Fuzzy Handoff Control in Edge Offloading

Fani Basic, Atakan Aral, Ivona Brandic
Institute of Information Systems Engineering
Vienna University of Technology, Vienna, Austria
{fani, atakan, ivona}@ec.tuwien.ac.at

***Abstract*— Edge computing is a promising paradigm that relies on heterogeneous computing resources located at the edge of the network, close to the end-user. Hence, applications with latency-sensitive and compute-intensive tasks rely on edge resources to offload and complete such tasks. In order to support non-intermittent service in case of user mobility, most of existing approaches focus on how to accelerate the handoff transfer time and not how to reduce its frequency. Moreover, the handoff mechanisms used in cellular networks do not consider the computational workload and therefore are not directly applicable to the edge offloading scenario. Considering dense edge deployment, it is vital to select the optimal edge node for offloading. Therefore, we take into consideration bandwidth, processor speed and latency capabilities in the proposed fuzzy logic node selection algorithm. We evaluate the improvements of the proposed selection, for perceived response time objective, in comparison to offloading to the closest or highest bandwidth node. In addition, we propose a handoff controller, to meet the same performance objective, when the user is moving further from the currently selected edge node. We evaluate our approach by offloading Directed Acyclic Graph (DAG) models of real-world mobile applications. The results show that we can significantly reduce both application response time and monetary cost of execution, by controlling the number of handoffs among edge nodes.**

***Index Terms*— Mobile Edge Computing (MEC), handoff, mobile offloading, fuzzy logic, control theory**

I. Introduction

Nowadays, the mobile applications' low-latency requirements and demands for executing complex tasks are increasing rapidly. This is especially the case for emerging mobile applications such as face recognition, natural language processing, interactive gaming, and augmented reality [1]. By allowing resource constrained mobile devices to offload data- and compute-intensive tasks to the remote cloud, we can improve their battery lifetime and both storage and computation capabilities. However, computation offloading (also known as cyber foraging [2]) to the cloud infrastructure may result in longer response time (RT) and higher network cost. Therefore, fog computing paradigm proposes moving the computation to the continuum between the data center and the user, reducing the RT. In this paper, we consider the case, where edge nodes at one hop distance to the user are utilized for task offloading, usually referred as Mobile Edge Computing (MEC).

Each edge node within a small physical scope can provide end-to-end latency low enough to meet the demands of time sensitive applications. In sparse edge deployment the user has only one or few available edge nodes to connect at nearby location. On the contrary, in dense deployment [3], numerous location-aware edge nodes are tactically spread in many places (such as user residences and workplaces, streets, subway stations, coffee shops, airports, etc.), and overlaps in coverage areas are common. Thus, a user has multiple options when selecting the optimal edge node for the services. Since these nodes can differ significantly in computation capabilities from each other, node nearest to the user is not always the best choice in terms of perceived RT. Instead, a set of parameters must be considered jointly during this selection process in dense deployment, such as available bandwidth, processor speed and latency. We use those parameters in the proposed fuzzy logic-based algorithm with an intuitive idea of offloading the arriving task to the edge node most suitable to serve requests within expected RT, referred as the optimal node. Fuzzy logic fits our purpose of handling imprecision in these multiple input values caused by the dynamicity of network and compute workloads. In non-linear systems with arbitrary complexity and large number of inputs, fuzzy logic reasoning is among the best applicable techniques [4]. In addition, fuzzy logic can be easily tuned using expert knowledge through manipulation of rules or fuzzy sets [5], therefore it is adaptable to various application requirements.

Once the edge node is selected and user successfully offloads an application task, we still have to find solutions for the issues that lead to degraded application performance such as network congestion, node overload, failures and user mobility. As the user is moving further from the currently selected edge node, next offloaded tasks on that node can have degraded performance far worse than physical distance may suggest [6]. A process called handoff, that is offloading upcoming requests to a more suitable edge node in vicinity, can be effective in such a scenario. However, frequent mobility of users causes frequent handoffs among edge servers [7] and each handoff may bring initial monetary and time cost. For instance, each initialization of the task processing platform (e.g. container or virtual machine (VM)) incurs delays and expenses. In this paper, a handoff means the application moves to another node after the completion of the active task.

In our mechanism, referred as the handoff controller, we use control theory to avoid so-called ping-pong effect, in which subsequent tasks are transmitted back and forth between different edge servers in the surroundings. Running locally at the client, the handoff controller tracks tasks and exchanges information with the offloaded service running on the edge nodes. In our controller design, measured RT is the monitored variable that is used as output of our system and then compared to the desired RT. The adoption of well-established, mathe-

matically grounded control theory in our work is motivated by the ability to design and implement feedback loops to guarantee the system stability and application performance despite disturbances.

The main contributions of this paper are: (i) a fuzzy logic algorithm that selects a target node based on its bandwidth, processor and latency parameters; and (ii) a handoff controller that takes application RT as the indicator to decide whether to move the task execution to another node. We answer the questions: *"On which node should a task be offloaded for the execution?"* and *"Is it an appropriate moment to move execution to another node or not?"* based on perceived application performance and monetary cost of execution. At the time we write, user mobility and consequently the handoffs among edge servers is an open research challenge [7]. Handoff or handover techniques for cellular networks are not directly applicable, due to the intensive computation that has to be considered in edge offloading scenario [8]. To the best of our knowledge, this is the first attempt to use the well-established control theory concepts to control handoffs in edge offloading. While existing approaches aim to accelerate the handoff transfer time [6], [8], the novelty of our work lies in reducing the total number of performed handoffs with broadly explored, mathematically ground techniques of control theory. By using the Facerecognizer and Navigator real-world application models, we show that this novel approach fulfills the performance guarantees while avoiding frequent and repeated handoffs, which in comparison to the the baseline approaches, brings up to 86.14% shorter RT and 85.3% cost savings.

The rest of the paper is structured as follows. In Section II, we describe our motivating scenario. Section III introduces fuzzy logic and the proposed algorithm. In Section IV, control theory background and the handoff controller design are provided. The experimental environment and results are presented respectively in Section V and VI. Finally, we discuss related work in Section VII and conclude the paper in Section VIII.

