# Mobility-Aware Service Placement for Vehicular Users in Edge-Cloud Environment

Rahul Mudam[1], Saurabh Bhartia[1], Soumi Chattopadhyay[1] and Arani Bhattacharya[2]

[1] Indian Institute of Information Technology, Guwahati, India
[2] Indraprastha Institute of Information Technology, Delhi, India

**Abstract.** In the era of Internet-of-things (IoT), both the number of web services and the number of users invoking them are increasing everyday. These web services utilize a cloud server for access to sufficient compute resources for service delivery. A disadvantage of cloud computing is that it is known to have a high latency because of its large distance (both physical distance as well as number of hops) from the end user device. A key technique of enabling low-latency web services, called edge computing, brings the compute resources closer to the end device. Edge computing enables better resource utilization and it reduces latency. However, since there are numerous compute resources or 'edge resources', determining where the services should be placed becomes a new challenge. In this paper, we consider the case of public transport vehicles utilizing edge computing to reduce latency while providing such web services. We first model the dynamic service placement problem considering user mobility. We then propose two algorithms to solve this problem. The first algorithm utilizes an Integer Linear Programming (ILP) to obtain an optimal solution, albeit at the cost of scalability. We then propose a heuristic algorithm to achieve a low latency, while also scaling to large problem instances. We validate the performance of both the techniques through extensive trace-driven simulations.

## 1 Introduction

With the recent improvement of wireless connectivity, services are generally delivered by software vendors from data centers. This paradigm of delivering services, called cloud computing [17], has made it easier for software vendors to provide new services or upgrade their existing ones. However, cloud computing being inherently centralized, fails to deliver low latency to users [2]. With latency becoming a major factor in satisfying users, multiple works have proposed using a more decentralized architecture, that complements the existing data centers. This decentralized architecture is known as edge computing [17]. In edge computing, data is placed in locations that are physically and logically closer to the user. This could be either mini-data centers associated with the network base stations [18], or compute resources provided by a third party [13]. Edge computing promises to deliver low-latency services, while maintaining the advantages of data centers.

However, utilizing edge computing to deliver services in practice has a number of challenges. For example, it is possible for users to move around in vehicles. Such movement of users makes it challenging to decide the edge device where data and state of services should be placed that are sought by users [24]. While a number of works have studied this problem of service placement, most of them do not consider the mobility of users [2, 16]. Works that have looked at the mobility of users either focus on a single user [1, 10] or depend on more compute-intensive techniques like path prediction or other forms of learning [3, 9, 10, 14, 25]. Many of them also do not consider the memory constraints imposed by edge devices.

In this paper, we study the service placement problem in a dynamic environment considering user mobility, viz., physical mobility of users while accessing said services. We consider the case of users situated in moving vehicles, from which they are accessing a set of services at different points of time. This also includes the possibility of the vehicles themselves accessing services, in particular, in the case of autonomous or semi-autonomous vehicles. The objective therefore is to dynamically place/re-place the services either on the cloud or on an edge device to minimize the overall latency felt by the users. Here, we first model the service placement problem in a dynamic environment. We then propose an optimal (oracle) algorithm to solve this problem using Integer Linear Programming (ILP). Though the optimal algorithm gives a solution, it assumes that information about the entire trajectory of the vehicle is available a priori. We show that it also suffers from scalability issues, i.e. for a large dataset, it is incapable of generating results in real-time.

To circumvent this problem, we utilize the ILP to solve the problem in stages. We utilize the fact that although the entire trajectory of the vehicles is usually not known in advance, users tend to know their next few destinations. This information is usually available from location-based applications such as Google Maps. Using this information, we are able to repeatedly run the ILP to obtain solutions for different time windows. Finally, we also propose a heuristic called First Come First Serve (FCFS) that utilizes information about only one destination to solve the service placement problem.

Our evaluation utilizes traces of publicly available datasets. We show that our technique reduces latency significantly (around 6.76 times better) compared to multiple baseline techniques. We also show that the FCFS heuristic in most cases performs close to the optimal solution given by ILP. Finally, we also show that the execution overhead of our heuristic is negligible compared to the amount of reduction in latency.

We summarize our contributions as follows:

1. We model the dynamic service placement problem considering user mobility and formulate it as a ILP to minimize the overall latency.
2. We propose two techniques of obtaining a realistic solution. The first technique solves the ILP in multiple time windows. The second technique uses a heuristic algorithm called FCFS using information about the next destination of the vehicle.
3. We perform extensive experiments based on real dataset to compare the optimal and FCFS algorithms with multiple baseline techniques. We show that FCFS provides on average 6.76 times lower latency than the best baseline, while adding an overhead of only order of tens of milliseconds.

## 2 Problem Formulation

In this section, we design a mathematical model of the dynamic service placement problem (DSPP). We consider our model as *dynamic* since here we assume that the location of the trajectory changes over time. In other words, the relative position between the user and each edge device keeps on changing over time. We begin with describing our system under consideration. We model our system as eight tuples $\mathcal{M} = (\mathcal{C}, \mathcal{E}, \mathcal{N}, \mathcal{R}, \mathcal{T}, \mathcal{H}, \mathcal{S})$:

1. A set of $n$ execution platforms $Ex = \mathcal{C} \cup \mathcal{E}$, where
   (a) $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$ is a set of cloud servers.

   (b) $\mathcal{E} = \{\mathcal{E}_{k+1}, \mathcal{E}_{k+2}, \ldots, \mathcal{E}_n\}$ is a set of edge devices.
2. A set of network parameters $\mathcal{N}$.
3. A vehicular road route map $\mathcal{R}$.
4. A set of vehicular trajectories $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_m\}$.
5. A set of handheld devices $\mathcal{H} = \{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m\}$, where each $\mathcal{H}_i \in \mathcal{H}$ is associated with a unique $\mathcal{T}_i \in \mathcal{T}$.
6. A set of services $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_r\}$.

We now discuss the characterization of each of the components of $\mathcal{M}$ in details.

