Received 27 October 2021; revised 5 December 2021; accepted 27 December 2021. Date of publication 6 January 2022; date of current version 4 February 2022.

Digital Object Identifier 10.1109/OJCOMS.2022.3140272

Optimal Container Migration/Re-Instantiation in Hybrid Computing Environments

SAM ALEYADEH[1], ABDALLAH MOUBAYED[1] (Member, IEEE), PARISA HEIDARI[2], AND ABDALLAH SHAMI[1] (Senior Member, IEEE)

[1]Department of Electrical and Computer Engineering, Western University, London, ON N6A 3K7, Canada

[2]IBM Client Engineering, IBM Canada, Montreal, QC H3B 4M7, Canada

CORRESPONDING AUTHOR: S. ALEYADEH (e-mail: saleyade@uwo.ca)

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) through NSERC Strategic Partnership Grant STPGP under Grant 521537.

ABSTRACT End-users and service providers have recently favored lower latency services. Edge computing has improved user quality of service (QoS) guarantees through the reduced geographical distance, decreased use of the network backbone, and flexible placement of the container hosting edge devices. The under-utilized isolated edge computing idling nodes are significant for service providers, especially for Internet of Things (IoT) applications. However, the nodes' minimal maintenance remains a hindrance due to related increased failures. Orchestrating over the edge alongside core environments allows tolerant services and more demanding ones to coexist without impacting the user experience. Therefore, the orchestrator's second priority is achieving and maintaining the QoS through optimal recovery method selection by either migrating the live containers or re-instantiating them. This paper proposes an Optimal Container Migration/Re-Instantiation (OC-MRI) model to optimize the orchestration methods focusing on downtime, container dependencies, and latency requirements. Next, we introduce a real-time heuristic-based solution, Edge Computing-enabled Container Migration/Re-Instantiation (EC2-MRI). Both models are bench-marked alongside staple greedy approaches. Simulation results showcase the lowest latencies and downtime with the OC-MRI model. Furthermore, the EC2-MRI model shows comparable results to the optimal model with minimal lag.

INDEX TERMS Container orchestration, hybrid computing, integer programming, migration/re-instantiation.

I. INTRODUCTION

THE RECENT rapid adoption of 5G networks significantly increased the pre-existing interest in edge computing. This is mainly due to the 5G paradigm's readiness for rapid network changes coupled with its ability to accommodate the increasing number of users and generated traffic. Within 5G, edge computing allows providers to shift their services away from the core cloud and towards end-users by utilizing the abundant resources found in pre-existing under-utilized edge communication devices. This shift further lowers latencies and offloads traffic from the network's main backbone. Real-time dependent applications, such as 4K streaming, instant speech translation, and intelligent transportation systems may then run seamlessly [1]–[3]. The push to adopt edge computing has generated a significant amount of literature targeting the optimization of its main attributes with more focus on latency and energy costs, capital (CAPEX), and operational expenditure (OPEX) [4].

The concept of containers was proposed to further build on the premise of edge computing. Container-type solutions were introduced to replace traditional virtual machines (VMs) to increase the system's flexibility and scalability further. This change is due to the containers' ability to coexist in a shared platform allowing more services to run with higher efficiency and mobility. Consequently, containers have decreased CPU requirements compared to VMs that require hardware stack virtualization. Containers can also recover more quickly and require a significantly lower share of the system's memory without requiring an entire OS image. These attributes allow for microservices to be hosted on

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

less sophisticated units. In contrast, robust computing units can be placed in the vicinity of end-users in the form of roadside units (RSU), base stations, and routers.

Containers hosted on the edge require additional constraints over core cloud-hosted containers to perform their intended tasks properly, as edge devices are typically less accessible and maintained less frequently. The containers must then achieve high robustness through other methods instead of relying on heavy hardware-based safeguards and traditional data center-based setups typically associated with core clouds [5]. This exposes the containers to more frequent outages, both planned (maintenance and updates) and unplanned (hardware failures and overloads) [6]. Redundant copies may be implemented to tackle this issue. However, this may affect the efficient utilization rates and system size, which are the advantages of using these containers. Thus, the desired approach is to maintain efficiency while lowering downtime when recovering, which lessens the impact on quality of service (QoS) once a failure is detected. This is vital in the edge environment due to the high availability requirement typical of its services, from urgent services, such as an emergency response, to standard services, such as IoT-based manufacturing and smart-city management. Traditionally, containers were allowed to either migrate from the failing edge node or re-instantiate by allowing the edge node to return to online status. Migrating with the anticipation of a failure serves to maintain their last state. The downtime will last until a new viable host is determined and the live capture image of the container is transferred. Conversely, the downtime in re-instantiation is dependent on the time a failed edge node needs to perform a hard reset and re-establish the connection with the end-user. This can be done by using the archived stateless image of the container instead of recovering the current data in the container.

An intelligent orchestration paradigm to decide between migration and re-instantiation is needed to achieve an optimal solution for placement while lowering downtime. Choosing the method depends on several network-related constraints to reduce global downtime. The decision process must also address the type of hosted services within the container and its compatibility and affinity towards migration and re-instantiation. For example, the migration should allow the system to recover seamlessly and maintain a high service up-time for a container with a stateful application. Comparatively, re-instantiation techniques are used when it is not necessary to preserve the application states [7]. The recovery method is not controlled solely by the application persistence requirement. Other metrics must be addressed, such as the end-user experience, cost, and security concerns. This paper introduces a novel container orchestrator based on an integer linear programming optimization model to address the challenges of traditional orchestration and find the optimal placement with minimal downtime in hybrid computing environments. In addition, due to the time complexity of the optimization model, this work introduces a comparably accurate heuristic model capable of achieving near-optimal results at a fraction of the time and complexity of the optimization model, thereby allowing it to provide real-time orchestration. The presented solutions are invoked once a failure is detected or deemed imminent. First, the host captures a snapshot of all hosted containers and generates a list of their given resources. It then provides a list of dependencies between the containers and similar constraints. A new placement is then generated using the decision, either migration or re-instantiation, to minimize the containers' downtime along with the access delay latency. To this end, the proposed approach enhances the QoS by minimizing the container downtime and satisfying the carrier-grade requirements of the provided services, namely availability and performance. The main contributions of this work are summarized as follows:

- Formulate the problem of container migration vs. re-instantiation while considering edge-related placement requirements such as latency and downtime.
- Develop an intuitive clustering method to generate representative user clusters capable of reducing solution space-related problems for optimization and heuristic-based solutions.
- Propose a real-time heuristic model orchestrator capable of providing comparable results to the optimization model.
- Evaluate the performance of the proposed optimization model and heuristic algorithm in comparison with the greedy migration and re-instantiation algorithms.

The remainder of this paper is organized as follows: Section II presents some of the previous related work addressing this problem. Section III describes the system model adopted in this paper. Section IV formulates the optimization problem. Section V presents the proposed heuristic algorithm along with a brief discussion of its complexity. Section VI presents and discusses the results of the system's performance evaluation process. Finally, Section VII concludes the paper.

II. RELATED WORKS

Current research trends describe a significant interest in exploiting the benefits of containers within an edge computing environment. A number of approaches were proposed to address the challenges facing the implementation of containers from both academia and industry.

Hawilo *et al.* created a solution focused on a specific type of VMs performing Network function virtualization (NFV) functionalities with the setup assuming all VMs are housed within a single data center [8]. The developed solution is based on an integer programming (IP) optimization model orchestrator. The system facilitates the placement of the virtual network functions (VNFs) taking into consideration different constraints such as inter-container relations and service function chain (SFC) delays.

Barbalace *et al.* Heterogeneous migration scheme for Containers migration for natively-compiled containerized applications across compute nodes with differing instruction

Set Architectures [9]. Their focus on edge computing and migration schemes specifically to address the issues of stateful services. Their approach produces negligible overhead most noticeable during migration.

Rodrigues *et al.* proposed an analytical model to resolve Service Delay in edge cloud computing (ECC) systems. The approach seeks to minimize the delay for both communication and computation elements. The results were compared to models addressing processing delay only [10]. Although the authors tackle additional types of delays and have achieved an improvement over traditional methods, the lack of placement, downtime-related constraints, and delay can be considered a drawback.

Alam *et al.* leveraged lightweight virtualization to generate a modular solutions that works with the Docker system [11]. their solution achieved high availability metrics by relying on the innate redundancies generated by docker with their proposed solution allowing for on-demand service deployment on heterogeneous architecture layers.

Kaur *et al.* addressed the optimization problem by considering that the core cloud is optimized based on delay and energy with the decision whether or not to offload to the edge. The multi-algorithm service model operates by jointly allocating workload assignment based on assigned weights and computation capacities of the respective VMs. They also implemented a method to ensure the acquired results remain consistent [12]. While their approach is effective, it failed to take into account the network operation, failures, and containers' dependencies.

Abdullaziz *et al.* focused on the migration aspect of container orchestration by making it a more reliable option [13]. The authors achieved this by leveraging live orchestration as a method of achieving low downtime comparable to re-instantiation. Their method is broken down into two tiers. The first migrates user connectivity, while the second migrates user containerized applications. They have also addressed the possible causes of prolonged container migration downtime. Their results boast lower downtime by up to 50% shorter than that of the state-of-the-art migration.

Oleghe presented in his work [14] the frameworks and algorithms most commonly used in container placement problem, the types of containers currently dominating the edge space and the heuristic approaches currently favored in the research community to offer real time solutions.