II. Motivating Scenario

Proliferation of smart mobile devices has opened a door to new futuristic, computationally intensive mobile applications with near real-time requirements. We identify typical near real-time request-response synchronous applications, applicable to our approach, such as natural language processing and question answering, live translation service, text-to-speech functionality, virtual personal assistant, navigator and voice/image/video recognition. In a face recognition scenario, for instance, application RT would be the duration from when the user device starts to transfer an image to the target edge node to when the user receives the result, such as a string containing the name of the recognized person. The same goes for navigator or live translation applications, in which the user device sends GPS coordinates or spoken words and receives calculated path or translated words from the remote edge node. We assume that these applications are stateless. Thus, their tasks can be re-offloaded on another available node in the case of unavailability or failure of the current node. In the MEC

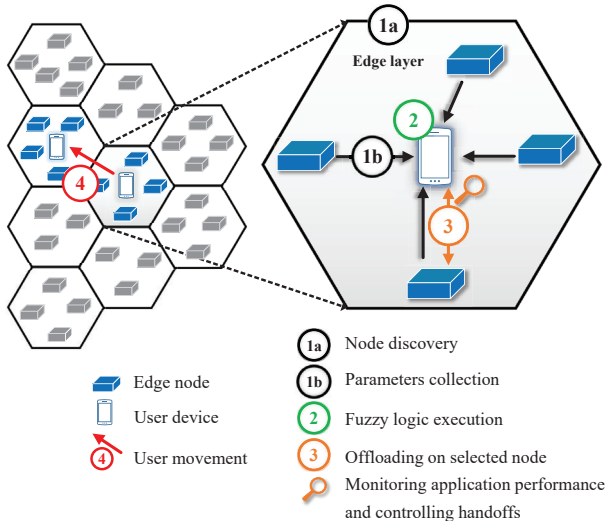
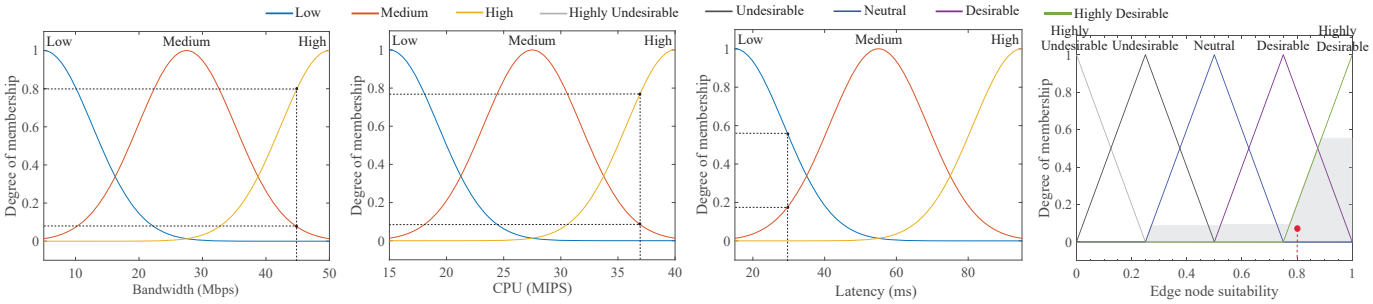


Fig. 1. Edge offloading model.

scenario, offloading of the upcoming tasks can be moved to a more suitable edge node, to improve user experience and facilitate smooth movement of mobile clients. However, each handoff increases execution time due to the initialization of a VM [6] or a container [8], [9] on the target edge node. Hence, controlling frequent handoffs, while ensuring required Quality of Service (QoS), can also reduce monetary cost and improve application performance.

A. Offloading model

Fig. 1 depicts the proposed offloading model. It starts when a user device receives user input in an application suitable for offloading. In step 1a, the user device starts discovery of the edge nodes in one-hop proximity using approaches from literature [10], [11]. Several edge nodes that are able to host the offloaded task are present on the physical region around the user illustrated with hexagons. We call them compatible edge nodes. Those nodes send their actual bandwidth, processor and latency capacities to the user device in step 1b. Using the collected edge node parameters, fuzzy logic-based target node selection starts in step 2, output of which is one target edge node. Using the result from the previous step, the application task is offloaded to the selected node. During step 3, a controller perceives the RT of each application's offloaded task. Meanwhile, the user device continuously offloads upcoming tasks until the controller output falls below the permitted threshold affected by the error, i.e. the difference between the desired and the measured RT. That happens in cases such as node overload or failure, network congestion, or user moving further away from the currently utilized node as shown in step 4. Accordingly, based on calculated control output, the controller can activate steps 1–3 again, searching for compatible nodes in the current physical region around user (step 1a) to offload subsequent tasks. Hexagon regions are symbolic in Fig. 1 and overlapping in the coverage areas of nodes is possible.



(a) Input MFs for the bandwidth. (b) Input MFs for the CPU. (c) Input MFs for the latency. (d) Output MFs for node suitability.

Fig. 2. Fuzzy inference system input and output sets.

B. Discussion

In our approach, it is possible to make offloading decision and node selection on the user device, since fuzzy logic simplifies system modeling by avoiding complex calculations. This way, a centralized orchestrator such as a cloud is not necessary. This allows, first, to avoid a single point of failure and ensure higher privacy of user data. Second, only client device and edge nodes are aware of perceived RT, precise user location needed to discover proximate nodes, and local network conditions. It is not efficient to transfer all this information to the cloud each time for a decision. To avoid side effects of local execution of the controller, such as increased battery consumption, possible computing overhead on the user device is reduced not only by a low computational complexity but also with infrequent execution of steps 1 and 2 described in Section II-A. This is because we perform a handoff only when the tasks' perceived RT is unacceptably deteriorated according to the controller, and not whenever there is an idle edge node in vicinity. To unburden the user device even more, another possible solution is to run the controller on a selected edge node. Once the handoff is performed to a more suitable node in user vicinity, the controller is initiated on that node as well. In the case of node failure, fallback design starts a fuzzy node selection on the user device, which then offloads tasks and triggers the controller on the selected edge node. This way, a user device battery lifetime is enhanced, but the edge node is additionally overwhelmed and loaded with the controllers of all running applications. However, this alternative implementation is out of the scope of this paper.

III. Fuzzy Logic-Based Edge Node Selection

Fuzzy logic, unlike classical Boolean logic, supports multiple intermediate values for attributes, rather than restricting them to binary extremes, making a decision based on in-between inputs [12]. This makes fuzzy inference systems effective in dealing with uncertainties and imperfect information among multiple input variables. In the fuzzy decision algorithm we proposed, three input parameters are used, namely: bandwidth capacity, processing power and latency. The edge node with the highest fuzzy output value is the most suitable to execute the offloaded task. We describe the three steps of the fuzzy reasoning mechanism as follows.

TABLE I
Fuzzy inference system rules.

