Dynamic Edge User Allocation with User Specified QoS Preferences

Subrat Prasad Panda, Kaustabha Ray, and Ansuman Banerjee(✉)

Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata, India
subratprasad.mail@gmail.com, {kaustabha_r,ansuman}@isical.ac.in

Abstract. Mobile Edge Computing (MEC) policies that bind user service requests to edge servers, seldom take into account user preferences of Quality-of-Service (QoS) and the resulting Quality-of-Experience (QoE). In this paper, we design a novel user-centric optimal allocation policy considering the QoS preferences of users, with an attempt to maximize the overall QoE. Additionally, we propose a real-time mobility aware user-centric heuristic algorithm to solve the allocation problem by accommodating the time varying QoS demands of users. Experimental results on real data sets demonstrate the efficiency of our allocation scheme and a comparison with state-of-art approaches in MEC literature.

Keywords: Edge computing · Server allocation · User migration

1 Introduction

In recent times, Mobile Edge Computing (MEC) [1] has emerged as a new paradigm that allows service providers to deploy services on MEC servers located near base stations. As users move around, their application service invocations are routed to proximate MEC servers to curtail the high latencies of cloud communication networks. A service allocation policy is designed to determine the user-service-server binding, i.e. which service requests from which users are provisioned by which MEC servers in their vicinity, as they move around. In recent years, several allocation policies, static and dynamic, considering different optimization metrics have been proposed in literature [3,4,6–8].

The general philosophy of service allocation policies is to design and optimize a user-mobility aware service-server-user binding that optimizes some quantitative metric (e.g.. latency, energy, throughput) to cater to user application service needs and ensure seamless usage experience. A recent work [6] has proposed a novel view of considering qualitative QoS level offerings by service providers in designing the service bindings. Additionally, the authors have quantitatively correlated QoS values with overall Quality-of-Experience (QoE) of users to demonstrate the existence of thresholds, beyond which, enhancing QoS values no longer enhances a user QoE. This work, however, does not consider a user's QoS preferences when deciding these bindings. Moreover, the binding is static, in other

E. Kafeza et al. (Eds.): ICSOC 2020, LNCS 12571, pp. 187–197, 2020.
https://doi.org/10.1007/978-3-030-65310-1_15

words, once an allocation is decided for a user service invocation to a specific QoS level at an edge server, he is continued to be served at the same level throughout, oblivious to the fact that the user may not be in a position to enjoy services at a higher QoS level always due to battery or other constraints. Also, the policy is not adaptive, in the sense that user movements, joining or leaving of users, and user QoS preferences and preference changes in terms of the required QoS levels, are not accounted for. This motivated us to design a dynamic self-adaptive allocation policy that can address these variations.

Designing an allocation that considers user preferences of QoS levels is challenging due to the dynamics of MEC systems, the stochastic nature of service invocation patterns and the large space of user-service-server binding configurations. In our view, allocation policies in literature are more catered towards the perspective of service providers [5,6], aiming to optimize quantitative metrics, often ignoring users' qualitative preferences of QoS levels when making allocation decisions. QoS levels typically translate to a monotonically increasing footprint on the resource consumption for both the user and the provider, at the server end where the service is provisioned, and at the user end where a communication latency depending on the size of transferred data is incurred. Policies like [6], being user agnostic, may allocate QoS levels to users leading to an added aggravation. In such scenarios, a service provider may also suffer a degradation in throughput since the high QoS levels translate to more resources allocated at the server end which could have been otherwise allocated to other users. In the worst case, an overtly aggressive user-agnostic QoS allocation can lead to new service requests being needlessly denied service.

Our proposal in this paper is a service allocation policy that caters to both user and provider views considering individual QoS preference levels to enhance overall QoE of users in a mobility-aware scenario. The QoS preferences of users can vary over time, for example, a user initially having high battery levels, and preferring to stream services at high QoS levels, may sometime later choose to downgrade his preference depending on the changing battery conditions to alleviate energy utilization spent in data communication. We take into account such user specified adjustments in an attempt to maximize the overall user experience. Additionally, we cater to mobility of users and changing conditions as well. We first formulate the problem of dynamic QoS preference aware edge user allocation and propose an Integer Linear Programming (ILP) formulation for the optimal solution, and a heuristic which produces near optimal QoE allocations. We use the EUA dataset [4–7], a real-world dataset as edge server locations, and the PlanetLab and Seattle Latency dataset [10] to generate latencies representative of MEC environments to validate our approach. Experimental results demonstrate the efficiency of our heuristic which produces near optimal allocations. We compare our results with two state-of-the-art approaches and show that our proposal outperforms both with respect to QoE.

