MoDEMS: Optimizing Edge Computing Migrations for User Mobility

Taejin Kim*, Sandesh Dhawaskar Sathyanarayana†, Siqi Chen†, Youngbin Im‡, Xiaoxi Zhang§,
Sangtae Ha† and Carlee Joe-Wong*
*Carnegie Mellon University, †University of Colorado Boulder, ‡UNIST, §Sun Yat-sen University
{tkim2, cjoewong}@andrew.cmu.edu, {sadh0344, siqi.chen, sangtae.ha}@colorado.edu, ybim@unist.ac.kr,
zhangxx89@mail.sysu.edu.cn

***Abstract*—Edge computing capabilities in 5G wireless networks promise to benefit mobile users: computing tasks can be offloaded from user devices to nearby edge servers, reducing users' experienced latencies. Few works have addressed how this offloading should handle long-term user mobility: as devices move, they will need to offload to different edge servers, which may require migrating data or state information from one edge server to another. In this paper, we introduce MoDEMS, a system model and architecture that provides a rigorous theoretical framework and studies the challenges of such migrations to minimize the service provider cost and user latency. We show that this cost minimization problem can be expressed as an integer linear programming problem, which is hard to solve due to resource constraints at the servers and unknown user mobility patterns. We show that finding the optimal migration plan is in general NP-hard, and we propose alternative heuristic solution algorithms that perform well in both theory and practice. We finally validate our results with real user mobility traces, ns-3 simulations, and an LTE testbed experiment. Migrations reduce the latency experienced by users of edge applications by 33% compared to previously proposed migration approaches.**

I. Introduction

Cloud computing is a popular way to access computing resources and generated 20.6 Zettabytes of traffic in 2021, compared to 6.8 Zettabytes in 2016 [1]. While it offers flexible and scalable access to plentiful resources, sending data to and from remote cloud servers may incur unacceptably high latencies. For example, augmented reality (AR) on mobile devices requires remote computing for compute-intensive tasks [2] that must be completed quickly.

The ongoing deployment of 5G technologies will soon allow cellular service providers to offer low-latency edge computing services. By separating the user plane from the control plane function, a 5G network now explicitly supports edge services by using a unified gateway called UPF (User Plane Function) [3] that can be integrated with MEC (Multi-access Edge Computing) [4] near or co-located with a 5G base station distributedly [5], [6]. Such systems have been proposed to reduce latency and bandwidth consumption in offloading computations from mobile devices to nearby servers [7]–[10] and reducing mobile device battery consumption [11]. A simple diagram of an edge computing system is shown in Figure 1. The mobile user in the vehicle offloads computations (e.g., inferences from large machine learning models) to the nearest edge server, which is geographically closer than the cloud server. However, to maintain their geographical advantage over cloud servers, multiple edge servers should be deployed across the service region, like the servers in Figure 1.

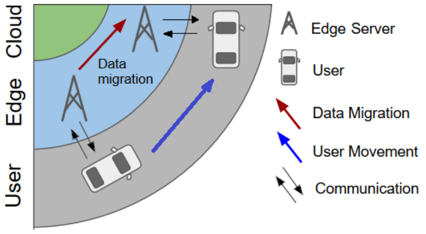


Fig. 1: A mobile user's task is migrated between edge servers.

A. Challenges: Edge Computing and Migrations

Despite latency benefits, deploying edge computing raises research challenges. Edge servers generally have fewer available resources and higher operating costs than cloud servers [7]. Edge computing solutions must also handle user mobility [12]–[14]: as users move, their applications should be serviced at a different and closer edge server to minimize latency. However, migrating an application's virtual machine (VM) or container from one edge server to another utilizes potentially limited bandwidth on links between the edge servers, and occupies resources on *both* the source and destination servers since applications need to maintain an active edge server connection during migration. Finding the optimal placement that balances these constraints with user needs is then difficult: user applications should coordinate to avoid overwhelming edge server and link capacities.

Placing applications on edge servers becomes even more difficult when user mobility is not fully known: application migration takes time, making reactive solutions that migrate only after a user has moved ineffective [12]–[14], and a misplaced migration may incur a higher cost than no migration at all. Users with different mobility patterns, e.g., car drivers versus pedestrians, may then need different migration plans, greatly enlarging our plan search space since users should also coordinate to respect capacity constraints. Machine learning algorithms embedded in 5G networks may predict short-term user movement [15], [16], but long-term patterns may not be known in advance. Simple solutions to alleviate uncertain mobility predictions, such as continuously replicating applications across multiple servers, require prohibitive amounts of edge resources. We address these challenges.

