Elastic Content Distribution Based on Unikernels and Change-Point Analysis

Polychronis Valsamas, Sotiris Skaperas and Lefteris Mamatas
Department of Applied Informatics,
University of Macedonia,
156 Egnatia Str, 54636, Thessaloniki, Greece,
{xvalsama, sotskap, emamatas}@uom.edu.gr

***Abstract*—The emerging 5G networks call for new approaches to CDNs through addressing challenging issues, such as: (i) scalable and holistic resource management, spanning from large data centers to the user device, including edge clouds; (ii) incorporation of heterogeneous physical and virtual resources; (iii) extensibility to support new capabilities or mechanisms; and (iv) adaptability to dynamic user requirements, server resources and network capacity constraints. User-generated content is the driving force for new services, while edge cloud solutions are being proposed to host (or cache) content locally. However, it is costly to deploy traditional clouds near end-users and such virtual machines (VMs) are inefficient for dynamic network conditions (i.e., may boot-up in minutes).**

Here, we propose an elastic content distribution platform that serves the Internet content using tiny Unikernel-based VMs. Such VMs are hosting one or a few videos each, appear rapidly in nearby cloud deployments, serve users and then disappear. In other words, the studied environment provides content dissemination through very dynamic, almost "fluid" VM placement, since the content is packaged with the server software with just a minor increase in size. So, we reposition the content caching and provisioning as a VM orchestration problem. We demonstrate the complete implementation of the proposed platform with proof-of-concept experimental results.

***Index Terms*—Content Delivery Networks (CDNs), Lightweight Clouds, Unikernels, Change Point Detection Analysis**

I. Introduction

The exponential growth of Internet content, in size, quantity, and network traffic demands, enabled network architectures realizing efficient hosting, discovery and dissemination of content, such as the Content Delivery Networks (CDNs) [1] [2]. CDNs are usually tightly coupled with cloud providers (e.g., Akamai [3], Mirror Image [4], Microsoft Azure [5], Amazon [6]) that use their own hardware, sometimes customized (e.g., NetApps FlexCase [7]). The CDN software may be proprietary, costly for SMEs and with specific hardware or OS requirements. Such approaches deliver transparently and efficiently content to the end-users. However, traditional CDN servers are typically far away from the content consumers and are unsuitable for the ultra delay-sensitive applications envisaged by the 5G networking initiatives [8]. Hence, we argue that there is a need for open, flexible, extensible, hardware-independent and resource-efficient CDN solutions hosting the content near to the users, and we suggest lightweight virtualization as an enabling technology for such unique features.

The main candidates for lightweight virtualization are the Containers and the Unikernels. The Containers (e.g., Docker [9] or LXC [10]) are standardized units implementing application packaging with all of its dependencies. Unikernels [11] (e.g., Mirage0S [12], Click OS [13], Rump Kernel [14], OSv [15]) are single-purpose VMs with only the essential part of the OS for the particular service, specialized at compile-time and sealed against modification when deployed in the cloud. The Unikernels have fewer MBs in size, boot up quicker and are more secure than Containers (e.g., the latter use shared kernel space). A Unikernel-based web server can even boot transparently in milliseconds with the reception of a DNS request packet [16]. A relevant thorough comparison between Unikernels and Containers can be found in [17]. For the above reasons, we suggest hosting Internet content using Unikernels. We named the latter Micro Content-Proxies (MCPs).

Along these lines, we propose a novel CDN platform, called Unikernel-Based CDN (UNIC), which provides efficient content caching through placing MCPs with popular content near the users. In comparison to traditional CDN solutions, UNIC provides the following advantages: (i) it can operate over heterogeneous hardware devices with diverse capabilities, including lightweight edge clouds; (ii) it provides significant elasticity capabilities through tiny VMs orchestration; (iii) it supports modular extensibility with new mechanisms; and (iv) it defines new research problems emerging from bringing together the content-caching approaches (e.g., [18], [19]) with the VM orchestration proposals.

Furthermore, UNIC supports the following novel features:

- modular orchestration of Unikernel-based VMs hosting replicas of Internet content (i.e., the MCPs), such as for configurable VM placement;
- content popularity changes detection mechanisms that drive the MCPs deployment based on a novel Change-Point Detection (CPD) methodology tailored to the specific problem;
- dynamic load balancing using a bespoke DNS service attached to the VM orchestration; and
- real-time monitoring of server resource utilization and end-user performance.

We designed and implemented UNIC in the context of the MONROE research project [20], [21]. MONROE provides

novel large-scale experimentation facilities and measurements of real buses, trains and tracks communicating over multiple mobile broadband providers. In this paper, we introduce UNIC and focus on its two core features: (i) the modular VM orchestration (e.g., placement); and (ii) the detection of content popularity changes. We tested our CPD methodology and experimented with UNIC using real youtube popularity measurements [22] to drive end-user content demand, as an approach to early detect changes in traffic and server resource utilization.