2 A Motivating Example

In this section, we present a motivating example to explain the problem context. Consider the scenario demonstrated in Fig. 1. There are two edge servers E_1 and E_2 and six users u_1, u_2, u_3, u_4, u_5 and u_6. The coverage area of a particular server is marked by a circle, hence any user within the coverage area of a server can use the services hosted at the particular server. For example, u_1 can only access the services from E_1, whereas, u_4 can access the services hosted at both E_1 and E_2. The resource capacity of each server is represented as a resource vector $\langle vCPUs, RAM, storage, bandwidth \rangle$ [6], where $vCPU$ denotes the number of virtual CPUs. For the example scenario, assume the resource capacities of server are denoted by vectors $s_1 = \langle 16, 32, 750, 8 \rangle$ and $s_2 = \langle 16, 16, 500, 4 \rangle$. Edge servers host services at different QoS levels. Provisioning a service at a QoS level consumes a certain amount of server resources. We assume both E_1 and E_2 host a service $\mathcal{P}$ with 3 QoS levels W_1, W_2 and W_3 as in Table 1. Each QoS level has a resource requirement represented by a 4-element resource vector $W = \langle vCPUs, RAM, storage, bandwidth \rangle$ and an associated QoE value. W_3 is the highest QoS level. Each user when invoking $\mathcal{P}$ specifies a desired QoS level, W_1, W_2 or W_3, at which he wishes to be served, and additionally, a lower tolerance threshold QoS level, below which the services are rendered unacceptable to him. The initial QoS preferences of the users are in Table 2. In the scenario demonstrated in Fig. 1, u_3 follows the trajectory as depicted by the curved line while all other users remain stationary. While in its trajectory, at time $t = 0$, demarcated by a black rectangle, u_3 invokes $\mathcal{P}$ with QoS preference as W_3. Simultaneously, u_1, u_2, u_4, and u_5 also invoke $\mathcal{P}$ at $t = 0$, while u_6 does the same at $t = 5\,\text{s}$. During the course of its trajectory, at $t = 5\,\text{s}$, u_3 downgrades its QoS preference from W_3 to W_2, at the point indicated by the blue diamond.

Table 1. Available QoS levels

QoS level	Resource requirement	QoE
W_1	$\langle 2, 2, 10, 1 \rangle$	1.5
W_2	$\langle 4, 4, 15, 1.5 \rangle$	4
W_3	$\langle 8, 4, 20, 2 \rangle$	5

Table 2. User QoS details

User	QoS level	QoS Min	Allocation t = 0 s [6]	Allocation t = 0 s Our	Allocation t = 5 s [6]	Allocation t = 5 s Our
u_1	W_1	Any	E_1, W_2	E_1, W_1	E_1, W_3	E_1, W_1
u_2	Any	Any	E_1, W_2	E_1, W_2	E_1, W_2	E_1, W_3
u_3	W_3	W_2	E_1, W_3	E_1, W_3	E_2, W_3	E_2, W_2
u_4	W_2	Any	E_2, W_3	E_2, W_2	E_1, W_2	E_1, W_2
u_5	W_3	W_2	E_2, W_3	E_2, W_3	E_2, W_3	E_2, W_2
u_6	W_1	Any	Idle	Idle	NA	E_2, W_1

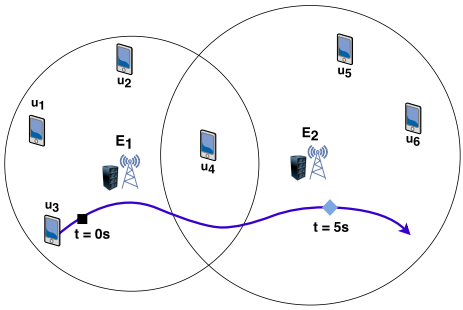


Fig. 1. Representative MEC scenario (Color figure online)

