Traffic-Aware Dynamic Container Migration for Real-Time Support in Mobile Edge Clouds

Sumit Maheshwari, Shalini Choudhury, Ivan Seskar and Dipankar Raychaudhuri
WINLAB, Rutgers University, North Brunswick, NJ, USA
{sumitm, shalini, seskar, ray}@winlab.rutgers.edu

***Abstract*—Edge clouds face challenges of resource assignment and load balancing due to variability of user location (mobility), server load and network state. Dynamic resource migration techniques are considered necessary to achieve load balance, fault tolerance and system maintenance objectives. Container migration is emerging as a potential solution that enables dynamic resource migration in virtualized networks and mobile edge cloud (MEC) systems. This paper proposes a traffic aware container migration approach and validates it with an end-to-end system implementation using a pure container hypervisor called LXD (Linux Container Hypervisor). The container migration model is then evaluated for real-time applications such as license plate recognition running in a mobile edge cloud scenario based on city-scale mobility traces from taxicabs in San Francisco. The system evaluation considers key metrics associated with application quality-of-experience (QoE) and network efficiency such as the average system response time and the migration cost for different combinations of load, compute resources, inter-edge cloud bandwidth, network and user latency. A specific compute resource and network-aware distributed resource migration algorithm called "ShareOn" is proposed and compared with alternative techniques using the San Francisco MEC model.**

***Index Terms*—QoE; Container Migration; Mobile Edge Computing; Virtualization; Real-time Applications; Edge Cloud;**

I. Introduction

Emerging cloud-assisted mobile applications – Augmented/Virtual Reality (AR/VR) involve intensive computation with real time response constraints. Mobile edge computing (MEC) [1], [2] is currently under active consideration as a promising approach which supports low-latency applications by bringing compute, network and storage close to the user. Edge cloud is generally deployed with limited compute resources to target local users. Thus, MEC is a distributed computing infrastructure that must respond to factors such as user mobility, fluctuating load, variability in network performance/congestion, and resource heterogeneity. Thus, maintaining user quality-of-experience (QoE) via distributed coordination between local edge cloud clusters is a challenge.

User QoE can be maintained in the MECs using a number of distinct mechanisms [3]–[6]. These include the use of either centralized or distributed resource assignment schemes which allocate computing servers to mobile user requests at nearby servers with available compute capacity. Resource virtualization (VM's and virtual networks) can also be used to partition and control resource use between multiple competing applications or users [7]. It is also possible to employ GPU's and/or parallelize computing resources across the network to accelerate the computation [8] or predict the network traffic [9]. Further, mobile user performance can be dynamically optimized via container migration in which the cloud process is moved from one computing node to another in response to mobility events and to load balance across the network. Container migration to be addressed in this paper implies moving a virtual machine (VM) or a container from one edge cloud to the other [10]. Virtualization based on containers allows users to run an application and its dependencies in an Operating System with flexible resource allocation, easy scaling and improved efficiency [11], [12]. Containers are gaining momentum due to their light running and deployment overhead, smaller start and stop time, size, and higher network bandwidth as compared to VMs [13], [14].

Container (LXD) [15]/Docker [16]) and VM migration are implemented in [17], [18]. In [19], [20], migration algorithms have been designed based on a limited set of parameters e.g. distance between edge cloud and user. Existing literature either studies VM or implements container migration without explicitly taking the container specific parameters e.g. dynamic resource allocation (available processing speed, RAM and bandwidth) and size, into account. The migration cost of a heterogeneous system is a complex combination of local, remote and network resources. System load, available processing resource and inter-node bandwidth affect the total migration time. Furthermore, considering above mentioned parameters to simulate container migration to test feasibility in a city-scale scenario is still unexplored.

This paper aims to develop a more general approach to container migration and to validate the proposed methods via simulation of a realistic MEC scenario. Thus, we propose ShareOn, a traffic-aware container migration algorithm using LXD and CRIU (Checkpoint Restore in Userspace) [21]. An end-to-end migration framework running real-time application has been deployed to analyze the impact of resource allocation, latency, bandwidth, size and migration time. A simulation model is also set up in which the containers are hosted in an edge cloud network running an automated license plate recognition (alpr [22]) application. Real traces from taxicabs in San Francisco [23] are used to model user mobility. Scalability of the system with respect to increasing traffic load is investigated using the above mentioned city-scale MEC model.