The UNIC platform applies CPD mechanisms [23] - [27] to provide an early signal of content popularity changes and deploy new MCPs. CPD is used extensively in network anomaly detection [28], [29], e.g., for intrusions detection [30] - [33]. We proposed a theoretical CPD methodology suitable for content popularity change estimation, aligned to the studied context. In our understanding, it is the only one applying CPD for content popularity in CDNs, and achieves promising results [34].

The efficient VM placement is a challenging issue in cloud computing. However, existing relevant proposals do not consider the high-dynamicity of Unikernels. For example, the survey paper [35] studies and categorizes a large number of existing VM placement approaches, but none considering Unikernels. UNIC is an ideal platform to experiment with such problems, since it supports modular VM orchestration, e.g., the definition of VM placement algorithms using short code snippets through a novel Graphical User Interface (GUI).

In our understanding, the only relevant CDN platforms to UNIC are [36], [37]. The MOSTO platform [36] deploys Unikernels as TCP proxies (i.e., to improve TCP's slow-start algorithm performance). An interesting CDN solution with impressive performance is [37], which provides content through Click OS Unikernels [13] and is evaluated with a CDN simulator [38]. In contrast to [37], which focuses on performance aspects, UNIC: (i) considers heterogeneity in terms of virtualization and Unikernel technology; (ii) conducts real experimentation; and (iii) achieves flexibility through Unikernel-oriented VM placement driven by novel early content popularity change detection. However, the two approaches could be potentially synchronized (i.e., support Click OS in our solution).

Last but not least, we provide our first proof-of-concept results, demonstrating the full system operation. We are currently extending our platform towards conducting experiments with the MONROE test-bed's mobile nodes and utilizing multi-homing capabilities of end-users.

The rest of this paper is organized as follows. Section II details the UNIC platform's architecture and highlights its core mechanisms. Section III presents our experimentation methodology and proof-of-concept results, demonstrating the full UNIC system operation. Finally, section V concludes the paper and highlights our future work.

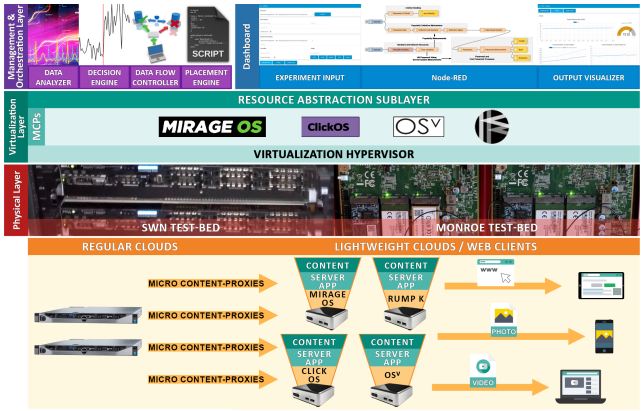

Fig. 1. The Architecture of the UNIC Platform

II. The UNIC Platform Architecture

UNIC is an intelligent lightweight cloud orchestration platform providing efficient content distribution to the end-users through MCPs scattered to a cloud hierarchy (i.e., with both regular and lightweight clouds). The UNIC platform realizes flexible and scalable content distribution over heterogeneous virtual and physical resources. We focus on two main UNIC aspects here: (i) the modular VM placement algorithms considering real-time server resource utilization and content provisioning requirements; and (ii) a novel approach to early content popularity change detection driving VM orchestration based on our CPD methodology, i.e., Cumulative Sum (CUSUM) procedures efficiently combining off-line and on-line CPD, mechanisms revealing the direction of changes and an improved time-series segmentation algorithm to detect multiple changes.

Here, we give a bottom-up description of the UNIC platform architecture (i.e., Figure 1), which consists of the following three main layers:

a) The *Physical Layer* utilizes federated hardware providing the required heterogeneity and scalability aspects. More precisely, we use both the MONROE and our own SWN test-beds(i.e., http://emulab.swn.uom.gr). The SWN test-bed provides the regular and lightweight cloud facilities and the MONROE test-bed hosts the end-users with challenging requirements (e.g., mobility, communication signal issues). The integration of the two test-beds enables the scalability aspect.

b) The *Virtualization Layer* supports lightweight cloud capabilities and multiple Unikernel technologies (e.g., Mirage OS, Rump Kernel or Click OS). The UNIC architecture is independent from the virtualization technologies used; hence, carefully designed abstractions hide the virtualization heterogeneity (i.e., the *Resource Abstraction Sublayer* exports a uniform interface for VM control).

c) The *Management and Orchestration Layer* controls and orchestrates the UNIC platform and test-beds, including providing the efficient VM placement through the *Placement Engine*, the traffic control through the *Data Flow Controller* and

network analytics enabling intelligent network configuration decisions through the *Data Analyzer* and the *Decision Engine*, respectively.