B. MoDEMS: Mobility Dependent Edge Migration System

In this paper, we propose MoDEMS (Mobility Dependent Edge Migration System) for generating migration plans. In more detail, our **technical contributions** are as follows:

- We design the first system that optimizes edge computing deployments using a *rigorous theoretical framework* to jointly address practical deployment challenges of resource constraints, mobility uncertainty, concurrent migrations for multiple users, and implementation overhead.
- We formulate a *linear integer optimization problem* to optimize latency and resource costs given diverse user-specific mobility models. We show the problem is NP-hard due to the concurrent service of users despite resource constraints. We propose a distributed heuristic that optimizes over user-specific mobility and intelligently minimizes users' coordination across different resources, showing analytically that it performs and scales well.
- We perform *extensive experiments* to evaluate the linear integer optimization solution and variations of our proposed heuristic on real user mobility traces [17], `ns-3` simulations, and an LTE testbed. The heuristic outperforms previous methods that do not consider mobility prediction or resource constraints by 20 to 35% .

The remainder of this paper is organized as follows. Section II contrasts MoDEMS with related works. Section III presents the system model and MoDEMS architecture. We then show in Section IV that the MoDEMS model allows us to formulate a linear integer optimization problem for edge migrations. Section V analyzes the complexity of this problem and theoretically examines our proposed scalable heuristic solution. Finally, we *experimentally validate our work* compared to prior approaches in Section VI. We conclude in Section VII. All proofs are provided in our technical report [18].

II. Related Work

Virtual machine (VM) migration is a major research challenge in edge computing [14], [19], [20]. Prior works of [21] and [22] develop integer programming problems similar to ours. However, these works propose reactionary migration policies that do not utilize user mobility predictions [21]–[25]. Other works have proposed dynamic policies given unknown costs of migration that follow Markovian user mobility patterns [26] or use Markov decision processes [25], [27]. However, such works generally do not consider resource capacity constraints at edge servers, which may force applications onto the cloud if resources run out. They also do not predict individual user movement, imposing the same migration policy on all users at the same location. Our evaluation shows that MoDEMS significantly (by $> 20\%$) lowers costs by considering both these factors. Lyapunov optimization frameworks are used as well [28], [29], while [30] simultaneously allocates bandwidth and compute resources to different users.

As we show in Section V, resource constraints make the optimal migration problem considerably more difficult. Works accounting for resource constraints that allow VMs to concurrently run on multiple edge servers simplifies the optimal migration problem and may not be practical [31], [32]. Other works analyze the allocation of computation and bandwidth at an operating system level, dividing tasks between local devices and edge servers [24], [33]. In comparison, we present the first theoretical framework and algorithms that (i) consider resource constraints, long term migrations, and variation between individual users, (ii) validate its effectiveness with realistic network experiments and (iii) design a distributed method to navigate the larger search space that results from individualized mobility predictions instead of uniform ones.

Migration in edge computing has also been studied in the context of complex event processing (CEP) applications [34], [35], in which streams of information from multiple mobile sources must be jointly analyzed at an edge server. However, that work does not use formal mobility models or optimization.

Network function virtualization (NFV) and software-defined networks (SDN) similarly aim to deploy service chains inside VMs hosted on geographically advantageous edge nodes to decrease bandwidth usage and latency [36], [37]. However, the middleboxes do not migrate according to user mobility.

III. MoDEMS System Model and Architecture

This section builds a system model that depicts the physical system of edge nodes and users (Section III-A). We formalize data migrations within this model in Section III-B.

A. Physical Model

We consider an edge computing service provider with multiple edge servers, e.g., located at mobile base stations. The provider has S servers to service U users, each with one process request, who can offload computation tasks to any of the edge servers by opening a personal VM or container on that server (our framework can model either), each of which serves one process at a time. Although we associate each user with a single process (e.g., one mobile application), our model can be easily extended to multiple processes per user.