BW	CPU	LAT	OUT	BW	CPU	LAT	OUT
LBW	LCPU	-	HU	MBW	HCPU	HLAT	N
LBW	MCPU	HLAT	HU	MBW	HCPU	MLAT	D
LBW	MCPU	MLAT	HU	MBW	HCPU	LLAT	D
LBW	MCPU	LLAT	U	HBW	LCPU	HLAT	D
LBW	HCPU	HLAT	HU	HBW	LCPU	MLAT	D
LBW	HCPU	MLAT	U	HBW	LCPU	LLAT	HD
LBW	HCPU	LLAT	U	HBW	MCPU	HLAT	D
MBW	LCPU	HLAT	U	HBW	MCPU	MLAT	HD
MBW	LCPU	MLAT	U	HBW	MCPU	LLAT	HD
MBW	LCPU	LLAT	N	HBW	HCPU	-	HD
MBW	MCPU	-	N				

A. Fuzzification

During fuzzification, input parameters are collected and mapped to relevant fuzzy sets with a membership value (or degree of membership) between 0 and 1 using the membership functions (MF) curve [13]. We use nine Gaussian MF to define input variables and five triangular MF for output, as they proved effective in [14] and in our evaluation. First, we define a Gaussian MF for each of three associated linguistic levels: *low* (L), *medium* (M) and *high* (H), for all the inputs: bandwidth (BW), processing power (CPU) and latency (LAT) as shown in Fig. 2a–2c. In addition, following triangular membership functions are used to represent the output (Fig. 2d): *Highly Undesirable* (HU), *Undesirable* (U), *Neutral* (N), *Desirable* (D), *Highly Desirable* (HD). To demonstrate fuzzification through an example assume that the following measures for the input parameters are collected: 45 Mbps of a bandwidth, 37 MIPS of processing capacity and 30 ms of latency. As shown with dashed lines in Fig. 2a–2c, we can observe that the BW value 45 Mbps lies in fuzzy set *medium* with a degree of membership (d.o.m.) 0.07 and in *high* with a d.o.m. 0.8; CPU value 37 MIPS lies in *medium* with a d.o.m. 0.08 and in *high* with a d.o.m. 0.77; latency value 30 ms lies in *low* with a d.o.m. 0.54 and in *medium* with a d.o.m. 0.18.

B. Inference engine

Inference engine infers the fuzzy output from the above defined fuzzy inputs through the if-then rule evaluation. Considering three possible values (L, M, H) for three inputs,

TABLE II
DEFINITIONS.

Symbol	Definition
C_{nodes}	Vector containing all discovered compatible nodes and parameters: BW, CPU, LAT
t_{node}	Index of a target node selected to serve offloading task
bw, cpu, lat	Bandwidth, processor and latency values
f_v	Output value that fuzzy logic returns
hf_v	Highest output value found
FuzzyLogic()	Call of fuzzy inference system mechanism

Algorithm 1 Fuzzy logic - target node selection

1: **function** FUZZYLOGICSELECTION(C_{nodes})
2: $t_{node}, bw, cpu, lat, f_v, hf_v \leftarrow 0$
3: **for** $i = 0$ to $|C_{nodes}| - 1$ **do**
4: $bw, cpu, lat \leftarrow C_{nodes}[i].bw, .cpu, .lat$
5: $f_v \leftarrow$ FuzzyLogic(bw, cpu, lat)
6: **if** $f_v > hf_v$ **or** $hf_v == 0$ **then**
7: $hf_v \leftarrow f_v$
8: $t_{node} \leftarrow i$
9: **end if**
10: **end for**
11: **return** $C_{nodes}[t_{node}]$
12: **end function**

we obtain 3^3 possible rule combinations as shown in Table I, where some rules are merged for brevity. Using these rules, an example of the reasoning process can be expressed as follows: if the BW is *high*, CPU is *high*, and LAT is *medium*, then consider this location as highly desirable for task offloading. Given prior exemplary inputs and their corresponding fuzzy sets (Medium/High BW, Medium/High CPU, Low/Medium LAT), eight rules are triggered in this step with the fuzzified output calculated as the minimum of the three membership values. Therefore, two implicated rules of {*MBW (0.07), MCPU (0.08), LLAT (0.54)/MLAT (0.18)*} give outputs in fuzzy sets *Neutral* to the extent of 0.07. Another two rules of {*MBW (0.07), HCPU (0.77), LLAT (0.54)/MLAT (0.18)*} give outputs in fuzzy sets *Desirable* to the extent of 0.07; both rules {*HBW (0.8), MCPU (0.08), MLAT (0.18)/LLAT (0.54)*} give output *Desirable* to the extent of 0.08; and finally both rules {*HBW (0.8), HCPU (0.77), LLAT (0.54)/MLAT (0.18)*} give output *Highly Desirable* to the extent of 0.54 and 0.18 respectively.

C. Defuzzification

In the defuzzification step, degree of membership of the output linguistic variables are converted to a resulting crisp output value. Following the above obtained edge node suitability values resulting in N (0.07), D (0.07 and 0.08) and HD (0.54 and 0.18), their maximum values result in the bounded area shaded in Fig. 2d. We use the widely accepted defuzzification method *center of gravity* (COG) [15] ($x^* = \int \mu(x)xdx \ / \int \mu(x)dx$ where μ is the bound formed by the output d.o.m and x is the edge node suitability). COG of the aforementioned area results in the output $x^* = 0.8$, highlighted with a red dot.

D. Node selection

Algorithm 1 shows the fuzzy logic-based target node selection, whereas Table II describes the notation used. The output of the fuzzy logic, which varies between 0 and 1, describes the suitability of each edge node for the offloading task, where greater values mean higher suitability. The algorithm runs on a user device and gets the list of all compatible nodes as the input. In lines 3 to 10, it calculates the fuzzy value for each edge node in a list considering bandwidth, processor and latency parameters. Parameters (line 4) are obtained from each single element in the compatible nodes list and mapped to the fuzzy sets according to their membership functions (line 5) as described in Section III-A. In addition, the inference rules from Table I are applied as explained in Section III-B. Defuzzification returns a quantitative value for each node, saved in variable f_v. Then, among the compatible nodes we select the one with the highest score (lines 6 to 9). The algorithm returns the edge node with the highest fuzzy value as the most appropriate one to serve offloaded task (line 11).

Computational complexity of the proposed Algorithm 1 is *O(n)* due to the *for* loop iteration (line 3), and related to *n*, the number of discovered nodes. Complexity of Fuzzy logic steps (line 5), is independent of *n*, hence they take constant time. Proposed algorithm targets only the edge nodes with low latency, in close proximity to user. Therefore, *n* is not expected to be large enough to cause notable delay.

IV. HANDOFF CONTROLLER

In a standard feedback control loop architecture, the controller periodically adapts the value of its output, ctl_i to reduce or eliminate the error. The error is the difference between the desired and measured output, strongly affected by the uncontrollable disturbances in the target system. In our approach, the system is the application offloaded on the *Selected Edge Node* (Fig. 3) and disturbances are the sudden changes in the traffic load, network and hardware failures as well as user mobility. Despite disturbances, mobile applications are expected to provide certain performance guarantees and response time has strong impact on user satisfaction. Therefore, we define the desired RT, $\widetilde{rt}$, that is a reference input in our model predefined by the developer, user or provider. The measured output at iteration i, rt_i, is the application's RT and it is used as a feedback variable. The difference between these two values is the control error, e_i (1) and a single iteration i is the time it takes for the execution of one offloaded task.

$$e_i = \widetilde{rt} - rt_i \tag{1}$$

At each iteration i, ctl_i value is compared with current control threshold value, τ_i, which we compute based on (2) in a *Threshold Evaluation* phase (Fig. 3). It changes dynamically based on how many tasks of the application suffer deteriorated response time on the same node. Here, deterioration means failing to achieve the performance goals of the application since measured RT becomes higher than the desired one. Therefore, more tasks with deteriorated performance results in higher τ_i and higher probability that ctl_i will fall under it.