As shown in Figure 1, the *UNIC dashboard* provides the experimentation input, results visualization and modular extensibility of the evaluated mechanisms through the Node-RED tool [40].

An important UNIC aspect is the ability to implement VM orchestration processes in the form of Node-RED workflows, in a plug-and-play fashion. All the UNIC *Management and Orchestration Layer* components have been implemented in the form of Node-RED nodes, such as: (i) the content popularity detection for the *Decision Engine*; (ii) the VM placement functions for the *Placement Engine*; (iii) the content popularity and web client performance monitoring for the *Network Analytics*; and (iv) the traffic load balancing (i.e., our own dynamic DNS server matching content replicas with web clients) for the *Data Flow Controller*. These nodes are standalone components that can be manipulated / configured independently of each other and be connected to form complete VM orchestration processes.

To further analyze the UNIC platform, we describe two of its core features below, i.e., the content popularity changes detection and the modular VM placement mechanisms. In section III, we show our first experimental results regarding these two aspects, utilizing and demonstrating the full platform operation.

A. Content Popularity Changes Detection

The UNIC platform detects early changes in the content popularity and signals new MCP placements, in case of an upward qualitative change in the content popularity or a removal of MCPs in case of a downward change (i.e., handled from the VM placement algorithms of subsection II-B). Such decisions are being communicated to a dynamic DNS server assigning end-users to the active content caches, in a round-robin fashion.

We propose a novel statistical change point analysis methodology to approach the content popularity detection problem. Our mechanisms target the following requirements: (i) low-complexity and quick estimations to match the dynamic and resource-constraint nature of Unikernels; (ii) to rely on a non-parametric framework to avoid restrictive assumptions (i.e., no particular model or distribution can fit a 'mixed content' provisioning and a large number of model parameters may lead to high convergence times); and (iii) to operate in an on-line manner and be able to estimate the existence, the direction and magnitude of changes.

To address the above requirements, we detect the existence of a change in the historical sequence of content views' observations using a retrospective test statistic for the unknown time of change in the mean [40]. Such procedure consists of a CUSUM based detector and the Newey-West long-run variance estimator [41] to capture the serial dependence between observations. We combine the method [40] with a new heuristic incorporating the two binary segmentation algorithms [42], [43], to be able to detect multiple change points (i.e., through a segmentation of the time-series). We apply the Moving Average Convergence/Divergence (MACD) trend indicator [44] to estimate the direction of detected changes. More information on our CPD approach can be found at our presentation [34].

Regarding our on-line CPD mechanism, we implemented a stopping-time procedure [27], based on a mean CUSUM detector. The stopping rule depends on a sensitivity parameter $\gamma \in [0, \frac{1}{2})$. For example, a $\gamma \ll 0.25$ allows slower but more accurate detections, in terms of type I errors (i.e., incorrect rejection of the null hypothesis of no change) and $\gamma \gg 0.25$ leads to quicker but more sensitive to false alarms detections.

We outline our main content change-point detection algorithm, assuming that the monitoring period starts at the arbitrary time $t = m.s$, as follows:

- **Step 0:** Define *a priori* a finite monitoring window $l > 0$, and denote the monitoring horizon as $m.h = m.s + l$.
- **Step 1:** Apply the segmentation algorithm supplied by the retrospective statistic R_h for the whole historical period $h = [1, m.s]$, or for the bounded interval (i.e., to experiment with smaller monitoring periods): $h = [w, m.s],\ w > 0$,
  - if no changes are detected, the training sample of the sequential procedure becomes $m = h$,
  - else the training sample becomes $m = [cp_{last}, m.s]$, where cp_{last} is the detected time of change.
- **Step 2:** Apply the sequential procedure $S(m, k)$, $k \geq m.s$,
  - if a change is detected for some $m.s \leq k_{cp} \leq m.h$, the procedure stops,
  - else if no change is occurred after $m.h$ observations set $k_{cp} = 0$ and the monitoring terminates, i.e., proceed to **Step 4**.
- **Step 3:** If $k_{cp} \neq 0$, define k_{cp} as a change-point and apply the trend indicator at the time of change $TI(k_{cp})$,
  - if $TI(k_{cp}) > 0$ then deploy a Unikernel (i.e., upward change),
  - else, displace a Unikernel (i.e., downward change).
- **Step 4:** Set a new starting point for the monitoring period,
  - if $k_{cp} > 0$, set $m.s = k_{cp} + d$, where d is a constant value defining a period assuming no change,
  - else, set $m.s = m.h$.

We maintain two parallel change-point detection processes with different significance level α and parameter γ values. We place one more Unikernel, in case the change is detected from both processes, otherwise we may remove one or two, in a similar way. The MCP placements and removals are being handled from the algorithms described in the following subsection.