Govindaraj and Artemenko followed a similar approach in [15]. With a focus on the complex orchestration of cyber-physical systems. Their work discussed the role of Edge Computing (EC) for factory automation applications. Taking Linux based containers as the basis, they built a live migration scheme called redundancy migration that reduced the downtime by a factor of 1.8 compared to the stock migration in Linux containers.

Wang *et al.* developed a unified mobile edge computing wireless power transfer (MEC-WPT) design framework with offloading and computing optimization for the edge while specifically relying on latency constrained computation. They minimized the total energy consumption subject to each node's individual computation latency constraints [16]. The authors leveraged the Lagrange duality method to obtain the optimal solution in a semi-closed form.

Machen *et al.* took the idea of migrating containers and virtual machines in a layered format [17]. Their approach using readily available technologies starts by migrating the non-state related aspects of services or VM and achieved much lower downtimes than traditional methods. The three-layer model also allows the pre-caching of popular applications at MECs, so that the time required for future instantiation of such applications can be shortened.

Alicherry and Lakshman developed a classic linear assignment algorithm using computational nodes and presented an algorithm for assigning VMs to data nodes that minimizes various latency metrics under different constraints [18]. The solution considered variable total access time allowances, with and without constraints, to showcase the system's adaptability.

Vaucher *et al.* developed a novel Container Orchestrator focusing on addressing issues such as having the containers being hosted on heterogeneous clusters [19]. Their focus was security-related issues stemming from the use of Intel-based SGX (software guard extension) enabled containers on a portion of the containers in a chain, and how to place them in such a way to maintain its relations to non-SGX enabled containers while preserving the security of the whole chain.

However, the solutions listed above focus on the recovery mechanism itself with focus on either re-instantiation or migration solely. In contrast, our work considers both potential recovery mechanisms as viable orchestration options. A second limitation of the related works is that their proposed solutions mostly exclusively consider the edge or core. This limits the optimality of the solutions to said solution space at a cost that could favor the users or providers at a time. Our solution considers the joint edge and core orchestration, allowing for a larger and more comprehensive solution space. Therefore, this paper proposes a comprehensive intelligent orchestrator based on an integer programming optimization model to minimize the impact of migration and re-instantiation on the containers' downtime and access delay. Accordingly, the proposed intelligent orchestration frameworks follow a more comprehensive approach concerning the recovery mechanisms used, adds the core to the solution space to act as an offset, and select the optimal placement that meets the container's demands, thereby achieving a highly optimized solution

III. SYSTEM MODEL

This work adopts a hybrid distributed computing environment similar to proposed works for modern networks. The global environment where the container and hosted services are placed is comprised of two platform types, namely the core cloud and the edge devices, as shown in Fig. 1. The core clouds are typically hosted using server farms and data centers. This allows for an abundance of resources, both

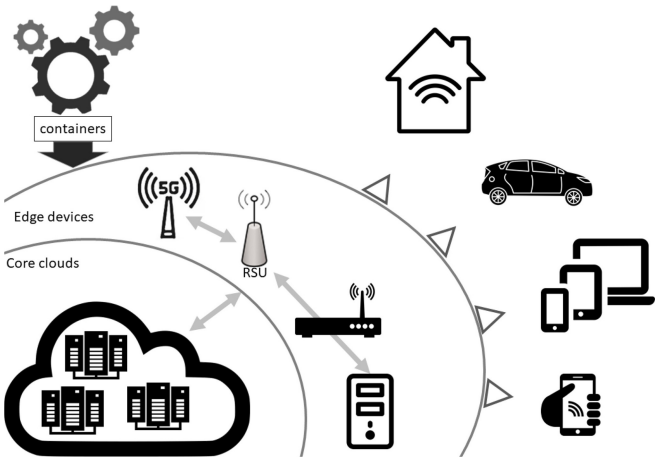


FIGURE 1. System Model: Core clouds and edge devices hosting service hosting containers.

computing and memory, in these environments. However, the size of the required structures to host them, the infrastructure, and the human intervention required to maintain them greatly limit suitable geographical placements in the real world.

Conversely, edge devices are a newly tapped resource that became available in the wake of the adoption of software-defined networks (SDN) and similar paradigms. That relies on the abstraction and virtualization of network functions to allow generic computing units to act as full-fledged networking units [20]. This presents a challenge for seeking an abundance of resources while also maintaining low latencies, which makes the optimization of this problem more crucial.

A. PHYSICAL RESOURCES

The core clouds physically operate from data centers or server farms. The servers are given abundant resources compared to their edge counterparts. The servers communicate with servers housed within the same rack or separate racks within the exact geological location. Cross-core communication is allowed due to the latencies, but it is not exercised extensively due to the increased complexity [21]. The latencies experienced within the core are kept to a minimum due to the advanced communication backbone and high bandwidth mediums used between servers, typically fiber-optic cables. The latencies between the core cloud and user are much higher, mainly due to the propagation delay resulting from the distance between them. The amount of computing power and memory resources in each server are limited in variation due to the homogeneous nature of the physical rack servers typically used in data centers.

In contrast, the edge devices are typically either repurposed communication nodes such as routers and RSUs, or larger units such as small dedicated edge nodes and 5G smart towers. The limited physical size of the edge units and the sporadic nature of their deployment enforces a singular rack structure. The racks typically host a limited number of servers compared to those found in large data centers. The

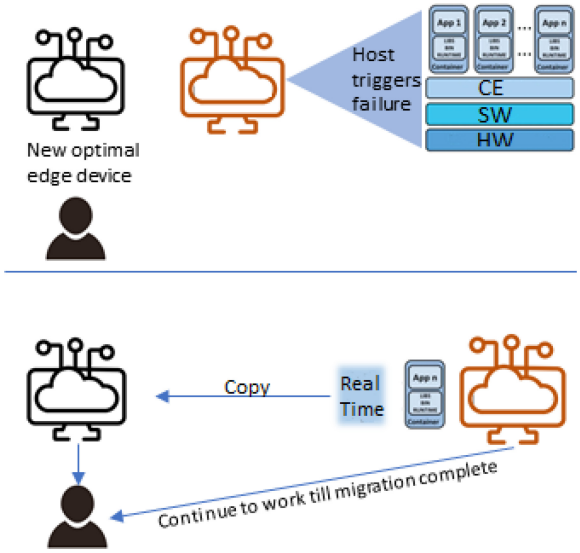


FIGURE 2. Migration mechanism with placement consideration.

latencies within the same rack are similar to those found in the core cloud. However, the delay between the edge device and end-user is more negligible than that of the core cloud. The resources are more limited than those of the core cloud and have a higher variance between each edge node's memory and computation resources.

B. CONTAINERS

The containers and the hosted services can vary based on their latency requirements. This ranges from latency-stringent services, such as those related to financial transactions and security, to looser requirements, such as those related to text messaging and advertising services. In addition to the containers' latency requirements, the required resources are uniquely tailored based on the task they serve. For example, from high-memory, low-computation in transcoding 4K videos and AR gaming to high-computation, low-memory requirements in navigation and sensor processing units.

IV. OPTIMAL CONTAINER MIGRATION/RE-INSTANTIATION (OC-MRI) MODEL FORMULATION

A. GENERAL MODEL DESCRIPTION

The model aims at minimizing two delay metrics. The first is the downtime resulting from the migration or re-instantiation process initiated for a container upon the imminent failure of its hosting computing node. The second is the access delay caused by the placement distance of the container to the end-user. The two processes have their own required static downtime, but choosing a new location has a significant impact on both the downtime and new access delay, as shown in Figs. 2 and 3. The following section details how we optimally minimize the downtime within the objective function. Meanwhile, the access delay is optimally minimized by enforcing costs upon both the objective function and its related constraints.

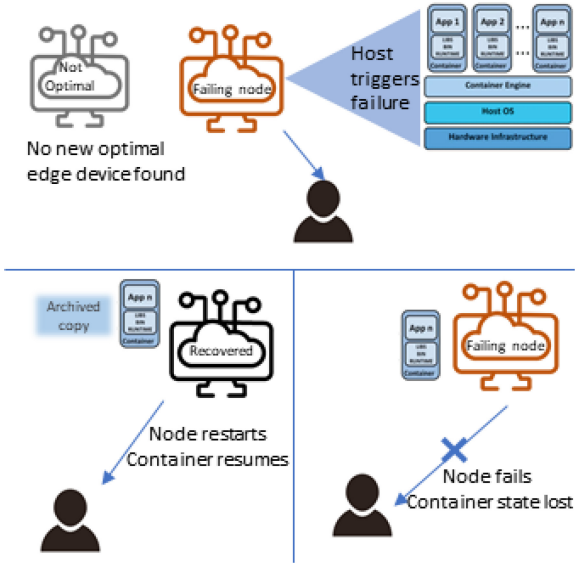


FIGURE 3. Re-instatiation mechanism with placement consideration.

TABLE 1. Table of notations.

