QoE-aware Elasticity Support in Cloud-Native 5G Systems

Sunny Dutta[1], Tarik Taleb[1]
[1]Aalto University
sunny.dutta@aalto.fi, talebtarik@ieee.org

Adlen Ksentini[2]
[2]Rennes University
adlen.ksentini@irisa.fr

Abstract – **Typically, maintaining static pool of cloud resources to meet peak requirements with good service quality makes the cloud infrastructure costly. To cope with this, this paper proposes an approach that enables a cloud-infrastructure to automatically and dynamically scale-up or scale-down resources of a virtualized environment aiming for efficient resource utilization and improved quality of experience (QoE) of the offered services. The QoE-aware approach ensures a truly elastic infrastructure, capable of handling sudden load surges while reducing resource and management costs. The paper also discusses the applicability of the proposed approach within the ETSI NFV MANO framework for cloud-based 5G mobile systems.**

I. Introduction

After Long Term Evolution (LTE) and IMT-advanced systems, the next major phase of mobile telecom industry is the 5th generation mobile networks (5G), also referred to as beyond 2020 mobile communications system [1]. With the ability to handle high bit rates in peak conditions, higher spectrum efficiency, better coverage, and support of potential numbers of diverse connectable devices, 5G systems are required to be cost-efficient, flexibly deployable, elastic, and above all programmable. The crucial issue in sustainability of the mobile operators in terms of 5G resides in the provisioning of high-data rate connectivity to an ever-growing mobile data traffic. Another issue pertains to the system scalability and reliability as it should be agile and elastic to accommodate huge amount of mobile applications. However, the shortfall in supporting high demands for diverse applications from various devices is mostly due to the highly centralized core networks and the usage of custom hardware components, which were not designed with elasticity in mind. Another challenge of 5G consists in supporting applications and services with an acceptable and consistent level of quality of experience (QoE) anywhere anytime. However, to accommodate the high dynamicity of mobile traffic and to ensure ultra-low latency and high QoE for delay-critical interactive services, operators are forced to overprovision components considering peak hours and keeping them unused during off-peak periods; a fact that effectively wastes resources in terms of energy, processing, and network.

The solution to the above limitations consists in the convergence of mobile networks and cloud, and the adoption of the network function virtualization (NFV) concept [2]. In a virtualized infrastructure, CPU, memory, storage, and networking are provided in abstract slices to a set of virtual machines (VMs). During the deployment phase, a Cloud Service Provider (CSP) typically offers a static set of available hardware configuration blocks (i.e., "flavors" in the context of Openstack – a unique combination of disk space, memory capacity and CPU) to a Virtual Network Function (VNF) provider (VNFP). The blocks are chosen to perfectly match the functional/operational requirements of the respective VNF. This enables operators create and deploy network components in accordance to their needs without involving dedicated hardware components. Whilst this could sound as a perfect solution, adopting static flavors despite the fluctuating nature of mobile traffic may result in poor performance as resources could be underutilize or overprovisioned. To tackle this issue, true elasticity of VNFs should be attained. For this purpose, the overall cloud orchestrator should be aware of the changing demands of users and the real-time conditions of the underlying resources. It should also dynamically reflect that in scaling up when there is a need for more resources or in scaling down to save resources and achieve cost-efficiency. The overall objective of this paper is to define a remedy to the above limitations by devising an approach that has the intelligence to understand the requirements based on varying nature of traffic and to automatically scale up/down the system. The approach considers the system's CPU/ RAM usage and QoE of end-users in the decision of elasticity and performs scalability in a cognitive manner making the system self-organized.

The remainder of this paper is organized as follows. Section II presents the state of the art. Section III describes our proposed scheme showcasing its technical benefits. Section IV describes the experiment setup and discusses the obtained results. The paper concludes in Section V.

II. State of the Art

The success of cloud-based mobile core networks highly hinges on an efficient lifecycle management of VNFs including VNF placement, instantiation, migration and scaling. Regarding the latter, it is important to decide the amount of resources to allocate for a VNF, parameters to consider for scaling and when to enforce the scaling operation.