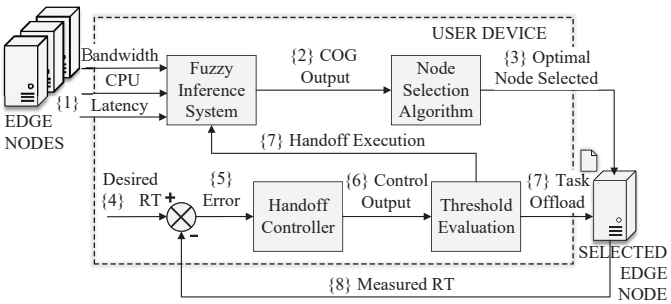
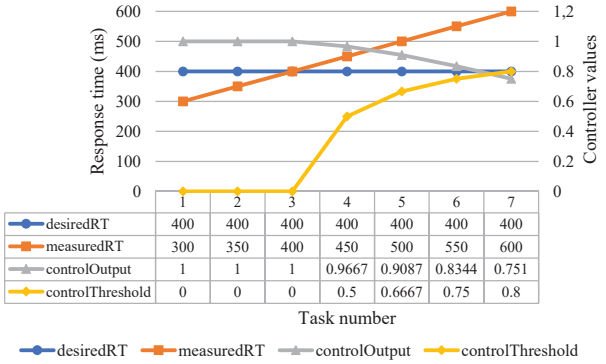


Fig. 3. Information flow within the proposed architecture.



	1	2	3	4	5	6	7
desiredRT	400	400	400	400	400	400	400
measuredRT	300	350	400	450	500	550	600
controlOutput	1	1	1	0.9667	0.9087	0.8344	0.751
controlThreshold	0	0	0	0.5	0.6667	0.75	0.8

Fig. 4. Simplified example of controller work, with the pole set to 0.7 and the desired RT of 400 ms.

When that happens, a *Handoff Execution* begins along with *Fuzzy Inference System* steps and a *Node Selection Algorithm* among the discovered nodes, as described in Section III. This process is depicted in Fig. 3. Execution terminates when there is no more user input in application or if application has no more task to offload. Otherwise, we offload next task and continue to use controller for calculating ctl_i of the current task.

$$\tau_i = 1 - \frac{1}{(1 + \text{Number of Deteriorated Tasks})} \tag{2}$$

The controller output ctl_i, is computed using equation (3), originally proposed in [16]. We adopt this approach since it is a generic yet practical method with mathematical background to devise ad-hoc control solution. ctl_i is based on its previous value, control error and value of α and $pole$, described as follows. The system model parameter α is estimated at each control interval based on the effect of the ctl_i on the measured RT, ($\alpha_i = rt_i/ctl_i$). A tunable parameter called $pole$, influences the stability of the controlled system, and determines how fast the system approaches to its equilibrium. To maintain the system stability, the pole values should be between zero and one [16]. The closer the pole is to 0, the shorter is the settling time since the controller reacts faster and more aggressively. On the contrary, setting the pole close to 1 produces a more robust and conservative controller, which does smaller adjustments of the control output and takes smaller steps towards the goal.

$$ctl_i = ctl_{i-1} + \frac{1 - pole}{\alpha_i} \cdot e_i \tag{3}$$

We provide a simplified example in Fig. 4, to demonstrate clearly how the proposed controller works. Assume the desired RT of an application running on the user device to be 400 ms. However, we observe an increasing trend in measured RT that goes over 400 ms after the first three tasks. Controller output is set to 1 and control threshold to 0 in the beginning, and their values are always in the range $[0, 1]$. Controller values change after offloading the fourth task since it has deteriorated RT. In this example, the pole is set to 0.7 and measured RT after the seventh task is 600 ms. At this point the calculated control value falls under the control threshold. Therefore, the eighth task will be executed on another edge node where control output and threshold will be again set to their default values 1 and 0, respectively. On the same configuration, controller with the pole 0.3 reacts faster and performs handoff after the fifth task with 550 ms measured RT, whereas controller with pole 0.9 is more conservative and handoff is needed only after the eighth task and 700 ms measured RT. These two alternative controllers are omitted for brevity.

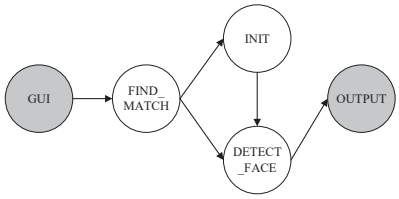
Our results in Section VI show that performing the service handoff each time when measured RT is deteriorated (indicated with (4) in Fig. 4) is too frequent and costly in terms of time and money. Our controller is able to avoid frequent handoffs and decide whether it is an appropriate moment to change the application deployment to a more powerful and less saturated node in the vicinity. Sometimes, there might not be any other node that is more suitable in terms of its calculated fuzzy value for the execution of the upcoming task. In such cases, we propose to continue offloading future tasks on the current node without handoff. Other scenarios such as cloud or local execution are beyond the scope of this paper.

V. Experimental Environment

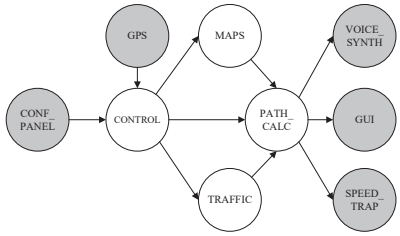
A. Simulation environment

Research on Edge computing is still in its infancy and at the time we write, no real-world MEC infrastructure is available to perform our experiments. IFogSim [17] and EdgeCloudSim [18] are the existing simulators, widely used in academia. However, they do not support either mobility models or handoff mechanisms and do not allow to specify when and where to migrate. Therefore, we build a simulation framework using Java programming language to perform the evaluation of our approach. It is a discrete event-based simulator and starts with offloading the first task and then iterates through all the application tasks, considered as the device workload. Each generated task is offloaded using the fuzzy handoff controller on a dynamic edge computing infrastructure. During this process, simulator logs the statistics about the tasks RT, the number of a performed handoffs and the time of the node usage. For the sake of reproducibility and to foster further research, we made all source code publicly available[1].

[1]https://github.com/BasFa/FHC-simulator/



(a) Facerecognizer application.



(b) Navigator application.

Fig. 5. DAG models of real-world mobile applications with their offloadable (light nodes) and not-offloadable (dark nodes) tasks.

