Mobility-Aware Container Migration in Cloudlet-Enabled IoT Systems using Integrated Muticriteria Decision Making

Mutaz A. B. Al-Tarawneh
Computer Engineering Department, Mutah University
Karak, JORDAN 61710

***Abstract*—Service migration plays a vital role in continuous service delivery in Internet of Things (IoT) systems. This paper presents a mobility-aware container migration algorithm for Cloudlet-enabled IoT systems. The proposed algorithm is based on an integrated multicriteria decision making (MCDM) approach. It has been implemented using a specialized simulation tool and compared to other existing migration algorithms. Simulation results demonstrate the ability of the proposed algorithm to achieve up to 48%, 48%, 20% and 36% improvement in migration time, service downtime, migration reliability and service loss rate, respectively as compared to other migration algorithms. The proposed algorithm is capable of perceiving the run-time dynamics of IoT systems and appropriately manage the process of container migration.**

***Keywords*—Internet of Things (IoT); container; migration; Cloudlet; criteria; decision making**

I. Introduction

Nowadays, the tremendous growth of commodity-available smart objects has resulted in the ubiquity of Internet of Things (IoT) applications. These objects can be either fixed or mobile and generate streams of big data with immense processing and memory requirements that surpass the capabilities of user devices. Therefore, IoT applications urged the use of Cloud Computing as a processing back-end that fulfills the processing requirements of such applications [1].

Following the expansion in IoT applications and the elevation of their processing requirements and the urgency of their latency constraints, the inseparable relation between the IoT applications and the Cloud has been hindered due to the quasi-central nature of the Cloud-based services and resources [2]. In order to cope with the processing and timing requirements of IoT applications and overcome the limitations of the Cloud-based IoT platforms, several edge computing models [3] such as Multi-access Edge Computing [4], Fog Computing [5] and Cloudlet Computing [6] have been proposed. For instance, Cloudlet Computing is considered as a middle-way between Fog Computing and Cloud Computing models. It places computing clusters with sufficient network bandwidth and processing capabilities in proximity to the IoT devices. A Cloudlet is basically a small data center, which is usually instantiated at few hops away from IoT devices and employ virtualization at either virtual machine or container levels [7].

According to the Cloudlet Computing paradigm, IoT application modules and services are placed on close tiny data centers aiming at satisfying the stringent timing requirements of the requests generated from IoT devices. Computing requests - generated from IoT devices are offloaded to such nearby tiny data centers which in turn perform the required functionality as dictated by the received requests. Once the received request is executed, execution results are sent back to the source IoT device [7]. In order to efficiently utilize the Fog/Cloudlet resources and minimize any potential performance overheads, a lightweight virtualization technique is employed, in which application modules and user data are encapsulated in containers [8, 9].

While Cloudlet Computing provides Cloud-like services in proximity to IoT devices and offers low-latency services to these devices, its advantages could be diminished in case of mobile or non-stationary IoT devices, where the access point that connects the IoT device to the Cloudlet may change [10]. In such scenario, user or IoT device mobility will increase the number of hops between the IoT device the Cloudlet hosting the associated container and, in turn, negatively impact the latency requirements of IoT devices. In other words, as the number of hops between the IoT device and its associated Cloudlet increases, the latency of IoT service requests will increase in a manner that resembles that of Cloud-based platforms, which shrinks the expected performance of Cloudlet-based platforms. Therefore, as the mobile IoT devices move away from their associated Cloudlets, both processing and control of their application modules should be handed over to the Cloudlet in proximity to the access point to which the IoT device is currently connected. This process can be performed by migrating the container, associated with the mobile IoT device, to a new Cloudlet that is placed near to the IoT device and respects that device's functional and non-functional requirements [11].

Apparently, IoT devices requesting services from nearby Cloudlets may be mobile and require continuity of service delivery across different locations. However, maintaining service delivery for mobile users and IoT devices is a challenging process. The study in [12] has proposed a framework for a Fog Computing architecture in which virtual machine migration is supported. They have assumed that each user has a virtual machine running on a particular Cloudlet. This Cloudlet is considered as an endpoint that provides services to the associated user. A virtual machine could be migrated to another Cloudlet based on user location, direction of movement, running applications, computing capacity of the destination Cloudlet and the network capacity. However, the proposed framework has not been evaluated and its suitability to existing Fog environments has not been verified. On the other hand, [13] has proposed a genetic algorithm-based model for virtual machine migration in Mobile Cloud Computing (MCC) environments considering both user mobility and the current workload of the potential destination Cloudlet in order to reduce the number of virtual machine migrations. In addition, [14] has proposed a simulation framework based on the iFogSim tool [15] to support user mobility by migrating virtual machines across Fog nodes. Furthermore, [16] has proposed a Fog computing architecture that supports user mobility by implementing a route optimization algorithm that improves the performance of the handover mechanisms. Similarly, [17] has proposed a framework for Mobile Edge Computing (MEC) environments in which service migration is implemented. Their goal was to ensure service continuity for mobile users and reduce both service downtime and overall migration time. Although previous research efforts have tackled service migration in various edge computing environments, their efforts have either supported migration at the virtual machine level or ignored important factors such as user location and mobility patterns.