Various research work have been carried out dealing with the resource allocation problem and VM management to achieve cost savings and that is from better utilization of computing resources while avoiding overload situations. In [3], cloud resource allocation is considered alongside VM's placement and migration based on VM's CPU, memory, storage, network bandwidth along with resource contention. There have been also numerous research work on dynamic resource provisioning using feedback control mechanism on the infrastructure level performance metrics. In [4], the trigger for scaling is considered based on threshold values relevant to resource usage. In [5], the work uses the concept of dynamic scaling of resources based on concurrent users count and the number of active connections. A front-end load-balancer (LB) is used to distribute the processing of services among parallel instances. In [6], an Amazon web service (AWS) auto scaling is introduced using indicators such as CPU utilization, network usage, and disk operations. The mechanism automatically scales up or scales down VM instances using threshold values of the indicators to trigger the operation. The research work described in [7] also proposes an auto-scaling mechanism based on budget constraints and job execution deadline. In [8], the proposed approach considers dynamic service level agreement (SLA) between the CSP and VNFP. Rather than allocating a fixed amount of resources for the whole life-cycle, the resources are varied based on predictable user load [9]. Similarly, in [10], another auto-scaling mechanism is introduced considering a pattern-based prediction algorithm to handle sudden appearance/ disappearance of traffic, which ensures good user experience. Unlike pattern-based approaches, the research work described in [11] suggests a work-profile based filtering concept, wherein the service quality of a cloud instance depends on specific service type. According to the service type, the cloud resources are allocated to achieve optimum results towards improving Quality of Service (QoS). To improve users' perceived QoE, various studies have come up with metrics, which are directly related to the performance of services such as video streaming. Studies in [12] proposed metrics in the form of network bandwidth, Round Trip Time (RTT), video-bit rate, page load times and video interruptions to evaluate the user QoE in terms of Mean Opinion Score (MOS) [9][13].

In the above-mentioned research work, optimal resource utilization and resource scaling decisions are made based on various indicators. They can be classified as either prediction-based approaches, feedback-controlled algorithms, or service-specific resource allocation schemes. To the best knowledge of the authors, no solution has considered QoE's feedback as a criterion to scale up/down cloud resources. To fill this gap, in this paper, we propose a novel scaling method that closely considers users' QoE. Besides real-time feedback on system resource usage (i.e., CPU and RAM usage), the proposed scaling method also considers real-time feedback on the quality of the service (i.e., service hosted inside the VNF) as perceived by the end-users. As will be described in the next section, the proposed scheme aims at building an intelligent, cognitive and self-healing system capable of scaling up/down resources in an autonomic fashion based on users' perceived QoE.

III. Proposed QoE-Aware Elasticity Enforcement Scheme

The overall objective of the proposed solution is to build a QoE-aware resource management of virtual instances; in order to automatically provision and scale network services (NS) in an elastic way. To make the proposed solution compliant with the existing ETSI NFV architecture, additional functions, namely QoE Assessor (QA), Resource Usage Monitor (RUM), and Elasticity Decision Maker (EDM) have been integrated with the architecture's Service Manager (SM) and Service orchestrator (SO).

The NFV Management and Orchestration (MANO) architecture, presented in [2], depicts the MANO framework. Three functional blocks, namely Virtualized Infrastructure Manager (VIM), the VNF Manager (VNFM) and the NFV Orchestrator (NFVO), play the key role in controlling the infrastructure and are inter-connected over specific reference points. Besides traditional management, the MANO framework focuses on novel management aspects introduced by NFV such as the creation and life-cycle management (LCM) of the virtualized resources for VNFs, collectively referred to as VNF management [1]. There are several VNF management tasks such as VNF scaling, migrating, updating, to name a few, but this paper focuses on cognitive scaling of VNF resources. Considering the ETSI NFV architecture [2], the three main architectural entities involved in the management and orchestration of VNFs are Service Orchestrator (SO), Cloud Controller (CC) and Service Manager (SM). SO is the entity responsible for lifecycle management of a NS. The primary functionality of SO also includes implementation of the service and complete end-to-end orchestration of service instance with creation and scaling. CC is mainly responsible for cloud infrastructure resource provisioning and deployment. It also manages virtual resources coordinating with SO and is also involved in provisioning, deployment and disposal of instances. The focus of SMs is in managing individual VNFs of different tenants based on predefined policies. We depict in Fig. 1 the role of the added functional blocks within the ETSI NFV architecture.