Variable	Description
X_{ce}	Placement decision variable
Y_c^{Dec}	Re-instantiation / Migration decision variables
C	Set of containers
N_c	Total number of containers
E	Set of host nodes
Ne	Total number of host nodes
U	Set of users
Nu	Total number of users.
Res_{cr}	Container c computational requirements
Res_{er}	Node e computational resources
R	Set of computational resources types
C^D	Set of slave containers
$X_{ce}^{original}$	Original placement of the container c
$T_{c'}^T$	Tolerance time slave container c'
T_c^R	Recovery time of master container c
SO_c^{Dec}	Container c hosting node's delay overhead
$AccD_{cu}$	Delay between end-user u and container c
$AccD_{eu}$	Delay between node e and end-user u
D_c^P	Delay generated from placement of container c
$D_{ce}^{ee'}$	Delay generated by moving container c from node e to e'
H_{mod}	Delay penalty based on containers affinity to migrate or re-instantiate
D_c^{Dec}	Delay overhead of migration or re-instantiation decision

- *Computational Resources Constraint:* Using this constraint, the proposed model selects the computing nodes that satisfy the containers' computational requirements. The resources are CPU cores and available memory.
- *Network Delay Constraint:* Using this constraint, the proposed model filters the nodes to select those that do not violate both the access delay and delay between master and slave containers in the cloud.
- *Availability Constraints:* Each container is classified as unattached for single container-based services, master, or slave for services that consist of multiple containers working in a hierarchical set up to perform their tasks. To maintain the usability of QoS of the overall services offered by various containers, the proposed model defines the following constraints:
  - *Affinity Constraint:* This ensures that the master container and its slaves must be hosted on the same physical server if the communication tolerance time between them is lower than the master's recovery time.
  - *Anti-Affinity Constraint:* Conversely, the slave containers and their master should be deployed on different servers if the slave has a higher tolerance time than their master's recovery time.

B. NOTATIONS AND DECISION VARIABLES

The developed model relies on the following set of variables as shown in Table 1. The table outlines each variable and how it relates to our problem setup covering all the types of constraints discussed in more detail below.

C. MATHEMATICAL FORMULATION

This subsection outlines the binary decision variables, the objective function, and the model's constraints.

- *Decision Variables:*

$$X_{ce} = \begin{cases} 1 & \textit{if Container `c' is placed on node } e \\ 0 & \text{otherwise} \end{cases}$$

$$Y_c^{dec\_two} = \begin{cases} Y_c^{Mig} = 1 & \text{`c' Migrated} \\ Y_c^{Mig} = 0 & \text{otherwise} \\ Y_c^{Re-inst} = 1 & \text{`c' Re-instantiated} \\ Y_c^{Re-inst} = 0 & \text{otherwise} \end{cases}$$

- *Objective Function:*

$$min \sum_{c}^{N_c} DownTime_c + AccD_{cu}. \tag{1}$$

- *Boundary constraints:*

$$X_{ce}, Y_c^{Dec} \in \{0,1\}\ \forall c \in C, e \in E \tag{2}$$
$$Dec \in \{Re-instantiation, Migration\}$$
$$Downtime_c \geq 0;\ \ \forall c \in C \tag{3}$$

- *Placement Constraints:*

$$\sum_{c=1}^{N_c} \left(X_{ce} \times Res_{cr} \leq Res_{er}\right);\ \ \forall e \in E, r \in R \tag{4}$$
$$\sum_{c=1}^{N_c} \left(X_{ce} \times AccD_{cu} \leq AccD_{eu}\right);\ \ \forall e \in E, u \in U \tag{5}$$
$$\sum_{E=1}^{N_e} X_{ce} = 1;\ \ \forall c \in C \tag{6}$$

- *Availability Constraints:*

$$(X_{ce} + X_{c'e}) \leq 2 \text{ or } \left(X_{ce} + X_{c'e}{}^{original}\right) \leq 2;$$
$$\forall e \in E, c \in C, c' \in C^D, T_{c'}^T \leq T_c^R \tag{7}$$
$$(X_{ce} + X_{c'e}) \leq 1 \text{ or } \left(X_{ce} + X_{c'e}{}^{original}\right) \leq 1;$$
$$\forall e \in E, c \in C, c' \in C^D, T_{c'}^T \geq T_c^R \tag{8}$$

- *Re-instantiation/Migration Delay Constraints:*

$$Y_c^{\mathrm{Re}-inst} + Y_c^{Mig} = 1;\ \forall c \in C \tag{9}$$

$$D_c^P = X_{ce} \times \left(\sum_{e=1}^{N_e} X_{ce}^{Original} \times D_{ce}^{ee'} \right);\ \forall c \in C \tag{10}$$

$$D_c^{Dec} = SO_c^{Dec} \times Y_c^{Dec};\ \forall c \in C \tag{11}$$

$$Downtime_c = D_c^{Dec} + D_c^P + H_{mod};\ \forall c \in C,$$
$$Dec \in \{Re - instantiaion, Migration\} \tag{12}$$

Constraint (2) determines that the placement and re-instantiation/migration decision variables are binary numbers. Constraint (3) determines that the container downtime must be a positive number. Constraint (4) determines that the candidate computing node must meet the computational requirements of the potential container. Constraint (5) specifies that the access delay to the container is less than the threshold access delay of the corresponding container. Constraint (6) determines that only one computing node can host a container.

The containers in the solution space are classified as three types: master, a container that relies on input from subordinate containers to finish its operation, slave, a container or group of containers that are needed by others to operate but require no inputs for others to operate normally, and finally free containers that are simply stand-alone single container service. To maintain the interdependent relationship between different containers, constraints (7)-(8) determines if a container placement is dependent on its relation to its possible slaves. Thus, it either forces all related containers to a suitable physical location or allows them to be placed more freely. Furthermore, constraint (9) determines that a container can be either migrated or re-instantiated. Finally, constraints (10) and (11) determine that a container should be placed on a server that satisfies the delay requirements while minimizing the migration or re-instantiation overhead. Based on the previous constraints, the model selects to either migrate or re-instantiate each container to minimize its downtime. Lastly, constraint (12) shows that the downtime of each container is calculated in terms of the placement latency and the overhead delay resulting from the choice of recovery process offset by the H_{mod} modifier. This is based on the container's type, the critical nature of its state, or the information it holds controlling its affinity to re-instantiate or migrate.

D. IMPLEMENTATION

The proposed model was implemented using python environment for ease of use during experimentation. For the objective function in eq. (1), the values for both latency and downtime were generated based on the test-bed. At the same time, the resources-based constraints were predetermined based on the environment size and allocated edge devices. Finally, the generated matrices based on the container types were generated to include a portion with inter-dependencies to be used in constraints (7)-(8).

E. COMPLEXITY

Although integer programming problems can be solved using traditional branch and bound algorithms [22], these problems are typically classified as NP-complete [23]. This is further emphasized by the size of the problem's search space. Accordingly, the problem's search space can be estimated to be $2^{2\times N_c \times N_e}$ where N_c is the total number of containers and N_e is the total number of host nodes. This is due to the fact that there are 2^2 possible values (migrate or re-instantiate) for each container to be placed at each host node. For example, for the case of $N_c = 10$ and $N_e = 5$, the search space is 1.267×10^{30}. Therefore, finding an optimal solution for this problem in a real-world scenario can be computationally infeasible due to the exponential growth in the search space size. Thus, a low-complexity heuristic algorithm capable of live decision making is needed to address this problem.

V. EDGE COMPUTING-ENABLED CONTAINER MIGRATION/RE-INSTANTIATION (EC2-MRI)

The optimization model has allowed us to find the optimal placements through orchestration to achieve carrier-grade quality, whereby the mathematical model focused on satisfying different criteria, mainly downtime and latency. While successful at achieving optimal results, the model is computationally complex. This drawback is mainly due to its lack of scalability, as illustrated by the aforementioned complexity. Without real-time orchestration, it is counter-intuitive to offer the system in its current state for orchestrating highly dynamic environments, such as mobile edge computing due to the following:

- The need for total access to the solution space for proper orchestration.
- Treatment and optimization of users and containers as individual unique elements.
- High computational time and significant resources required to perform the orchestration task.

Given that downtime is the primary optimization objective, the model's orchestration processing time must be taken into consideration. This additional delay is especially significant when considering the highly mobile nature of edge users, and the sporadic nature of SDN and 5G based edge devices where it may become detrimental. The optimization model cannot be retrofitted or adjusted in a way that would address all the aforementioned issues while maintaining its accuracy. A better lightweight real-time solution is thus required. We choose to pursue a heuristic-based approach due to its flexible nature and capability of handling frequent changes to the network. They can also circumvent the global environment's staggering size and its effect on any system's ability to provide seamless orchestration unnoticeable to the end-user. Orchestration can benefit from taking place within the edge environment by removing the communication overhead of offsite orchestration. This design aspect minimizes the additional delay caused to the orchestration communication overhead by a core cloud that is typically required when done using the centralized approach. This requirement is