1. Each execution platform $Ex_i \in Ex$ is characterized by three tuples $(U_i, D_i, TC_i)$, where $U_i, D_i$ and $TC_i$ refer to the uplink speed, downlink speed and the total memory capacity of $Ex_i$, respectively. We assume each cloud server to have sufficient memory, i.e., $\forall \mathcal{C}_i \in \mathcal{C}, TC_i$ is infinite, so that the execution of the web services is not constrained by the memory capacity of a cloud server. This assumption is true for most commercial cloud servers.
2. The set of network parameters $\mathcal{N}$ contains an average propagation delay $PD_{i,j}$ of an execution platform $Ex_i \in Ex$ from a location $\mathcal{L}_j$.
3. The route map $\mathcal{R} = (\mathcal{L}, E)$ is given as a graph, where
   - $\mathcal{L}$ is the set of vertices of the graph. Each vertex $\mathcal{L}_i \in \mathcal{L}$ of $\mathcal{R}$ represents a location.
   - $E$ is the set of links of the graph. Each link $e_{ij} = (\mathcal{L}_i, \mathcal{L}_j) \in E$ indicates the existence of a vehicular road between locations $\mathcal{L}_i$ and $\mathcal{L}_j$.
4. Each trajectory of a vehicle $\mathcal{T}_i$ is modeled as the tuple of the following tuples: $\mathcal{T}_i = ((\mathcal{L}_{i1}, TS_{i1}, TS'_{i1}, \hat{\mathcal{S}}_{i1}), (\mathcal{L}_{i2}, TS_{i2}, TS'_{i2}, \hat{\mathcal{S}}_{i2}), \ldots, (\mathcal{L}_{ix}, TS_{ix}, TS'_{ix}, \hat{\mathcal{S}}_{ix}))$, where
   (a) The vehicle passes through the locations $\mathcal{L}_{i1}, \mathcal{L}_{i2}, \ldots, \mathcal{L}_{ix}$.
   (b) At timestamp $TS_{ij}$ the vehicle reaches the location $\mathcal{L}_{ij}$ and at timestamp $TS'_{ij}$ the vehicle leaves $\mathcal{L}_{ij}$.
   (c) $\hat{\mathcal{S}}_{ij}$ is the set of services invoked by $\mathcal{T}_i$ at location $\mathcal{L}_{ij}$.

   Since we target services invoked on public transport vehicles, we expect the timestamps and the routes taken to be known in advance.
5. Each handheld device $\mathcal{H}_i \in \mathcal{H}$ is characterized by two tuples $(U_i^h, D_i^h)$, where $U_i^h$ and $D_i^h$ refer to the uplink and downlink speeds, respectively.
6. $\mathcal{S}$ is the set of services invoked by the vehicles. The data requirement of each service varies across the trajectories. Therefore, each service $\mathcal{S}_i \in \mathcal{S}$ of a trajectory $\mathcal{T}_j \in \mathcal{T}$ is defined as 3-tuple: $\mathcal{S}_i = (Ip_{i,j}, Op_{i,j}, PM_{i,j})$, where
   (a) $Ip_{i,j}$ is the average input file size required to be uploaded to an execution platform from the handheld device of the trajectory $\mathcal{T}_j$ to invoke the service $\mathcal{S}_i$.
   (b) $Op_{i,j}$ is the average output file size generated by $\mathcal{S}_i$ and to be downloaded from the respective execution platform to the handheld device of $\mathcal{T}_j$.
   (c) $PM_{i,j}$ is the worst case peak memory required by $\mathcal{S}_i$ to be executed when invoked from $\mathcal{T}_j$.

Our model is generic enough to handle interactive services as well. An interactive service can be divided into multiple blocks, where each block can be represented by a service as defined above. Therefore, an interactive service, in our model, can be treated as multiple atomic services.

We now define the notion of latency for $\mathcal{S}_i$ of $\mathcal{T}_j$. Consider $\mathcal{S}_i$ being invoked from $\mathcal{H}_j$ from $TS_x$ to $TS_{x+k'}$ while traveling through $\mathcal{L}_{l0}, \mathcal{L}_{l1}, \ldots, \mathcal{L}_{lk'}$. Also consider in each $TS_k$, for $k \in \{x, x+1, \ldots, x+k'\}$, $\mathcal{S}_i$ is executed in $Ex_{pk} \in Ex$. The latency for $\mathcal{S}_i$ of $\mathcal{T}_j$ is computed as:

1. *Initial Upload:* At the initial timestamp $TS_x$, when $\mathcal{S}_i$ of $\mathcal{T}_j$ starts its execution, the input file (i.e., $IP_{i,j}$) of $\mathcal{S}_i$ of $\mathcal{T}_j$ is uploaded to $Ex_{p0}$ from location $\mathcal{L}_{l0}$.
2. *Final Download:* At the final timestamp $TS_{x+k'+1}$, when $\mathcal{S}_i$ of $\mathcal{T}_j$ finishes its execution, the output file (i.e., $OP_{i,j}$) of $\mathcal{S}_i$ of $\mathcal{T}_j$ is downloaded from $Ex_{pk'}$ to $\mathcal{H}_j$ at $\mathcal{L}_{l(k'+1)}$.
3. *Intermediate Transfer:* In an intermediate timestamp $TS_k$, for $x < k \leq (x + k')$, when $\mathcal{S}_i$ of $\mathcal{T}_j$ continues its execution, the state of $\mathcal{S}_i$ of $\mathcal{T}_j$ may be transferred from $Ex_{p(k-1)}$ to $Ex_{pk}$ through the handheld device $\mathcal{H}_j$ at location $\mathcal{L}_{lk}$. The state size of a service is represented by its peak memory, i.e., $PM_{i,j}$.

The uploading/downloading latency for $\mathcal{S}_i$ of $\mathcal{T}_j$ has two key components: (a) transmission delay and (b) propagation delay. The service uploading/downloading is associated with three different events: (i) data uploading from the sender device, (ii) data propagation from a sender device to the receiver device, (iii) data downloading to the receiver device. While (i) and (iii) together determine the transmission delay, (ii) decides the propagation delay. Mathematically, the latency is defined as:

$$\begin{aligned}&\underbrace{\left(\underbrace{\frac{IP_{i,j}}{U_j^h}}_{\text{uploading}} + \underbrace{PD_{l0,p0}}_{\text{propagation delay}} + \underbrace{\frac{IP_{i,j}}{D_{p0}}}_{\text{downloading}}\right)}_{\text{Initial Upload}} + \underbrace{\left(\frac{OP_{i,j}}{U_{pk'}} + PD_{pk',l(k'+1)} + \frac{OP_{i,j}}{D_j^h}\right)}_{\text{Final Download}} + \\ &\underbrace{\sum_{k=1}^{k'}\left(\underbrace{\left(\frac{PM_{i,j}}{U_{p(k-1)}} + PD_{p(k-1),lk} + \frac{PM_{i,j}}{D_j^h}\right)}_{\text{download from } Ex_{p(k-1)}} + \underbrace{\left(\frac{PM_{i,j}}{U_j^h} + PD_{lk,pk} + \frac{PM_{i,j}}{D_{pk}}\right)}_{\text{upload to } Ex_{pk}}\right) I_{p(k-1),pk}}_{\text{Intermediate Transfer}}\end{aligned} \tag{1}$$

where, $I_{p(k-1),pk}$ is an indicator variable, indicates if state of $\mathcal{S}_i$ of $\mathcal{T}_j$ is transferred from one execution platform to another in the intermediate timestamps. Formally:

$$I_{p(k-1),pk} = \begin{cases} 1, & \text{if } Ex_{p(k-1)} \neq Ex_{pk} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

The key objective of this work is to reduce the overall latency across all services of all trajectories. Since the edge compute resources available are usually fixed, this requires us to design an algorithm to decide where to place each of the services. We model this objective as that of reducing the sum of latencies across all the services and all the users.