To illustrate the idea behind our proposed cloud resource management scheme, we consider a web based video service (i.e. based on HTTP) as a NS, which is hosted as a VNF on top of a VM, utilizing shared resources such as CPU, RAM, I/Os of the parent physical machine (PM). Typically, the NFV-MANO coordinates the instantiation of VM(s) on selected PM(s) [14-16] and assign/allocate resources based on the selected flavors to the VM(s) that will eventually host the VNF(s). However, the process of hosting/instantiating/deploying VNFs on VMs has inherent limitations because of the static process of choosing flavors [1] which may meet the functional requirements but the complexity can be found in the form as mentioned below:

a) An over-provisioned allocation of resources, exceeding the requirements, may increase the overall cost of the system and may also result in scarcity of resources in the pool.
b) The allocation process may not take into account the unforeseen traffic/load surges, which may ultimately impact the service performance.
c) The strict resource allocation and isolation policy prevents the sharing and run-time/dynamic (re)allocation and/or (re)organization of the unused capacities of the underlying physical resources that are "pinned" to other VMs on the same PM.

Given the abovementioned limitation, for a service such as a web-based video streaming whereby multiple clients connect to the network to receive video streams, they normally end up exhausting the resources. The nature of traffic in terms of connection count and bandwidth utilization is pretty much dynamic and unpredictable. If the allocated static set of resources (i.e., processing power of CPU or the buffer/ cache/ swap / physical memory) are not sufficient to handle the number of connections, it will impact the service in terms of delay, packet loss, play-out interruptions, and will eventually downgrade the users' perceived quality. On the other hand, when the resources are overprovisioned, the cloud will operate in an under-utilized manner and will experience high energy consumption, increased power budget and reduced profit. To address the above limitations, there is need for a solution that intelligently allocates resources to VNFs when they are instantiated as well as during their run-time, attaining full and true elasticity of the cloud. In this paper, this is achieved by constantly checking the service quality and accordingly enforcing elasticity and that is through integration of RUM, QA and EDM into the MANO architecture (Fig. 1).

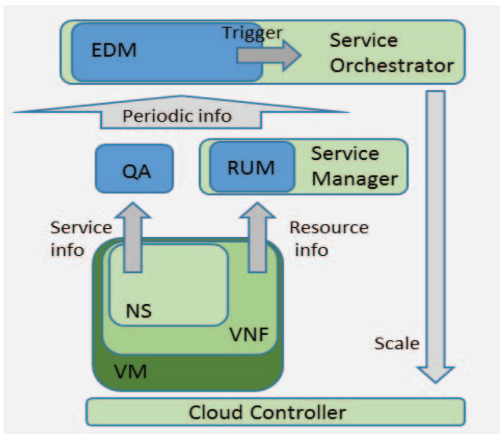

Fig. 1. Key components of the proposed scheme as part of the ETSI MANO framework.

A. Resource monitoring

Dedicated resources for a VNF can be of different types. For a web-service VNF, the most important resources are CPU and RAM as they are crucial for efficiently processing and handling connection requests. Indeed, large CPU and RAM sizes enable fast processing of a large number of connections [17][18]. In Openstack, the concept of "flavors" is used whereby each flavor represents a block of fixed resources in terms of CPU cores, RAM, storage, ephemeral disk and swap disk. Accordingly, when a VNFP allocates a flavor to a VNF, it allocates a certain amount of resources that remain fixed during the operation course of that VNF. To keep real time information about the resource usage of a VM, we consider the RUM functionality that can be part of Openstack's Ceilometer. The main functionality of RUM includes real-time monitoring of the resources. Integrated with the SM, RUM takes into account the aforesaid parameters and periodically monitors the running instance's utilization percentage of CPU to find the idle states and the percentage amount of RAM left to handle additional processes. As part of the management architecture, RUM performs this periodical monitoring to gather information on the health of the VM instance, considering whether it is becoming exhausted or is underutilized. The gathered data is then sent periodically to the Elasticity Decision Maker (EDM) for further processing.