User QoS Preference Agnostic Allocation: A user preference agnostic policy such as [6] does not even take into the account the initial QoS preferences. The allocation is shown in Column 4 of Table 2 as E_k, W_p pairs indicating the edge server

E_k and the QoS level W_p to which the user u_i is bound. Moreover, at $t = 5$ s, this policy continues to provision u_3 at W_3 as shown in Column 6, agnostic of the fact that u_3 had requested for a downgrade to W_2. The QoE value experienced by u_3 is 5. In such a scenario, since the bandwidth requirement of W_3 is 2 Mbps, u_3 incurs an additional latency overhead due to increased data transfer. Also, at $t = 5$ s, when u_6 invokes the service, E_2 no longer has the needed resources to serve him, considering its serving capacity and the resources already consumed. Given the coverage constraint and the locations shown, u_6 cannot be served by E_1. However, had u_3's QoS level been reduced to W_2 when u_3 changed its preference level, u_6 could be onboarded at E_2.

Our Method at Work: Our user preference aware policy considers the initial preferences, and allocates levels as depicted in Table 2 to the users. Further, at time $t = 5$ s, when u_3 indicates its change of preference level, we reduce the QoS level allocated from W_3 to W_2. In such a scenario, for QoS level W_2, the bandwidth requirement is 1.5 Mbps, hence, the additional latency incurred by u_3 earlier is no longer applicable. When we assign W_2 to u_3, the QoE index of u_3 is 4, lower than W_3. Since u_3 requested for a lower QoS level, we consider the corresponding QoE value is good enough. Additionally, since a lower QoS level corresponds to lower resource consumption at the server, we can re-distribute the resources to better serve other users. u_6 can now be onboarded at $t = 5$ s.

The example shows the trade-off between resource consumption, latency and QoE in user QoS agnostic versus user QoS preference aware provisioning. The latter is challenging to design considering time-varying user QoS requirements while catering to user mobility. To the best of our knowledge, this is the first work towards mobility-aware dynamic user allocation with user QoS preferences.

3 System Model and ILP Formulation

In this section, we first formalize the system model. We consider a discrete time-slotted model [7]. We denote by $U^t = \{u_1, u_2 \ \dots \ u_n\}$ the set of active users and by $S^t = \{s_1, s_2 \ \dots \ s_m\}$ the set of active edge-servers at time t. Each server s_j has a radius R_j and a capacity vector C_j^t $\langle CPU, RAM, storage, bandwidth \rangle$ at t, denoted as $C_j^t = \langle \left(c_j^1\right)^t, \left(c_j^2\right)^t, \left(c_j^3\right)^t, \left(c_j^4\right)^t \rangle$ in that order. We denote by W_l the demand vector $\langle CPU, RAM, storage, bandwidth \rangle$ of QoS level l, denoted as $\langle w_l^1, w_l^2, w_l^3, w_l^4 \rangle$ in that order. A server can only cater to service requests from users within the service radius. For user u_i, the preferred QoS level is denoted as H_i^t, and the threshold L_i^t for the lowest QoS level tolerable. A service allocation policy can choose to serve him at *any* QoS level between the threshold and the preferred level (both inclusive), with an attempt to serve maximum number of users at their preferred levels, thereby, maximizing the overall QoE of all stakeholders, while keeping in view the capacity of each edge server, and the coverage constraint induced by the relative separating distance between the user and the servers. If a user cannot be allocated to any edge-server a suitable QoS level inside the preference range, he has to wait till the required resources are

available. We assume a set of q QoS levels. Let E_{il}^t denote the QoE value for u_i at QoS level l, q_i^t the QoS level assigned to u_i at time t, d_{ij}^t the distance between u_i and server s_j, Δ_{ij}^t the latency experienced by u_i allocated to s_j at t. We compute latency Δ_{ij}^t as a function of q_i^t and d_{ij}^t. The latency experienced in any user-server allocation has to honor a maximum limit denoted by δ. We formulate an Integer Linear Program (ILP) for the problem below.