B. Modular VM Placement

As discussed above, UNIC supports modular extensibility of VM orchestration functionalities in the form of independent software entities, called Node-RED nodes. We exploited such capability to implement three alternative VM placement mechanisms: (i) a *Random Placement* algorithm, choosing randomly one of the available physical nodes and providing reference measurements; (ii) a *Quantity-Based*, choosing the physical node each time hosting the lower number of VMs; and (iii) an *Objective Weight Function (OWF)* approach, considering the real-time CPU, RAM and network utilization measurements of each node, weighted by particular coefficients (i.e., tuning the involved performance trade-offs).

Since the first two algorithms are self-explanatory, we detail the *OWF* algorithm only. We consider as $PN = \{PN_1, ..., PN_n\}$ the set of available physical nodes (PN) to host MCPs, where n is the PN number.

We define below the constraints for each $PN_i,\ 1 \leq i \leq n$,

$$0\% \leq CPU_{PN_i}, MEM_{PN_i}, TT_{PN_i}, RT_{PN_i} < 100\%$$

The variables CPU, MEM, TT and RT represent the percentage of CPU utilization (i.e., the average values of available CPU cores), memory allocation, outgoing and incoming link utilization, respectively.

The subset $APN = \{APN_1, ..., APN_m\} \subseteq PN,\ m \leq n$, denotes the PNs that satisfy the above constraints (i.e., they have available resources to host one more VM).

The *OWF* calculates periodically the following objective function, representing the resource utilization for each $APN_j,\ 1 \leq j \leq m$:

$$Node.Utilization_{APN_j} =$$

$$= \alpha CPU_{APN_j} + \beta MEM_{APN_j} + \gamma TT_{APN_j} + \delta RT_{APN_j}$$

The coefficients $\alpha,\ \beta,\ \gamma,\ \delta \geq 0$ weight the importance of each resource type, e.g., to match particular application-level requirements.

We place the VM on the one APN_j out of the APNs that provides the minimum resource utilization, given by:

$$\min_{APN_j} Node.Utilization_{APN_j},\ \forall j \in [1, m]$$

In section III, we experiment with the above three algorithms. The capability of UNIC to specify alternative placement algorithms allows it to act as a novel experimentation facility for more sophisticated algorithms. For example, we are currently working on alternative proposals considering the high-dynamicity of Unikernels.

III. Proof-of-Concept Experimental Results

Here, we describe our experimentation methodology, including the details and configurations of main platform implementation aspects, the test-bed used and the two experimentation scenarios we carried out. The section concludes with our proof-of-concept results demonstrating the full system operation and its capabilities for content popularity detection and dynamic VM placement.

A. Experimentation Methodology

We conducted real experiments that required the implementation and configuration of separate technical features, such as: (i) the VM orchestration; (ii) the content popularity detection mechanisms; (iii) the end-user traffic emulation and control; and (iv) the physical server resource utilization and end-user performance monitoring. We briefly outline each technical aspect below, i.e., configuration parameters, basic implementation details or open-source tools we used.

The VM Orchestration: We created lightweight web-servers delivering content with Mirage OS Unikernels. Such tiny VMs are being orchestrated according to a work-flow diagram created through the Node-RED tool. Such work-flow defines the communication of independent software entities, i.e., the Node-RED nodes. We created one node per VM orchestration process (e.g., the VM deployment, the placement decision making, etc). An experimenter can introduce new nodes (e.g., placement algorithms) or connect them in alternative ways (e.g., to create new orchestration work-flows). All processes communicate with the hypervisor through the *Resource Abstraction Sublayer (RAS)*, exposing a unified north interface to the orchestration features but virtualization-technology specific south interfaces. The latter communicate through ansible scripts [45]. *RAS* allows us to introduce heterogeneous virtualization technologies, in the near future.

The Content Popularity Detection Mechanism: We implemented the CPD mechanisms in Matlab. To ensure an online operation, we created at a separate host a Matlab TCP server application that receives periodic content popularity measurements and returns a notification for each detected change-point and an estimation of its basic characteristics (i.e., direction and rough magnitude). The other end is a Node-RED node that triggers VM deployments or removals.

The end-user traffic emulation and control: We emulated the web-clients using the httperf open-source tool [46]. We create and deploy web-clients based on real content popularity measurements extracted from youtube with the tool [22]. In Figure 2, we show the content requests per minute we used as an input for the emulation of web clients in these experiments. The duration of the particular measurements match the duration of the experiments, i.e., 310 minutes for each run. We created a DNS-based load-balancing Node-RED node that keeps track and redirects the web requests to particular content caches (i.e., Unikernel VMs), in a round-robin fashion.

The physical server resource utilization and end-user performance monitoring: We are monitoring the servers' resource utilization, in terms of CPU, memory, incoming / outgoing traffic and the performance of web-clients using the open-source tool CollectD [47]. We store the measurements in InfluxDB [48], a time-series database, and visualize them with the Grafana tool [49]. The measurements reach to the orchestration processes that take informed decisions for the context environment, e.g., to the VM placement algorithms. The monitoring takes place at regular time intervals (i.e., every 10 secs).