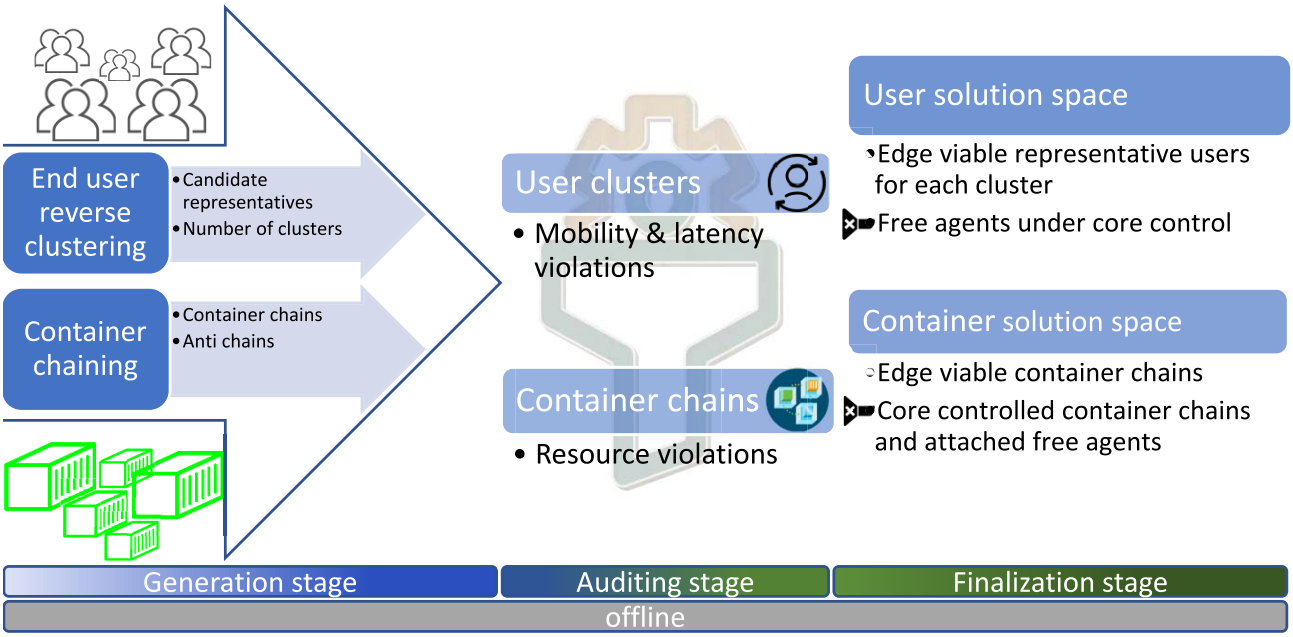


FIGURE 4. User and container solution space setup.

addressed by limiting the heuristic model's complexity, thus allowing it to run on the edge environments, eliminating the drawback. However, to achieve this, the overall solution space must be segmented into isolated sub-spaces to maintain the required low complexity.

The requirements above present their own set of challenges when creating the heuristic model. To address them efficiently, the system's behavior is comprised of two main stages. The first intends to run offline and sporadically, and focuses on simplifying the solution space by tackling the number of elements through clustering and chaining. It also further simplifies the problem by segmenting the generated clusters and chains based on their interactions and proximity to each other. This stage is intended only as an environment initialization point. It is a fail-safe when the existing edge-space results in many failures that indicate a considerable divergence from the last segmentation cycle. Its offline nature gives it access to core cloud environments' exclusive resources, especially the abundance of low-cost computational power and access to the global solution space. Fig. 4 shows the breakdown of the undertaken task. Firstly the generation stage is where the solution space is simplified, followed by the auditing stage where the candidates are vetted for accuracy and reliability. Lastly, the finalization stage where the heuristic assigns control of the outputted elements to the best-suited orchestration controller, as discussed below.

A. GENERATION STAGE

The live requirement placed on the edge-computing container migration/re-instantiation (EC2-MRI) model demands limited complexity. However, relying on a greedy or similarly generic approaches, while boasting the highly sought-after simplicity, not only does it not guarantee near-optimal placements, its output can vary vastly. To achieve our target of matching the placement benefits of the OC-MRI model, we must first solve the solution space size issue. This solution can be achieved by reducing and segmenting the solution space into isolated and self-managing subspaces.

1) USER CLUSTERING

The OC-MRI was allotted several concessions regarding the edge user's main distinguishing attributes since it was not developed as a live orchestrator, with the main concession its ability to ignore the mobility of user clusters. However, the heuristic approach cannot follow suit. As such, the process of generating the user solution space must be given additional attention. This is to ensure that the system is generating a healthy and stable solution space and not fall into a counterproductive cycle of "build-to-fail" environments. This outcome will keep triggering the offline stage more frequently, with little actual orchestration taking place. When generating a new edge space, simply permitting all users entry into the space as individual entities lacks scalability and reproduces the OC-MRI's drawbacks in our heuristic solution. On the other extreme, using advanced clustering techniques can be too time-intensive even for offline operations. Their high granularity will also steer the solution back towards the build-to-fail state. We address these hurdles in an intuitive approach by first subjecting the user solution space to the subtractive clustering process. This process generates the clusters necessary to provide coverage for the overall solution space and a corresponding list of representative candidate users acting as the centroid of their host cluster. While not highly complex, this clustering method has

Algorithm 1 Chain Generator

Input: $C = \{1, 2, \ldots, |Nc|\}$, *CD*, Segment tables
Output: Cluster & Anti Cluster tables

for $e \in Clustertables(i)$ **do**
  for $c \in Nc$ **do**
    if $(X_{ce} + X_{c'e}) \leq 2$ *or* $(X_{ce} + X_{c'e}{}^{original}) \leq 2$ **then**
      Rename c to *ChainID.c*
      update $CnChainTable(u(i))$
    end if
    if $(X_{ce} + X_{c'e}) \leq 1$ *or* $(X_{ce} + X_{c'e}{}^{original}) \leq 1$ **then**
      update $AntiCnChainTable(u(i))$
    end if
  end for
end for
return Cluster & Anti Cluster tables

been implemented several times and proven to work well. It was used most recently in the research efforts within 5G ultra-dense networks [24]. The main premise is segmenting user space to ensures the distance between the candidate clusters is significant by raising the Squash factor related to the overall user space and adjusting the Accept/Reject ratios based on the average mobility index of the users occupying each edge space.

The results of the subtractive clustering process are a number of disjoint clusters, from said list we choose a candidate that is near the center of the generated segment to act as the representative user, for the purposes of generating the audit threshold value and the mobility audit as well. After the centroid list is generated, we apply a straightforward rigid distance-based clusterer that produces a radial border around the centroid. This captures as many users adhering to equation (13) where S is the user's diameter coordinate space divided over the number of potential clusters. T is an integer variable ranging from 1 to 3 representing the mobility of each user (stationary, pedestrian, vehicular) and I represents the tolerances of the users present in the overall user space. This equation is used to limit the cluster upper limits in both the physical coverage area and the number of users hosted. It also nearly eliminates the presence of build-to-fail user clusters scenario.

$$Max_{\text{clusteringdistance}} = \left(ST^{\text{unclustered}} - SI_{\min}\right). \tag{13}$$

2) CONTAINER CHAINING

The optimization model allowed containers with links to exist on multiple edge devices given the tolerances outlined in equation (7). The heuristic model cannot handle such complex tasks as the placements and interactions can easily grow in complexity quite rapidly. To address this issue, a reduction step similar to what was done in the user solution space is implemented. Unlike the user solution space, segmenting and clustering are not required in their traditional meaning. Instead, containers are chained together based on their affinities as shown in Algorithm 1.

3) CHAIN GENERATOR

The chain generator aims at maintaining the size and complexity requirements of the second algorithm. Individually orchestrating the containers will be difficult. Thus, the algorithm generates a master list of all container chains based on the affinity constraint (7). Container chains are then clustered into singular entities and assigned new IDs. The new format changes the IDs from integers to floating variables with the integer digits representing the cluster ID and the fraction digits preserving the container unique IDs. Anti-affinity is then generated based on constraint (8). Once the key elements of the table are generated, the corresponding computational requirements for each container chain is amended based on the aggregated value of contained clusters. Lastly, an affinity binary variable is generated based on the highest H_{mod} modifier for each cluster chain. While this approach guarantees a semi-optimal solution, it is still limited by the loss of multi-host based solutions due to the over simplification of the affinity constraint to maintain a lower complexity in the later algorithm.

4) AUDITING STAGE

Once the users have been placed into clusters and the containers have become grouped into more monolithic chains, we have to ensure the feasibility of our new solution space in various aspects. We begin addressing this with the user clusters. Two major causes of failure have to be avoided. First the tolerance offset from the representative user violates that of the masked user (cluster host user). The second is a preventative step to avoid immediate failures related to mobility of the users. To address this issue, an auditing threshold is first enforced on each cluster guided by the following equation (14) where I is the mobility index the user and i is the cluster number:

$$Audit_{\text{threshold}} = \left(Max^{i}{}_{\text{clusteringdistance}}/2\right) - I \tag{14}$$

Once the auditing threshold is invoked, all users found in violation of it, based on their tolerances or mobility, will be checked to ensure that they can remain members of the cluster and the representative user can be an effective stable substitute. Failed users are pulled from their host cluster and designated as a free agent under the control of the core until the next clustering cycle. The next cycle is triggered periodically or when triggered based on significant degradation in the EC2-MRI performance. The degradation can be caused by significant changes in the host devices number and available resources or the live portion of the auditing stage offloading a more extensive than allowed number of users from the clusters to become free agents under the direct core cloud control. Fig. 5 illustrates the generation of a threshold and an example of a mobility violation-based user expulsion.

Container auditing on the other hand is relatively more straightforward. By using the overall size of the container chain, it checks to ensure that the available containers are placeable while maintaining the latency constraints. In addition, to maintain feasibility through multiple iterations, a

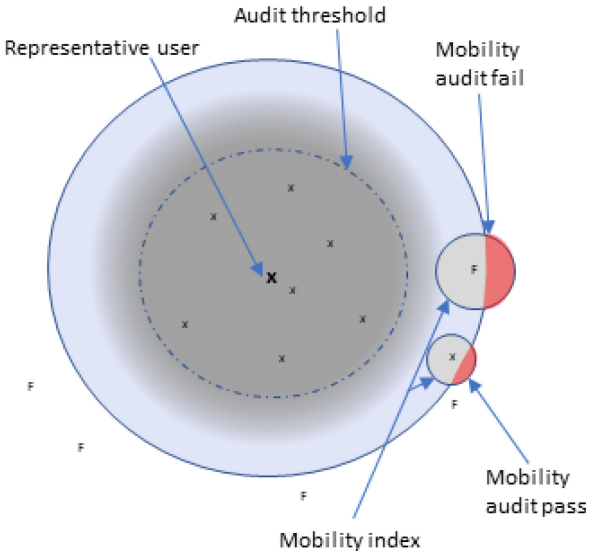


FIGURE 5. Auditing stage, the red portion highlights the margin of mobility violation.