More recently, the work in [18] has proposed an autonomic container migration approach for Fog computing environments. Their approach seeks to ensure continuous service delivery by migrating application modules to Fog nodes that are deemed to respect service delivery deadlines. The efficacy of their approach has been quantified in terms of network usage, execution cost and service execution delays. However, they did not consider other important factors such as service downtime, migration time, service loss and the reliability of the migration process. On the other hand, the work in [19] is the most recent effort that models container migration in Fog/Cloudlet computing environments. It models container-based migration and supports realistic user mobility patterns based on the SUMO urban simulation tool [20, 21]. In addition, they have proposed several baseline container migration algorithms that are categorized as: lowest latency (LL) in which containers are migrated to the Cloudlet that is expected to deliver the lowest service latency, closest access point (CAP) where containers are migrated to the Cloudlet connected to the closest access point to current user location and closest server Cloudlet (CSC) in which containers are migrated to the closest server Cloudlet to current user location. However, none of these migration algorithms has considered migration reliability, service downtime and overall migration time when making the migration decision.

This work proposes a container migration algorithm for Cloudlet computing environments based on an integrated multicriteria decision making (MCDM) approach that integrates the Entropy [22] and Technique of Order Preference Similarity to the Ideal Solution (TOPSIS) [23] methods. The proposed approach seeks to migrate a container associated with a particular mobile IoT device to a nearby server Cloudlet considering user location, mobility pattern, migration time and migration reliability. While MCDM has been used in related disciplines such as cloud service selection [24, 25] and task scheduling in Cloud Computing [26, 27], no prior attempts have been made to apply MCDM for container migration in Cloudlet computing environments. The main contributions of this work can be summarized as follows:

- Proposing an MCDM-based algorithm for container migration in Cloudlet-enabled IoT systems.
- Implementing the proposed algorithm using a specialized simulation tool with realistic migration models and user mobility patterns.
- Performing a series of experiments to assess the performance of the proposed migration algorithm and compare it against that of existing algorithms.

The rest of this paper is organized as follows. Section II explains the research methodology. Section III presents and discusses the obtained results and Section IV summarizes and concludes this paper.

II. Research Methodology

This section explains the research methodology followed in order to implement the proposed migration algorithm including the system model, mathematical formulation of the MCDM techniques and the algorithmic design of the proposed algorithm.

A. System Model

In this paper, a hierarchical Cloudlet-enabled IoT system is assumed as shown in Fig. 1. Service requests generated from IoT devices are sent to the nearby Cloudlet which in turn performs the required task and communicate the results back to the IoT devices. If a task required by a particular IoT device can not be performed by the Cloudlet or requires further processing, the Cloudlet will forward the received request to the Cloud. Each Cloudlet supports a lightweight virtualization technique in which user's data and application modules are encapsulated in containers.

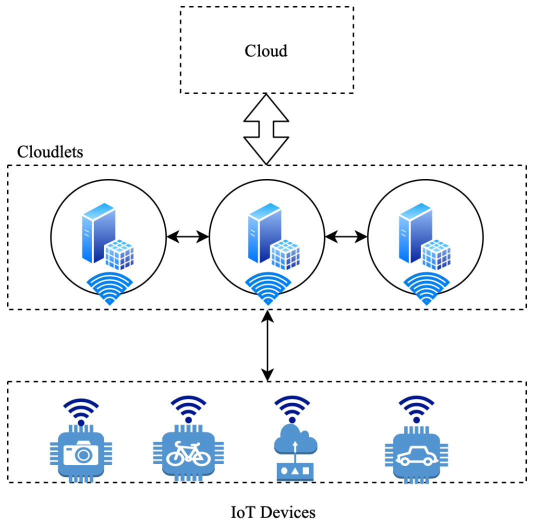

Fig. 1. IoT System Model.

On the other hand, IoT devices are assumed to be mobile i.e. change their locations over time. At any time instant, each IoT device will be connected to the Cloudlet hosting its associated container and can access that Cloudlet's services through a particular access point. In order to prevent service interruption and guarantee low-latency service delivery, container migration (from a source to a destination Cloudlet) should be performed in a timely manner. Determining the time instant at which the migration process should be initiated and choosing the most appropriate Cloudlet to host the migrating container are two major issues in container migration. In this work, each Cloudlet is considered as the decision maker responsible for addressing the two aforementioned issues. In other words, each Cloudlet is assumed to monitor the current location of its associated IoT devices and decide if a migration should be performed and what is the most suitable Cloudlet to host the migrating container considering both migration reliability and expected migration time.

The Cloudlets layer considered in this work consists of a set of Cloudlet nodes $C = \{SC_1, SC_2, .., SC_N\}$. The total number of cloudlets in the environment is denoted as N. In addition, each Cloudlet node is characterized by a triplet $\langle Cn_i(cpu)^{cap}, Cn_i(mem)^{cap}, Cn_i(bw)^{cap}\rangle$ where $Cn_i(cpu)^{cap}$ is the number of CPU cores available on Cloudlet i, $Cn_i(mem)^{cap}$ is the amount of memory available on node i and $Cn_i(bw)^{cap}$ is the available bandwidth. The mobile IoT devices receive services from their associated containers which are deployed on various Cloudlet nodes available in the IoT environment. Each container (con_j) is allocated to a suitable Cloudlet that satisfies its resource requirements. The resource requirements of a particulare container can be denoted as a triplet $\langle Con_j(cpu)^{req}, Con_j(mem)^{req}, Con_j(bw)^{req}\rangle$ where $Con_j(cpu)^{req}$ is the number of CPU cores required by container j, $Con_j(mem)^{req}$ is the required memory and $Con_j(bw)^{req}$ is the required bandwidth.