B. QoE Assessor

Since there is a direct impact of allocated resources on QoE, the proposed scheme decides on whether to trigger elasticity or not based on QoE perceived by users. Therefore, the proposed scheme does not only optimize the resource pool, but also aims for ensuring the desired QoE. For this purpose, there is need to measure and monitor, in real-time, users' perceived QoE in terms of the MOS score. To this end, we used the Pseudo-Subjective Quality Assessment (PSQA) method, which allows measuring user perceived quality in an automatic manner and without any human intervention. PSQA is based on random neural networks, which capture the relation between quality impacting parameters and the MOS value. PSQA runs in two phases. The first phase is a subjective test phase, where users give MOS score to video sequences allocated by a predetermined parameters. The second phase consists in training the random neuronal network, which derives a non-linear function that captures the relation between MOS and the criteria that impact QoE. The used version of PSQA is dedicated for HTTP-based video streaming [13][19]. Two parameters impacting QoE are considered, namely play-out interruptions and quantization parameter (QP – index providing a scaling matrix of amount of video compression in an encoded video). In HTTP streaming, the underlying protocol is TCP, where service distortion is eliminated through the recovery of lost packets but interruptions happen in the form of retransmissions and bandwidth fluctuations. Considering this as a playout interruption, the model is trained to generate the QoE estimation based on values of total numbers of interruptions, average delays and maximum delays experienced along with QP. Using PSQA on the delivered service in real-time, the QoE estimation is obtained by QA and eventually the MOS values are then periodically fed into EDM as an input attribute for elasticity decision.

C. Elasticity Decision Maker (EDM)

The main EDM functionality consists in deciding when to trigger SO to enforce the elasticity operation of an instance indicating what resources should be allocated for the instance, while preventing service interruption. EDM periodically receives monitoring information from both RUM and QA. Based on that, it carries out the following operations:

d) Selecting adequate thresholds for triggering elasticity; this operation is conducted in a self-learning fashion based on ad-hoc measurements, past experiences, and benchmarking of the VNF performance.

e) Specifying the right amount of resources to allocate to an instance to meet the service level agreement in terms of QoE and to also avoid underutilization of cloud resources.

To analyze data, EDM employs a Multi Attribute Decision Making (MADM) algorithm as depicted in Fig. 2 and that is using multiple inputs such as resources allotted for an active instance, resources being effectively in use, maximum resources that can be allocated for the service, instance flavor size, the available flavor sizes in the pool, and MOS of the running service. The thresholds of each of these inputs are preset for variable flavors and for different services (i.e., video streaming) ensuring optimal working conditions. This can be achieved before bringing services into the production environment by manually performing load-benchmarking test for different flavors. The evaluation starts by comparing the CPU usage, RAM usage and the MOS score against their respective threshold values, and accordingly EDM decides on the elasticity as follows:

i) In case the resource usage exceeds the maximum defined threshold and the MOS score is below the optimal value, EDM initiates vertical scale-up.

ii) In case the resource usage is low and below a minimum threshold while MOS is acceptable, EDM initiates vertical scale-down.

iii) In case the resource usage is within the threshold range but the MOS value is below optimal value, EDM initiates vertical scale-up.

iv) In case the resource usage is within the threshold range but the MOS value exceeds the optimal value, EDM takes no action in terms of elasticity deeming the system being healthy and instead updates the relevant thresholds.

Table 1. Notations and details used in the Algorithm of Fig. 2.

Uc	Instance CPU usage (%)
Um	Instance RAM usage (%)
MOS	Index of quality perceived
MOSopt	Optimal MOS score
Tmax	Max Threshold of CPU & RAM
Tmin	Min threshold of CPU & RAM
MAXquota	Maximum resources that can be allocated

```
function scaleup ( )
  if allotted resource == MAXquota then
    scaleup not possible; exit to main ( )
  else
    if vertical scale not possible then
      launch smaller instance and attach to LB // do parallel scaling
    else
      launch bigger instance and attach to LB
      wait for new instance to be active
      remove smaller instance
    end if
  end if

function scaledown ( )
  if allotted resource == MAXquota then
    remove smaller instance
  else
    if already lowest flavor then
      exit to main ( ) //scaledown not possible
    else
      launch smaller instance and attach to LB
      wait for new instance to be active
      remove bigger instance
    end if
  end if

function operation ( )
  if (Uc Um $\geq$ Tmax) && (MOS $\leq$ MOSopt) then
    scaleup ( )
  elif (Uc Um $\leq$ Tmin) && (MOS $\geq$ MOSopt) then
    scaledown ( )
  elif ((Uc Um $\geq$Tmin && Uc Um $\leq$ Tmax) || MOS $\leq$ MOSopt) then
    if resource available then
      scaleup ( )
    else
      exit to main ( ) // upper limit reached
    end if
  else ((Uc Um $\geq$ Tmin && Uc Um $\leq$ Tmax) && MOS == MOSopt)
    exit to main ( ) // System healthy
  end if

function main ( )
  receive monitoring data
  perform operation( )
```

Fig. 2. The adopted Multi Attribute Decision Making algorithm.