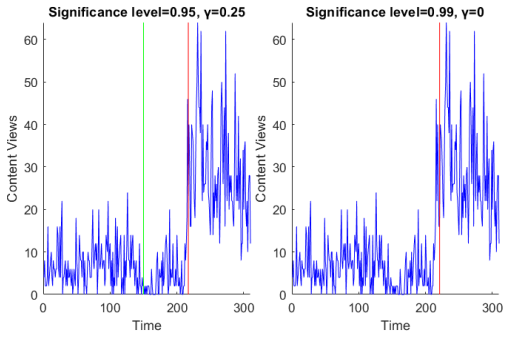

Fig. 2. Content-views per minute of a particular youtube video and detected change-points for different α and γ values. Red and green lines symbolize the upward and downward change respectively.

We grouped our experimental runs into two scenarios. The first scenario investigates the **impact of early content popularity change detection** driving by VM orchestration, while the second investigates the **impact of placement algorithms** utilizing real-time server resource utilization measurements.

We use our own SWN test-bed to conduct the experiments. More precisely we used: (i) seven physical servers, five to host the MCPs, one as a Management and Orchestration server and one to host the CPD mechanisms; (ii) ten Raspberry PIs to emulate the web-clients requesting web content from the MCPs; and (iii) one L3 100Mbps switch.

We detail the results of our two experimental scenarios below.

B. Experimental Results

We carry out two experimental scenarios, the first validating the impact of our change-point detection mechanisms and the second of the placement algorithm used. We evaluated both of them in terms of physical servers' resource utilization. For both scenarios: (i) we assume a running operation of UNIC with three MCPs hosting particular web content; and (ii) we allocated web-clients based on the real content popularity traces illustrated in Figure 2. We see there is a small change (i.e., reduction) in the end-user demand to watch the particular video at around the 150 minute of the experiment and a significant change after the 220 minute.

We apply two CPD processes in parallel with variable sensitivity, i.e., a more sensitive with parameters $\gamma = 0.25$, $\alpha = 0.95$ and a less sensitive with parameters $\gamma = 0$, $\alpha = 0.99$, as shown in Figure 2. In case both of them estimate a change point at the same time interval, we deploy two VMs, assuming a larger magnitude of change. In the typical case the sensitive approach detects a change but not the less sensitive one, we deploy one VM.

We set the *OWF* algorithm's coefficients $\alpha,\ \beta,\ \gamma,\ \delta \geq 0$ to the values 60%, 30%, 5%, 5%, respectively. In the following figures (i.e., 3 to 8), the different colors represent measurements from different physical machines.

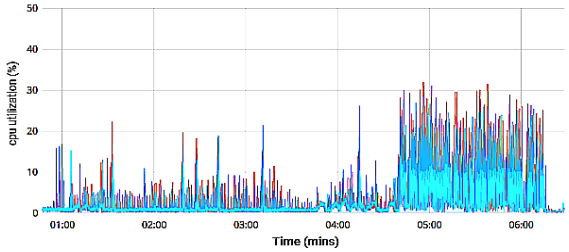

Fig. 3. The servers' CPU utilization with the change-point detection mechanisms disabled, using the Objective Weight Function placement algorithm

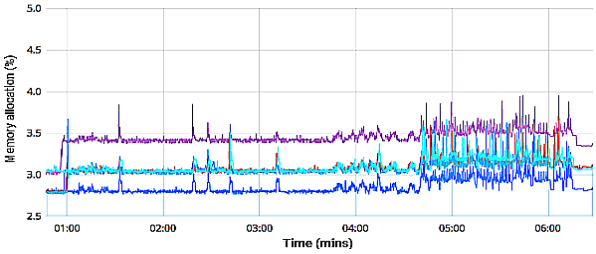

Fig. 4. The servers' memory allocation with the change-point detection mechanisms disabled, using the Objective Weight Function placement algorithm

Scenario 1 - Impact of the change-point detection mechanisms: To evaluate the impact of the change-point detection mechanisms, we run the experiment twice, one with the CPD mechanisms enabled and one with them disabled.

The Figures 3 and 5 contrast the percentage of CPU utilization and Figures 4 and 6 the percentage of memory allocation per physical server, with the change-point detection mechanisms disabled and enabled, respectively. According these figures, we have the following observations:

- for the first time period (i.e., 0 to 220 minutes), the CPU and memory allocation is similar for both cases.
- for the second time period (i.e., 220 to 310 minutes), the maximum CPU utilization was reduced around 10%, while there is a 0.5% increase in the memory allocation.