## 3 Detailed Methodology

In this section, we present our methodology to solve DSPP. We first propose the optimal solution for DSPP, followed by a heuristic solution based on First Come First Serve (FCFS).

### 3.1 Optimal Algorithm

Our optimal solution is based on the Integer Linear Programming (ILP) formulation. We first define a set of Boolean variables $\mathcal{B}$ as follows:

$$y_{i,j,k,l,p,u} = \begin{cases} 1, & \text{if } \mathcal{S}_i \text{ of } \mathcal{T}_j \text{ is uploaded to } Ex_p \text{ from } \mathcal{L}_l \text{ at } TS_k \\ 0, & \text{otherwise} \end{cases}$$

$$y_{i,j,k,l,p,d} = \begin{cases} 1, & \text{if } \mathcal{S}_i \text{ of } \mathcal{T}_j \text{ is downloaded from } Ex_p \text{ at } \mathcal{L}_l \text{ at } TS_k \\ 0, & \text{otherwise} \end{cases}$$

$$z_{i,j,k,p} = \begin{cases} 1, & \text{if } \mathcal{S}_i \text{ of } \mathcal{T}_j \text{ is executing in } Ex_p \text{ at } TS_k \\ 0, & \text{otherwise} \end{cases}$$

We now design the objective function and the set of constraints required to formulate the ILP using $\mathcal{B}$. It may be noted that in this formulation, the objective is to minimize the overall latency across all services of all trajectories, which is obtained by choosing the appropriate value of each Boolean variables in $\mathcal{B}$ by the ILP solver. Therefore, the objective function is formulated by summing up the latency of all services of all trajectories. However, the latency expression of $\mathcal{S}_i$ of $\mathcal{T}_j$, which is used in the objective function of ILP, is different from Expression (1). In the definition of the latency of $\mathcal{S}_i$ of $\mathcal{T}_j$, we consider that the execution platform, where $\mathcal{S}_i$ of $\mathcal{T}_j$ to be placed in each timestamp is known to us. However, in the objective function of this formulation, the execution platform is to be decided by the ILP solver itself. Therefore we need to reformulate the latency expression for each service of each trajectory. We now discuss the three cases again, which we discussed earlier to define the latency.

Let the execution time span of $\mathcal{S}_i$ of $\mathcal{T}_j$ from timestamp $TS_x$ to $TS_{x+k'}$ be denoted by $\mathit{\Gamma}_{ij} = \{TS_{x+0}, TS_{x+1}, \ldots, TS_{x+k'}\}$, while $\mathcal{T}_j$ is at location $\mathcal{L}_{lk}$ at $TS_{x+k} \in \mathit{\Gamma}_{ij}$.

– *Initial Upload*: At the initial timestamp (i.e., at $TS_x$), $\mathcal{S}_i$ of $\mathcal{T}_j$ is uploaded to an execution platform to be decided by the ILP solver, and captured by the following.

$$\lambda^1_{ij} = \sum_{Ex_p \in Ex} \left( \frac{IP_{i,j}}{U^h_j} + PD_{l0,p} + \frac{IP_{i,j}}{D_p} \right) y_{i,j,x,l0,p,u} \tag{3}$$

We note that in Expression (3), only one Boolean variable corresponding to the execution platform, where $\mathcal{S}_i$ of $\mathcal{T}_j$ is to be uploaded initially, is set to 1 by the ILP solver. We ensure this by adding a constraint, which is discussed later.

– *Intermediate Transfer*: In each intermediate timestamp, $\mathcal{S}_i$ of $\mathcal{T}_j$ has two options: either $\mathcal{S}_i$ of $\mathcal{T}_j$ continues its execution in the same platform executing in the previous timestamp, or it gets downloaded from the previous execution platform and uploaded to some other execution platform. Mathematically,

$$\lambda^2_{ij} = \sum_{k=1}^{k'} \Bigg( \sum_{Ex_p \in Ex} \Big( \frac{PM_{i,j}}{U_p} + PD_{p,lk} + \frac{PM_{i,j}}{D^h_j} \Big) y_{i,j,k,lk,p,d} + \sum_{\substack{Ex_q \in Ex, \\ Ex_p \neq Ex_q}} \Big( \frac{PM_{i,j}}{U^h_j} + PD_{lk,q} + \frac{PM_{i,j}}{D_q} \Big) y_{i,j,k,lk,q,u} \Bigg) \tag{4}$$

– *Final Download*: In this case, $\mathcal{S}_i$ of $\mathcal{T}_j$ has to be downloaded from its last execution platform, which is expressed by the following expression.

$$\lambda^3_{ij} = \sum_{Ex_p \in Ex} \left( \frac{OP_{i,j}}{U_p} + PD_{p,l(k'+1)} + \frac{OP_{i,j}}{D^h_j} \right) y_{i,j,(k'+1),l(k'+1),p,d} \tag{5}$$

We now formulate the objective function of the ILP as follows:

$$\text{Minimize: } \sum_{\mathcal{T}_j \in \mathcal{T}} \sum_{\mathcal{S}_i \in \mathcal{T}_j} \left( \lambda^1_{ij} + \lambda^2_{ij} + \lambda^3_{ij} \right) \tag{6}$$

We now discuss the set of constraints required for this formulation. First, the number of times each $\mathcal{S}_i$ of $\mathcal{T}_j$ has been uploaded to an execution platform $Ex_p$ has to be equal to the number of times the same has been downloaded from $Ex_p$.

$$\forall \mathcal{T}_j \in \mathcal{T} \text{ and } \forall \mathcal{S}_i \in \mathcal{T}_j; \forall Ex_p \in Ex \qquad \sum_{TS_x \leq TS_k \leq TS_{k'}} y_{i,j,k,lk,p,u} = \sum_{TS_x \leq TS_k \leq TS_{k'+1}} y_{i,j,k,lk,p,d} \tag{7}$$

where, the execution time span of $\mathcal{S}_i$ of $\mathcal{T}_j$ is from $TS_x$ to $TS_{x+k'}$, while $\mathcal{T}_j$ passes through location $\mathcal{L}_{lk}$ at $TS_k$.

Each $\mathcal{S}_i$ of $\mathcal{T}_j$ continues its execution on $Ex_p$ at $TS_k$ if $\mathcal{S}_i$ of $\mathcal{T}_j$ has been uploaded to $Ex_p$, but not yet downloaded from $Ex_p$. We have the following constraint to capture this fact:

$$\begin{aligned} &\forall \mathcal{T}_j \in \mathcal{T} \text{ and } (\forall \mathcal{S}_i \in \mathcal{T}_j)\,; \quad \forall TS_k \in \Gamma_{ij}; \quad \forall Ex_p \in Ex \\ &z_{i,j,k,p} = \sum_{TS_\psi \le TS_k} y_{i,j,\psi,l\psi,p,u} - \sum_{TS_\psi \le TS_k} y_{i,j,\psi,l\psi,p,d} \end{aligned} \tag{8}$$

where, $\mathcal{T}_j$ passes through location $\mathcal{L}_{l\psi}$ at $TS_\psi$.