When EDM needs to take an action in terms of elasticity, it triggers SO to scale up/down the running instance specifying the new size/flavor of the instance. Hereby, the term "scaling" refers to increasing or decreasing the resources in terms of CPU, RAM, and storage. Scaling can be either vertical or horizontal. Through vertical scaling, SO increases the computing power on a running VM, whereas the horizontal scaling adds computing power in the form of adding another VM in parallel to the one in question. Whilst horizontal scaling is relatively straightforward, it is costly as it involves more VMs and is also more complicated from the management perspectives. Furthermore, horizontal scaling requires load balancers to be more effective. In contrast, vertical scaling exhibits less operation cost but is highly complex from the implementation point of view. For this reason, in this research work, we report on results obtained in case of vertical scaling as its horizontal counterpart is relatively easier to implement.

As the allocated resources shall not exceed the available PM resources; SO consults CC for the available resources before allocating any. For a particular NS, a set of resources is already fixed but not allocated from the beginning. The service starts with a minimal flavor of the instance attached to a load balancer (LB). LB works as a front-end service and re-routes the incoming requests in a round-robin fashion to the newly built instance (in case of scaling) and accordingly prevents service disruption. In case scaling up becomes required and the service is already running with the minimal flavor, a bigger flavor instance is added to the LB pool. Once the service migration occurs between the old and new instances and the new instance becomes ready and functional, SO removes the old busy instance. In case of an underutilized VM, scaling down is performed in the same fashion. As explained before, RUM keeps sending periodically the status of the newly instantiated VM to EDM. If the information received from QA conveys that the perceived quality is not satisfactory, the whole process is re-run to achieve the desired MOS score, thus resulting in the selection of the best fit flavor with the best QoE. In this way, EDM, supported by QA, reflects QoE in its elasticity decisions in a dynamic and autonomic manner saving cost, energy consumption, and equipping the overall system with the self-healing feature.

IV. PERFORMANCE EVALUATION

For the sake of performance evaluation, a real-life testbed was built using Ubuntu 14.04.03 LTS desktop workstation with 8 core CPU and 16GB RAM. The cloud environment was built using Openstack (i.e., Devstack Juno) inside a VirtualBox Ubuntu server with dedicated 4 vCPUs and 8 GB RAM. The setup consists of all-in-one Openstack environment with the controller, compute, heat and neutron components running on the same node. Heat was used to orchestrate the initial setup of a LB and a single instance within the LB pool with a built-in Nginx server for streaming a preloaded video file. LB balances load in round-robin fashion. A HTTP-based live network streaming (i.e., progressive stream) was used. Chrome (i.e., incognito mode) and VLC player were used at the client side to view the video stream. Moreover, the Apache Bench and WRK2 frameworks were used to simulate load, while a modified VLC player was used to measure the MOS score at the user side. Instances were created using Ubuntu cloud image inside customized flavors, wherein the minimal one (Flavor1) had 1vCPU, 512 MB of RAM, and 4 GB storage; and the largest one (Flavor 4) had 2 vCPUs, 2048 MB of RAM and 10GB storage. Two more flavors, Flavor3 and Flavor 4 were created with 1vCPU, 1048 MB RAM, 4GB storage and 2 vCPUs, 1048 MB RAM, 10GB storage, respectively. The RUM, QA and EDM modules were implemented in the same node along with the controller. At RUM, resource usage are monitored and fed to EDM every 60 seconds. To deliberately overload the VM hosting the video service (i.e., busy CPU and exhausted RAM), parallel concurrent connections towards the server were launched: 2000 concurrent connections and a total of 100000 requests per second were sent to the server. The resource usage thresholds were set to 90% for both CPU and RAM as the upper limit to trigger a scale up operation. As for scaling down, the thresholds were set to 25% and 65% for CPU and RAM, respectively. The above thresholds were empirically retrieved after testing instances multiple times under varying loads using the AB benchmarking. The threshold limits were set accordingly and with the intention to keep the system responsiveness in enforcing elasticity (i.e., particularly scaling up) within a reasonable range. Indeed, if the system responsiveness is too long, the service will be totally disrupted before even scaling up the relevant instance.