The rest of the paper is organized as follows. Section II

Research supported under NSF Future Internet Architecture - Next Phase (FIA-NP) Award CNS-134529

978-1-5386-8134-3/18/$31.00 ©2018 IEEE

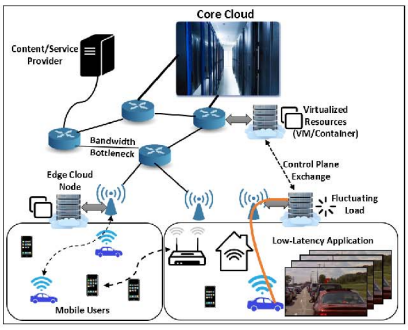

Fig. 1. General MEC System

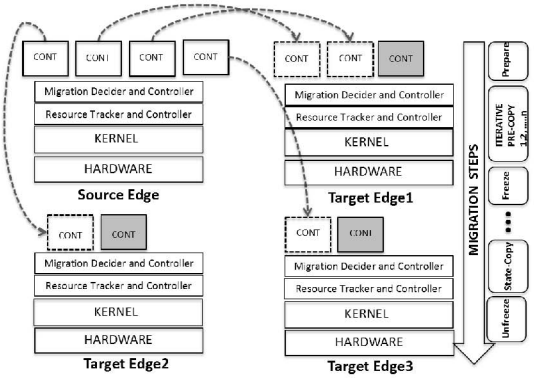

Fig. 2. Container Migration Flow Diagram

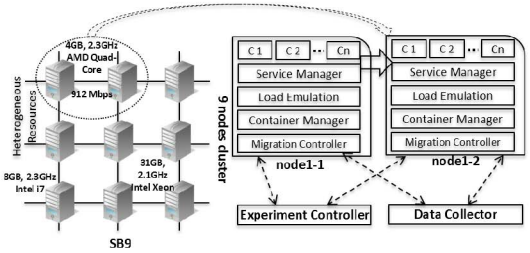

Fig. 3. Container Migration Set-up in ORBIT

highlights the need of container migration in MEC. Section III describes the container migration system with specific details on the migration flow and the emulation set-up. Section IV introduces the simulation model and its parameters. Simulation scenarios and analytics are detailed in Section V. Results are discussed in Section VI and Section VII concludes the paper.

II. MEC and Container Migration

The architecture of MEC is based on three layers of computing, the first at the mobile client, the second at a local network attached edge cloud, and the third at a centralized data center/cloud in the core of the network. As shown in Fig. 1, this architecture offers the advantage of low latency response to real-time applications which cannot tolerate a typical edge-to-core round trip delay that typically exceeds ∼100 ms.

With the help of flexible resource provisioning and sharing among neighboring edge cloud nodes, MEC can meet the unpredictable traffic demands and quickly scale the network due to its multitenancy feature. Further, in order to reduce the application latency and to provide the required user QoE, service requests are handled by the resource virtualized environment. Container-based virtualization is finding increasing adoption in MEC systems to realize slice isolation and fine resource control. Resource isolation (especially, memory) across components of different applications is necessary for the integrity of individual applications. The light-weight containerized resources can be shared with neighboring nodes using migration techniques [24]. The dynamic container migration approach therefore can be used to address deteriorating user QoE, arising from the processing latency (system load) and/or the network latency (user mobility). In the rest of this paper, we describe and validate a container migration system suitable for MEC scenarios with latency constrained applications.

III. Container Migration System

Containers are hosted on a physical shared resource enabling service as well as user level virtual separation. Migrating these containers in a controlled environment allows to monitor and analyze physical resource usage such as memory, processing, and network. This section details our end-to-end container migration emulation system underlining migration flow, system details and a method to assess migration cost.

A. Flow Diagram

We start with an outline of the software stack at each MEC computing node. The migration process has two entities, a source, which is the host that initially has the container, and a sink, the container receiver. The prepare phase is succeeded by iterative pre-copy, freeze, state-copy, unfreeze and post-copy phases. Fig. 2 illustrates the distributed process flow by placing control and decision logic at each of the node. The resource tracker and controller at the source node assess the network and neighboring node resources. The migration decider determines best destination node while the controller initiates the selected container migration.