Such outcome can be explained as follows. In the second experimental run, the change-point detection mechanisms take the decision to boot two more MCPs due to the abrupt increase of content views (i.e., both CPD processes detect the change, as shown in Figure 2). This decision reduces the CPU utilization, but has a minor impact on the memory allocation (i.e., due to the additional VM deployment). This is consistent with the content popularity traces used for the web-clients deployment, dictated by the real measurements, where there is a change-point at the same time-period (i.e., see Figure 2). We note the smaller change-point, detected from the sensitive CPD process only, leads to the removal of an MCP, without significant impact on both CPU utilization and Memory.

Scenario 2 - Impact of the VM placement algorithm: Due to space constrains we illustrate two placement algorithms, the first using the *Random* and the second the *OWF*. The latter mechanism considers the real-time measurements of all

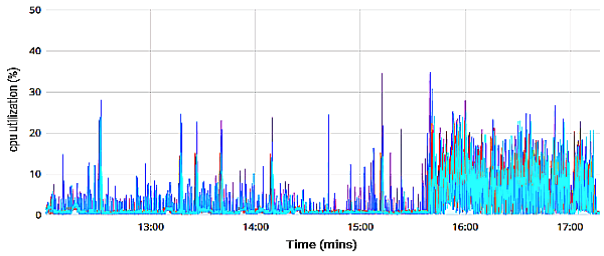

Fig. 5. The servers' CPU utilization with the change-point detection mechanisms enabled, using the Objective Weight Function placement algorithm

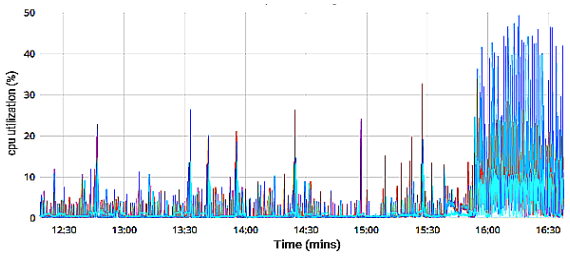

Fig. 7. The servers' CPU utilization with the change-point detection mechanisms enabled, using the Random placement algorithm

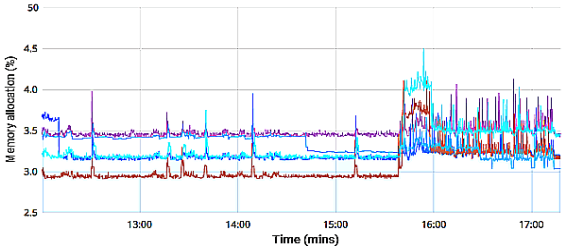

Fig. 6. The servers' memory allocation with the change-point detection mechanisms enabled, using the Objective Weight Function placement algorithm

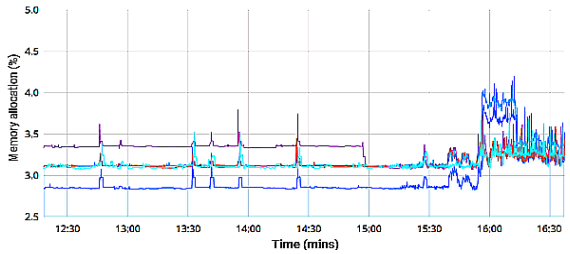

Fig. 8. The servers' memory allocation with the change-point detection mechanisms enabled, using the Random placement algorithm

available physical machines. In both cases the change point detection mechanisms are enabled.

Figures 5 and 7 highlight the impact of the placement algorithm on the CPU utilization, while Figures 6 and 8 the same impact on the memory allocation of physical servers. According to these four figures, we observe the following:

- for the first time period (i.e., 0 to 220 minutes), the CPU and memory measurements scale at the same low-levels, for both placement algorithms (i.e., there are not very many content requests, as shown in Figure 2).
- for the second time period (i.e., 220 to 310 minutes), the *OWF* placement algorithm causes the consumption of at least 10% less maximum CPU allocation, which even reaches the 30% at some points. This without significant changes in the memory allocation.

The above outcome can be justified by the choice of the *Random* placement algorithm to boot one additional MCP in a server that hosts other VMs as well. This result calls for further investigations in the sophistication of the placement algorithms.

IV. Conclusions and Future Work

In this paper, we introduced and detailed UNIC, our elastic Content Distribution facility based on Unikernel VMs, motivated by the unique requirements of 5G networks evolution (e.g., ultra low-delays, flexibility, programmability and heterogeneity). We particularly focused on two of its primary aspects: (i) its modular VM orchestration capability; and (ii) its novel mechanisms for content popularity detection. We provided proof-of-concept results demonstrating the above two aspects and its full system operation.

