the most heavily loaded machine's load falls below a low water mark, the active pool size is decreased. The high/low water marks would be adjusted with the rate of subscription and departure of media functions, to give enough reaction time to allocate resources. Such considerations, however would depend on the nature of the service(s) offered in practice.

Orchestration: The connection between monitoring and processing is easily achieved through a management function. The control requests a processing resource for a specific operation; the management function will schedule the activity on a suitable machine, and return to the control the characteristics required to integrate it in an end-to-end media flow, e.g., IP address and port numbers.

The operation would have to be pre-registered with the management function, in the form of an execution script to establish its performance characteristic, to be used by the scheduler, with a test and calibration protocol. This model provides strict resource confinement and acts as a contract for the execution of a specific function, valid over all its instances.

The control also notifies the management of the end of the execution of a function, so that the resource can be terminated and recycled.

A variant: To illustrate the flexibility of our architecture and its scheduling model, we present here another context, which is suitable for videoconference, where quality can be degraded within reason if resources are saturated. We have imagined four quality scenarios, characterized by the profiles presented in Table I.

TABLE I: DIFFERENT SERVICE PROFILES

Profile	Best	Default	Fast	Ultrafast
CPU(%)	13	7.2	5.4	3.6
PSNR	40.52	38.22	37.00	35.28
MOS	Excellent	Excellent	Good	Good

The idea, in this case, it to work with a fixed pool of resources, but to degrade the quality of the communication, within the confines of quality constraints characterized by a satisfactory MOS. As the load increases beyond a limit set a percentage of CPU load, the quality of the flows will be lowered, and similarly increased, with suitable hysteresis to avoid oscillations, when the load decreases. Fig. 5 shows the behaviour of the load as the number of instances increases beyond the best quality and flow quality is slowly downgraded; the algorithm itself is based on thresholds and straightforward. The threshold maximum load is set at 75 % to allow temporary overruns. Note that GStreamer allows a transition between quality profiles without interruption of the video transmission.

VI. Discussion and Related Work

Complementarity between grid and cloud has been discussed by Foster in [12], and [13]. Taking a subset of a cloud

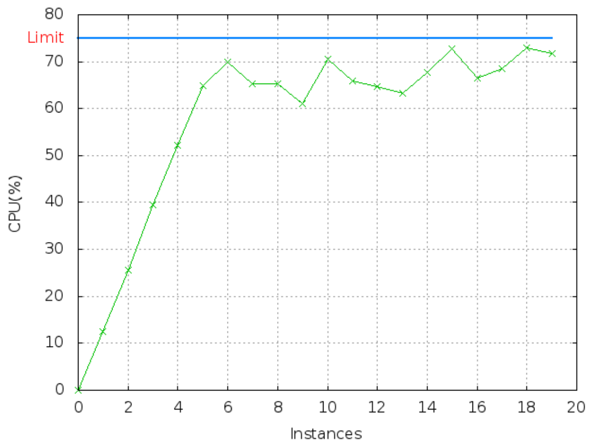


Figure 5: CPU Load

and using it as a computational grid, as we describe here, is not a new concept. It is also consistent with the use of a cloud as a streaming farm, which has itself been the focus of some research [5]. Our work differs in its concentration on interactive media processing.

It also complements the large body of work on the management of QoS in clouds and the establishment of SLAs [14], [15], as it is meant to provide a predictable model of performance requirements for management. Our approach is simpler as we show that we do not have to worry about the effects of virtualisation, which is unnecessary for our purposes. We also differ from streaming applications since we do not have to worry about load balancing or disk access, which can lead to other scheduling issues.

Most important, media distribution is closely related to the research done on gaming clouds [16]. The similarities in time constraints and computing load between both types of problem are clear although some elements are clearly different. Taking into consideration single user or group video games only, they will be implemented in a single server, with no need for transcoding. Furthermore, specialized hardware, typically GPUs, will be used to achieve better performance results. Finally, some latency in game set-up is acceptable, which leads to more flexibility in scheduling. Nevertheless, it is clear that the same infrastructure can benefit both types of application and such convergence will be the focus of future work.

VII. Conclusion

We have presented a performance analysis for a media function and shown how these results can be used for their scheduling in a grid environment. We have chosen this demanding function to assess the practical limits and the results show that it is quite reasonable to not only mix such functions on the same processor with a simple containment, but to also mix them with other functions, also interactive, but possibly less demanding.