In our setup, elasticity is performed using Flavor 1 and Flavor 4. The environment starts with Flavor 1 and once the resource usage at the instance reaches the upper limit thresholds, EDM triggers HEAT (i.e., Openstack orchestration engine used as SO) to instantiate a new instance in the same pool with Flavor 4. A sleep time was considered for the newly created instance before it becomes active. Once the second instance becomes active, the first one is released ensuring the service is intact. Apart from this vertical scale-up, horizontal scale-up was also carried out to meet the high number of concurrent connections and that is by instantiating additional VMs running in parallel. Scaling down was also carried out in the same fashion. It shall be noted that since the instances were active, resize or stack update operations were not performed as that would result in suspend mode of the instance which would eventually disrupt the video stream service. Figs. 3 and 4 depict the full scale-up and scale-down operations, respectively. The 'firm' line indicates which instance is in service and how LB takes care of the switchover. The firm line at level 80% represents Flavor 4 is used for streaming and at 40% indicates when Flavor 1 is in use. The pointers 's', 'e', and 'c' on Figs. 3 and 4 indicate respectively the elasticity triggering start time, the end of the scaling operation, and the actual instance changeover time considering a predefined sleep time set up for the new instance to ensure service continuity.

These figures show that right after the changeover in flavor, the usage ratios of both RAM and CPU change, although the injected load is constant in all cases. The figures also indicate that LB rotates the service as soon as the new instance is instantiated. However, the complete release of the old instance takes some time as a sleep time interval was deliberately considered to ensure a smooth streaming service. It is worth noting that the delay since triggering vertical elasticity till the release of old instance is merely '15s to 20s'.

To further investigate the impact of the VM flavor on the perceived QoE, different sizes of VMs were instantiated and videos

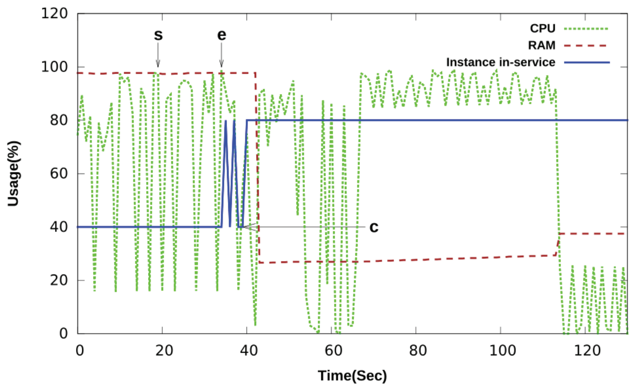

Fig. 3. An example of a scale up operation. (s: elasticity trigger time; e: VM migration end time; c: VM actual changeover time).

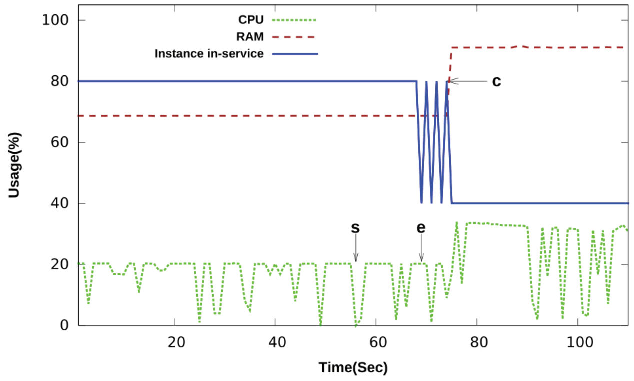

Fig. 4. An example of a scale down operation (s: elasticity trigger time; e: VM migration end time; c: VM actual changeover time).