The following constraint ensures that each $\mathcal{S}_i$ of $\mathcal{T}_j$ must execute on exactly one execution platform in each timestamp throughout its time span.

$$\forall \mathcal{T}_j \in \mathcal{T} \;\&\; (\forall \mathcal{S}_i \in \mathcal{T}_j)\,; \forall TS_k \in \Gamma_{ij}; \sum_{Ex_p \in Ex} z_{i,j,k,p} = 1 \tag{9}$$

Our final constraint is related to the memory capacity of each edge device. At each timestamp $TS_k$, the memory constraint of each edge device has to be satisfied. An edge device cannot accept any service, if it does not have residual memory capacity to satisfy the service's memory requirement.

$$\forall TS_k \in \mathcal{T}; \forall Ex_p \in \mathcal{E}; \quad \sum_{\mathcal{T}_j \in \mathcal{T}} \sum_{\mathcal{S}_i \in \mathcal{T}_j} PM_{i,j} * z_{i,j,k,p} \le TC_p \tag{10}$$

where $TC_p$ is total capacity of $\mathcal{E}_p$.

Although the ILP provides an optimal solution to DSPP that minimizes total latency, it does not scale for large problem instances. Moreover, we may not always have complete knowledge about all the trajectories in advance. Thus, we propose a window-based optimal strategy to overcome this problem.

### 3.2 Window-based Optimal Algorithm

The crux of the window-based optimal strategy is that we do not require to have complete information about all the trajectories in advance. However, if we have prior knowledge of the next few timestamps, say $\omega$ number of timestamps, then also we can apply the same optimal algorithm on each sub-part of the trajectories. The main idea of this algorithm is to divide the entire set of timestamps into multiple windows of size $\omega$ and solve the problem optimally for each window individually. We note that we need to transfer the previous state of the system (i.e., which service of which trajectory is executing on which execution platform) to its next state to obtain the optimal solution for the next window. Clearly, when $\omega$ is the total number of timestamps across all trajectories, the window-based optimal algorithm generates the optimal solution. For a smaller value of $\omega$, although this approach does not produce an optimal solution, this approach increases the scalability as compared to the optimal algorithm, since it handles a smaller set of variables at a time.

In case of large number of timestamps, the window-based optimal algorithm scales better compared to the optimal approach. Therefore, we can use this approach as an online technique. However, for a large number of trajectories, edge devices, or the number of services per timestamp per trajectory in one window, the window-based optimal algorithm does not scale as well in real-time. This can increase the computation overhead of running it online. Therefore, in the next subsection, we propose a scalable heuristic algorithm, which can solve the placement problem dynamically in real-time.

### 3.3 Heuristic using FCFS

We now discuss our heuristic algorithm, which solves DSPP by first come first serve (FCFS) scheduling. In FCFS, if a service $\mathcal{S}_i$ of a trajectory $\mathcal{T}_j$ starts its execution earlier on an execution platform $Ex_p$ than another service $\mathcal{S}_{i'}$ of $\mathcal{T}_{j'}$, $\mathcal{S}_i$ of $\mathcal{T}_j$ gets higher priority over $\mathcal{S}_{i'}$ of $\mathcal{T}_{j'}$ on $Ex_p$. In case of tie, it gets resolved arbitrarily. The FCFS algorithm is an online algorithm as it runs on each timestamp. Therefore, this algorithm does not require the trajectory information in advance.

If a service $\mathcal{S}_i$ of a trajectory $\mathcal{T}_j$ executes at timestamp $TS_k$, we have three possibilities. Analyzing each of the possibilities, the service placement decision is taken. We now discuss the principle of the FCFS algorithm.

1. $\mathcal{S}_i$ of $\mathcal{T}_j$ is placed to the fastest execution platform accessible from the current location having the residual capacity to accommodate the service for processing at $TS_k$, if $\mathcal{S}_i$ of $\mathcal{T}_j$ starts its execution at $TS_k$.
   The fastest execution platform is the one having latency equal to $\min_{Ex_q \in Ex}\big(\frac{IP_{i,j}}{U_j^h}$ $+PD_{lk,q} + \frac{IP_{i,j}}{D_q}\big)$.
2. If $\mathcal{S}_i$ of $\mathcal{T}_j$ starts its execution before $TS_k$ and it is already on the fastest edge device accessible from the current location or on the cloud, no action needs to be taken.
3. If $\mathcal{S}_i$ of $\mathcal{T}_j$ starts its execution before $TS_k$ and it is neither on the fastest edge device accessible from the current location nor on the cloud, $\mathcal{S}_i$ of $\mathcal{T}_j$ may need to be transferred from the current edge device to the fastest execution platform accessible from the current location having capacity to accommodate it. However, this decision is taken based on a look-ahead in the next $K$ timestamps, where $K$ is an input to this algorithm, as discussed below.

Consider the execution time span of $\mathcal{S}_i$ of $\mathcal{T}_j$ is up to $TS_{x+k'}$, while $\mathcal{T}_j$ passes through location $\mathcal{L}_{lk}$ at $TS_k$. Also consider $\mathcal{S}_i$ of $\mathcal{T}_j$ executed on $Ex_q$ at $TS_{k-1}$. We first define a transfer latency $Tr(Ex_q, TS_\psi, \mathcal{L}_{l\psi})$ from $Ex_q$ to the fastest execution platform accessible from $\mathcal{L}_{l\psi}$ at $TS_\psi$ as:

$$Tr(\cdot) = \left(\frac{PM_{i,j}}{U_q} + PD_{l\psi,q} + \frac{PM_{i,j}}{D_j^h}\right) + \min_{Ex_p \in Ex}\left(\frac{PM_{i,j}}{U_j^h} + PD_{l\psi,p} + \frac{IP_{i,j}}{D_p}\right) \tag{11}$$

The execution platform chosen $Ex_c$ for $\mathcal{S}_i$ of $\mathcal{T}_j$ at $TS_k$ is:

$$Ex_c = \begin{cases} Ex_q, & \text{if } Tr(Ex_q, TS_k, \mathcal{L}_{lk}) \geq \min_{TS_\psi} Tr(Ex_q, TS_\psi, \mathcal{L}_{l\psi}) \\ Ex_p, & \text{otherwise} \end{cases} \tag{12}$$

where, $TS_k \leq TS_\psi \leq \min(TS_{k+K}, TS_{x+k'})$, $Ex_p$ is the fastest platform accessible from $\mathcal{L}_{lk}$ and has residual capacity to accommodate $\mathcal{S}_i$ of $\mathcal{T}_j$. The above equation checks whether the total time required to transfer $\mathcal{S}_i$ of $\mathcal{T}_j$ at $TS_k$ from $\mathcal{E}_q$ to $\mathcal{E}_p$ is less than the time required to transfer it in the later timestamps. If that is the case, $\mathcal{S}_i$ of $\mathcal{T}_j$ is transferred at $TS_k$. Otherwise, $\mathcal{S}_i$ of $\mathcal{T}_j$ continues its execution on $Ex_q$. We make the following observations about FCFS:

1. Since FCFS is an online algorithm, it is executed in each timestamp in each handheld device, which adds an additional overhead in the overall latency. Experimentally, we have shown that the overhead incurred due to the execution of this algorithm is very small.
2. Whenever an edge device accepts any service for execution, it broadcasts its own residual capacity. Therefore, computation of residual capacity of edge devices does not have any impact on the latency computation.