All simulation results are obtained on a 64-bit Windows 10 machine, configured with a 2.70-3.50 GHz Intel i7-7500U CPU and 16 GB memory. To measure the benefits of our approach accurately and to ensure a confidence interval of 95%, we repeat the simulations 100000 times, each time on a different edge node topology generated. Fuzzification, inference, and defuzzification steps of fuzzy reasoning mechanism are implemented utilizing jFuzzyLogic library [19].

B. Mobile applications

Applications are modeled as a set of tasks (T). Each task is defined as $T(DATA, MI)$, where DATA is the size of task data and MI is the task length in millions of instructions. To the best of our knowledge, there are no real-world traces to cover these mobile application parameters, available to the research community at the present time. Therefore, we employ Directed Acyclic Graph (DAG) models of real-world mobile applications, used also in other works such as [20], [21]. Two popular applications with strong computational requirements are considered, namely, *Facerecognizer* and *Navigator* (Fig. 5). For each DAG, nodes represent the application tasks and edges are the dependencies between them. Some tasks are offloadable (light nodes), and others, like graphical user interface (GUI) rendering or GPS usage, are device dependent and hence not offloadable (dark nodes). Facerecognizer application consists of five compute-intensive tasks and navigator application of nine data-intensive tasks defined in [20]. The offloadable tasks structure and requirements are outlined in the first three columns of Tables III and IV. The tasks are run sequentially with the respect of the topological order of the DAG. The goal of the Facerecognizer application is to identify faces on a given image using image processing, whereas in the navigator application the goal is to show the fastest route on the map, and recalculate it in case of changes. Observing these applications, we define realistic sizes for the *image* $\{1, 5\}$MB and *map* $\{5, 10, 15\}$MB and use them as $\frac{1}{\lambda}$ parameter of the exponential distribution at each simulation.

TABLE III
FACERECOGNIZER APPLICATION SETTINGS

Task	MI	DATA (MB)	RT (ms) 1MB	RT (ms) 5MB
FIND_MATCH	$\frac{1}{\lambda}$ = 4	$\frac{1}{\lambda}$ = image	413	686
INIT	$\frac{1}{\lambda}$ = 4	$\frac{1}{\lambda}$ = image	414	687
DETECT_FACE	$\frac{1}{\lambda}$ = 8	$\frac{1}{\lambda}$ = image	686	959

TABLE IV
NAVIGATOR APPLICATION SETTINGS

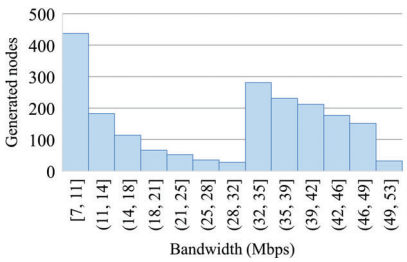
Task	MI	DATA (MB)	RT (ms) 5MB	RT (ms) 10MB	RT (ms) 15MB
CONTROL	$\frac{1}{\lambda}$ = 2	5	552	552	552
MAPS	$\frac{1}{\lambda}$ = 3	$\frac{1}{\lambda}$ = map	619	955	1304
TRAFFIC	$\frac{1}{\lambda}$ = 5	$\frac{1}{\lambda}$ = map	754	1089	1439
PATH_CALC	$\frac{1}{\lambda}$ = 5	$\frac{1}{\lambda}$ = map	754	1090	1439

During a single simulation, we measure how long it takes to execute 200, 500 and 1000 tasks, how many handoffs are made, and what is the monetary cost. Such high number of tasks are not unexpected, since we do not consider offloading a single task by a static user, but a moving user who continues to use the application for a longer period of time as a physical distance between the user and node increases. We can think of the navigator application used in a vehicle or by a pedestrian where traffic calculation, speed trap indication, route updates and recalculation followed by voice instructions are repetitively triggered. Another use case includes surveillance/tracking applications with face/object detection functionalities; or a tourist that walks through the city and explores it using a augmented reality glasses with face/object recognition feature.

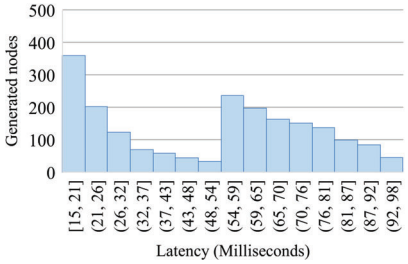
C. Edge nodes and the network infrastructure

To simulate a heterogeneous MEC infrastructure for evaluation, we define an edge node as $EN(BW, MIPS, LAT)$, where BW represents available bandwidth between the user and the node, MIPS represents the millions of instructions that can be executed per second, and LAT is the network latency. The closer the user is to the edge node, the lower is the latency.

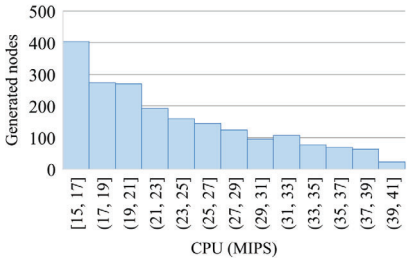
Previous work shows that heavy-tailed distribution is a good fit to approximate geographical variety of network latency [22], [23]. We use Pareto distribution to generate edge nodes latencies and choose 1.25 as the shape parameter, which is experimentally shown to be more similar to real-world values [24]. Similarly for bandwidth values, we follow the methodology in the most-widely used random network topology generator, BRITE [25], which use Pareto distribution among others. Pareto is also a good approximation for the CPU utilization of virtualized servers in data centers [26]. We take scale parameters for the distributions from [20], where average CPU power is 15 MIPS, average latency is 15 or 54 ms depending on Wi-Fi or cellular (3G) connection, and average



(a) Bandwidth PDF of the two aggregated Pareto distributions with scale parameters 7.2 Mbps for 3G and 32 Mbps for Wi-Fi.



(b) Latency PDF of the two aggregated Pareto distributions with scale parameters 15 ms for Wi-Fi and 54 ms for 3G.



(c) CPU PDF of Pareto distribution with a scale parameter 15 MIPS.

Fig. 6. Edge topology parameters for 2000 generated nodes.

bandwidth is 32 Mbps for Wi-Fi or 7.2 Mbps for 3G. Resulting probability density functions (PDF) are shown in Fig. 6.

In our simulation, we consider the impact of the network and edge nodes' workload on applications' QoS requirements. Due to the novelty of MEC technology, and concerns about making commercial systems' workload traces publicly available, we uses dynamic workloads at edge nodes, where percentage BW and CPU change (Δ_{BW} and Δ_{CPU}) is chosen uniformly at random in each time step. The range of the random function is also unique to each edge node in order to generalize our results with various workload trends (increasing or decreasing with different slopes). To that end, Δ_i is chosen from the range $[x_i, y_i]$ where $x_i = rand(-1, 1)$, $y_i = rand(-1, 1)$, and $x_i < y_i$ for all $i \in \{\text{BW}, \text{CPU}\}$. Limits x_i and y_i are fixed and unique to each node. Positive values represent increasing trends and vice versa. When two values are farther from each other, workload trend is highly unstable. In this way, we are able to cover many possible workload characteristics. For latency changes, we choose a value of Δ_{LAT} between 0 and 1 using uniform random distribution and it always represents the percentage of increase since we assume that the user is moving further away from the selected node. Initially, the logical topology consists of 20 edge nodes, but each time handoff should be done due to RT deterioration and user movement, a new scope of 20 nodes is found. We consider the previous nodes unavailable from that time on.