On the other hand, Fig. 2 shows the migration model assumed in this work. This migration model is based on [14, 19]. This model defines both the migration zone and the migration point on the map. Whereas the migration zone defines the area in which migration decisions are always computed and evaluated, the migration point represents the point on the map at which container migration should be initiated. The decision on whether to migrate a container associated with a particular user (or IoT device) should be made by the Cloudlet to which that IoT device is currently connected. For instance, Fig. 2 shows two users (user-1 and user-2) moving on the map with different speeds, directions and geographical positions. Their attributes should be continuously monitored by their current Cloudlet that is responsible for making the right decision for each user. As shown, user-1 is moving towards the access point (AP) that connects it to its current Cloudlet while user-2 is moving away from the AP. Hence, no migration should be performed for the container associated with user-1 while a migration for the container associated with user-2 should be initiated once it reaches the migration point. The migration point can be set either statically based on the coverage area of the AP or dynamically based on some dynamic attributes such as the user speed. For instance, Fig. 2 shows an example of a fixed migration point that is set at a distance that is 70% of the radius of the coverage area of the AP.

Once a Cloudlet decides to migrate the container associated with a particular IoT device, it must consult its migration algorithm to select the most suitable destination Cloudlet from a set of potential Cloudlets confined within the migration cone. The migration cone is defined in terms of the two adjacent directions to the current direction of the user and an angle that defines the relative region between the user and the AP. For example, the cone associated with user-2 is confined between the Northeast and the Southeast directions. The value of the cone's angle could be a fixed value or a dynamic value that is changed at run-time subject to some optimization constraints.

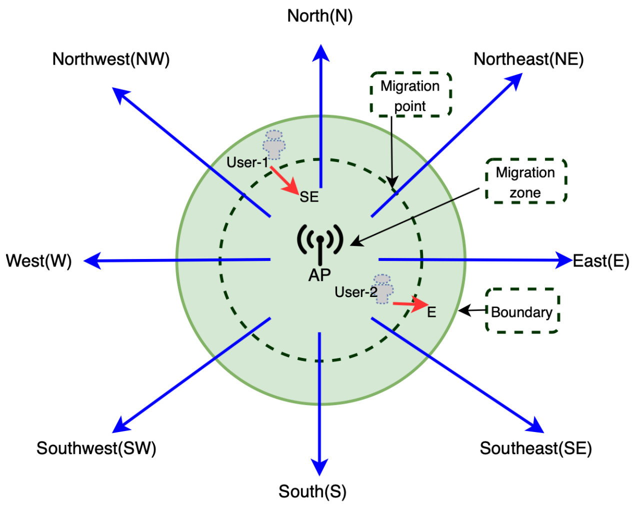

Fig. 2. Migration Model [19].

B. MCDM-Based Migration Algorithm

This section explains the proposed MCDM-based migration algorithm. Fig. 3 highlights the main steps followed by the proposed migration algorithm in order to select the most appropriate destination Cloudlet for a migrating container. As mentioned in Section II-A, each Cloudlet continuously monitors the movement of its associated IoT devices. Once a Cloudlet observes that an associated mobile IoT device is moving away from its associated AP, it must prepare and initiate the migration process by performing a number of steps based on an integrated Entropy-TOPSIS multicriteria decision making approach. First, the migration cone defined by the migration model must be constructed. Second, if the migration cone is not empty (i.e. it contains at least two possible destination Cloudlets that can satisfy all processing, memory and bandwidth requirements of the migrating container), the source Cloudlet will construct a decision matrix D_{ij} based on two criteria , namely, migration time (mt_{ij}) and migration reliability (mr_{ij}).

Apparently, migration time represents the expected time required to complete transferring the migrating container from source Cloudlet i to destination Cloudlet j. The total time required to migrate a container from Cloudlet i to Cloudlet j is computed as shown in equation 1.

$$mt_{ij} = \frac{M_{con_i}}{bw_{ij}} + T_{U_i} + T_{D_{ij}} \tag{1}$$

Where mt_{ij} is the total migration time, M_{con_i} is the memory size of the container migrating from Cloudlet i, bw_{ij} is the available network bandwidth between Cloudlets i and j, T_{U_i} is the uplink latency of Cloudlet i and $T_{D_{ij}}$ is the latency by distance between Cloudlets i and j.