3. The quality of this algorithm depends on the value of $K$. In general, with an increase in the value of $K$, the solution quality, i.e., the latency monotonically improves. However, after certain value of $K$, this improvement stagnates. We also note that with an increase in the value of $K$, the computation time of the FCFS algorithm increases up to a certain value of $K$. Unless mentioned otherwise, we consider the value of $K$ as 1. However, in the experimental section, we have shown the trade-off between the computation time and the solution quality in terms of the latency for different values of $K$.

**Time Complexity:** The FCFS algorithm iterates over each timestamp, and in one timestamp, the algorithm iterates over each trajectory to find out what all services are executed in that timestamp. It accordingly places the services on an appropriate execution platform to obtain a low latency. Therefore, the complexity of the FCFS algorithm is polynomial in the size of the set of trajectories. More specifically, the worst-case time complexity of the FCFS algorithm is the order of the size of the set of trajectories, i.e., $O(|\mathcal{T}|)$, since each trajectory is defined as the set of services accessed across all timestamps.

## 4 Experimental Results

In this section, we present our experimental results with analysis. We implemented our proposed framework in python. All experiments were performed on a system with the following configuration: AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx, 2100 Mhz, 4 Cores(s), 8 logical processor(s) @ 2.10 GHz with 8GB DDR4 RAM. We used Gurobi [5] as the ILP solver. We begin with a discussion of the datasets used for our evaluations.

### 4.1 Dataset Generation

We conducted our experiments on a real dataset, which we generated for the evaluation and a set of synthetically generated datasets. We now discuss these two datasets in detail.

**Real Dataset Generation** We could not find any real benchmark dataset that can be used directly to evaluate the performance of our proposed framework. Therefore, we designed our own dataset by combining multiple datasets to model various dimensions of our problem model. We now demonstrate each component of $\mathcal{M} = (\mathcal{C}, \mathcal{E}, \mathcal{N}, \mathcal{L}, \mathcal{R}, \mathcal{T}, \mathcal{H}, \mathcal{S})$ below.

In our real dataset, we considered only one cloud server. We used the Pantheon[3] dataset to generate the uplink and downlink speeds of the cloud. We assumed one edge device per location and generated the uplink and downlink speeds of each edge device randomly between 3 MBps to 10 MBps. We obtained these values by observing actual transmission speeds using a Wi-Fi dongle connected to a Raspberry Pi. The size of the memory of each edge device was generated randomly between 512 MB to 4 GB considering the configuration of Raspberry Pi. The propagation delay from a location to an edge device was generated randomly following a distribution, which was obtained from our collected ping latency data. We conducted an experiment to collect the ping latency of our institute server from different locations. The propagation delay from a location to the cloud was generated from the distribution

[3] `https://www.pantheon.stanford.edu/summary/?page=1`

obtained from our collected data containing the ping latency of Amazon and Google servers from different locations. The set of locations, route map and the set of vehicular trajectories were constructed from gatech[4] dataset. The gatech dataset contains 10 different user trajectories, where the positions of the vehicles were captured in terms of latitude and longitude pairs. We first extracted the latitude-longitude pairs from the dataset. We then used K-means [8] algorithm with Haversine distance [20] function to discretize the set of locations. The route map was generated from the vehicular trajectories of gatech dataset. To obtain the set of services and their duration of invocations per user, we used Carat[5] dataset. The uplink and downlink speeds of each handheld device were estimated from a distribution, obtained from the uplink and downlink speeds of a set of cell phones. Finally, we generated the input, output and the worst case memory requirement of each service by random sampling from our collected dataset. We performed an experiment on publicly available service APIs to obtain the input, output and the worst case memory requirement.

**Synthetic Dataset Generation** To show the performance scalability of our framework, we extended our experiments on synthetically generated dataset, which we discuss below. For each instance of the dataset, externally, we provided the number of edge devices and clouds, the number of trajectories, the total number of services, the number of timestamps, and the number of services per trajectory per timestamp. For each dataset, we first randomly generated the graph representing the route map. The number of vertices in the graph was equal to the number of edge devices. We used a probability $p$ following the uniform distribution to generate a link between each pair of vertices. We note that a trajectory of our system is nothing but a path of the graph, which was chosen randomly. Finally, for each trajectory tuple, we randomly assigned two timestamps. The first timestamp shows the time to enter into the location, while the second timestamp shows the time to leave the location. The rest of the part of each dataset were generated similarly, as described above.

### 4.2 Comparative Methods

We compared our methods with three baseline techniques:

***1) Proactive method:*** This method assumes that if a service of a trajectory is uploaded to any execution platform, it continues the execution until the execution platform becomes inaccessible from the current location of the vehicular trajectory. In this approach, a service of a trajectory is uploaded, if required, to the fastest execution platform accessible from the current location of the trajectory having the residual capacity to accommodate the service. The inaccessibility of an execution platform is measured in terms of its propagation delay. A service of a trajectory is transferred from one execution platform to another when the propagation delay of the former execution platform is more than a given threshold value from the current location of the trajectory.

***2) Reactive method:*** In this method, each service of a trajectory changes the execution platform along with the vehicular trajectory across the span of the service. Here, in each location of a trajectory, a service of the trajectory is uploaded to the fastest execution platform accessible from the current location of the trajectory having the residual capacity to accommodate the service.

[4] `https://www.crawdad.org/gatech/vehicular/20060315/`
[5] `https://www.cs.helsinki.fi/group/carat/data-sharing/`

Table 1. Comparative study on the real dataset (all in seconds)

| Network parameters | | | | | |
|---|---|---|---|---|---|
| $\|\mathcal{T}\|$ | $\|\mathcal{S}\|$ | $\|\mathcal{E}\|$ | $\|TS\|$ | $\#\mathcal{S}/\mathcal{T}, TS$ | |
| 10 | 121 | 10 | 10 | 2 to 5 | |
| Comparative study with different algorithms | | | | | |
| Subject | Optimal | FCFS | Proactive | Reactive | Cloud |
| Lat | - | 12.52 | 12.52 | 17.37 | 291.11 |
| CT | - | 0.01 | 0.02 | 0.024 | 0.02 |
| FCFS with $K$-look ahead | | | | | |
| $K$ | 2 | 3 | 5 | 7 | 10 |
| Lat | 8.61 | 6.14 | 6.07 | 6.07 | 6.07 |
| CT | 0.01 | 0.01 | 0.01 | 0.02 | 0.02 |

**note: $\#\mathcal{S}/\mathcal{T}, TS$ the number of services per trajectory per timestamp

***3) Cloud-based method:*** In this method, each service of each trajectory is uploaded to the cloud. The service continues its execution in the cloud until it finishes. Finally, the service gets downloaded from the cloud to the handheld device.