D. Performance measures

Our aim is to offload tasks on the selected target node and keep the response time below a predefined set point. The response time includes the communication delay, the network transmission delay of sending data to the edge server, and the execution time on that server [3]. The communication delay is the perceived two-way latency between client and edge $(2 * EN(LAT))$. The network transmission delay is measured as the data size of task sent and response received, divided with the available bandwidth $(2 * T(DATA)/EN(BW))$. We assume that sent and received data have the same size. The execution time on the server is measured as the number of instructions divided by edge MIPS $(T(MI)/EN(MIPS))$. The full response time calculation is given in (4).

$$RT = 2 * EN(LAT) + 2 * \frac{T(DATA)}{EN(BW)} + \frac{T(MI)}{EN(MIPS)} \quad (4)$$

In case of RT deteriorations, controller triggers a handoff process to a more appropriate node. However, each handoff brings additional overhead and increases the application execution time. For generality, we cover three empirical values for this additional delay: $0.42s$ when the application image is already downloaded [9], $15.8s$ when launching a container [8], or $39.3s$ when launching a VM on the newly selected edge node [6].

Monetary cost (M) is calculated using Amazon EC2, On-Demand Pricing[2] for the region EU (London) as shown in (5), since similar pricing can be expected for edge resources once the edge infrastructure becomes publicly available. The unit amount $price$ is charged only for what is used by the second with the minimum of a minute, e.g. $\$3.00 \times 10^{-5}$ or $\$6.33 \times 10^{-5}$ per second for m5.large and m4.large instances. The price is based on the time when the edge node is initialized and the first task is executed, until the time the resources on that node are released due to the performed handoff. We consider that each handoff brings an additional monetary cost if the node n is utilized less than a minute. This is reflected in our cost calculation in (5).

$$M = \sum_n \max(time_n, 60) * price \quad (5)$$

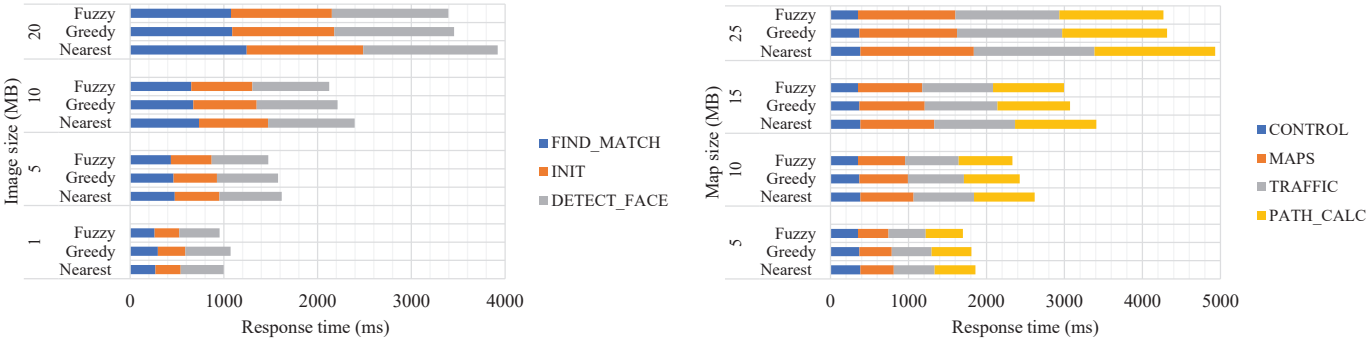
VI. Numerical Results

In this section, first we demonstrate that the fuzzy logic node selection can minimize the application RT. In addition, we define a model for desired RT used in our feedback loop. Then, we evaluate the controller through exhaustive simulation.

A. Fuzzy logic node selection

We compare the proposed policy (*Fuzzy*) to two other baseline strategies of choosing the node with the highest bandwidth (*Greedy*) and offloading to the nearest edge node (*Nearest*), which is a very common approach in the literature [8], [27], [28]. In our implementation, the nearest node to the user is the one with the lowest latency value. The proposed

[2]https://aws.amazon.com/ec2/pricing/on-demand/



(a) Facerecognizer application.

(b) Navigator application.

Fig. 7. RT perceived for offloadable tasks of Facerecognizer and Navigator applications.

approach, as shown in Fig. 7, offers substantial gains over those baseline approaches.

For the Facerecognizer application (Fig. 7a) with an image size of 1 MB, *Fuzzy* approach decreases the average RT, in comparison to the *Nearest* approach, by 4.35% and in comparison to the *Greedy* approach, by 10.93%. This is because we consider more relevant parameters (BW, CPU, LAT) for this computation intensive task. However, considering task sizes in range from 5 to 20 MB, the *Greedy* approach outperforms the *Nearest*, since increase in data size puts more weight on data transfer time. In these cases, comparing to the *Greedy* approach, ours decreases the time cost by 6.65% in the best case and by up to 1.73% in the worst case.

For the Navigator application (Fig. 7b) with a map size of 5 MB, our *Fuzzy* approach decreases the average RT, in comparison to the *Greedy* approach, by 5.99%. Whereas, for a 25 MB task, the difference between our and the *Greedy* approach decreases to 1.12%, because network transmission delay gets more dominant as the data size of task is increased. Accordingly, for higher data sizes other methods might outperform ours, but such a large data size is unrealistic for the applications relevant to edge computing. Obtained results confirm our assumption that taking into account parameters such as computation power, distance from the user and bandwidth in dense edge environment brings substantial RT benefits.

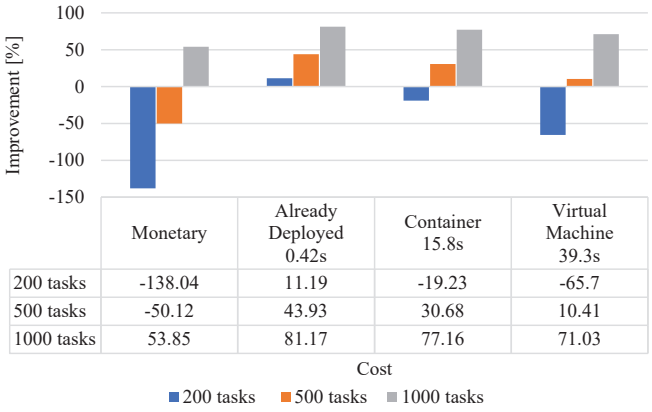
After each offloaded and processed task, the client receives the response and records the RT. Different applications have different requirements, especially in terms of RT, which are assigned by the application developer, service provider or user. Using the results from the simulations above, we define the desired RT for each application task. It is calculated as the average task RT measured in *Fuzzy*, *Greedy* and *Nearest* policy, with an allowed additional overhead of 50%. This overhead is included since used exponential distribution results in wide range of MI and DATA values. With higher granularity this overhead can be reduced or removed. However, we present the desired response time estimate calculated for *image* = {1, 5}MB and *map* = {5, 10, 15}MB in the remaining columns of Tables III and IV respectively.