Algorithm 2 Segmenter

Input: $E = \{1, 2, \ldots, |Ne|\}$, $U = \{1, 2, \ldots, |Nu|\}$
Output: Segment tables

for $u \in Nu$ **do**
  for $e \in Ne$ **do**
    if $Dp(e(i)) < UserCluster(MaxTolerance)$ **then**
      update $SegmentTable(u(i))$
    end if
  end for
  Sort $SegmentTable(u)$ based on latency to $u(i)$
end for
return Sorted Segment Tables

resource-specific over-provisioning is enforced. The margin limits the maximum resources for a container and removing a whole chain ensures that the remaining chains remain hostable. This approach is necessary given the relative opportunistic nature of the used placement algorithm and to avoid build-to-fail states. Any chains found to be violating this threshold will be removed along with their attached users from the clusters.

5) EDGE DEVICE ALLOCATION STAGE

Once the auditing stage is finalized, the edge devices that can house the remaining cluster chains are polled sequentially and an ordered candidate list is prepared based on Algorithm 2.

Segmenter: Generates a single dynamically sized table of candidate edge devices based on their availability within a user latency allowance. The table size is based on the highest latency threshold. This can result in a significantly large table in cases of densely placed servers coupled with loose latency requirements, which is addressed in step 4. The chain generator is polled for each region after all the tables have been generated to eliminate any edge devices deemed incapable of containing the generated container chains. The servers are then organized based on latency and computing power. Then, only the top candidates for each region are maintained and the rest are discarded. The number of top candidates maintained is based on the density of edge devices found in each region.

Algorithm 3 Orchestrator

Input: Cluster & Anti Cluster tables, Segment tables, *ChainID.c*
Output: placement request

for $i =<$ *sortedsigmenttablesize* **do**
2: **while** $X_{s_u} \neq 1$ **do**
    find $d^c_{s_u} = \min_{c \in C_u} \{\frac{1}{|V|} \sum_{v \in V} d^c_{s,v}\}$
4: **if** $ChainID \neq AntiClustertable(i)$ **then**
      if candidate resource allowance $>$ *ChainID* resource requirments **then**
6: **poll** candidate
        if placement successful **then**
8: **EXIT**
        end if
10: **end if**
      end if
12: **end while**
end for
14: **if** no candidate found **then**
    remove *ChainID.c* from Cluster, Anti Cluster tables, and Segment tables.
    orchestration raised to core cloud controller
16: **end if**
return

B. EDGE-CONTROLLED LIVE ORCHESTRATION PLACEMENT STAGE

The model is trigger activated once a container or VM signals a failure or fails to respond, it enacts Algorithm 3. Starting from the hosting edge node, an iterative check of the candidate table is initiated. This is followed with sequential requests based on their ranking. The requested edge devices check their computational availability and their currently hosted containers for any anti-affinity violations. If none are found, the request is approved and the algorithm terminates. If the table is exhausted with no solutions, the edge device raises the request to the core hosting secondary algorithm to orchestrate a viable solution. At this stage, the drawbacks of the clusters in the first stage are rectified, but the container chain is removed from the candidates list for that region until the next periodic operation of the provisioning stage.

C. CORE-CONTROLLED LIVE ORCHESTRATION PLACEMENT STAGE

The heuristic chooses to treat a container differently if it is a free agent from the start or was removed due to repeated

failures. This method maintains overall system complexity. However, relying on a generic approach, such as a greedy migration or re-instantiation, would be deemed counter-intuitive because it can keep victimizing a group of users to an unhealthy edge device or a sub-optimal latency. To address this, the core can make use of its abundance of resources to orchestrate such stragglers using the original methods outlined in the OC-MRI model. The main limitation of the optimization model was the lack of scalability with relation to the solution space. However, while acting as a backup to the EC2-MRI model, this is no longer a requirement. When orchestrating has either failed during the live stage or in the case of the free agents, the core can use the distance S to create the solution space limits for a single user and perform live orchestration. The core can tackle this orchestration in two methods based on the number of entities under its control at the instance of a failure:

- Use the free agent's tolerance to generate the solution space
- Use the distance S to generate a slightly complex solution space and optimize all controlled nodes within the influenced region.

D. COMPLEXITY

In contrast to the OC-MRI model, the proposed EC2-MRI heuristic has taken a number of steps to avoid such a limitation and focused on achieving a lower computational complexity, even with the discussed two stages. The overall complexity is governed by the complexity of each of the stages. The complexity of the first stage being $O(N_u \times N_e + N_c \times N_e)$ where N_u is the number of users, N_e is the number of host nodes, and N_c is the number of containers to be placed. However, since this is done only once offline, this will not impact the system's overall complexity. On the other hand, the complexity of the second stage is $O(N_c \times N_e)$ under the assumption of the worst case being that each container is placed freely with zero affinity. Thus, the overall complexity is linear in the number of containers and host nodes. Using the same values as in the previous example, the complexity would be in the order of 50 operations.

VI. PERFORMANCE EVALUATION

To best test the two solutions put forth, a suitable device is necessary for both bench-marking and environment simulation. To evaluate the performance of the proposed solutions, a physical workstation with 6 Cores and 12 threads of CPU, a 11 GB GPU, and 32 GB of RAM is used to build the test bed and implement both the OC-MRI and EC2-MRI models.

Three test beds of varying size are implemented to represent settings from sporadically populated to densely populated edge environments and to better represent and highlight the variable nature of edge environments. Once the test beds are properly populated, both models are tested in contrast to two generic algorithms, namely greedy re-instantiation and migration. These greedy algorithms are introduced to act as base benchmarks to highlight each of the proposed models' benefits and drawbacks. The algorithms' behavior is created through assigning a higher affinity to either the migration or the re-instantiation decision variables to force the intended behavior without allowing for non-solvable states. The resulting latency and downtime generated by each orchestration approach is measured from the users' point-of-view to act as the main metrics for the models' performance. In addition, other metrics specific to the clustering algorithm are presented to highlight the efficacy of the proposed clustering approach. Finally, given that the EC2-MRI is the only implantable solution, further discussion and metrics related to its inner modules are presented.

TABLE 2. Test bed size.

	edge	core	user clusters
small	15	2	4
medium	55	3	12
large	125	3	21

A. TEST BED

A number of test beds are generated to perform the most comprehensive method of testing the two models and the bench markers, with equal distribution to each size as shown in Table 2. Accordingly, the simulation environment is generated through five stages starting with the user clusters, core placement, edge devices, resource allocation, and finally container placement. Before any testing takes place, the earlier seeding of the users and containers gets remapped based on their cross latencies where the distance between any related entities such as core cloud and edge device is represented as distance. The only caveat is for any communication taking place over the core-edge separation region, typically between core cloud and edge devices or end-users, will incur a flat latency penalty to represent the typical lag of traditional network backbone, which we are trying to minimize.

1) USER CLUSTER SEEDING

The EC2-MRI offline stage's related processing delay is not taken into account to maintain objectivity, as the users' clusters assumed to be present in the OC-MRI are used from the output of the EC2-MRI finalization stage. Additionally, all processing time for the models left are ignored when measuring the downtime to focus on the orchestration effect alone. However, the heuristic failed call and response attempts are included as they are not deemed to be related to processing and are an integral part of the heuristic model's behavior. The user clusters are generated based on the user clustering algorithm outlined in Section V-A1 of the heuristic model. Each cluster is occupied with a minimum of five users and capped at 25 users. To ensure proper clustering, a randomized mobility metric is attached to the users while ensuring that no highly mobile users are also given stringent latency requirements. This is done to maintain the solvability of the solution space for all algorithms and maintain the practicality of the simulation environment. Once the clustering and

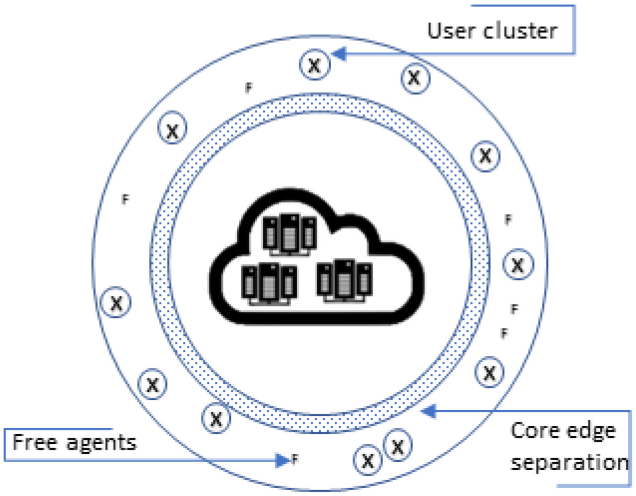


FIGURE 6. User clusters seeding and core edge latency penalty.