### 4.3 Analysis on real dataset

In this section we briefly discuss our experimental analysis. We begin with analyzing the results obtained on the real dataset. Table 1 shows the results on the real dataset. From the table we have the following observations:

1. **Performance of FCFS:** The proposed First Come First Serve (FCFS) algorithm (with look-ahead 1) was as good as the proactive technique and better than the reactive and cloud-based techniques in terms of latency.
2. **Performance of Optimal:** The optimal algorithm was unable to produce any result due to the size of the dataset.
3. **Execution time of FCFS:** Our FCFS algorithm was able to generate the results in the order of tens of milliseconds.
4. **Impact of look-aheads on FCFS:** As we increased the number of look-aheads $K$, the latency monotonically reduced. However, this improvement stagnates beyond $K = 5$. This is because for our available dataset, the entire set of decision parameters can usually be obtained by FCFS when $K \geq 5$. As is evident from Table 1, the execution timespan of any service is bounded by 5 timestamps.

To generalize the overall characteristics of our proposed optimal and FCFS algorithms, we further extended our experiments on synthetically generated datasets.

### 4.4 Analysis on synthetically generated dataset

In this analysis, we varied different network parameters, i.e., the number of trajectories ($|\mathcal{T}|$), the number of edge devices ($|\mathcal{E}|$), the number of timestamps ($|TS|$), the number of services ($|\mathcal{S}|$), and the number of services per trajectory per timestamp ($\#\mathcal{S}/\mathcal{T}, TS$), to analyze the performance of different algorithms. At a time we varied only one parameter while keeping the rest of the parameters constant.

We first performed an experiment with a smaller dataset. Figures 1(a) - (e) shows the comparative study between different algorithms. We have the following observations:

1. **Performance of Optimal Algorithm:** As evident from Figures 1(a) - (e), the optimal algorithm produced the best results in terms of latency. However, the optimal algorithm was quite expensive in terms of computation time. For a relatively large dataset, the optimal algorithm was, therefore, unable to produce any result.

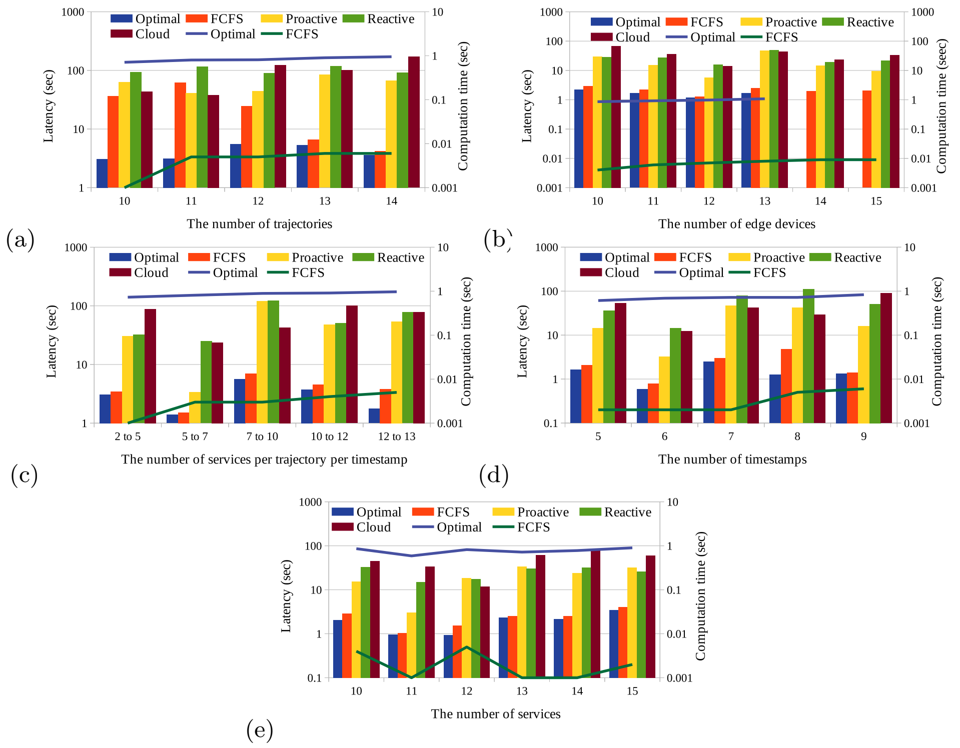
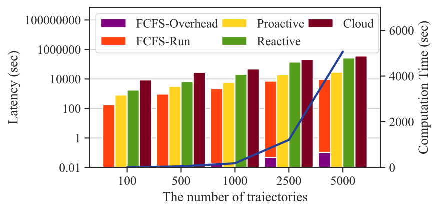


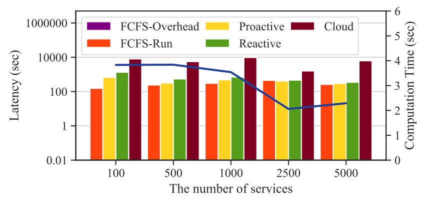
Fig. 1: Comparative study between different methods by varying the number of (a) trajectories ($|\mathcal{S}| = 10$; $|\mathcal{E}| = 5$; $|TS| = 5$; $\#\mathcal{S}/\mathcal{T}, TS = 2-5$); (b) edge devices ($|\mathcal{T}| = 5$; $|\mathcal{S}| = 10$; $|TS| = 5$; $\#\mathcal{S}/\mathcal{T}, TS = 2-5$); (c) services per trajectory per timestamp ($|\mathcal{T}| = 5$; $|\mathcal{S}| = 20$; $|\mathcal{E}| = 5$; $|TS| = 5$); (d) timestamps ($|\mathcal{T}| = 5$; $|\mathcal{S}| = 10$; $|\mathcal{E}| = 5$; $\#\mathcal{S}/\mathcal{T}, TS = 2-5$); (e) services ($|\mathcal{T}| = 5$; $|\mathcal{E}| = 10$; $|TS| = 5$; $\#\mathcal{S}/\mathcal{T}, TS = 2-5$)

2. **Comparison between FCFS and Optimal Algorithms:** On average, the optimal algorithm was 2.43 times better than the FCFS algorithm in terms of latency, while the FCFS algorithm was 306 times faster than the optimal algorithm. This signifies the purpose of the FCFS algorithm.
3. **Comparison between FCFS and baseline techniques:** In few cases, the FCFS algorithm was worse than the proactive or cloud-based techniques. As apparent from Figures 1(a) - (e), in only 1 out of 27 cases, the proactive technique and the cloud-based technique had 1.5 times and 1.6 times lower latency than the FCFS, respectively. However, on average, the FCFS algorithm had 6.76 times lower latency than the best algorithm among proactive, reactive, and cloud-based techniques (in each case).
4. **Variation of network parameters:** As observed from Figures 1(a) - (e), with the increase in the number of trajectories, edge devices, timestamps or services per trajectory per timestamp, the computation time of both the optimal and the FCFS algorithms monotonically increased. However, the number of services did not influence the computation time. While the number of services increases the variation of services to be invoked, it does not increase the total number of services invoked from each trajectory.