B. Handoff controller

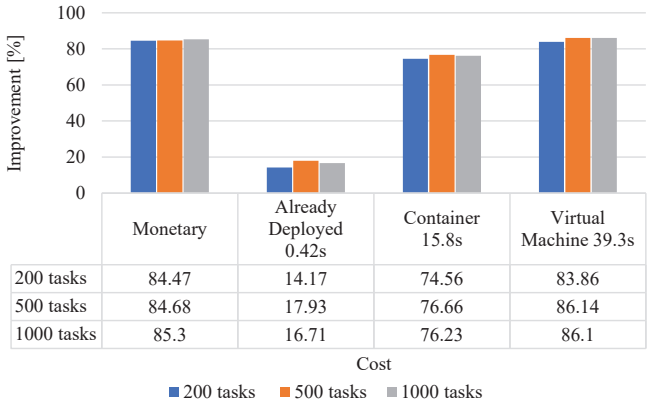
As explained before, node selection also affects the number of handoffs made during the application execution. For the second set of experiments to evaluate the controller, we use the pole value of 0.7 based on our empirical analysis of various results. However, due to the expected high initial monetary cost (calculated in (5)), we set pole value to 0.9 during the first minute of execution to make a controller more robust and less prone to handoffs. A device workload is composed of the Facerecognizer and Navigator applications' workload executed sequentially by the user.

Our approach, Fuzzy node selection with Handoff Controller (FHC), is compared to two baselines. In both, to make the comparison fair, we choose the nodes with the fuzzy selection algorithm, since we show that it outperforms *Greedy* and *Nearest* offloading (Section VI-A). In the first one (*Never Handoff*), all tasks are offloaded to the same single node that is found in the beginning of the application execution. No handoffs are made even if RT violations occur. In the second scenario (*Always Handoff*) the user starts offloading tasks to one node and performs handoffs as soon as the previous offloaded task experiences the first RT violation. We show that the proposed *FHC* approach offers significant improvement over those baseline approaches. In Fig. 8 we present percentage improvement in terms of monetary savings (*Monetary*) and average response time. Application RT includes the additional time cost each handoff can bring, as follows: 39.3 seconds for launching a VM on the newly selected node (*Virtual Machine 39.3s*), 15.8 seconds for a container (*Container 15.8s*) and 0.42 seconds for launching a container when application image is already downloaded (*Already Deployed 0.42s*). Negative values show the cases when other approaches outperform *FHC*.

Fig. 8a shows the improvement our approach brings comparing to *Never Handoff*. For a workload with 200 and 500 tasks, *FHC* brings 138.04% and 50.12% additional monetary cost respectively, since the user stays quite near to the initially selected node and latency does not significantly increase the RT and hence the monetary cost with the baseline approach. The longer the application execution is, e.g. 1000 tasks, *FHC*



(a) *FHC* and *Never Handoff*.



(b) *FHC* and *Always Handoff*.

Fig. 8. Approaches comparison in terms of monetary and time costs that handoffs bring.

brings up to 53.85% monetary saving, since the task RT is increasingly shorter due to handoff to the closer nodes with better performance. In total application response time including the handoff time, *FHC* brings improvements up to 81.17%. Therefore, these increments in monetary cost are outweighed by shorter response time, except for short workload with 200 tasks combined with very high handoff time of 15.8s and 39.3s, in which case *FHC* takes from 19.23% up to 65.7% longer RT.

Fig. 8b shows the comparison of our approach to the *Always Handoff* approach. Improvements in RT are not affected by the number of offloaded tasks, due to continuous handoffs to the closer better performing nodes. Whereas in *Never Handoff* approach, workloads with a bigger number of executed tasks, result in longer RT due to the latency increase as the user is moving further away from the initially selected node. Moreover, *FHC* lowers the number of handoffs and improves application RT by 74.56% and 86.14%, considering handoff time cost of 15.8s and 39.3s respectively. In cases when handoff brings lower time cost, i.e. 0.42s, improvements in the response time are smaller and go up to 17.93%. This is due to the fact that our handoff controller postpones handoff until the controller's output falls under the control threshold, consequently some tasks perceive deteriorations in RT and the handoff time cost is not large enough to compensate even more. Furthermore, due to the lower number of handoffs, *FHC* brings monetary savings of up to 85.3% by means of the handoff controller.

VII. Related work

The advantages of edge technologies is proven in many use cases [7] with some major research challenges open, such as dealing with frequent handovers among edge servers caused by user mobility. Different offloading techniques with mobility management covered for the edge context are discussed in [29]. The effectiveness of edge infrastructure employment to support mobile application is considered in [30]. As distinct from these works, we also consider monetary costs. Offloading cost models have been discussed in [20], [31]. However, they compare user cost in interplay between remote cloud and edge cloud usage and not in terms of handoffs.

Analogous to the seamless handoff or handover in cellular networks, where the client is changing the base station while communicating, the edge node handoff as well allows interruption-free client mobility [8], [32]. With a key difference that intensive task computation has to be done in edge offloading scenario. Therefore, the edge node has to be pre-provisioned with needed capabilities or all the runtime states of offloaded workload must be transferred to the node, as authors emphasize in [8]. To overcome the differences in a fog computing handoff mechanism, with those studied in cellular networks, authors in [32] define general architecture components for location-based VM migration. Moreover, they emphasize the importance of a decision-making process that triggers VM migration in fog computing, not yet covered in the literature. Authors in [33] used a control theory to synthesize a controller for vertical memory elasticity of cloud applications. However, authors do not consider horizontal scaling important in large-scale volatile edge computing scenarios. While most studies aim to reduce the handoff transfer time [6], [8], the novelty of our work is controlling the number of handoffs both by selecting the optimal target node and by monitoring applications performance with a respect of response time objectives. Moreover, we consider the monetary cost of hosting the service at the MEC.

Rather than selecting target node in dense environment, there are many studies that focus on what to offload [21], [34] and whether to execute a task locally or offload it to edge or cloud as in [35], [36]. However, a fog service placement problem, with the goal of reducing the execution time of applications, implemented as integer linear programming is discussed in [27]. Works like [8], [27], [28] perform offloading to the nearest edge node, without considering other parameters such as significant differences in computation capabilities nodes may have. Therefore, authors in [11] propose cloudlet selection heuristic based on requirements of the application

and capability and stability of the mobile cloudlet. Nonetheless, they do not consider handoffs among servers.