We let $[X]$ represent the set $\{1, 2, ..., X\}$. We consider discrete time steps $t \in [T]$. While our model is agnostic to the exact time step length, in practice, the time steps would likely last a few minutes. At this granularity, we can meaningfully represent user movement around a city via their locations at different times, while ensuring that migrations complete within a single time step. For simplicity and following prior work [38], all users enter the system at time $t = 1$ and exit at $t = T$, though their requests may start after $t = 1$ and end before $t = T$. Table I summarizes our notation.

Servers and links. We suppose that the servers are dispersed across a given geographical region. Each server s has its resource capacity defined by the vector R_s, which may include CPU, RAM, and storage provided to process VMs.

We assume that servers will have wired connections with one another following a given network topology. For example,

R_s	$n \times 1$ vector for capacities of n resource types at server s	$\overrightarrow{c_s}$	$n \times 1$ vector of each unit resource cost at server s
$h^{u,t_1,t_2}_{s_1,s_2}$	Decision variable returning 1 for migration for user u between time steps t_1 and t_2 from server s_1 to s_2 and 0 o.w	$\overrightarrow{w_u}$	$n \times 1$ vector representing the amount of resources required of each type for user u's process
$q^t_{u,s}$	Binary indicator variable that returns 1 if the process for user u is at server s at time t and 0 otherwise	ϵ_u	Migration amount for the VM of user u in terms of total bandwidth consumption
$j^{u,t}_{s_1,s_2}$	Migration rate fraction ($[0,1]$) that returns the portion of the VM migrated for user u at time t from server s_1 to s_2	W_u	Consumption of service bandwidth per unit time step between user and process VM for user u
$g^t_{u,s}$	Variable that returns 1 if a VM migration is taking place to server s at time t for user u and 0 otherwise	Z_{s_1,s_2}	Cost of using unit amount of bandwidth for single time step between servers s_1 and s_2
B	$S \times S$ vector of bandwidth capacity between servers	$Y^{u,t}_r$	Actual latency experienced by user u at time t
$i^t_{u,s}$	Binary indicator of 1 if user u at server s at time t and 0 o.w.	Y^u_x	Maximum latency limit for user u
$P[i^t_{u,s}]$	Probability ([0,1]) of user u being at server s at time t	D^u_Y	Monetary value of latency violation per unit for user u

TABLE I: Physical system variables (left), cost and migration graph variables (right).

edge servers may be connected to regional controllers in a hierarchical topology or with direct-wired links to each other [34]. For simplicity in presentation and analysis, we assume that all servers are connected directly with one another. All communication between any two servers (migrations and service) occurs only through the link directly connecting the two servers. Our system set up and analysis can easily extend further to scenarios with more realistic topologies as well.

User mobility. If we know how users will move over time, or if we are only concerned with past locations, we use the variable $\{i^t_{u,s};\, t = 1, 2, \ldots, T\}$, a binary indicator of whether user u is closest to server s at time t. If user mobility is being predicted for future time steps and not known with full confidence, we set the variable $i^t_{u,s} = P[i^t_{u,s}]$ to indicate the probability that user u is at server s at time t; thus $P[i^t_{u,s}]$ is a continuous variable in $[0, 1]$. This model is quite general and can represent individual user movement that follows a range of typical mobility models, e.g., Markovian mobility as in [38].

User service. We use $q^t_{u,s}$ to indicate at which server a VM for user u is located. Note that this is not necessarily the closest server to the user u's location. Users at any location in the region will always connect to the geographically closest server, captured by the variable $i^t_{u,s}$ and predicted by $P[i^t_{u,s}]$. From there, the user will pull data as needed from the VM at the server indicated by $q^t_{u,s}$ over the network backbone.

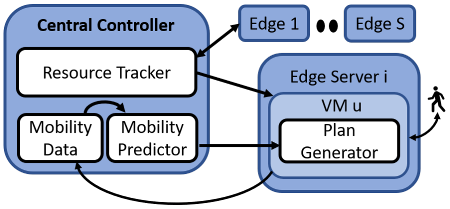


Fig. 2: Flowchart of the distributed approach on solving the edge computing migration problem via MoDEMS.