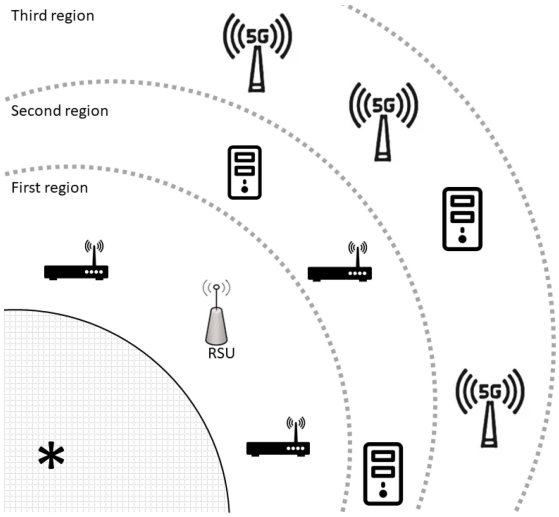


FIGURE 7. Edge device placement with respect to user cluster.

auditing stages are completed, separate sub spaces are generated around each cluster as shown in Fig. 6 the outline of core edge latency barrier is outlined along with the representation of the user clusters and free agents. At this stage, the only restriction aside from only seeding in the edge space is the overlapping between user clusters, were the smaller clusters are quashed and made into free agents. This step is enforced to ensure that no unrealistic latencies occur in the later stages of test bed setup given that the edge devices are generated related to the cluster centroid.

2) CORE CLOUD SEEDING

The core clouds are first assigned a location within the core designated space. Following this, a number of servers will be assigned to each of the cores to represent tray workstation style devices with minimal latencies. Once the servers are assigned to a distinct core, they are then designated to their specific racks within each core. This generates the necessary latencies within the data centers and between the core clouds and user clusters. The delays generated by their placement within the core are not represented in the latency map. But to maintain accuracy, a latency for inter-rack communication is randomly assigned in the range of 2-5ms, and 7-10ms for cross-rack communication.

3) EDGE DEVICE SEEDING

The edge devices' placement in the vicinity of the user clusters impacts both the possible size of the edge device and its offered resources. The sizes of used edge devices fall into three generalized tiers: small for devices such as routers, medium representing devices such as an RSU, and large for devices similar in size and capability to 5G towers. The probability of these placements has been controlled as shown in Fig. 7. The regions are split based on their latency with respect to the user clusters. The mapping based latency uses a scale from 10ms to 32ms representing typical edge latencies within levels 2-4 of 5G environments as outlined in [25]. The distribution for each edge device type is static

TABLE 3. Edge device size based placement.

	Region		
Edge Device Size	First	Second	Third
Small	70%	20%	0%
Medium	30%	65%	35%
Large	0%	15%	65%

TABLE 4. Edge node resource availability.

	Computation affinity		Memory affinity	
	CPU	Memory	CPU	Memory
Small	4	2	2	5
Medium	9	5	5	9
Large	15	7	8	13
Core	18	12	18	12

TABLE 5. Container resource requirements.

	Computing affinity		Memory affinity	
	CPU	Memory	CPU	Memory
Small	2	1	1	2
Medium	4	2	1	4
Large	7	4	2	6

in all test beds generated. The ratios for each used type as shown in Table 3 outline each region's ratio for total device types from the main pool based on the assigned percentage.

4) RESOURCE ALLOCATION

The available resources for services are categorized as either computational power or memory. The edge devices will have an abundance of either but not both. The unit of measure for both will be in their capacity to meet the demands of a singular small container of either type. Table 4 shows the allocation based on the size and type.

5) CONTAINER PLACEMENT

The containers were categorized through a similar method to the resource allocation based on the size requirements and their resource affinity. Table 5 shows the relation between their size and the required resources. However, they additionally require a latency threshold. The containers are assigned affinities to represent container chains performing a singular

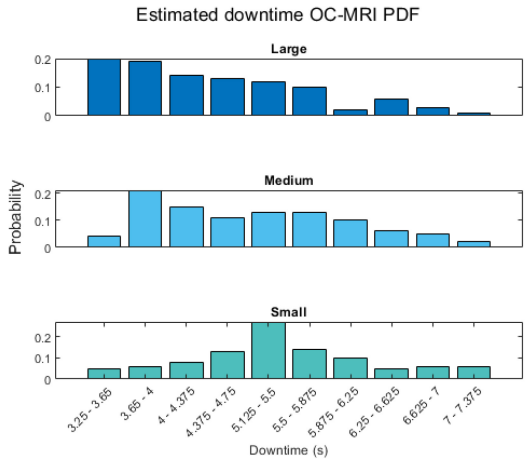


FIGURE 8. Downtime distribution OC-MRI.

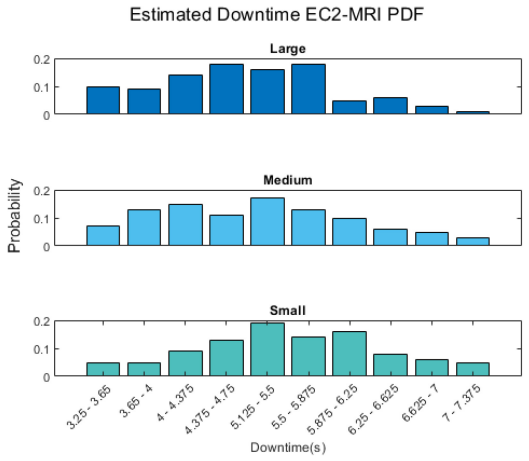


FIGURE 9. Downtime distribution EC2-MRI.

service where cross communication is required, then the latencies between them are restricted. Once the containers are randomly distributed between user cluster, the downtimes assigned to each container relates to its type and size with values ranging from 2s to 5s with additional time tacked on GPU heavy containers following the approach in testing used in [26]. The aforementioned affinities and overall latency requirements are adjusted based on the length of the chain to maintain both solvability and plausibility. Finally, a 15% anti affinity is assigned to any container not belonging to the same chain. These probabilities are maintained throughout the general testing.

B. DOWNTIME AND LATENCY ANALYSIS

Both the latency and downtime were generated by the placement distance in the latency mapping during edge device seeding stage, the ranges used stem from typical downtimes expected with general current memory based migration techniques [27], [28]. the downtime used as input for the models tests relies on the type of container recovering along with the transfer time associated with the typical size of said containers.

1) DOWNTIME

The proposed optimization and heuristic models are benchmarked against two base greedy algorithms. The models experience a highly varied downtime response when transitioning between the small test bed towards the large test bed. The resulting range of downtime experienced has been divided into 10 segments of equal length to better contrast the different models' behavior within each time range. The OC-MRI, as is shown in Fig. 8 performs best in the sporadic environments with high adherence to the effect of the container type modifier H_{mod}, with a majority of placements falling within the 3.25 to 4.75 seconds range. The optimization model maintained great performance throughout with minimal additional lag even when expanding to the medium and large test beds peaking at 4 and 5.5 seconds respectively. Following the performance of

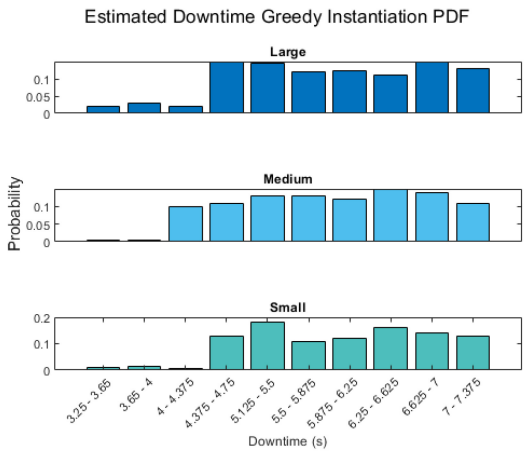


FIGURE 10. Downtime distribution greedy re-instantiation.

OC-MRI, the heuristic based approach used in EC2-MRI performed comparably well with respect to downtime as shown in Fig. 9 boasting similar results. However, it is noteworthy that it experiences a consistent dip in the placement opportunities within the initial 2.5 second range. This is attributed to two factors, the monolithic chains presence making the use of the low latency of small edge devices less likely due to their limited resources, and the presence of conflicting modifier H_{mod} values within a single chain both leading to higher latencies. Finally, the performance of the base bench marking algorithms has resulted in a much higher downtimes with very limited placements in the first 1.5 seconds totaling less than 25% for all test bed sizes combined in each algorithm as shown in Figs. 10 and 11. Overall, the OC-MRI model maintained the lowest downtime values throughout the latencies with high adherence rate of the H_{mod}. The heuristic approach, while showing a few drawbacks related to container chaining, still maintained comparable downtime values to the optimization model. The greedy algorithms using the H_{mod} modifier were turned into a forced migration orchestration modeling the behavior similar to the recovery method in [26] had significantly higher downtime regardless of the

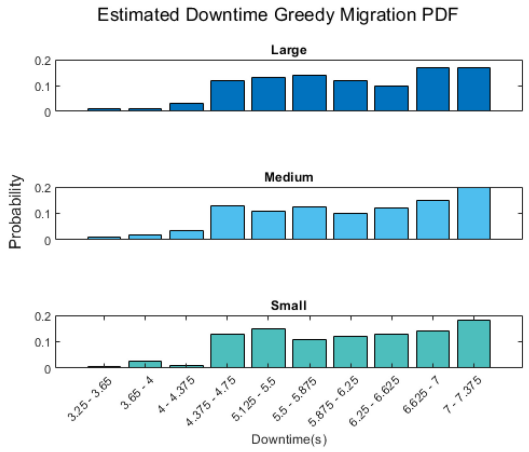


FIGURE 11. Downtime distribution greedy migration as per [17].