Our future work includes improvements in the main UNIC features and experimentation with new novel mechanisms, such as:

- **support of real mobile clients**. We are currently experimenting with web-clients in real buses and multi-homing capabilities over multiple mobile broadband providers, in the context of the MONROE project.
- **alignment to the Mobile Edge Computing paradigm**, e.g., through MCPs that appear dynamically at edge clouds near the mobile users.
- **network slicing support**. We consider, in the context of the NECOS project, a network slicing scenario with alternative content delivery services being offered (e.g., HTTP, video content and augmented reality services), assuming a different slice per type of service.
- **improvements in the VM placement algorithms**. We are currently working on more sophisticated placement algorithms considering the very-dynamic behavior of MCPs.
- **extensions in our change-point detection mechanisms**, e.g., their integration with stochastic models to support prediction of content popularity and more sophisticated ways to detect the magnitude of changes.
- **to synchronize with other content provisioning paradigms**, e.g., Information-Centric and Content-Centric Networking [50] research areas, content offloading to Mobile Edge Clouds solutions, etc.
- **to investigate complementary research problems**. UNIC can also validate intelligent algorithms for other research topics, e.g., network conditions forecasting, detection of anomalies or security issues, etc.

ACKNOWLEDGMENTS

This work is partially supported by the open call scheme of the H2020 MONROE (grant agreement number 644399) project and the H2020 NECOS (grant agreement number 777067) project. The views expressed are solely those of the author(s).

REFERENCES

[1] Pathan, Mukaddim, Rajkumar Buyya, and Athena Vakali. "Content delivery networks: State of the art, insights, and imperatives," Content Delivery Networks, Springer, pp. 3-32, 2008.

[2] Pathan, Al-Mukaddim Khan, and Rajkumar Buyya. "A taxonomy and survey of content delivery networks," Technical Report Grid Computing and Distributed Systems Laboratory (GRIDS-TR-2007-4), University of Melbourne, 2007.

[3] Akamai Technologies, Inc., www.akamai.com, 2007.

[4] Mirror Image Internet, Inc., www.mirror-image.com, 2007.

[5] Azure Microsoft Interntet, Inc., https://docs.microsoft.com/el-gr/azure/.

[6] Amazon.com, Inc., https://www.aws.amazon.com/.

[7] NetApp. NetApp FlexCache. http://www.netapp.com/us/products/storage-systems/flashcache/index.aspx, 2013.

[8] N.Alliance "5G white paper." Next generation mobile networks, white paper (2015).

[9] "Docker," Docker, 2016. [Online]. Available: https://www.docker.com/. [Accessed: 06-Apr-2016].

[10] "Linux Containers", LXC [Online]. Available: https://www.linuxcontainers.org/.

[11] A. Madhavapeddy and D. J. Scott, "Unikernels: Rise of the virtual library operating system," Queue, vol. 11, no. 11, 2013.

[12] Xen project, "MirageOS," 2016. [Online]. Available: https://mirage.io/. [Accessed: 30-Nov-2016].

[13] NEC, "ClickOS," A minimalistic, tailor-made, virtualized operating system to run Click-based middleboxes [Online]. Available: http://cnp.neclab.eu/clickos/. [Accessed: 03-Apr-2016].

[14] J. Cormack, "The rump kernel: A tool for driver development and a toolkit for applications."

[15] Cloudius systems, "OSv - the operating system designed for the cloud," 2016. [Online]. Available: http://osv.io/. [Accessed: 30-Nov-2016].

[16] A. Madhavapeddy et al., "Jitsu: Just-in-time summoning of unikernels," in 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), pp. 559-573, 2015.

[17] Manco, Filipe, et al. "My VM is Lighter (and Safer) than your Container." Proceedings of the 26th Symposium on Operating Systems Principles. ACM, 2017.

[18] Wang, Xiaofei, et al. "Cache in the air: exploiting content caching and delivery techniques for 5G systems," IEEE Communications Magazine, vol. 52, no. 2, pp. 131-139, 2014.

[19] Cho, Kideok, et al. "Wave: Popularity-based and collaborative in-network caching for content-oriented networks," Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on. IEEE, 2012.

[20] Alay, O., Lutu, A., Garca, R., Pen-Quirs, M., Mancuso, V., et al. "Measuring and assessing mobile broadband networks with MONROE." World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2016 IEEE 17th International Symposium on A. IEEE, 2016.

[21] Alay, O., Lutu, A., Ros, D., Garcia, R., Mancuso, V., et al. "MONROE: Measuring mobile broadband networks in Europe." Proceedings of the IRTF and ISOC Workshop on Research and Applications of Internet Measurements (RAIM). 2015.

[22] Zeni, Mattia, Daniele Miorandi, and Francesco De Pellegrini. "YOUStatAnalyzer: a tool for analysing the dynamics of YouTube content popularity." Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013.

[23] Aue, A., Hormann, S., Horvath, L. and Reimherr, M. "Break detection in the covariance structure of multivariate nonlinear time series models," The Annals of Statistics vol. 37, no. 6B, pp. 4046-87, 2009.