Objective:

$$Maximize : \sum_{t \in T} \sum_{i=1}^{|U^t|} \sum_{j=1}^{|S^t|} \sum_{l=L_i^t}^{H_i^t} x_{ijl}^t \times E_{il}^t \tag{1}$$

where,

$$x_{ijl}^t = \begin{cases} 1, & \text{If user } u_i \text{ is allocated to server } s_j \text{ at QoS level } l \text{ at time } t \\ 0, & \text{Otherwise} \end{cases}$$

Subject to:

1. Coverage Constraint:

$$d_{ij}^t \leq R_j^t \tag{2}$$

2. Capacity Constraint:

$$\sum\nolimits_{i=1}^{|U^t|} \sum\nolimits_{l=L_i^t}^{H_i^t} w_l^k \times x_{ijl}^t \leq \left(c_j^k\right)^t : \forall t \in T, \forall j \in \left\{1, \ldots |S^t|\right\}, \forall k \in \{1, \ldots 4\} \tag{3}$$

3. Latency Constraint:

$$\sum\nolimits_{j=1}^{|S^t|} \sum\nolimits_{l=L_i^t}^{H_i^t} \Delta_{ij}^t \times x_{ijl}^t \leq \delta : \quad \forall t \in T, \ \forall i \in \left\{1, \ldots |U^t|\right\} \tag{4}$$

4. User-Server Mapping:

$$\sum\nolimits_{j=1}^{|S^t|} \sum\nolimits_{l=L_i^t}^{H_i^t} x_{ijl}^t \leq 1 : \quad \forall t \in T, \ \forall i \in \left\{1, \ldots |U^t|\right\} \tag{5}$$

5. Integer Constraint:

$$x_{ijl}^t \in \{0, 1\} : \forall t \in T, \forall i \in \left\{1, ..|U^t|\right\}, \forall j \in \left\{1, ..|S^t|\right\}, \forall l \in \left\{L_i^t..H_i^t\right\} \tag{6}$$

The objective function aims at maximization of the overall QoE of users over the set of time slots t over a period T. The indicator variable x_{ijl}^t at any time instant t, encodes all possible server-user-qos preferences. The objective function implicitly encodes all individual preferences and the threshold in the summation, hence no additional constraints are needed to specify the minimum threshold QoS level as required. At any time instant t, a user u_i can be allocated to s_j if the user is within radius R_j, as expressed by the constraint in Eq. 2. To

allocate u_i to s_j at a QoS level l, the resource requirement at s_j is denoted by W_l. The total resources allocated must honor the capacity constraint of each server. Equation 3 ensures that the combined requirements of users allocated to a server remains within the server's total capacity for each dimension CPU, RAM, storage and bandwidth of the resource vector. Equation 4 ensures that users are allocated to servers such that the latency bound is honoured. Equation 5 is used to express that a single service can only be allocated to a single server at a QoS level at any t. Equation 6 specifies that x^t_{ijl} variables are Boolean indicator variables denoting service requests from users, the respective server to which the requests are allocated and required QoS values. As observed in [6], QoS is non-linearly correlated with the QoE for any service, and we represent the QoS-QoE correlation using the logistic function (Eq. (7)) as in [6] with an additional scaling according to the QoS level preference and threshold specified by a user. The QoE E^t_{il} experienced by u_i at time t for level l is expressed as:

$$E^i_l = \frac{E_{max}}{1 + \exp\{-\alpha\,(\gamma^t_{il} - \beta^t_i)\}} \tag{7}$$

The scaling assists to assign lowest QoE value to lowest QoS level and highest QoE value to highest QoS level. E^t_{il} depends on the QoS level W^t_l, his QoS preference H^t_i and the threshold level L^t_i at time t. Here, $\gamma^t_{il} = \frac{\sum_{k=1}^{4} w^k_l}{4}$ is the mean computational demand of QoS level W_l of user u_i at time t; $\beta^t_i = \frac{\gamma^t_{iH^t_i} - \gamma^t_{iL^t_i}}{2}$ is the mid-point of QoE value of user u_i at t. The value E_{max} is the maximum value of QoE and α is the growth factor of the logistics function.

A solution to the ILP gives us for each time slot t, an optimal allocation of user service requests to QoS levels at edge servers, honoring QoS preferences, the latency upper bound and radius constraints. If the ILP solver returns unsatisfiable, we conclude that the user set cannot be allocated to their proximate edge servers, given the constraints. To cater to dynamic mobility and preference changes, we re-evaluate the ILP when any of the following scenarios occur: (a) any user changes the QoS specification; b) users or edge-servers become inactive; c) users move in and out of the service zone of servers; and d) new service requests are placed. However, given the associated computational needs, re-evaluating the ILP frequently turns out to be a non-scalable strategy, as demonstrated in our experimental results presented in Sect. 5. To address this, we design a scalable heuristic to cater to real-world dynamic scenarios, as described in the following.