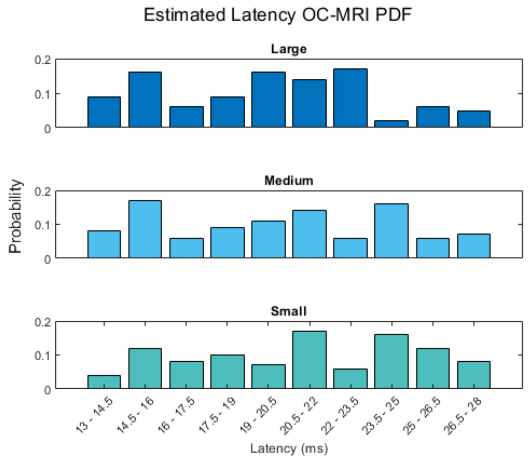


FIGURE 12. Latency experienced by the user, OC-MRI.

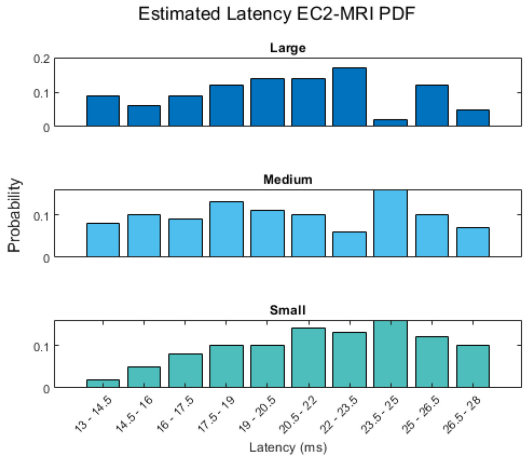


FIGURE 13. Latency experienced by the user, EC2-MRI.

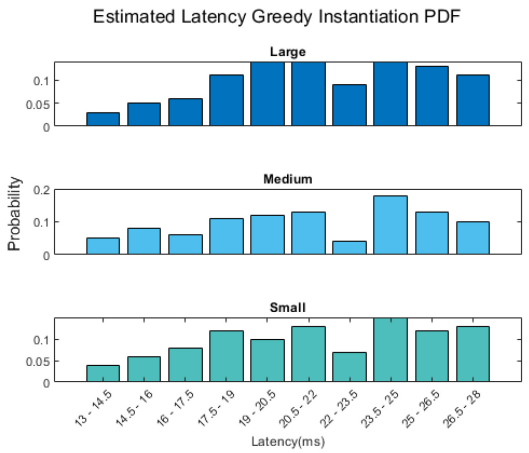


FIGURE 14. Latency under greedy Re-instantiation orchestration.

effect of the container type. This is due to their simplistic approach, ignoring the container requirements, and relying solely on their pre-configured biases.

2) LATENCY

Similar to downtime. the min and max latencies generated by all models were used to create 10 time based segments to allow for better contrast when comparing the models' behavior to each other with each latency range. The latencies offered by the OC-MRI followed a similar trend to its performance in the downtime metric with great distribution heavily weighted towards lower latencies as shown in Fig. 12. While the performance between the small and medium test beds was comparable, a significant lag was generated during the large test bed bench-marking with a noticeable reduction in the placement opportunities in the lower latency thresholds. The performance of the EC2-MRI favored the middle range of latencies with a slight offset towards the upper range in the large test bed stage as shown in Fig. 13. This can be attributed to the necessary shift away from the user caused by the container chaining favoring the middle and large sized edge computing units. Both base benchmark algorithms have shown inconsistent latency response to the increase in the test bed size. The results of the greedy recovery mechanisms were heavily weighted towards the upper 4ms with little to no change in the distribution throughout the testing process as shown in Figs. 14 and 15. the distribution notices a failure to allocate proper placements when approaching the outer region of the edge space adjacent to the core with the H_{mod} variable impact on their operation becoming more negligible. While the EC2-MRI has consistently achieved comparable results to the OC-MRI optimal placements, the cost of reducing the size of the solutions space through container chaining remains an obstacle. A better approach such as splitting the larger chains or restricting the chaining of medium and large containers can be beneficial, but at the cost of increasing the complexity and size of the solution space. Another approach to investigate is more offloading of reliable containers onto the core.

C. HEURISTIC ANALYSIS

While the heuristic model has a call and response aspect to it with regards to confirming placement availability before attempting to migrate or opting for re-instantiation, this

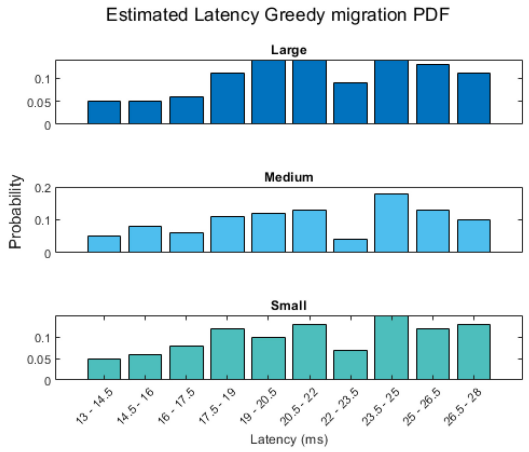


FIGURE 15. Latency under greedy Migration orchestration as per [17].

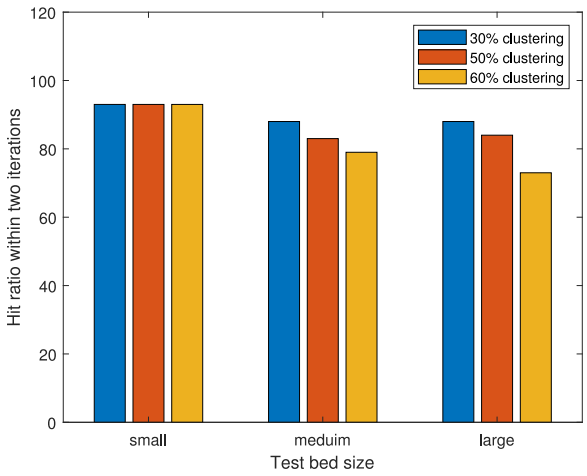


FIGURE 16. Successful placement within the first two iterations of the EC2-MRI.

behavior will naturally create additional overhead delay that can prolong downtime beyond preset tolerances. To investigate the amount of delay generated from this aspect of the model's functionality, Fig. 16 shows the percentages a solution was achieved in the first two iterations for varying cluster sizes. The testing was focused on the amount of clustering for each test bed to ensure better stress testing through forcing longer chains to compete for a lower number of viable edge-devices capable of hosting the chains as a singular object. As shown, the system maintained a high success rate despite the varying chain lengths with little degradation when stress testing under a high rate of container chaining of 60%.

The heuristic approach segmentation is based on a unified set of clustering techniques that don't differ significantly regardless of the addressed test-bed size. This approach is necessary to limit the complexity of the system and the variable nature of the test-beds. The clustering techniques used showed minor improvements regardless of test-bed size compared to the OC-MRI model; this is most visible in Figures 9 and 13.

TABLE 6. Clustering accuracy.

Test Bed Size	Clustering pre-audit	
	User clusters edge control	Free agents core control
Small	92%	8%
Medium	84%	16%
Large	86%	14%

TABLE 7. Effect of auditing on error avoidance.

	EC2-MRI edge fail	
Test Bed Size	No audit	Audit
Small	4	0
Medium	12	5
Large	28	7

The clustering stages' accuracy has a direct impact on the EC2-MRI downtime and latencies through two means. Beginning with the clustering accuracy as shown in Table 6, the system's setup stage's ability to cover the solution space efficiently, creating reliable clusters, is of high importance. But another aspect that needs to be taken into account is the number of free agents generated as the overhead of core based orchestration is best avoided when aiming for lower downtimes. The table shows a high ratio of edge controlled clusters regardless of the test bed size, with free agents ratio never reaching 20%.

The audit stage of the offline setup stage requires a dedicated metric when testing its efficacy. To measure its effectiveness, an isolated test run on all three test bed is performed with and without the auditing module. This was done to check the changes it evokes in the number of failure-generated free agents. as shown in Table 7. The audit stage is able to offload all errors within the small test bed environment and approximately 40% and 25% reduction during the medium and large tested respectively; greatly lowering the cases or un-placeable container user pairs offloaded as free agents in the orchestration stage.

VII. CONCLUSION

Proper placement of edge services has become increasingly critical for both network service providers (NSPs) and end-users. This paper explored container orchestration in a hybrid computing environment. The various challenges hindering container edge adoption were identified and discussed. The paper presented two solutions while providing detailed insights on system modeling and building blocks of a container orchestrator. The first is an integer programming optimization model, namely the OC-MRI model, that addressed the container orchestration between edge devices and core clouds. The model adhered to a number of performance and availability-aware constraints. The main target of the model was to achieve minimal downtime for fault recovery through calculating the decision variable to either re-instantiate or migrate. The model also achieved lower latencies as a secondary objective indirectly enforced through the manipulation of constraints and impact of the downtime variable. Although the proposed model minimized the downtime and provided noticeable improvements to the

latencies, it remained limited by its lack of scalability and prohibitive time complexity, making real time implementation infeasible. To address this issue, a heuristic solution was presented through the EC2-MRI model. The algorithm consisted of two stages to maintain the overall system's ability to run in real time. The two stages allowed for the highly complex portions to be run on the core cloud where computation resources are abundant and inexpensive while the latter stage is implemented on the edge devices where real time decision making is required and computation resources are scarce. Additional metrics were used to critique the system's unique modules and their impact on the overall system's accuracy to better highlight the benefits of the EC2-MRI algorithm.

Using the current results as a starting point, we aim to convert the optimization model into a multi-objective one. This move is necessary to address the issue of the optimization cost of running the edge device as they have highly variable costs stemming from their placement and unit size. Another possible improvement is the use of the H_{mod} variable. To better utilize it and avoid conflicting magnitudes in container chains, it is best to convert it into a multiplier type modifier (values below 1 representing ranges of affinity to migrate, and values above 1 representing re-instantiation) to allow for better granularity when assigning affinity.