were streamed from these instances. The MOS values were measured at the client side and the impact of the MOS score in deciding when and by how much to scale up/down an instance was also evaluated. The testbed was run multiple times using a high definition 112-seconds long video. The presented results are an average of multiple testbed runs. The testbed was performed using the considered four flavors under the same load (i.e., 200 concurrent users viewing the video at a rate of 600 requests per second using wrk2). The MOS score and the resource usage patterns are shown in Figs. 5 and 6, respectively. MOS was calculated for every 16 sec chunk of the received video. It was calculated based on how much additional time was involved to playback that 16 sec chunk of actual video when streamed from the loaded instances. The MOS scale is from "0" (low) to "1" (high). Considering the network constraints (i.e., delay and bandwidth), the buffering and interruption times experienced in all scenarios varied within the range of 0.5 sec and 64.2 sec for the different flavors. Every 16 sec span of measuring the video quality is represented as "Measuring Window" in Fig. 5. Initial 16 sec chunk was not considered in calculation. The average video play length for Flavors 1, 2, 3 & 4 were 176.2 sec, 157.89 sec, 112.5 sec and 121.5 sec, respectively.

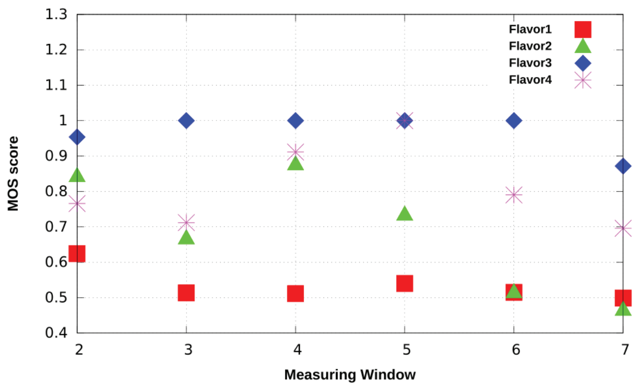

Fig. 5. QoE values for different flavors.

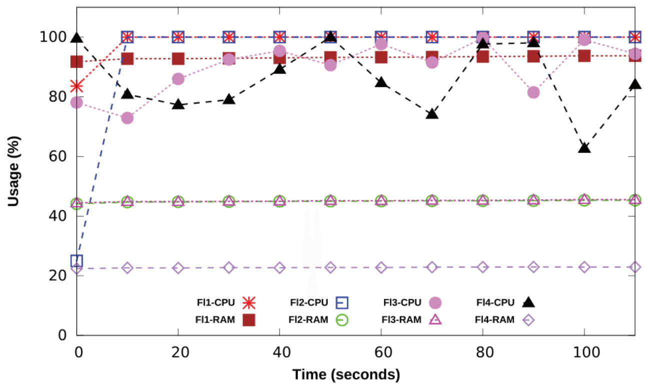

Fig. 6. Average resource usage ratios for different flavors.

From Fig. 6, it is apparent that Flavor 4 is the least utilized. However, from Fig. 5, Flavor 3 exhibits the best QoE. This implies that considering the MOS score only may yield an optimal solution in terms of both quality and cost (i.e., resource size). It also represents an effective metric in assessing the system "health": the instance with the highest resources does not necessarily guarantee the highest QoE. Indeed, a tradeoff between the allocated resources (i.e., cost) and QoE needs to be attained.

V. Conclusion

In this paper, we proposed a scheme that reflects QoE in deciding, in an autonomic manner, when and by what magnitude to enforce elasticity in a cloud environment. The proposed scheme supports the self-healing feature of the cloud as the cloud becomes cognitive and can take autonomic corrective measures, when needed, to maintain an optimal level of QoE of its offered services. The scheme was incorporated in the ETSI MANO framework and was evaluated in a real-life testbed. The obtained results demonstrated the responsiveness of the scheme in effectively enforcing elasticity (e.g., within 15s) and that is based on real-time values of MOS, obtained from the client side. The results also showed that there is an important tradeoff between the allocated resources and QoE. Determining such a tradeoff empirically and theoretically is a research subject worth investigation and shall define one of the authors' future research directions in this area.

Acknowledgment

This work was partially supported by the TAKE 5 project funded by the Finnish Funding Agency for Technology and Innovation (TEKES) and in part by the Finnish Ministry of Employment and the Economy.

References

[1] F.Z. Yousaf and T. Taleb, "Fine Granular Resource-Aware Virtual Network Function Management for 5G Carrier Cloud," to appear in IEEE Network Magazine.