4 Heuristic Solution

In this section, we present the design of an efficient polynomial time heuristic which generates near-optimal solutions. We use a Red-Black Tree [2] as an indexing data-structure. The algorithm maintains a Red-Black Tree for each edge server and uses a metric defined as i-factor for each user in its service zone as index. This heuristic is used in place of the ILP, and executes whenever any

of the events mentioned earlier occur, necessitating a reevaluation of the allocation. However, this being a polynomial time algorithm, is lightweight and can be executed more efficiently than the ILP. Our heuristic has the following steps.

- We first divide the new users into two classes, single-server class (S-class) and multi-server class (M-class). The users within the range of only one edge-server are clustered into S-class and the users withing the range of more than one edge-server are put into the M-class. For example, in Fig. 1, the users u_1, u_2, u_3, u_5 and u_6 are within the range of only one server i.e. E_1 and are hence clustered into S-class. However, u_4 can access both E_1 and E_2, hence is put into the M-class. This categorization is done once for all users at the start, and adjusted at every time slot only if there is a change in user locations, new users join in, or existing users leave.
- The users in both S-class and M-class are allocated an initial QoS level at their minimum threshold specified. Referring to the scenario in Sect. 2, u_1, u_2, u_3 u_4, and u_5 are initially assigned at QoS level W_1, W_1, W_2, W_1 and W_2 respectively. The increment factor (i-factor), discussed later in this section, is computed for all the users in both the S-class and M-class. The i-factor is determined by user's QoS preference and presently assigned QoS level (*plevel*). For determining the allocation, S-class is considered before the M-class since S-class users are bound to a single edge server. Each user is assigned to the edge server according to his i-factor. Users with low i-factor get higher preference to an edge server during the assignment. For M-class users, the allocation policy tries to assign an user to the nearest server with required remaining computation resource, with a motivation to serve him with better latency experience. We examine the users according to their i-factor, compute an initial assignment and update the Red-Black Tree with i-factor as key for each server.
- Our heuristic then attempts to enhance the QoS level of each user (upper bounded by their respective preference levels) and re-evaluates the i-factor after incrementing the QoS level. This process of incrementing continues till all users receive their QoS preference levels or the server exhausts its available resources and we move on to examine the next server in the vicinity of the user from where he can be served.
- For servers which have exhausted their resources, users from M-class may be migrated to the other nearby servers having free resources. Once users have been migrated across nearby servers, the QoS levels have to be re-evaluated. QoS upgrade is re-performed after migration.

The heuristic selects the user with smallest i-factor and increments the QoS level of that user. It then proceeds to update the Red-Black Tree with the re-computed i-factor. Considering our example, at $t = 0$, on enhancement of QoS levels, the users $u_1 \ldots u_5$ are alloted W_1, W_2, W_3, W_2 and W_3 respectively.

Computation of i-factor: The i-factor helps to determine which user causes more alterations to QoS values if the QoS level of a user is increased. Users with lower i-factor values are given higher preferences when the QoS values allocated

to them are upgraded. Equation 8 determines the i-factor of a certain user u_i having level preference and threshold of H_i^t and L_i^t respectively with presently assigned QoS level of l at time t. The QoE function E_i^t, E_{max} and α are from Eq. 7 discussed previously. The numerator affects the i-factor by scaling the QoE value according to the present QoS level, i.e., it assigns a higher i-factor as user's reach their preferred QoS levels. The denominator demarcates the difference between H_i^t and L_i^t, the higher the difference, the lower is i-factor.

$$ifactor = \frac{E_{max} \times (E_i^t + l)}{\alpha \times max(H_i^t - L_i^t, 1)} \tag{8}$$

Migrating Users for Improving QoE: Once all the Red-Black trees corresponding to all edge servers have been updated, we find the list of users who can be migrated from the servers which have exhausted their resource capacities and hence, no further QoS upgradation for users are possible. Upon successful migration, our allocation algorithm is re-initiated for possible QoS upgradation.