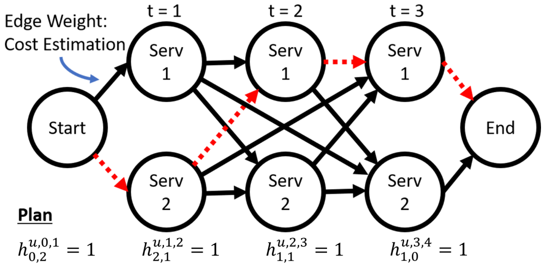


Fig. 3: Simple migration graph used to capture physical model and cost of edge computing system. A migration plan is extracted from the dotted path in red. The migration plan is also described in terms of the optimization variable $h^{u,t1,t2}_{s_1,s_2}$.

B. MoDEMS Architecture

System Modules. Figure 2 displays a flow diagram of MoDEMS' system modules, implemented in a distributed manner. The process begins with the user spawning a VM at the closest available edge server. The central controller, e.g., the 5G Radio Intelligent Controller [15] potentially at a cloud, coordinates access to multiple 5G base stations, each equipped with an edge server. Using the *resource tracker* and *mobility data*, the controller gathers compute and bandwidth resource availability as well as user movement patterns by communicating with the edge servers. The *mobility predictor* uses the stored mobility data to generate probabilistic predictions of future movements for specific users. This information is then sent to a *plan generator* at each user's VM, where it is used to generate migration plans. In a centralized MoDEMS system, the plan generator is instead placed at the central controller. Our goal is to develop an effective and efficient plan generator, which can be understood as optimizing over a *migration graph*, as we discuss below and in Section IV.

Migration Plan Generation. The *migration graph* data structure visualizes a process's migration decisions and associated costs [34], [39]. It has a start and end node before the first time step and after the last time step respectively. For notation, we set the server index for both the start and end nodes as $s = 0$. We define the other nodes in the migration graph as server-time pairs, representing the possible placements of a VM at every time step. An edge between the vertices (s_1, t_1) and (s_2, t_2), $s_1 \neq s_2$, represents a migration from server s_1 to server s_2 that starts at time t_1 and ends at time $t_2 > t_1$. During a migration, the process remains at the source server, while a copy VM is built at the destination. Unlike prior work [26], [38], [40], we may have $t_2 > t_1 + 1$, i.e., migrations may take place over multiple time steps. This ability may allow more migrations to take place if there is limited bandwidth to migrate. If $s_1 = s_2$, the edge represents simply staying at server s_1. Such edges only last one time step since long-term edges are redundant. We associate each edge with a weight that represents the per-unit costs of taking that path on the migration graph, as defined in Section IV. Figure 3 depicts a migration graph over three time steps in a two-server system.

For each VM of user u, we define a feasible **migration plan** as any path, or continguous sequence of edges, from the start node to the end node in the migration graph. Thus, VMs can only migrate to one server at any time. We represent a migration plan for user u's VM by defining the indicator h_{s_1,s_2}^{u,t_1,t_2} as equal to 1 if the path from server s_1 to s_2 and from time t_1 to t_2 is included in the migration plan, e.g., the dashed migration plan in Figure 3, and 0 otherwise.

We ensure the feasibility of the chosen migration plan by constraining $\left\{h_{s_1,s_2}^{u,t_1,t_2}\right\}$ to ensure that (i) every entry to a node on a migration graph must have a departure, and (ii) a migration plan leaves the start node exactly once while arriving at the end node exactly once. In terms of notation, $x \in ([X] > x_o)$ indicates the set $\{x_o+1, x_o+2, ..., X\}$, with analogous definitions for $x \in ([X] \leq x_o)$ and $x \in ([X] \neq x_o)$. Furthermore, $s \in \{0, [S]\}$ represents a set $\{0, 1, ..., S\}$ to include the server index for the start and end node. Formally:

$$\begin{gathered}\sum_{t_1 \in ([T]<t)} \sum_{s_1 \in (0,[S])} h_{s_1,s}^{u,t_1,t} = \sum_{t_2 \in ([T+1]>t)} \sum_{s_2 \in (0,[S])} h_{s,s_2}^{u,t,t_2} \\ \sum_{s \in [S]} h_{0,s}^{u,0,1} = \sum_{s \in [S]} h_{s,0,1}^{u,T,T+1} = 1.\end{gathered} \tag{1}$$

To ensure that edges that do not exist cannot be taken, we constrain $h_{s_1,s_2}^{u,t_1,t_2} \leq H_{s_1,s_2}^{u,t_1,t_2}$. H_{s_1,s_2}^{u,t_1,t_2} returns 1 if an edge from s_1 to s_2 and from t_1 to t_2 is viable, and 0 otherwise.