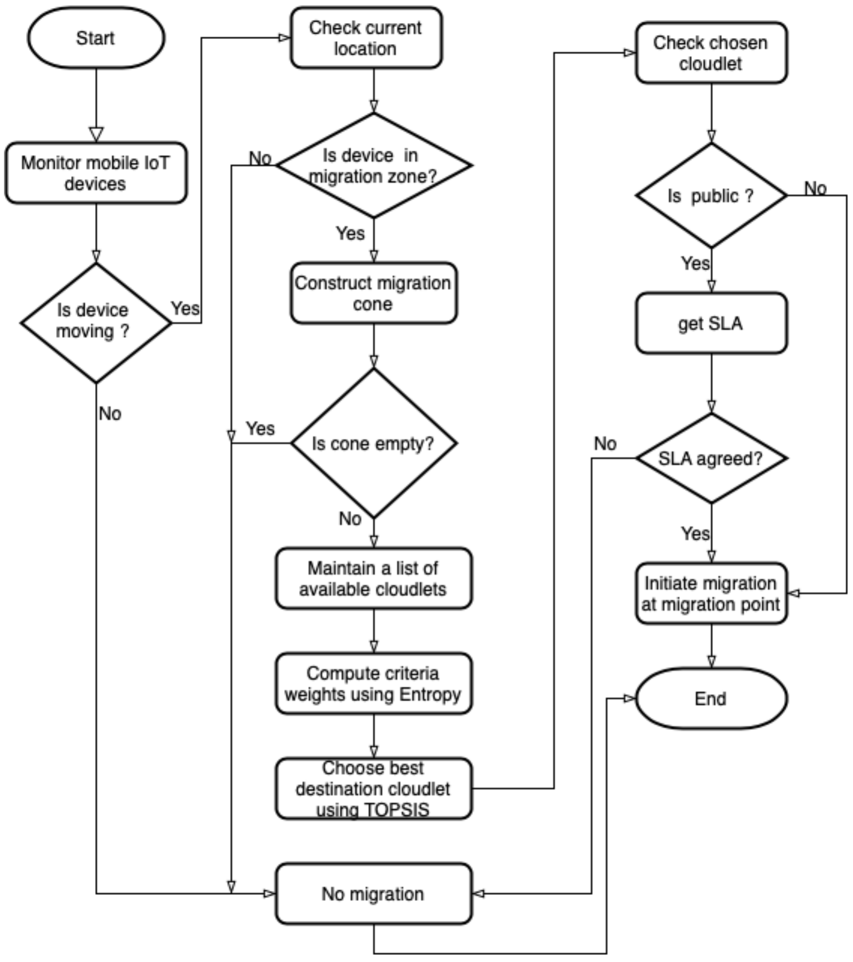
Start
Monitor mobile IoT devices
Is device moving ?
Yes
No
Check current location
Is device in migration zone?
No
Yes
Construct migration cone
Is cone empty?
Yes
No
Maintain a list of available cloudlets
Compute criteria weights using Entropy
Choose best destination cloudlet using TOPSIS
No migration
Check chosen cloudlet
Is public ?
No
Yes
get SLA
SLA agreed?
No
Yes
Initiate migration at migration point
End

Fig. 3. Migration Algorithm Flow-chart.

On the other hand, migration reliability (mr_{ij}) quantifies the probability that the migration process (from Cloudlet i to Cloudlet j) will be completed successfully during the anticipated migration time. In this work, migration reliability is calculated based on the reliability of the communication link between Cloudlets i and j and the reliability of the destination Cloudlet j. In general, the reliability of a particular subsystem s can be computed as shown in equation 2 [28].

$$R_s(t) = e^{-\int_0^t h(\tau)d\tau} \tag{2}$$

Where $h(t)$ is the hazard rate function associated with subsystem s. Assuming that both the communication link, between Cloudlets i and j, and destination Cloudlet j have constant hazard functions, their reliability values can be computed as as shown in equation 3.

$$\begin{aligned} R_{link(i,j)}(t) = e^{-\lambda_{ij} t} \\ R_{C_j}(t) = e^{-\lambda_{C_j} t} \end{aligned} \tag{3}$$

Where $R_{link(i,j)}$ is the reliability of the communication link between Cloudlets i and j, R_{C_j} is the reliability of Cloudlet j, λ_{ij} is the constant hazard rate of the communication link and λ_{C_j} is the constant hazard rate of Cloudlet j. Since the communication link and the destination Cloudlet constitute a serial reliability model, the reliability of the migration process (from Cloudlet i to Cloudlet j) can be computed by multiplying these individual reliability values [29], as shown in equation 4.

$$mr_{ij}(t) = R_{link(i,j)}(t) * R_{C_j}(t) = e^{-(\lambda_{ij}+\lambda_{C_j})t} \tag{4}$$

Where mr_{ij} is the reliability of migrating a container from Cloudlet i to Cloudlet j. Intuitively, by setting the value of t to mt_{ij}, the reliability of a migration process whose expected finish time is mt_{ij} can be computed. Having obtained the migration time and migration reliability of each possible destination Cloudlet j, the source Cloudlet will construct the decision matrix as shown in equation 5.

$$D_{ij} = \begin{bmatrix} mt_{i1} & mr_{i1} \\ mt_{i2} & mr_{i2} \\ mt_{i3} & mr_{i3} \\ . & . \\ . & . \\ mt_{in_c} & mr_{in_c} \end{bmatrix} \tag{5}$$

Where n_c is the number of Cloudlets available within the constructed migration cone. Third, once the decision matrix is obtained, the proposed migration algorithm will employ an Entropy-based method to assign a weight to each decision criterion as weight calculation is an essential step towards MCDM-based alternative (Cloudlet) selection approaches. The computed weight signifies the importance given to each criterion when making the migration decision. The computation of criteria weights based on the Entropy method can be summarized as follows:

Step 1: Computation of normalized feature weight (P_{ij}) for the i^{th} alternative and the j^{th} criterion.

$$P_{ij} = \frac{d_{ij}}{\sum_{i=1}^{n_c}(d_{ij})}, (1 \leq i \leq n_c, 1 \leq j \leq m) \tag{6}$$

Where d_{ij} is the value of j^{th} criterion under the i^{th} alternative, n_c is the number of possible alternatives (i.e. Cloudlets) and m is the number of criteria.