5 Experiments and Analysis of Results

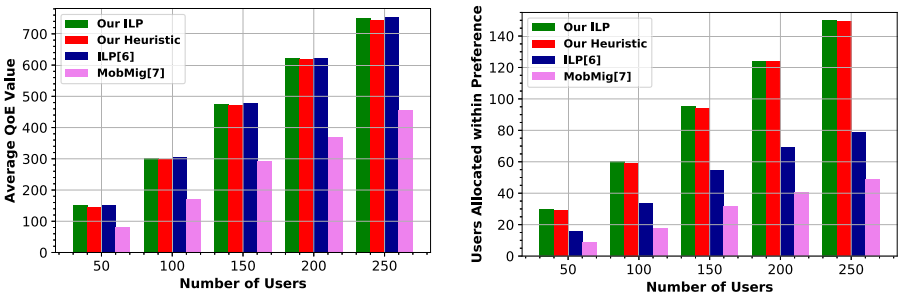
All experiments were conducted on a machine with Intel Core i5-8250U processor and 8 GB RAM. The ILP model discussed in Sect. 3 was solved using the Python Mixed-Integer-Programming library. The results from our heuristic are compared with the baseline ILP formulated in Sect. 3, the optimal algorithm presented in [6] and the dynamic mobility aware policy in [7].

Experimental Setup: We use the EUA data-set for edge server locations, which includes data of base stations and users within the Melbourne Central Business District area. The coverage area of edge servers are set randomly to values between 200-400 m radius. To simulate different attributes of users over time, we randomly select several users and do the following: a) randomly assign 20% users with 0 m/s for static users, 30% users with random speed between $1-2$ m/s for walking users, and the remaining 50% users with speed between $10-20$ m/s for users in vehicle; b) randomly assign an initial direction between 0° to 360° which then follows the random way-point mobility model [7]; and c) randomly assign the users' high and low QoS preferences.

We generate latencies from the real world PlanetLab and Seattle latency data-set [10]. Since the PlanetLab and Seattle latency data-set comprises latencies from across the world, which is not fully representative of latencies in an MEC environment, we cluster the data-set into 400 clusters considering devices which are in proximity of each other. A cluster is randomly picked and a representative latency is assigned according to our latency measure derived based on the distance and QoS level, as in [9]. We consider the product of distance and QoS level, which is scaled down according to the number of clusters. A discrete-time slotted model with each slot of 25 s is considered in which the users move and change their QoS preferences dynamically. At the end of each time slot, some user locations are updated, and to 20% of users, we randomly assign new

preference levels to simulate dynamic QoS preferences. The number of discrete time slots is kept at 20 for each experiment. To consider various sizes of user population, we vary the number of users from 50 to 250 at intervals of 50 users, while keeping the number of servers to 50 and the server resources at 100% of the cumulative resource requirement of all users at the highest QoS level, distributed uniformly over all servers. Each experiment is averaged over 50 runs. For the QoE model, we set $E_{max} = 5$, $\alpha = 1.5$. We compare the results of our ILP, our heuristic, the static ILP proposed in [6] and MobMig [7], a Mobility-aware dynamic allocation policy. We consider the ILP in [6] by running it in each discrete time step since it is a static formulation. We use MobMig by setting the QoS level as highest possible since MobMig does not support dynamic QoS changes. For comparison, we study the following metrics: a) Average QoE achieved per time slot; b) Average number of users allocated within their QoS preference per time slot; c) Average execution time (CPU time) for evaluation of algorithms; and d) Average latency experienced by users.