We can now define $q_{u,s}^t$ in terms of the migration plan variables h_{s_1,s_2}^{u,t_1,t_2}. If a process for user u is being migrated from s_1 to s_2, it is serviced by s_1 until the migration finishes.

$$\begin{aligned}q_{u,s}^t = &\sum_{s_1 \in [S]} \sum_{t_1 \in ([T]<t)} \sum_{t_2 \in ([T] \geq t)} h_{s_1,s}^{u,t_1,t_2} \\ - &\sum_{s_2 \in (0,[S])} \sum_{t_3 \in ([T]<t)} \sum_{t_4 \in ([T] \geq t)} h_{s,s_2}^{u,t_3,t_4}\end{aligned} \tag{2}$$

We can then define the variable $j_{s_1,s_2}^{u,t} \in [0,1]$ as the rate of transfer at time t during a migration from s_1 to s_2. For example, if $h_{s_1,s_2}^{u,t,t+1} = 1$, then $j_{s_1,s_2}^{u,t} = 1$ as the entirety of the process has been migrated between time t and $t+1$.

$$j_{s_1,s_2}^{u,t} = \sum_{t_1 \in ([T] \leq t)} \sum_{t_2 \in ([T]>t)} \frac{h_{s_1,s_2}^{u,t_1,t_2}}{t_2 - t_1} \tag{3}$$

For convenience, we also define $g_{u,s}^t$ to equal 1 if the process of user u at time t is in the midst of a migration to server s, and 0 otherwise. Thus, $g_{u,s}^t = 1$ if and only if $h_{s_1,s}^{u,t_1,t_2} = 1$ for some server $s_1 \neq s$, and time $t_1 \leq t < t_2$:

$$g_{u,s}^t = \sum_{s_1 \in ([S] \neq s)} \sum_{t_1 \in ([T] \leq t)} \sum_{t_2 \in ([T]>t)} h_{s_1,s}^{u,t_1,t_2} \tag{4}$$

IV. Optimization Problem Formulation

In this section, we show that finding the migration plan in the centralized MoDEMS scenario can be formulated as a linear integer program. We consider two types of costs: the *operational cost*, which is incurred by the service provider of operating the edge servers and links; and the *user dissatisfaction cost*, the compensation required when the service provider cannot meet users' quality-of-service (QoS) requirements, e.g., if it incurs high latencies. This compensation may be enforced via service level agreements between users and the edge provider. In order to formulate the problem, we assume known arrival and departure times of all processes, which we relax in Section V. Table I summarizes our notation.

Operational Cost. The operation cost includes both *placement* and *bandwidth* cost. The placement cost incurred for a single user during a single time step is the sum of the costs of using each type of resource at an edge server s. With $\overrightarrow{c}_s$ and $\overrightarrow{w_u}$ as the cost and demand vectors of each resource at s for user u respectively, the placement cost is:

$$C_P = \sum_{t \in [T]} \sum_{u \in [U]} \sum_{s \in [S]} (g_{u,s}^t + q_{u,s}^t)(\overrightarrow{w}_u^\intercal \overrightarrow{c}_s), \tag{5}$$

since user u utilizes resources at s both when it is located at s $\left(q_{u,s}^t = 1\right)$ and in the process of migrating to s $\left(g_{u,s}^t = 1\right)$.

The *bandwidth* cost includes the cost of migrations, C_{B_m}, and the use of network bandwidth to service processes, C_{B_s} (e.g., for conveying the results of edge server computations to the user device). The amount of bandwidth used for migration, B_m, is then defined as $\sum_{t \in [T]} \sum_{u \in [U]} B_m^{u,t}$ where: $B_m^{u,t} = \epsilon_u \sum_{s_1 \in [S]} \sum_{s_2 \in ([S] \neq s_1)} j_{s_1,s_2}^{u,t}$, i.e., the migration size ϵ_u multiplied by the rate of migration $\left(j_{s_1,s_2}^{u,t}\right)$. The cost of bandwidth used for migrations is $C_{B_m} = \sum_{t \in [T]} \sum_{u \in [U]} C_{B_m}^{u,t}$, where $C_{B_m}^{u,t}$ is the sum of each term in $B_m^{u,t}$ multiplied by the link cost Z_{s_1,s_2}.