In terms of the heuristic model, the EC2-MRI is currently only reactive. However, a reactive approach is the natural path when aiming to lower downtime. We must first address the behavior of the system in a dynamic environment to better adjust the system. The best approach to address this issue is to start with a mobility model capable of identifying changes in the trajectories of the user clusters using the current method of subtractive clustering and making adjustments to maintain the lowest latency and highest robustness. To develop these venues of research, a heterogeneous intelligent mobility model will be necessary. Unlike the conventional mobility models currently used, we propose to use a restricted path based model that will not allow for free motion. Instead, a preset map of restricted paths representing roads and pedestrian pathways within streets or large building such as malls and stadiums, where edge computing faces the highest demands, will be developed. Lastly, we aim to use the input of the OC-MRI and EC2-MRI to train a machine-learning based model to investigate its efficacy as a real-time solution when compared to the EC2-MRI from both an accuracy and complexity point of view. This is paramount given the time-based nature of the solution and the necessary orchestration to account for failure recovery and users leaving and entering predesignated statically assigned clusters.

REFERENCES

[1] A. Moubayed, A. Shami, P. Heidari, A. Larabi, and R. Brunner, "Edge-enabled V2X service placement for intelligent transportation systems," *IEEE Trans. Mobile Comput.*, vol. 20, no. 4, pp. 1380–1392, Apr. 2021.

[2] A. Moubayed, A. Shami, P. Heidari, A. Larabi, and R. Brunner, "Cost-optimal V2X service placement in distributed cloud/edge environment," in *Proc. WiMob*, 2020, pp. 1–6.

[3] A. Moubayed and A. Shami, "Softwarization, virtualization, & machine learning for intelligent & effective V2X communications," *IEEE Intell. Transp. Syst. Mag.*, early access, Sep. 29, 2020, doi: 10.1109/MITS.2020.3014124.

[4] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "OpenStack: Toward an open-source solution for cloud computing," *Int. J. Comput. Appl.*, vol. 55, no. 3, pp. 38–42, 2012.

[5] B. I. Ismail *et al.*, "Evaluation of docker as edge computing platform," in *Proc. IEEE Conf. Open Syst. (ICOS)*, Bandar Melaka, Malaysia, Aug. 2015, pp. 130–135.

[6] U. Awada and J. Zhang, "Edge federation: A dependency-aware multi-task dispatching and co-location in federated edge container-instances," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, 2020, pp. 91–98.

[7] C. Costache, O. Machidon, A. Mladin, F. Sandu, and R. Bocu, "Software-defined networking of linux containers," in *Proc. RoEduNet Conf. 13th Edition Netw. Educ. Res. Joint Event RENAM 8th Conf.*, Chisinau, Moldova, Sep. 2014, pp. 1–4.

[8] H. Hawilo, M. Jammal, and A. Shami, "Orchestrating network function virtualization platform: Migration or re-instantiation?" in *Proc. IEEE 6th Int. Conf. Cloud Netw. (CloudNet)*, Prague, Czech Republic, Sep. 2017, pp. 1–6.

[9] A. Barbalace, M. L. Karaoui, W. Wang, T. Xing, P. Olivier, and B. Ravindran, "Edge computing: The case for heterogeneous-ISA container migration," in *Proc. 16th ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environ.*, 2020, pp. 73–87.

[10] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato, "Hybrid method for minimizing service delay in edge cloud computing through VM migration and transmission power control," *IEEE Trans. Comput.*, vol. 66, no. 5, pp. 810–819, May 2017.

[11] M. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen, "Orchestration of microservices for IoT using docker and edge computing," *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 118–123, Sep. 2018.

[12] K. Kaur, S. Garg, G. S. Aujla, N. Kumar, J. J. P. C. Rodrigues, and M. Guizani, "Edge computing in the Industrial Internet of Things environment: Software-defined-networks-based edge-cloud interplay," *IEEE Commun. Mag.*, vol. 56, no. 2, pp. 44–51, Feb. 2018.

[13] O. I. Abdullaziz, L.-C. Wang, S. B. Chundrigar, and K.-L. Huang, "Enabling mobile service continuity across orchestrated edge networks," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 3, pp. 1774–1787, Jul.–Sep. 2020.

[14] O. Oleghe, "Container placement and migration in edge computing: Concept and scheduling models," *IEEE Access*, vol. 9, pp. 68028–68043, 2021.

[15] K. Govindaraj and A. Artemenko, "Container live migration for latency critical industrial applications on edge computing," in *Proc. IEEE 23rd Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, vol. 1, 2018, pp. 83–90.

[16] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 17, no. 3, pp. 1784–1797, Mar. 2018.

[17] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, "Live service migration in mobile edge clouds," *IEEE Wireless Commun.*, vol. 25, no. 1, pp. 140–147, Feb. 2018.

[18] M. Alicherry and T. Lakshman, "Optimizing data access latencies in cloud systems by intelligent virtual machine placement," in *Proc. IEEE 32nd Int. Conf. Comput. Commun. (INFOCOM)*, Turin, Italy, Apr. 2013, pp. 647–655.

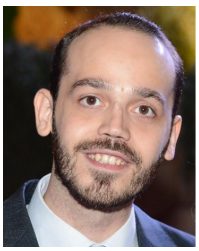
[19] S. Vaucher, R. Pires, P. Felber, M. Pasin, V. Schiavoni, and C. Fetzer, "SGX-aware container orchestration for heterogeneous clusters," in *Proc. ICDCS*, Vienna, Austria, Jul. 2018, pp. 730–741.

[20] "Datacenters of the future: A shifting landscape from the core to the edge," Raritan, Franklin Township, NJ, USA, Rep., May 2017. [Online]. Available: https://www.raritan.com/eu/landing/datacenters-of-the-future-whitepaper/thanks

[21] R. Alsurdeh, R. N. Calheiros, K. M. Matawie, and B. Javadi, "Hybrid workflow scheduling on edge cloud computing systems," *IEEE Access*, vol. 9, pp. 134783–134799, 2021.

[22] J. W. Chinneck, "Chapter 13: Binary and mixed-integer programming," in *Practical Optimization: A Gentle Introduction*, Ottawa, ON, Canada, 2004.
[23] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*. Boston, MA, USA: Springer, 1972, pp. 85–103.
[24] Q. Liu, C. F. Kwong, S. Zhang, L. Li, and J. Wang, "A fuzzy-clustering based approach for MADM handover in 5G ultra-dense networks," *Wireless Netw.*, vol. 2019, pp. 1–14, Sep. 2019.
[25] D. Burstein. "Edge Computing Architecture Impact on Latency." 2020. [Online]. Available: https://stlpartners.com/edge-computing/edge-computing-architecture-impact-latency/
[26] R. A. Addad, D. L. C. Dutra, M. Bagaa, T. Taleb, and H. Flinck "Fast service migration in 5G trends and scenarios," *IEEE Netw.*, vol. 34, no. 2, pp. 92–98, Mar./Apr. 2020.
[27] M. Terneborg, J. K. Rïnnberg, and O. Schelén, "Application agnostic container migration and failover," in *Proc. IEEE 46th Conf. Local Comput. Netw. (LCN)*, 2021, pp. 565–572.
[28] H. M. Dur, "A container-based code offloading framework for mobile edge computing applications," M.S. thesis, Dept. Inf. Syst., Middle East Tech. Univ., Ankara, Turkey, 2021.

SAM ALEYADEH received the Bachelor of Engineering Science degree from Queens University in 2013, and the Master of Science degree in computer science program. He is currently pursuing the Ph.D. degree in electrical and software engineering with Western University with a focus on 5G networking orchestration and optimization focusing on edge computing.



ABDALLAH MOUBAYED (Member, IEEE) received the B.E. degree in electrical engineering from Lebanese American University, Beirut, Lebanon, in 2012, the M.Sc. degree in electrical engineering from the King Abdullah University of Science and Technology, Thuwal, Saudi Arabia, in 2014, and the Ph.D. degree in electrical and computer engineering from the University of Western Ontario in August 2018, where he is a Postdoctoral Associate with the Optimized Computing and Communications Lab. His research interests include performance and optimization modeling, machine learning and data analytics, computer network security, cloud computing, and e-learning.

PARISA HEIDARI received the master's and Ph.D. degrees in computer engineering from the École Polytechnique de Montreal, Canada, in 2007 and 2012, respectively. She is a Cloud Developer with IBM Canada. Before that, she was an IoT Developer and an Security Master with Ericsson. She worked as an Research Associate with Concordia University in collaboration with Ericsson from 2013 to 2014. In 2015, she joined Ericsson Research in Montreal as a Postdoctoral Fellow. She holds to her credit several publications and patents. Her research interests include edge computing, cloud next generation, container technologies and server-less approach, smart resource dimensioning, optimal placement, and different aspects of QoS assurance in cloud systems.

ABDALLAH SHAMI (Senior Member, IEEE) is a Professor with the ECE Department, University of Wester Ontario, Londond, ON, Canada. He is the Director of the Optimized Computing and Communications Laboratory, Western University. He is currently an Associate Editor for IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE NETWORK, and IEEE COMMUNICATIONS SURVEYS AND TUTORIALS. He has chaired key symposia for IEEE GLOBECOM, IEEE ICC, IEEE ICNC, and ICCIT. He was the elected Chair of the IEEE Communications Society Technical Committee on Communications Software.