B. System Details

Our set-up is deployed at the SB9 (sandbox9) in the ORBIT [25] testbed (Fig. 3) which enables a software based emulation. We use SSH tunneling to connect to the edge cloud nodes. The core of LXD is a privileged daemon, which provides a REST API over a local UNIX socket and the network. Container orchestrated with Shell and Python scripts, allows us to run alpr remotely and to emulate users. CRIU does container checkpoint and restore on the host as a snapshot.

Container Creation: The primary sockets used in container migration are: control stream, CRIU images stream and filesystem stream. LXD supports creating/managing bridges, IPv4 address, NAT (Network Address Translation) and DHCP (Dynamic Host Configuration Protocol) range. Hence, before setting up the container we set-up LXD for storage and networking needs e.g. daemon settings, storage pools, network devices and profiles. Hostname is pre-added to the LXD group and the LXD tools are pre-installed to the destination edge with valid keys.

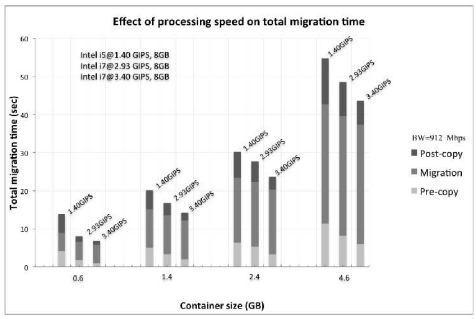

Fig. 4. Impact of Processing Speed, Container Size on Total Migration Time

Migration Phases: There are three phases of migration: decision, initiation and completion. During the decision phase, a node accumulates resource, network, and application QoS information. A low overhead control plane protocol is used to exchange both routing and computing state information between edge cloud clusters in the region. Upon selecting the container to be migrated, neighboring edge clouds are queried using an extended inter-domain protocol such as EIR (edge-aware interdomain routing) [26] which has the above mentioned features. The neighboring nodes then respond with their utilization level: high, medium or low. A suitable destination node is chosen and the migration is initiated. Container state is then copied and traffic switched. On migration completion, the source node discards the old container states.

Application Details: The alpr is used to detect license plates of cars. The frames are obtained from the UE (User Equipment) video stream. After processing these frames in a container, the output is possible plate numbers with the set confidence level. The application phases include detection, binarization, char analysis, plate edges, character segmentation, OCR (Optical Character Recognition) and post processing.

C. Assessing Migration Cost

We developed an experiment to measure migration cost parameters: pre-copy, migration, and post-copy time, with respect to machine type, network bandwidth and container size for alpr. In order to assess migration requirement and cost, different size containers (0.6-4.6GB) are exchanged between two similar test nodes by varying the processing speed. Fig. 4 shows the total migration time (tmt) which includes pre-copy, migration and post-copy time. The migration time for fixed sized containers and given inter-edge bandwidth remains same. The pre-copy and post-copy time are inversely proportional to the processing speed since it is also shared by the containers for the application specific computations.

IV. Modeling Container Migration

We escalate the emulation set-up to carry out a large scale simulation model for a continuous container migration approach. To evaluate the efficiency of our work we choose San Francisco city as the geographical location with nine nodes spaced out across the city to deploy the edge cloud network as shown in Fig. 5. Real SFO taxicab traces are used from the heavy traffic routes across this location.

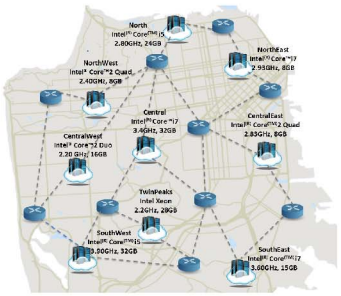

Fig. 5. Edge Cloud Network Topology

A. Simulation Parameters

The parameters used in simulation are described below.

- Page dirty rate (r_{pd}): Memory pages modified per second
- Page size (s_{page}): Default size of a memory size
- Processing speed (s_p): Processor speed in GIPS
- RAM (m): Random access memory in GB
- Inter-edge bandwidth (b_i): BW between two edge nodes
- Network latency (t_n): User to edge cloud network delay
- Queuing latency (t_q): The wait time of request at an edge cloud node defined using M/M/c queuing model where c are the number of servers in an edge cloud node (cluster).
- Processing latency (t_p): Computation delay at a node
- Control latency (t_c): Control plane delay between nodes
- Total response time (t_{total}): Sum of t_n, t_q and t_p
- Edge cloud load ($load$): Current load at an edge node
- Container size (s_{con})
- Containers-up (k): No. of running containers at a node
- Total migration time (tmt): Sum of pre-copy, migration and post-copy time