VIII. Conclusion

The main contributions of this study can be summarized as follows. First, we describe our offloading model with the following steps: (i) optimal node selection algorithm using fuzzy logic, which outputs a single node; and (ii) handoff-decision-making via control theoretical approaches based on application performance. Furthermore, we design a simulation framework and we use DAG models of real-world applications for the evaluation. Our experimental results demonstrate the benefits our controller brings in reducing the monetary cost by up to 85.3% through avoiding frequent handoffs. We also compare the initialization time cost for different configurations (VMs and containers) and achieve improvements in terms of application response time by up to 86.14% in comparison to baseline approaches. As future work, we plan to evaluate our approach through real-world user mobility traces. Moreover, we plan to handle workload balancing more explicitly and in combination with proactive handoff control.

Acknowledgment

The work described in this paper has been funded through the Rucon project (Runtime Control in Multi Clouds), FWF Y 904 START-Programm 2015.

References

[1] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[2] M. Satyanarayanan, "Pervasive computing: vision and challenges," *IEEE Personal Communications*, vol. 8, no. 4, pp. 10–17, 2001.

[3] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *IEEE Int'l Conference on Distributed Computing Systems*, 2017, pp. 2573–2574.

[4] E. H. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *International journal of man-machine studies*, vol. 7, no. 1, pp. 1–13, 1975.

[5] A. N. Toosi and R. Buyya, "A fuzzy logic-based controller for cost and energy efficient load balancing in geo-distributed data centers," in *IEEE/ACM International Conference on Utility and Cloud Computing*, 2015, pp. 186–194.

[6] K. Ha, Y. Abe, Z. Chen, W. Hu, B. Amos, P. Pillai, and M. Satyanarayanan, "Adaptive vm handoff across cloudlets," CMU School of Computer Science, Tech. Rep., 2015.

[7] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[8] L. Ma, S. Yi, and Q. Li, "Efficient service handoff across edge servers via docker container migration," in *ACM/IEEE Symposium on Edge Computing*. ACM, 2017, p. 11.

[9] E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwälder, "Incremental deployment and migration of geo-distributed situation awareness applications in the fog," in *ACM International Conference on Distributed and Event-based Systems*. ACM, 2016, pp. 258–269.

[10] G. A. Lewis, P. Lago, and P. Avgeriou, "A decision model for cyber-foraging systems," in *Working IEEE/IFIP Conference on Software Architecture*, 2016, pp. 51–60.

[11] S. Chilukuri, S. Bollapragada, S. Kommineni, and K. C. C., "Raincloud - cloudlet selection for effective cyber foraging," in *IEEE Wireless Communications and Networking Conference*, 2017, pp. 1–6.

[12] L. A. Zadeh, "Fuzzy sets," in *Fuzzy Sets, Fuzzy Logic, And Fuzzy Systems*. World Scientific, 1996, pp. 394–432.

[13] J. Roger and G. Ned, "Fuzzy logic toolbox for matlab," The Math Works Inc., USA, Tech. Rep., 1995.

[14] A. V. Dastjerdi and R. Buyya, "Compatibility-aware cloud service composition under fuzzy preferences of users," *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 1–13, 2014.

[15] Y. Bai and D. Wang, "Fundamentals of fuzzy logic control - fuzzy sets, fuzzy rules and defuzzifications," in *Advanced Fuzzy Logic Technologies in Industrial Applications*. Springer, 2006, pp. 17–36.

[16] C. Klein, M. Maggio, K.-E. Årzén, and F. Hernández-Rodriguez, "Brownout: Building more robust cloud applications," in *International Conference on Software Engineering*. ACM, 2014, pp. 700–711.

[17] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[18] C. Sonmez, A. Ozgovde, and C. Ersoy, "Edgecloudsim: An environment for performance evaluation of edge computing systems," in *Second International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE, 2017, pp. 39–44.

[19] P. Cingolani and J. Alcala-Fdez, "jfuzzylogic: a robust and flexible fuzzy-logic inference system language implementation," in *2012 IEEE International Conference on Fuzzy Systems*. IEEE, 2012, pp. 1–8.

[20] V. De Maio and I. Brandic, "First hop mobile offloading of dag computations," in *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2018, pp. 83–92.

[21] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *International conference on mobile systems, applications, and services*. ACM, 2010, pp. 49–62.

[22] W. Zhang and J. He, "Modeling end-to-end delay using pareto distribution," in *International Conference on Internet Monitoring and Protection*. IEEE, 2007, pp. 21–21.

[23] T. Høiland-Jørgensen, B. Ahlgren, P. Hurtig, and A. Brunstrom, "Measuring latency variation in the internet," in *Int'l Conference on Emerging Networking Experiments and Technologies*. ACM, 2016, pp. 473–480.

[24] G. Hooghiemstra and P. Van Mieghem, "Delay distributions on fixed internet paths," Delft University of Technology, Tech. Rep., 2001.

[25] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: Universal topology generation from a user's perspective," Boston University Computer Science Department, Tech. Rep., 2001.

[26] S. Di, D. Kondo, and F. Cappello, "Characterizing cloud applications on a google data center," in *International Conference on Parallel Processing*. IEEE, 2013, pp. 468–473.

[27] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards qos-aware fog service placement," in *IEEE International Conference on Fog and Edge Computing*, 2017, pp. 89–96.

[28] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck, "Mobile edge computing potential in making cities smarter," *IEEE Communications Magazine*, vol. 55, no. 3, pp. 38–43, 2017.

[29] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[30] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 26–35, 2017.

[31] T. Zhao, S. Zhou, X. Guo, Y. Zhao, and Z. Niu, "Pricing policy and computational resource provisioning for delay-aware mobile edge computing," in *IEEE/CIC International Conference on Communications in China*. IEEE, 2016, pp. 1–6.

[32] L. F. Bittencourt, M. M. Lopes, I. Petri, and O. F. Rana, "Towards virtual machine migration in fog computing," in *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, 2015, pp. 1–8.

[33] S. Farokhi, P. Jamshidi, E. B. Lakew, I. Brandic, and E. Elmroth, "A hybrid cloud controller for vertical memory elasticity: A control-theoretic approach," *Future Generation Computer Systems*, vol. 65, pp. 57–72, 2016.

[34] Y. Zhang, H. Liu, L. Jiao, and X. Fu, "To offload or not to offload: an efficient code partition algorithm for mobile cloud computing," in *IEEE Int'l Conference on Cloud Networking*. IEEE, 2012, pp. 80–86.

[35] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[36] T. Zhao, S. Zhou, X. Guo, Y. Zhao, and Z. Niu, "A cooperative scheduling scheme of local and internet cloud for delay-aware mobile cloud computing," in *IEEE Globecom Workshops*, 2015, pp. 1–6.