Step 2: Calculation of the output entropy (e_j) for each criterion j.

$$\begin{aligned} e_j &= -K * \sum_{i=1}^{n_c}(P_{ij} * ln(P_{ij})), (1 \leq j \leq m) \\ K &= \frac{1}{ln(n_c)} \end{aligned} \tag{7}$$

Step 3: Calculation of the degree of diversification (g_j) for each criterion j.

$$g_j = |1 - e_j|, 1 \leq j \leq m \tag{8}$$

Step 4: Computation of the weight (w_j) of each criterion.

$$w_j = \frac{g_j}{\sum_{j=1}^{m}(g_j)}, (1 \leq j \leq m) \tag{9}$$

Fourth, having obtained the criteria weights using the Entropy method, the migration algorithm proceeds to select the most suitable destination Cloudlet to receive the migrating container using the TOPSIS method. The rationale behind the TOPSIS method is to choose the alternative with the shortest euclidean distance from the ideal solution and the farthest distance from the negative-ideal solution. The TOPSIS method employed to select the ideal alternative (Cloudlet) proceeds as follows:

Step1: The decision matrix (D_{ij}) is normalized using vector normalization technique and then weighted using the criteria weights - computed using the Entropy method to obtain the weighted normalized performance matrix.

$$\begin{aligned} \bar{d_{ij}} &= \frac{d_{ij}}{\sqrt{\sum_{i=1}^{n_c}(d_{ij})^2}}, (1 \leq i \leq n_c, 1 \leq j \leq m) \\ \bar{d_{ij}} &= w_j * \bar{d_{ij}}, (1 \leq j \leq m) \end{aligned} \tag{10}$$

Step 2: Determine the best condition (A_j^+) and the worst condition (A_j^-) with respect to each criterion (j) and construct the best conditions vector (A_b) and the worst conditions vector (A_w).

$$A_j^+ = \begin{cases} \max_i(\bar{d_{ij}}) & j \in J^+ , (1 \leq i \leq n_c) \\ \min_i(\bar{d_{ij}}) & j \in J^- , (1 \leq i \leq n_c) \end{cases} \tag{11}$$

$$A_j^- = \begin{cases} \min_i(\bar{d_{ij}}) & j \in J^+ , (1 \leq i \leq n_c) \\ \max_i(\bar{d_{ij}}) & j \in J^- , (1 \leq i \leq n_c) \end{cases} \tag{12}$$

Where J^+ indicates beneficiary criteria i.e. criteria with positive impact (e.g. migration reliability) while J^- represents non-beneficiary criteria i.e. criteria with negative impact on the migration process (e.g. migration time). The best and worst conditions vectors are then constructed as:

$$\begin{aligned} A_b &= \{A_j^+ \mid (1 \leq \text{j} \leq \text{m})\} \\ A_w &= \{A_j^- \mid (1 \leq \text{j} \leq \text{m})\} \end{aligned} \tag{13}$$

Step 3 Calculate the Euclidean distance between each alternative i and each of the best condition vector (A_b) and the worst condition vector (A_w).

$$\begin{aligned} L_{ib} &= \sqrt{\sum_{j=1}^{m}(\bar{d_{ij}} - A_b[j])^2}, (1 \leq i \leq n_c) \\ L_{iw} &= \sqrt{\sum_{j=1}^{m}(\bar{d_{ij}} - A_w[j])^2}, (1 \leq i \leq n_c) \end{aligned} \tag{14}$$

Where L_{ib} and L_{iw} are the distances from the target alternative (Cloudlet) i to the best and worst condition vectors, respectively.

Step 4 Calculate the performance score (P_i) assigned to each alternative.

$$P_i = \frac{L_{iw}}{L_{iw} + L_{ib}}, (1 \leq i \leq n_c) \tag{15}$$

Step 5 Rank alternatives (Cloudlets) according to their performance scores (P_i).

After performing the steps dictated by the TOPSIS method, the Cloudlet with the highest performance score is chosen as the most suitable destination Cloudlet to receive the migrating container and the migration process will be initiated upon device's arrival at the migration point. If the chosen Cloudlet is a public Cloudlet, its service level agreement (SLA) parameters will be compared against the requirements of the IoT device (user) whose container is involved in the migration process. If the chosen Cloudlet satisfies the requirements of the associated IoT device, the migration process will be initiated once that device reaches the predefined migration point. On the other hand, if the chosen Cloudlet is not guaranteed to fulfil the requirements of the associated IoT device, the migration process will be aborted and the associated container will remain on its original Cloudlet.

III. Results and Discussion

In order to evaluate the performance of the proposed migration algorithm, it has been implemented using the MobFogSim simulation tool [19]. MobFogSim is a recently developed tool that allows development and evaluation of container migration in Fog and Cloudlet computing environments with concrete support for realistic user mobility patterns. The proposed algorithm has been validated and compared against other migration algorithms that are originally implemented within the MobFogSim tool. Table I summarizes the most important input simulation parameters. Whereas some simulation parameters were given fixed values, other parameters such as network bandwidth and the constant hazard rates were randomly picked from the designated ranges. Hence, the results presented in this section represent the arithmetic mean of the results of 20 different simulation runs.

TABLE I. Simulation Settings.