B. Migration Cost

For a given container to be migrated, the cost is a function of pre-copy time, post-copy time and migration time. Therefore, the total migration cost is defined as follows:

$$C_m = \sum_{j=1}^{N} \left[\frac{(k_{in,j} + k_{out,j}) * r_{pd,j} * s_{page,j}}{s_{p,j} * (1 - load_j)} \right] + \sum_{j=1}^{N} \sum_{\substack{l=1 \\ (l \neq j)}}^{N} \left[\frac{k_{in,j,l} * r_{pd,j} * s_{page,j}}{b_{i,j,l}} \right] \quad (1)$$

In the Eq. 1, the first part denotes the compute time and the second part estimates the migration time due to inter-edge bandwidth, where N is the number of edge cloud nodes, and k_{in} and k_{out} are the number of incoming and outgoing containers from the given node respectively. The objective is to minimize system migration cost as well as average latency (application). In a continuous migration process, multiple container migration time overlaps which can be optimized by minimizing the time of maximum cost migration as follows:

$$\min . \left[\max . \left\{ \frac{(k_{in,j} + k_{out,j}) * s_{page,j} * r_{pd,j}}{s_{p,j} * (1 - load_j)} + \sum_{\substack{l=1 \\ (l \neq j)}}^{N} \frac{k_{in,j,l} * s_{page,j} * r_{pd,j}}{b_{i,j,l}}, \forall j \in [1, N] \right. \right. \quad (2)$$

C. ShareOn: Migration Decision Algorithm

Migration is a two way process. First, the source node selects containers based upon their resource usage and application performance. Second, the source node determines *best* available destination node based upon its utilization region, estimated application performance and the available inter-edge bandwidth. The utilization region, $util$, captures the node specific parameters such as load, available CPU and RAM. Estimating application performance at the destination node is challenging because of network variability, fluctuating node utilization and the application type. Also, the number of migrations between target nodes affect the available inter-edge bandwidth and therefore, the migration decision must take this into account.

The average processing per container at a node is $s_{p,avg} = s_p/k$ and the average RAM is $m_{avg} = m/k$. The utilization region of a node can be found using algorithm 1 which defines High, Med or Low zones by capturing available processing speed and RAM, and comparing them with pre-defined values needed to run an application.

Algorithm 1: Utilization Region of a Node

```
function findUtil(s_{p,avg}, m_{avg});
Output: High, Med, Low
if s_{p,avg} ≤ s_{p,th1} || m_{avg} ≤ m_{th1} then
    return High;
else
    if (s_{p,avg} > s_{p,th1} || m_{avg} > m_{th1}) AND
     (s_{p,avg} ≤ s_{p,th2} || m_{avg} ≤ m_{th2}) then
        return MED;
    else
        return LOW;
    end
end
```

The network latency, t_m, between the user and the destination edge cloud node can be estimated as $d_{u,e} * l_d + var(l)$ where $d_{u,e}$ is the distance between the user and the destination edge node, l_d is the network latency per unit distance and $var(l)$ is the past moving average latency variance of the user geographic area and the destination node. The processing latency, t_p, of a destination node can be approximated as $t_{p,avg} + utilFac * \alpha_e$ where $utilFac$ is the current utilization factor of a node, and α_e is the latency factor associated with the current utilization.

Migration gives rise to the computation and network overheads. Therefore, the decision algorithm has to adequately determine whether, when and where to migrate depending on aspects such as application QoS, user mobility, inter-edge bandwidth, and resource availability at MECs. Considering these parameters, we design ShareOn, a dynamic container migration technique which uses traffic (number of requests per second at a node) as a primary metric. Each node tracks its own compute and networking resources, and is aware of the application latency of each request. These nodes have control plane connectivity to each other and hence can query the neighboring nodes periodically for the above mentioned parameters. Thus, in our system, each node can decide, control and migrate its containers in a distributed manner.

ShareOn works as follows. First, a node shortlists high total application latency containers and determines the primary reason — high processing latency, networking latency or both. Second, for each container, a few suitable neighbors are selected using two conditions: (a) falls in low or med $Util$ regions and/or (b) geographically closer to the user. The above procedure enlists all containers with their suitable neighbors. Finally, a destination is chosen based upon number of migrations from the source to the destination considering the available inter-edge bandwidth and compute. The complete procedure is described in Algorithm 2.