This work overall establishes the suitability of the edge cloud, as opposed to dedicated hardware or boxes, as a host and support for media processing. The latency of the operations is quite acceptable for such applications as IMS provided the cloud be deployed closer to the edge, e.g., for mobile service providers.

In future work we plan to develop and refine our management function and design a scheduling toolbox to support a variety of operations along the line of those we have presented. More specifically, we plan to integrate learning mechanisms to assess the nature of the processing.

Acknowledgment Our technique to measure latency was inspired by [17], but adapted to the tools offered by GStreamer.

References

[1] Multimedia Resource Function Controller (MRFC) - Mp interface: Procedures Descriptions, 3rd Generation Partnership Project TS 23.333, Rev. V11.0.0, Jun. 2012.

[2] Media server control using the IP Multimedia (IM) Core Network (CN) subsystem, 3rd Generation Partnership Project TS 24.880, Rev. V8.2.0, Jun. 2008.

[3] M. Armbrust et al. “A view of cloud computing,” Commun. ACM, vol. 53, no. 4, Apr. 2010, pp. 50–58.

[4] S. Dey, “Cloud mobile media: Opportunities, challenges, and directions,” in Computing, Networking and Communications (ICNC), 2012 Int’l Conference on, Feb. 2012, pp. 929–933.

[5] Y. Wu, C. Wu, B. Li, X. Qiu, and F. Lau, “Cloudmedia: When cloud on demand meets video on demand,” in Distributed Computing Systems (ICDCS), 31st Int’l Conference on, June 2011, pp. 268–277.

[6] J.-L. Chen, S.-L. Wuy, Y. Larosa, P.-J. Yang, and Y.-F. Li, “IMS cloud computing architecture for high-quality multimedia applications,” in Wireless Communications and Mobile Computing Conference (IWCMC), July 2011, pp. 1463–1468.

[7] P. Bellavista, G. Carella, L. Foschini, T. Magedanz, F. Schreiner, and K. Campowsky, “QoS-aware elastic cloud brokering for IMS infrastructures,” in Computers and Communications (ISCC), IEEE Symposium on, July 2012, pp. 157–160.

[8] S. Islam and J.-C. Grégoire, “Giving users an edge: A flexible cloud model and its application for multimedia,” Future Generation Computer Systems, vol. 28, no. 6, 2012, pp. 823–832.

[9] V. Sentongo, K. Ferguson, and M. Dlodlo, “Real-time performance evaluation of media pipeline plug-in architectures,” in Southern Africa Telecommunication Networks and Applications Conference (SATNAC 2011), East London, South Africa, Sep. 2011.

[10] M. Peixoto et al., “A metascheduler architecture to provide QoS on the cloud computing,” in Telecommunications (ICT), IEEE 17th Int’l Conference on, Apr. 2010, pp. 650–657.

[11] I. N. Goiri, F. Julià, J. Fitó, M. Macías, and J. Guitart, “Resource-level QoS metric for cpu-based guarantees in cloud providers,” in Economics of Grids, Clouds, Systems, and Services, Lecture Notes in Computer Science Series, J. Altmann and O. Rana, Eds. Springer Berlin / Heidelberg, vol. 6296, 2010, pp. 34–47, 10.1007/978-3-642-15681-6_3.

[12] I. Foster, Y. Zhao, I. Raicu, and S. Lu, “Cloud computing and grid computing 360-degree compared,” in Grid Computing Environments Workshop, GCE ’08, Nov. 2008, pp. 1–10.

[13] I. Foster, “There’s grid in them thar clouds,” http://ianfoster.typepad.com/blog/2008/01/theres-grid-in.html, 2008, last access on February 14th, 2014.

[14] S. Ferretti, V. Ghini, F. Panzieri, M. Pellegrini, and E. Turrini, “QoS-aware clouds,” in Cloud Computing (CLOUD), IEEE 3rd Int’l Conference on, July 2010, pp. 321–328.

[15] J. Pedersen et al., “Assessing measurements of QoS for global cloud computing services,” in Dependable, Autonomic and Secure Computing (DASC), IEEE Ninth Int’l Conference on, Dec. 2011, pp. 682 –689.

[16] R. Shea, J. Liu, E. C.-H. Ngai, and Y. Cui, “Cloud gaming: Architecture and performance,” IEEE Network, July/August 2013, pp. 16–21.

[17] O. Boyaci, A. Forte, S. Baset, and H. Schulzrinne, “vDelay: A tool to measure capture-to-display latency and frame rate,” in Multimedia, 2009. ISM ’09. 11th IEEE Int’l Symposium on, Dec. 2009, pp. 194–200.