Parameter	Value
Size of container's execution state	12.8 MB
Number of Cloudlets	144
Number of Cloudlets per access point	1
Radius of access point coverage(r)	500 m
Migration point policy	fixed (r-40 m)
IoT device speed	20 Kmph
Network bandwidth (between Cloudlets)	[2,8] MB/s
Hazard rate	$[1 * 10^{-7}, 15 * 10^{-7}]$

Fig. 4 depicts the total migration time achieved under the proposed policy along with the migration time achieved under existing algorithms, namely, lowest latency (LL), closest access point (CAP) and closest server Cloudlet (CSC) [19]. The results are annotated with their 95% confidence interval. In order to reasonably assess the performance of different migration algorithms, they were simulated under different migration techniques: clod migration (Cold) and live migration (Live). In cold migration: first, the container to be migrated is frozen/stopped to make sure that it no longer modifies its state. Second, the whole execution state of that container is checkpointed and then transferred while being stopped. Third, the container is resumed at the destination once all its execution state is available there. On the other hand, live migration first halts container on the source Cloudlet and copies its minimal and kernel execution state to the destination Cloudlet so that the container can resume its execution there. Only after that and while the container is running, it transfers all the remaining state including the memory pages, which represent the major portion of the whole state. As shown in Fig. 4, the proposed algorithm has achieved the lowest migration time as compared to other migration algorithms under both the Cold and Live migration techniques; the employed Entropy-TOPSIS decison making approach has allowed the proposed algorithm to observe the degree of diversification available among the considered potential destination Cloudlets (i.e. those available within the migration cone) and choose the Cloudlet that would yield the lowest migration time as compared to other algorithms. In addition, the Cold migration technique has lower migration time when compared to the Live migration technique under all possible migration algorithms. On average, the proposed algorithm has achieved 48%, 42% and 43% improvement in the migration time as compared to the LL, CAP and CSC algorithms, respectively.

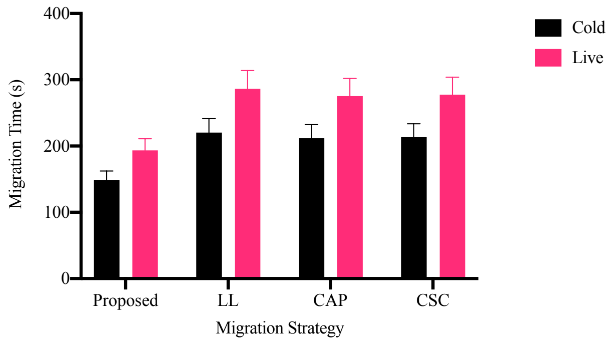

Fig. 4. Migration Time.

On the other hand, Fig. 5 illustrates the total downtime observed under the proposed and existing migration algorithms. The downtime is the time interval during which the service required by the mobile IoT devices will not be available due to the migration process. As shown, the proposed algorithm has resulted in lower downtime as compared to other algorithms under both migration techniques. It can be observed that the downtime is equal to the migration time when the Cold migration technique is employed. On the other hand, the downtime is much lower than the migration time when the Live migration technique is used; under the Live migration technique, the migrating container keeps on running while the majority of its state is being moved to the chosen destination Cloudlet. The container is stopped only for the transfer of a minimal amount of its overall state, after which the container runs at the destination Cloudlet. Nevertheless, the proposed algorithm can still achieve lower downtime as compared to other migration algorithms due to its ability to select the most appropriate destination Cloudlet with sufficient network bandwidth to handle the migration process. The amount of improvement the proposed algorithm has achieved in downtime - as compared to other algorithms is equivalent to that observed in migration time.

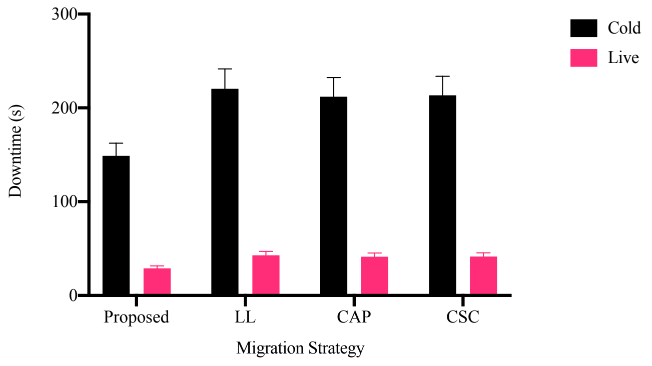

Fig. 5. Service Downtime.

Fig. 6 shows migration reliability obtained by the proposed and the existing migration algorithms under Cold and Live migration techniques. As shown, the proposed algorithm was able to surpass other migration algorithms in terms of migration reliability. Apparently, migration reliability decreases when moving from Cold to Live migration technique due to the increase in migration time - under the same hazard rate. This can be explained by the exponential relationship shown in equation 4. However, the proposed algorithm can still outperform other algorithms and pick the most suitable destination Cloudlet considering the reliability factor. Overall, the proposed algorithm has achieved 17%, 20% and 20% average improvement in migration reliability as compared to LL, CAP and CSC algorithms, respectively. On the other hand, Fig. 7 portrays the service loss rate under the considered migration algorithms. Service loss rate is the percent of service requests that could not be handled due to the migration process. This rate is clearly dependent on the downtime. As shown, the proposed algorithm was able to yield the lowest service loss rate as compared to other algorithms under both Cold and Live migration techniques. The proposed algorithm has achieved 36%, 27% and 28% reduction in service loss rate when compared to LL, CAP and CSC, respectively.