Algorithm 2: ShareOn: Traffic Aware Container Migration

```
function findDestEdge(t_{total,m}, Util_j, b_{i,j,l});
Output: List of containers and selected nodes
for all nodes findUtil(j) MED or LOW;
STORE favorableUtils;
if t_{total,m} > t_{app,th} AND t_{total,est,j} < t_{total,m} then
    return favorableUtils_m AND selectContainers
else
    return NULL
end
for all selectContainers AND favorableUtils_m;
ShareOn(input)
function ShareOn(b_{i,j,l}, selectContainers);
Output: DST_m for a container m;
if C_m < C_{m,th} then
    return DST_m
else
    return NULL;
end
```

D. Result — Emulation

The emulation based total application delay is shown in Fig. 6 to observe the effect of server load on the container migration and the application performance. ShareOn is compared with a case when there is no migration. The load is varied from 0 to 1 for node1 and 1 to 0 for node2 in steps. When the load reaches 0.4 for the node1, the total application delay crosses a set threshold (100ms in this case), triggering migration. Node2 is chosen using ShareOn which shows a drop in the total delay upon migration completion.

V. Simulation Set-up

We develop a simulation set-up similar to the emulation model described in the Section III, to test the scalability in a large geographical region such as San Francisco city with real mobility traces. The topology is same as shown in Fig. 5. This section details the simulation scenario and the numerical values for the parameters described earlier.

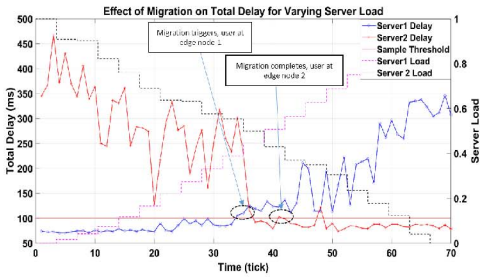

Fig. 6. Baseline (no migration) vs. ShareOn

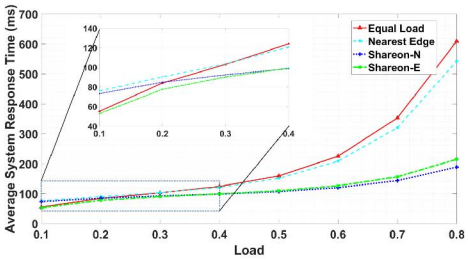

Fig. 7. Average System Response Comparison for Static Allocations and Migrations

A. Simulation Scenario

Initially, we determine the system performance without considering migration, using following approaches.

Equal-Load (E): The user requests are equally divided among the heterogeneous edge cloud nodes, routing a set number of requests based upon user vicinity. The remaining requests are routed to the next closest node and so on.

Nearest-Edge (N): The user requests are always routed to the closest node irrespective of the node's current load.

Migration is simulated using the approaches listed below.

Bandwidth-only: The users are connected at the nearest edge cloud node and the migration is done based on the application QoS and available inter-edge cloud bandwidth.

Processing-only: Following the above process, the migration is done based on the application QoS and the destination node resources (processing speed).

In all the cases, user mobility is introduced and load is varied by injecting multiple requests per taxicab. The delay induced by the control latency, t_c, is negligible and is omitted in the simulations. Our proposed approach, $ShareOn$, can be instantiated from the nearest or the equal load scenario considering their first connected request as the initial state.

B. Simulation Parameters

The numerical values for the simulation are listed in Table I. There are total 536 taxicabs in the city with a known mobility pattern. The load is varied from 0 to 1 by initiating multiple requests from a taxi. The processing speed determines the page dirty rate which allows us to push most of the memory pages to destination before suspending the container at the source.

TABLE I
SIMULATION PARAMETERS

Parameter	Value	Parameter	Value
s_p	2.2-3.9 GIPS	b_i	10-100 Gbps
r_{pd}	2.5-4 kpps	s_{page}	4-64 KB
m	8-32GB	$load$	0-1
s_{con}	0.6-4.6 GB	#taxi	536

VI. Results and Discussion

This section presents the results obtained from the simulation model introduced in the previous section.