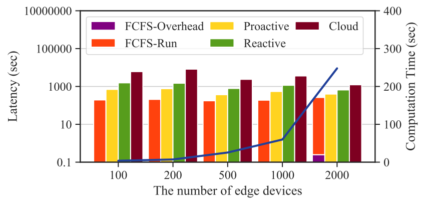
As discussed earlier, for a large dataset, the optimal algorithm was not able to produce any result. However, our FCFS algorithm is scalable enough to generate results in a reasonable time limit. Figures 2(a) - (e) show the results on larger datasets. A similar trend was observed from the larger datasets as well. Here, in Figures 2(a) -



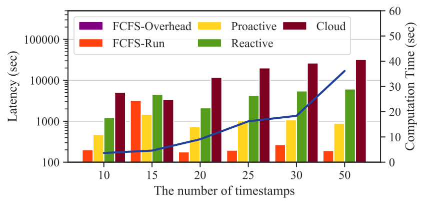
(a)



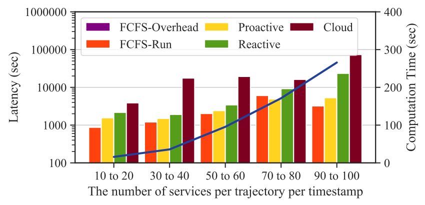
(b)



(c)



(d)



(e)

Fig. 2: Comparative analysis on large datasets by varying the number of (a) trajectories; (b) edge devices; (c) services per trajectory per timestamp; (d) timestamps; (e) services; Unless otherwise mentioned, the general configuration of the network parameters for this experiment are: $|\mathcal{T}| = 100$; $|\mathcal{S}| = 100$; $|\mathcal{E}| = 100$; $|TS| = 10$; $\#\mathcal{S}/\mathcal{T}, TS = 5 - 10$

(e), we reported the overall time taken by the FCFS algorithm across all trajectories across all timestamps (as shown by the blue line graph in Figure 2). However, in each case, we also reported the latency overhead (i.e., the computation time of the FCFS algorithm in each trajectory in each timestamp) added due to the computation time of the FCFS algorithm. We note that for the larger dataset, the latency overhead generated due to the computation time of the FCFS algorithm was significant, as shown in Figures 2(a) - (e). However, the FCFS algorithm was still 2.57 times better than the best technique among proactive, reactive, and cloud-based methods (in each case) in terms of latency. We further note that only in 3 out of 26 cases, the proactive technique was 1.5 times better than the FCFS in terms of latency.

## 4.5 Impact of Tunable Parameters

We now discuss the impact of two tunable parameters on the trade-off between solution quality and computation time.

**Impact of window size ($\omega$):** We first discuss the impact of window size in case of window-based optimal algorithm. We compared the solution quality (i.e., latency) of the window based optimal algorithm for different window sizes with the optimal algorithm. Figure 3(a) shows the latency and overhead due to the computation time of the algorithm across different window sizes for five different datasets (i.e. Cases 1-5), where each dataset represents a different set of parameter configurations. As evident from Figure 3(a), on average, with the increase in the size of window, the solution quality improved. This is expected, as with an increase in the window size, in general the window-based optimal algorithm gradually approaches the optimal solution.

**Impact of lookaheads ($K$):** We now discuss the impact of $K$ in the case of the $K$-look-ahead FCFS algorithm. We compared the solution quality (i.e., latency) of

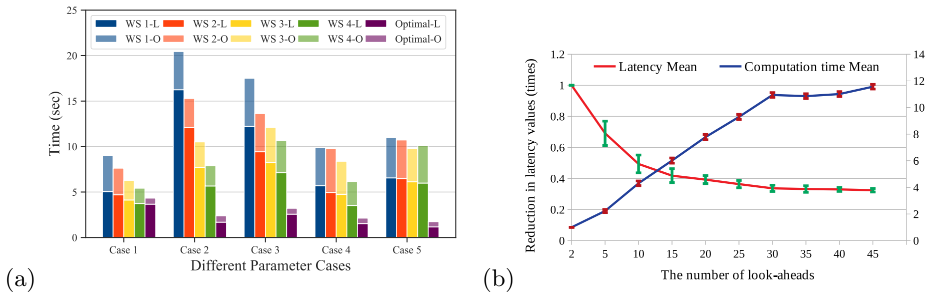


Fig. 3: Trade-off between computation delay and latency for (a) window-based optimal algorithm; (b) FCFS with $K$-look ahead. For (a), the darker colors (labels ending with a suffix '-L') denote latency, while the lighter colors (labels ending with a suffix '-O') denote overhead.

the $K$-look-ahead FCFS algorithm for a specific value of $K$ with the 1-look-ahead FCFS algorithm. Figure 3(b) shows the means and standard deviations of latency improvement and computation time degradation. While the latency improvement was calculated as the ratio between the latency obtained by 1-look-ahead FCFS and $K$-look-ahead FCFS, the computation time degradation was determined by the ratio between the time taken by $K$-look-ahead FCFS and 1-look-ahead FCFS. As evident from Figure 3(b), with the increase in the value of $K$, the solution quality monotonically improved, and computation time degraded. However, we note that after a certain limit, the solution quality did not improve significantly with an increase in the value of $K$. The trade-off between the solution quality and the computation time gets captured by the value of $K$.

In summary, FCFS provides a good balance between the solution quality (i.e., latency value) and computation time. The solution quality can be improved further at the cost of computation time using $K$-look-ahead FCFS.

## 5 Related Work

The rise of low-latency applications for the Internet of Things has made it necessary to utilize edge devices, instead of depending only on cloud services [2, 7, 14]. Multiple studies have appeared in the literature about providing such low-latency services. The first category deals with service placement in an edge-cloud environment, whereas the second category handles service requests for users of vehicles.

**Service Placement in Edge-Cloud Environment:** The problem of service placement in edge-cloud environment has received attention recently [11, 16, 19, 23]. One of the earliest solutions to the service placement problem was proposed in [16], where the authors first provided an (IoT) model along with the Quality of Service (QoS) requirement of the services and formulates Fog Service Placement Problem (FSPP) based on QoS requirements. In [4, 6, 15, 19, 23], the authors model the application placement problems and then propose a solution based on the different changing dynamics of the network and requests. None of these studies focus on user mobility. Our work builds on these ideas to propose an algorithm that considers the mobility of users.