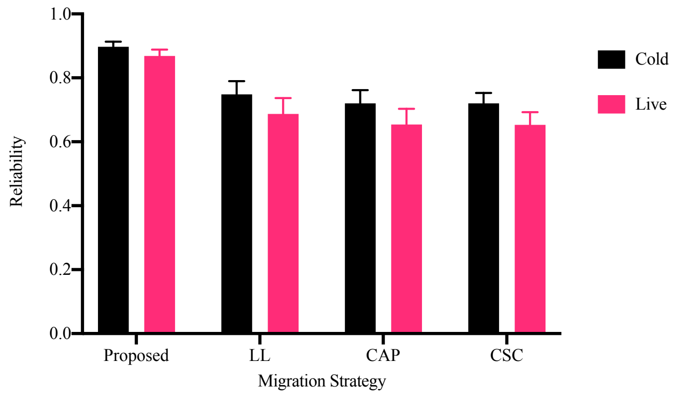

Fig. 6. Migration Reliability.

The presented results prove the ability of the proposed migration algorithm to perceive the run-time dynamics of the Cloudlet-enabled IoT environment and choose the most appropriate destination Cloudlet that would optimize the considered performance metrics. This is due to the fact that the proposed MCDM-based migration algorithm is able to observe the degree of variation - with respect to each migration criterion within the considered destination Cloudlets, reasonably assign criteria weights (using the Entropy method) and appropriately assign performance scores to the considered Cloudlets (using TOPSIS) when making a migration decision. This fact is confirmed by the results shown in Tables II and III which represent two sample migration scenarios. As shown, migration time has been assigned higher weight than migration reliability in the first scenario (Table II) while the opposite is observed in the second scenario (Table III).

On the other hand, Fig. 8 illustrates how criteria weights were assigned during the simulation process for 12 consecutive migration events. These results prove the ability of the proposed migration algorithm to sense the dynamic changes in the decision variables (criteria) which, in turn, steers the decisions making process i.e. destination Cloudlet selection towards the most appropriate alternative taking into account the current status of the IoT environment.

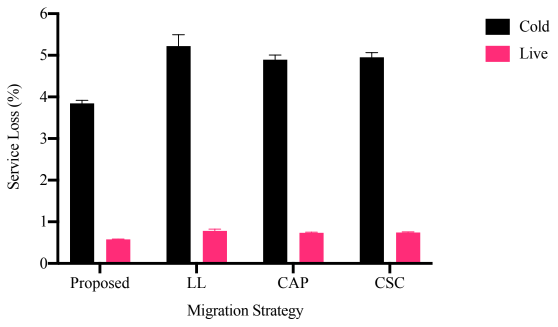

Fig. 7. Service Loss Rate.

TABLE II. Migration Scenario 1.

Alternative ID	Migration Time (s)	Migration Reliability	Performance Score
A	128.6	0.85	0.97
B	128.7	0.82	0.95
C	128.7	0.73	0.91
D	144.4	0.80	0.82
E	163.6	0.91	0.62
F	167.7	0.72	0.56
G	219.5	0.89	0.13
H	220.8	0.85	0.11
I	219.2	0.62	0.02
Criteria Weight	0.81	0.19	

TABLE III. Migration Scenario 2.

Alternative ID	Migration Time (s)	Migration Reliability	Performance Score
A	177.1	0.86	0.99
B	177.2	0.79	0.76
C	186.9	0.80	0.73
D	177.1	0.78	0.71
E	177.1	0.76	0.63
F	177.1	0.75	0.61
G	177.0	0.69	0.45
H	177.2	0.63	0.35
I	223.9	0.64	0.02
Criteria Weight	0.39	0.61	

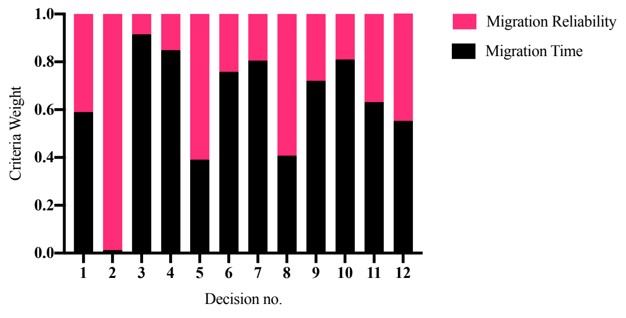

Fig. 8. Criteria Weights for Different Migration Decisions.

IV. Conclusion

In this paper, a mobility-aware container migration algorithm for Cloudlet-enabled IoT systems has been presented and evaluated. The proposed algorithm has employed an Entropy-TOPSIS integrated multicriteria decision making approach to select the most appropriate destination Cloudlet for a migrating container. It has been implemented using a specialized simulation tool and compared against exiting migration algorithms. Simulation results have proved the ability of the proposed algorithm to outperform other algorithms in terms of migration time , service downtime, migration reliability and service loss rate. They have also confirmed the ability of the proposed algorithm to perceive the run-time dynamics of the IoT environment and accurately steer the destination Cloudlet selection process.

References

[1] M. De Donno, K. Tange, and N. Dragoni, "Foundations and evolution of modern computing paradigms: Cloud, iot, edge, and fog," *IEEE Access*, vol. 7, pp. 150 936–150 948, 2019.