A. System Performance

Fig. 7 compares our migration approach, ShareOn with no migration cases. In the equal load case, the average system response time at load 0.1 is low as compared with the other approaches since the service requests have been equally distributed among nine edge cloud nodes. However, as load gradually increases, the average system response time starts degrading since without any optimization this approach cannot handle the volume of requests with the available resources at each node. While in the case of the nearest edge (users connect to the closest MEC) the average system response time is prominent. The reason for this rise is that in the real-time mobile taxicab trace, many of the users connect to the North-East edge cloud node (ref: Fig. 5) because of their close physical proximity to that node when they are introduced into the system. The limited resources at that node are unable to support all the connected users and hence the system enters into overload resulting in a large average system latency.

Using ShareOn-E, initially there are not many migrations since all the requests are well distributed between nodes. On increasing the load, the system response is well below the non-migration approaches, as the load gets efficiently distributed across the geographical regions. This is due to ShareOn taking processing capability, inter-edge bandwidth and network latency of each node into consideration for migrations. In the case of ShareOn-N, the users are initially connected at the nearest edge cloud where the available resources are exhausted resulting in a slight increase in the migration cost.

B. Migration Cost

Fig. 8 presents the migration cost (C_m) for different system load. The maximum C_m is the peak migration time recorded for the system at a given load. Similarly, minimum C_m is the least while the average C_m is mean time. The total number of containers migrated are not linearly proportional to the load. At a higher load, some containers are omitted even though the latency threshold is met for the migration due to scarce compute and networking resources. Therefore, even though the number of migrating containers rise with the load, a drop in the average migration cost can be observed, as seen at load=0.3.

C. ShareOn vs. Other Approaches

Fig. 9 compares ShareOn with bandwidth-only and processing-only migration methods. The former performs better than the later at lower loads as the migration time dominates the pre-copy and post-copy time for the fewer container migrations in these cases. As the bandwidth-only approach keeps track of bandwidth before initiating migration, the average system latency does not suffer from the migration

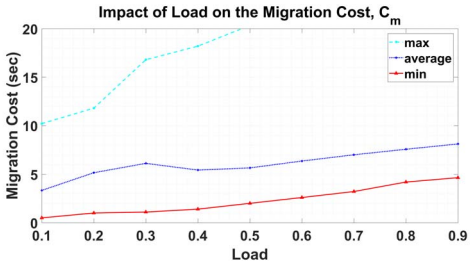

Fig. 8. Effect of Load on the Migration Cost

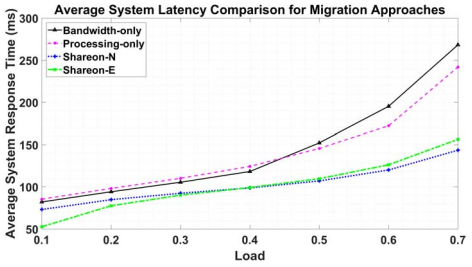

Fig. 9. Average System Response Comparison for Different Migration Approaches

time factor. For the higher load scenario, checking bandwidth only is not sufficient as the migration time dominates the pre-copy and post-copy time, thereby increasing the total system latency. In either case, ShareOn performs significantly better than these methods. The project code is available at [27].

VII. Conclusion

This paper has proposed dynamic container migration as a mechanism for supporting user-mobility, server load and network fluctuations. A traffic-aware container migration method is developed using a testbed based set-up, to emulate an edge cloud network and user mobility. The migration cost is evaluated by running a real-time application and a distributed migration algorithm. Using parameters obtained from the emulation, a large-scale simulation model for migration is developed incorporating real taxicab traces from San Francisco city. A heuristic traffic-aware container migration scheme, ShareOn, is proposed which considers multiple parameters: application QoS, network latency, edge cloud resources, and inter-edge bandwidth. The system performance of ShareOn is compared with two non-migration approaches: equal-load and nearest-edge, and two migration-based approaches: bandwidth-only and processing-only. From our experiments, we make following observations: (a) migration is a viable approach when sufficient computation (at source and destination) and inter-edge bandwidth are available, (b) a low-load scenario incurs substantial migration cost while there is no significant drop in the average system response time as compared to no-migration approaches, and (c) processing-only and bandwidth-only approaches fail to lower the average system response time at higher load as compared to the the multi-parameter ShareOn approach. For future work, we plan to evaluate alternative network and compute-aware algorithms in an edge-cloud enabled large-scale outdoor testbed such as COSMOS [28], and further use prediction based models to improve QoE.

References

[1] Hu, Yun Chao, et al. "Mobile edge computingA key technology towards 5G." ETSI white paper 11.11 (2015): 1-16.