[2] T. Taleb, M. Corici, C. Parada, A. Jamakovic, S. Ruffino, G. Karagiannis, and T. Magedanz, "EASE: EPC as a Service to Ease Mobile Core Network," in IEEE Network Magazine, Vol. 29, No. 2, Mar. 2015. pp.78 – 88.

[3] G. Somani, P. Khandelwal, and K. Phatnani, "VUPIC Virtual Machine Usage Based Placement in IaaS Cloud," arXiv preprint arXiv:1212.0085 (2012).

[4] H. C. Lim, S. Babu, J. S. Chase, and S. S. Parekh, "Automated Control in Cloud Computing: Challenges and Opportunities," in Proc. 1st workshop on Automated Control for Datacenters and Clouds, Chicago, Illinois, Jun. 2009.

[5] T. C. Chieu, A. Mohindra, A. A. Karve, A. Segal, "Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment," in Proc. IEEE ICEBE Int'l Conf. e-business Eng. , Macau, China, Oct. 2009

[6] AWS auto-scaling – URL: http://aws.amazon.com/autoscaling/

[7] M. Mao, J. Li, M. Humphrey, "Cloud Auto-scaling with Deadline and Budget Constraints," in Proc. 11th IEEE/ACM Int'l Conf. Grid Computing, Brussels, Belgium, Oct. 2010

[8] B. B. Nandi, A. Banerjee, S. C. Ghosh, "Dynamic SLA Based Elastic Cloud Service Management: A SaaS Perspective," in Proc. IFIP/IEEE Int'l Symp. on Integrated Network Manage. , Ghent, Belgium, May 2013

[9] A. Ksentini, T. Taleb, and K. Benletaief, "QoE-based Flow Admission Control in Small Cell Networks", to appear in IEEE Trans. on Wireless Communications.

[10] J. Bao, Z. Lu, J. Wu, S. Zhang, Y. Zhong, "Implementing a Novel Load-aware Auto Scale Scheme for Private Cloud Resource Management Platform", in Proc. IEEE Network Operations and Manage. Symp. (NOMS), Krakow, Poland, May 2014

[11] T. Magedanz, F. Schreiner,, "QoS-aware Multi-Cloud Brokering for NON Services- Tangible benefits of elastic resource allocation mechanisms", in Proc. 5th IEEE Int'l Conf. on Communications and Electronics (ICCE), Danang, Vietnam, July 2014

[12] P. Casas, A. Sackl, S. Egger, and R. Schatz, "YouTube & Facebook Quality of Experience in Mobile Broadband Networks", in Proc. GC'12 workshop: Quality of Experience for Multimedia Communications, Anaheim, CA, Dec. 2012

[13] A. Ksentini and T. Taleb, "QoE-Oriented Adaptive SVC Decoding in DVB-T2," in IEEE Trans. on Broadcasting, Vol. 59, No. 2, Jun. 2013. pp.251-264

[14] T. Taleb, M. Bagaa, and A. Ksentini, "User Mobility-Aware Virtual Network Function Placement for Virtual 5G Network Infrastructure," in Proc. IEEE ICC 2015, London, UK, Jun. 2015.

[15] M. Bagaa, T. Taleb, and A. Ksentini, "Service-Aware Network Function Placement for Efficient Traffic Handling in Carrier Cloud," in Proc. IEEE WCNC'14, Istanbul, Turkey, Apr. 2014.

[16] T. Taleb and A. Ksentini, "Gateway Relocation Avoidance-Aware Network Function Placement in Carrier Cloud," in Proc. ACM MSWIM 2013, Barcelona, Spain, Nov. 2013.

[17] T. Taleb, A. Ksentini, and R. Jantti, "Anything as a Service for 5G Mobile Systems", to appear in IEEE Network Magazine.

[18] P. Frangoudis, L. Yala, A. Ksentini, and T. Taleb, "An architecture for on-demand service deployment over a telco CDN," in IEEE ICC'16, Kuala Lumpur, Malaysia, May 2016.

[19] K. Singh, Y. Hadjadj-Aoul, G. Rubino, " Quality of Experience estimation for Adaptive HTTP/TCP video streaming using H.264/AVC", in Proc. CCNC - IEEE Consumer Communications & Networking Conf., Las Vegas, United States, Jan. 2012.