[2] S. Parikh, D. Dave, R. Patel, and N. Doshi, "Security and privacy issues in cloud, fog and edge computing," *Procedia Computer Science*, vol. 160, pp. 734 – 739, 2019.

[3] Y. Ai, M. Peng, and K. Zhang, "Edge computing technologies for internet of things: a primer," *Digital Communications and Networks*, vol. 4, no. 2, pp. 77 – 86, 2018.

[4] S. Shahzadi, M. Iqbal, T. Dagiuklas, and Z. Qayyum, "Multi-access edge computing: open issues, challenges and future perspectives," *Journal of Cloud Computing*, vol. 6, 12 2017.

[5] S. P. Singh, A. Nayyar, R. Kumar, and A. Sharma, "Fog computing: from architecture to edge computing and big data processing," *The Journal of Supercomputing*, vol. 75, no. 4, pp. 2070–2105, 2019.

[6] H. Yao, C. Bai, M. Xiong, D. Zeng, and Z. Fu, "Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 16, p. e3975, 2017, e3975 cpe.3975.

[7] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *Journal of Systems Architecture*, vol. 98, pp. 289 – 330, 2019.

[8] J. P. Martin, A. Kandasamy, and K. Chandrasekaran, "Exploring the support for high performance applications in the container runtime environment," *Human-centric Computing and Information Sciences*, vol. 8, no. 1, p. 1, 2018.

[9] A. Celesti, D. Mulfari, A. Galletta, M. Fazio, L. Carnevale, and M. Villari, "A study on container virtualization for guarantee quality of service in cloud-of-things," *Future Generation Computer Systems*, vol. 99, pp. 356 – 364, 2019.

[10] 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.

[11] C. Puliafito, C. Vallati, E. Mingozzi, G. Merlino, F. Longo, and A. Puliafito, "Container migration in the fog: A performance evaluation," *mdpi sensors*, vol. 19, 2019.

[12] L. F. Bittencourt, M. M. Lopes, I. Petri, and O. F. Rana, "Towards virtual machine migration in fog computing," in *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, 2015, pp. 1–8.

[13] M. Islam, A. Razzaque, and J. Islam, "A genetic algorithm for virtual machine migration in heterogeneous mobile cloud computing," in *2016 International Conference on Networking Systems and Security (NSysS)*, 2016, pp. 1–6.

[14] M. M. Lopes, W. A. Higashino, M. A. Capretz, and L. F. Bittencourt, "Myifogsim: A simulator for virtual machine migration in fog computing," in *Proceedings of The 10th International Conference on Utility and Cloud Computing*, New York, NY, USA, 2017, p. 47–52.

[15] 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.

[16] Y. Bi, G. Han, C. Lin, Q. Deng, L. Guo, and F. Li, "Mobility support for fog computing: An sdn approach," *IEEE Communications Magazine*, vol. 56, pp. 53–59, 05 2018.

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

[18] J. Martin and A. Kandasamy, "Mobility aware autonomic approach for the migration of application modules in fog computing environment," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–20, 2020.

[19] C. Puliafito, D. M. Gonçalves, M. M. Lopes, L. L. Martins, E. Madeira, E. Mingozzi, O. Rana, and L. F. Bittencourt, "Mobfogsim: Simulation of mobility and migration for fog computing," *Simulation Modelling Practice and Theory*, vol. 101, p. 102062, 2020.

[20] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo - simulation of urban mobility: An overview," in *SIMUL 2011, The Third International Conference on Advances in System Simulation*, 2011, pp. 63–68.

[21] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wiessner, "Microscopic traffic simulation using sumo," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 2575–2582.

[22] S. R. Komaragiri and D. N. Kumar, *Multicriterion Analysis in Engineering and Management*. PHI Learning Pvt. Ltd., New Delhi, India, 2010.

[23] J. Papathanasiou and N. Ploskas, *Multiple Criteria Decision Aid : Methods, Examples and Python Implementations*. Cham: Springer International Publishing, 2018.

[24] J. Araujo, P. Maciel, E. Andrade, G. Callou, V. Alves, and P. Cunha, "Decision making in cloud environments: an approach based on multiple-criteria decision analysis and stochastic models," *Journal of Cloud Computing*, vol. 7, no. 1, p. 7, 2018.

[25] C. Jatoth, G. R. Gangadharan, U. Fiore, and R. Buyya, "Selcloud: a hybrid multi-criteria decision-making model for selection of cloud services," *Soft Computing*, vol. 23, no. 13, pp. 4701–4715, 2019.

[26] R. Khorsand and M. Ramezanpour, "An energy-efficient task-scheduling algorithm based on a multi-criteria decision-making method in cloud computing," *International Journal of Communication Systems*, vol. 33, no. 9, p. e4379, 2020.

[27] S. C. Nayak, S. Parida, C. Tripathy, B. Pati, and C. R. Panigrahi, "Multi-criteria decision-making techniques for avoiding similar task scheduling conflict in cloud computing," *International Journal of Communication Systems*, vol. 33, no. 13, p. e4126, 2020.

[28] E. A. Elsayed, "Overview of reliability testing," *IEEE Transactions on Reliability*, vol. 61, no. 2, pp. 282–291, 2012.

[29] C. E. Ebeling, *An Introduction to Reliability and Maintainability Engineering*. Waveland Press, 2019.