[24] Fryzlewicz, P. "Wild binary segmentation for multiple change-point detection," The Annals of Statistics vol. 42, no. 6, pp. 2243-2281, 2014.

[25] Hoga, Y., "Monitoring Multivariate Time Series," Journal of Multivariate Analysis vol. 155, pp. 105-121, 2017.

[26] A. G. Tartakovsky, A. S. Polunchenko, and G. Sokolov, "Efficient computer network anomaly detection by changepoint detection methods," IEEE Journal on Selected Topics in Signal Processing,, vol. 7, no. 1, pp. 4-11, 2013.

[27] Fremdt, S., "Asymptotic distribution of the delay time in Pages sequential procedure," Journal of Statistical Planning and Inference vol. 145, pp. 74-91, 2014.

[28] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," ACM computing surveys (CSUR), vol. 41, no. 3, article 15, 2009.

[29] A. K. Marnerides, A. Schaeffer-Filho, and A. Mauthe, "Traffic anomaly diagnosis in Internet backbone networks: a survey," Computer Networks, vol. 73, pp. 224-243, 2014.

[30] I. Nevat, D. M. Divakaran, S. G. Nagarajan, P. Zhang, L. Su, L. L. Ko and V. L. L.Thing , "Anomally detection and attribution in networks with temporally correlated traffic," IEEE/ACM Transactions on Networking, pp. 1-14, 2017.

[31] S. S. Kim and A. L. N. Reddy, "Statistical techniques for detecting traffic anomalies through packet header data," IEEE/ACM Transactions on Networking, vol. 16, no. 3, pp. 562-575, 2008.

[32] G. Thatte, U. Mitra, and J. Heidemann, "Parametric methods for anomaly detection in aggregate traffic," IEEE/ACM Transactions on Networking, vol. 19, no. 2, pp. 512-525, 2011.

[33] D. Turner, K. Levchenko, A. C. Snoeren, and S. Savage, "California fault lines: understanding the causes and impact of network failures," ACM SIGCOMM CCR, vol. 41, no. 4, pp. 315-326, 2011.

[34] S. Skaperas and L. Mamatas, "Change point detection for load balancing based on content popularity," 16th Mathematics of Networks (MoN) meeting, September 12, Sussex, 2017.

[35] Lopez-Pires, Fabio, and Benjamn Barn. "Virtual machine placement literature review," Polytechnic School, National University of Asuncion, Tech. Rep., 2015, Available: http://arxiv.org/abs/1506.01509.

[36] G. Siracusano, R. Bifulco, M. Trevisan, T. Jacobs, S. Kuenzer, S. Salsano, N. Blefari-Melazzi and F. Huici. "Re-designing Dynamic Content Delivery in the Light of a Virtualized Infrastructure," IEEE Journal on Selected Areas in Communications, vol. 35, no. 11, pp. 2574-2585, 2017.

[37] S. Kuenzer, A. Ivanov, F. Manco, J. Mendes, Y. Volchkov, F. Schmidt, K. Yasukata, M. Honda and F. Huici, "Unikernels Everywhere: The case for Elastic CDNs," in Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 17). ACM, New York, NY, USA, 2017.

[38] CDNSim - a stream-level simulator for large content delivery networks, https://github.com/cnplab/cdnsim

[39] Node-RED: http://nodered.org/.

[40] Horvth, L., Kokoszka, P., and Steinebach, J. "Testing for changes in multivariate dependent observations with an application to temperature changes". Journal of Multivariate Analysis,vol. 68, no. 1, pp. 96-119, 1999.

[41] Newey, Whitney K., and Kenneth D. West. "A simple, positive semi-definite, heteroskedasticity and autocorrelation consistent covariance matrix." National Bureau of Economic Research Cambridge, Mass., USA, 1986.

[42] Ju, Vostrikova L. "Detecting disorder in multidimensional random process." Soviet Math. Dokl. Vol. 24. 1981.

[43] Inclan, C., and Tiao, G. C. "Use of cumulative sums of squares for retrospective detection of changes of variance". Journal of the American Statistical Association, vol. 89, no. 427, pp. 913-923, 1994.

[44] G. Appel, "Become Your Own Technical Analyst," The Journal of Wealth Management, vol.6, no.1, pp. 27-36, 2003.

[45] Ansible:https://www.ansible.com

[46] Httperf HTTP load generator: https://github.com/httperf/httperf

[47] Collectd - The system statistics collection daemon: https://collectd.org/download.shtml

[48] InfluxData (InfluxDB) — Time Series Database Monitoring & Analytics: https://www.influxdata.com/developers/

[49] Grafana - The open platform for analytics and monitoring: https://grafana.com/

[50] Ahlgren, B., Dannewitz, C., Imbrenda, C., Kutscher, D., and Ohlman, B. "A survey of information-centric networking". IEEE Communications Magazine, vol. 50, no. 7, 2012.