[2] Maheshwari, S., et al. "Scalability and Performance Evaluation of Edge Cloud Systems for Latency Constrained Applications." Proceedings of the Third ACM/IEEE Symposium on Edge Computing. ACM, 2018, 286-299.

[3] Garcia L., P., et al. "Edge-centric computing: Vision and challenges." ACM SIGCOMM Computer Communication Review 45.5 (2015): 37-42.

[4] Westphal, C., "Challenges in networking to support augmented reality and virtual reality." Int. Conf. on Computing, Networking and Communications (ICNC 2017). 2017.

[5] Dinh, Hoang T., et al. "A survey of mobile cloud computing: architecture, applications, and approaches." Wireless communications and mobile computing 13.18 (2013): 1587-1611.

[6] Dasari, M., et al. "Impact of Device Parameters on Internet-based Mobile Applications." Proceedings of the 2018 Internet Measurement Conference. ACM, 2018.

[7] Bronzino, F., et al. "NOVN: Named-Object Based Virtual Network Architecture." Proceedings of the 20th International Conference on Distributed Computing and Networking. ACM, 2019.

[8] Zhao, Z., et al. "Game cloud design with virtualized CPU/GPU servers and initial performance results." Proceedings of the 3rd workshop on Scientific Cloud Computing. ACM, 2012.

[9] Maheshwari, S., et al. "A joint parametric prediction model for wireless internet traffic using Hidden Markov Model." Wireless networks 19.6 (2013): 1171-1185.

[10] Ma, Lele, et al. "Efficient service handoff across edge servers via docker container migration." Proceedings of the Second ACM/IEEE Symposium on Edge Computing. ACM, 2017.

[11] Vaughan-Nichols, et al. "New approach to virtualization is a lightweight." Computer 11 (2006): 12-14.

[12] Maheshwari, S., et al. "Comparative Study of Virtual Machines and Containers for DevOps Developers." arXiv:1808.08192 (2018).

[13] Pahl, C., et al. "Containers and clusters for edge cloud architectures–A technology review." Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on. IEEE, 2015.

[14] Tay, Y. C., et al. "A Performance Comparison of Containers and Virtual Machines in Workload Migration Context." Distributed Computing Systems Workshops (ICDCSW), 2017 IEEE 37th International Conference on. IEEE, 2017.

[15] The LXD container hypervisor. Retrieved January 20, 2018, from https://www.ubuntu.com/containers/lxd.

[16] Docker. (2018). Docker. [online] Available at: https://www.docker.com/ [Accessed 25 Jul. 2018].

[17] Nadgowda, S., et al. "Voyager: complete container state migration." Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on. IEEE, 2017.

[18] Machen, Andrew, et al. "Live service migration in mobile edge clouds." IEEE Wireless Communications 25.1 (2018): 140-147.

[19] Wang, S., et al. "Dynamic service migration in mobile edge-clouds." IFIP Networking Conference (IFIP Networking), 2015. IEEE, 2015.

[20] Ghribi, C., et al. "Energy efficient vm scheduling for cloud data centers: Exact allocation and migration algorithms." Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on. IEEE, 2013.

[21] CRIU. Retrieved January 20, 2018, from http://www.criu.org/

[22] Openalpr/openalpr. Retrieved January 20, 2018, from https://github.com/openalpr/openalpr/wiki/OpenALPR-Design

[23] Piorkowski, M., et al. CRAWDAD dataset epfl/mobility. [online] Crawdad.org. Available at: https://crawdad.org/epfl/mobility/20090224.

[24] Clark, C., et al. "Live migration of virtual machines." Proceedings of the 2nd Conference on Symposium on Networked Systems Design Implementation-Volume 2. USENIX Association, 2005.

[25] Open-Access Research Testbed for Next-Generation Wireless Networks (ORBIT). Retrieved April 12, 2018, from http://www.orbit-lab.org/.

[26] Mukherjee, S., et al. "EIR: Edge-aware inter-domain routing protocol for the future mobile internet." Computer Networks 127 (2017): 13-30.

[27] Maheshwari, S., (2018, October 08). Sumitece87/shareon. Retrieved from https://github.com/sumitece87/shareon

[28] Cosmos-lab.org. (2018). COSMOS Project. [online] Available at: http://www.cosmos-lab.org [Accessed 3 Aug. 2018].