**Edge Service for Vehicular Users:** The authors in [24] identified the requirement of the mobility problem, highlighted the advantages of mobility and discovered open challenges in this direction. In [1] and [25], the authors considered an application with multiple components to be placed on the set of edge devices across multiple timestamps for a moving user. Finally, [3] utilized a simulation tool to benchmark the performance of various algorithms. In contrast, our objective is to minimize the overall latency while multiple mobile users access different services at various points of time of their journey.

A number of works also consider optimizing service placement for moving user devices [3, 9, 10, 21, 22]. Mobmig [12] focused on solving the service placement problem in the context of edge users from moving vehicles by looking at the direction of its movement. However, its primary focus is on load balancing, and not on minimizing overall latency. References [21] and [22] model the problem of service placement as an Markov Decision Process (MDP). Unlike our work, these analytical model do not consider multiple users and multiple services to reduce the complexity of their model. Reference [10] utilized Thompson Sampling to handle the uncertainties inherent in placing services on edge clouds. However, it considers the response time of only a single user at a time, and considers a much simpler service model without considering the diversity of data requirements for different services. Moreover, these studies [9, 10] do not consider the dimension of memory requirement and availability, preferring to focus only on optimizing latency. In contrast, our work focuses on optimizing service latency while adhering to the memory constraint imposed by edge devices.

## 6 Conclusion

In this paper, we study the dynamic service placement problem in a distributed edge-cloud environment with emphasis on user mobility. We first model the problem and propose an optimal solution to this. To address the scalability issue, we further propose a heuristic algorithm considering FCFS scheduling. The experimental results on real and synthetic datasets show the effectiveness of our proposal. One limitation of this work is the assumption of having prior knowledge of the service invocation logs. In the future, we will utilize techniques shown by prior studies to predict services invoked to relax this assumption.

## 7 Acknowledgment

We would like to acknowledge Dr. Ansuman Banerjee, Indian Statistical Institute and Dr. Nanjangud C Narendra, Ericsson Research Bangalore for their initial discussions on this project.

## References

1. Bahreini, T., Grosu, D.: Efficient placement of multi-component applications in edge computing systems. In: ACM/IEEE Symposium on Edge Computing. p. 5. ACM (2017)
2. Bhattcharya, A., De, P.: Computation offloading from mobile devices: Can edge devices perform better than the cloud? In: ARMS-CC Workshop. p. 1–6 (2016)
3. Deng, S., Huang, L., Taheri, J., Yin, J., Zhou, M., Zomaya, A.Y.: Mobility-aware service composition in mobile communities. IEEE TSMC: Systems **47**(3), 555–568 (March 2017). https://doi.org/10.1109/TSMC.2016.2521736
4. Farhadi, V., et al.: Service placement and request scheduling for data-intensive applications in edge clouds. In: IEEE INFOCOM. pp. 1279–1287 (2019)
5. Gurobi Optimization, L.: Gurobi optimizer reference manual (2019), `http://www.gurobi.com`
6. He, T., et al.: It's hard to share: joint service placement and request scheduling in edge clouds with sharable and non-sharable resources. In: IEEE ICDCS. pp. 365–375 (2018)

7. Lin, L., et al.: Computation offloading toward edge computing. Proceedings of the IEEE **107**(8), 1584–1607 (Aug 2019). https://doi.org/10.1109/JPROC.2019.2922285
8. MacQueen, J., et al.: Some methods for classification and analysis of multivariate observations. In: Fifth Berkeley symposium on mathematical statistics and probability. vol. 1, pp. 281–297 (1967)
9. Ouyang, T., Zhou, Z., Chen, X.: Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing. IEEE Journal on Selected Areas in Communications **36**(10), 2333–2345 (Oct 2018). https://doi.org/10.1109/JSAC.2018.2869954
10. Ouyang, T., Li, R., Chen, X., Zhou, Z., Tang, X.: Adaptive user-managed service placement for mobile edge computing: An online learning approach. In: IEEE INFOCOM. pp. 1468–1476. IEEE (2019)
11. Pasteris, S., Wang, S., Herbster, M., He, T.: Service placement with provable guarantees in heterogeneous edge computing systems. In: IEEE INFOCOM. pp. 514–522 (2019)
12. Peng, Q., et al.: Mobility-aware and migration-enabled online edge user allocation in mobile edge computing. In: IEEE ICWS. pp. 91–98 (July 2019). https://doi.org/10.1109/ICWS.2019.00026
13. Rausch, T., Avasalcai, C., Dustdar, S.: Portable energy-aware cluster-based edge computers. In: 2018 IEEE/ACM Symposium on Edge Computing (SEC). pp. 260–272 (Oct 2018). https://doi.org/10.1109/SEC.2018.00026
14. Rejiba, Z., Masip-Bruin, X., Marín-Tordera, E.: A survey on mobility-induced service migration in the fog, edge, and related computing paradigms. ACM Comput. Surv. **52**(5), 90:1–90:33 (2019)
15. Selimi, M., et al.: Practical service placement approach for microservices architecture. In: IEEE/ACM CCGRID. pp. 401–410 (2017)
16. Skarlat, O., Nardelli, M., Schulte, S., Dustdar, S.: Towards qos-aware fog service placement. In: IEEE ICFEC. pp. 89–96 (2017)
17. Tong, L., Li, Y., Gao, W.: A hierarchical edge cloud architecture for mobile computing. In: IEEE INFOCOM. pp. 1–9 (2016)
18. Tran, T.X., et al.: Collaborative mobile edge computing in 5g networks: New paradigms, scenarios, and challenges. IEEE Communications Magazine **55**(4), 54–61 (2017)
19. Urgaonkar, R., Wang, S., He, T., Zafer, M., Chan, K., Leung, K.K.: Dynamic service migration and workload scheduling in edge-clouds. Performance Evaluation **91**, 205–228 (2015)
20. Van Brummelen, G.: Heavenly mathematics: The forgotten art of spherical trigonometry. Princeton University Press (2012)
21. 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 Transactions on Mobile Computing pp. 1–1 (2019)
22. Wang, S., Urgaonkar, R., Zafer, M., He, T., Chan, K., Leung, K.K.: Dynamic service migration in mobile edge computing based on markov decision process. IEEE/ACM Transactions on Networking **27**(3), 1272–1288 (June 2019). https://doi.org/10.1109/TNET.2019.2916577
23. Wang, S., Zafer, M., Leung, K.K.: Online placement of multi-component applications in edge computing environments. vol. 5, pp. 2514–2533. IEEE Access (2017)
24. Waqas, M., Niu, Y., Ahmed, M., Li, Y., Jin, D., Han, Z.: Mobility-aware fog computing in dynamic environments: Understandings and implementation. IEEE Access **7**, 38867–38879 (2018)
25. Zhao, H., Deng, S., Zhang, C., Du, W., He, Q., Yin, J.: A mobility-aware cross-edge computation offloading framework for partitionable applications. In: IEEE ICWS. pp. 193–200. IEEE (2019)