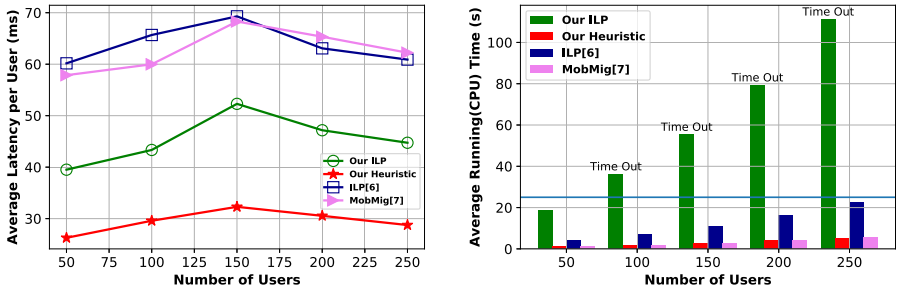
Results and Discussion: Figure 2 depicts the average QoE and the average number of users allocated within their QoS preference on the experimental setup with varying users. The results show the effectiveness of the heuristic in being able to generate near optimal solutions comparable with the results from the optimal ILP for both average QoE and average number of users allocated within their QoS preferences. The ILP achieves better allocation of users within their QoS preference having QoE values similar to the ILP in [6]. MobMig [7], being unaware of user QoS preferences allocates users at highest available QoS level when used in a variable QoS scenario. Consequently, the policy leads to a violation in preference levels in a large fraction of users as inferred from Fig. 2b. However, the ILP [6], which seeks to optimize overall QoE, generates near equivalent QoE and number of allocated users as compared to our ILP and heuristic.



(a) Total QoE (b) Number of Allocated Users

Fig. 2. Varying users experiment results

The average latency per user is depicted in Fig. 3a. As can be inferred from Fig. 3a, both our optimal and heuristic policies significantly outperform MobMig and the ILP in [6] in terms of average latency incurred by the users. This is because our preference aware policies provide the flexibility to dynamically



(a) Average latency per user (b) Average running time

Fig. 3. Latency and running time for allocation policies

adapt QoS values depending on user-qos preference levels and hence conserve resources both at the server end and at the user end. Additionally, at the user-end, adapting to changing QoS levels, prevents higher communication data transfer latencies. As such, our heuristic, which initially assigns the lowest assignable QoS value to users, while progressively upgrading the QoS values depending on resource availability, results in a much lower average latency owing to lower communication overhead. Figure 3b additionally depicts the efficiency of our algorithm in a mobility-driven dynamic scenario where the heuristic takes a fraction of the running time of our ILP. Our heuristic requires lower running times as compared to the ILP in [6] while requiring similar running times to MobMig simultaneously taking QoS-preferences into account. For each algorithm, we consider time-out as 25 s, i.e., the length of each slot. In Fig. 3b, however, we illustrate the time it would have actually taken by the algorithms for the allocation to compare effectiveness.

6 Conclusion and Future Work

In this paper, we have proposed a novel approach to the user-centric dynamic QoS edge user allocation problem. We formulated an optimal ILP and a near optimal heuristic to aid scalability in mobility driven real-world scenarios. As future work, we are working on learning based strategies for modeling user movements, QoS preferences, service invocations and migrations.

References

1. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: MCC, pp. 13–16. ACM (2012)
2. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. The MIT Press, Cambridge (2009)
3. Guo, H., Liu, J., Qin, H.: Collaborative mobile edge computation offloading for IoT over fiber-wireless networks. IEEE Network **32**(1), 66–71 (2018)

4. He, Q., et al.: A game-theoretical approach for user allocation in edge computing environment. IEEE TPDS **31**, 515–529 (2020)
5. Lai, P., et al.: Optimal edge user allocation in edge computing with variable sized vector bin packing. In: Pahl, C., Vukovic, M., Yin, J., Yu, Q. (eds.) ICSOC 2018. LNCS, vol. 11236, pp. 230–245. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03596-9_15
6. Lai, P., et al.: Edge user allocation with dynamic quality of service. In: Yangui, S., Bouassida Rodriguez, I., Drira, K., Tari, Z. (eds.) ICSOC 2019. LNCS, vol. 11895, pp. 86–101. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33702-5_8
7. Peng, Q., et al.: Mobility-aware and migration-enabled online edge user allocation in mobile edge computing. In: ICWS, pp. 91–98 (2019)
8. Wang, C., Liang, C., Yu, F.R., Chen, Q., Tang, L.: Computation offloading and resource allocation in wireless cellular networks with mobile edge computing. IEEE TWC **16**(8), 4924–4938 (2017)
9. Wang, S., Guo, Y., Zhang, N., Yang, P., Zhou, A., Shen, X.S.: Delay-aware microservice coordination in mobile edge computing: a reinforcement learning approach. IEEE TMC, pp. 1–1 (2019)
10. Zhu, R., Liu, B., Niu, D., Li, Z., Zhao, H.V.: Network latency estimation for personal devices: a matrix completion approach. IEEE/ACM ToN **25**(2), 724–737 (2016)