Similarly, the amount of bandwidth used for service B_s is defined as $\sum_{t \in [T]} \sum_{u \in [U]} B_s^{u,t}$ where: $B_s^{u,t} = W_u \sum_{s_1 \in [S]} \sum_{s_2 \in ([S] \neq s_1)} i_{s_1}^{u,t} q_{s_2}^{u,t}$. Here, W_u is the throughput of the service bandwidth demanded by u that travels to and from the user $\left(i_{s_1}^{u,t}\right)$ and VM server $\left(q_{s_2}^{u,t}\right)$. The resulting cost is $C_{B_s} = \sum_{t \in [T]} \sum_{u \in [U]} C_{B_s}^{u,t}$, where $C_{B_s}^{u,t}$ is the sum of the terms of $B_s^{u,t}$ multiplied by the link usage cost Z_{s_1,s_2}. When the bandwidth is owned and allocated by the operator [30], it is possible to omit the costs related to bandwidth.

User Dissatisfaction Cost. We measure user dissatisfaction by the *latency* of user's experienced service, i.e., the time of communication between the user and the server hosting its process. We suppose that each user u specifies a threshold of maximum latency Y_x^u, with an increasing user dissatisfaction cost at latency above Y_x^u. For example, this cost may represent a user's future unwillingness to use the edge system or monetary compensation for the system being unable to provide the specified maximum latency. VR and AR applications, for instance, may have such costs when the latency rises above a user perception threshold [2]. The latency violation cost $C_Y = \sum_{t \in [T]} \sum_{u \in [U]} C_Y^{u,t}$ is defined as: $C_Y^{u,t} = \max(0, Y_r^{u,t} - Y_x^u) D_Y^u$. The value $Y_r^{u,t}$ is the actual latency experienced by user u at time step t, which depends on the physical distance between the VM server and the server to which a user connects: $Y_r^{u,t} = \sum_{s_1 \in [S]} \sum_{s_2 \in ([S] \neq s_1)} L(s_1, s_2) i_{s_1}^{u,t} q_{s_2}^{u,t}$. The value of $L(s_1, s_2)$ represents the latency incurred over a direct link between servers s_1 and s_2. The constant D_Y^u represents the monetary value of the usability the user has lost.

Formulation. Given the placement, bandwidth, and user dissatisfaction costs, we wish to solve the problem:

$$\min_{h \in H} C_{total} = C_P + C_{B_m} + C_{B_s} + C_Y \qquad (6)$$

$$\text{s.t. (1)}, \sum_{u \in [U]} (g_{u,s}^t + q_{u,s}^t)(\vec{w}_u) \leq R_s \forall s,\ B_m + B_s \leq B$$

where we have imposed server and link capacity constraints.

V. Solving the Optimal Migration Problem

In this section, we discuss solution algorithms for problem (6). We show that the problem is NP-hard and propose heuristic solution algorithms that have only quadratic complexity. We then analyze our algorithms' ability to handle the challenges of resource constraints and unknown user mobility.

A. Complexity Analysis

We begin by establishing the complexity of the migration graph that we define in Section III-B.

Proposition 1 (Number of migration paths)**.** *The number of migration paths for a process grows at least as fast as* $O(S^T)$.

Proof: For each $t \in [T]$, there are S possible locations for a given process. Since a migration plan must place the process at a server at each time step, the result follows. ■

As we might expect from Proposition 1, the optimization problem of finding the best path for each process is NP-hard:

Proposition 2 (NP-hardness)**.** *Solving (6) is NP-hard.*

Proof: The generalized assignment problem, which is NP-hard [41], is a special case of (6). ■

Despite the exponential growth of the migration graph with respect to the number of time steps, the proof of NP-hardness considers only a single time step. The main difficulty in solving this problem arises from the need to *concurrently determine migration plans for multiple users*.

Choosing the migration plan for a single user in isolation is similar to finding the shortest path through the migration graph, which is solvable in polynomial time [42]. This intuition informs our proposed solution heuristic, Seq-Greedy.

B. Seq-Greedy Solution Heuristic

Our proposed *Seq-Greedy* method can be run in either a centralized or distributed (Figure 2) MoDEMS deployment. The distributed Seq-Greedy method begins by generating a migration graph for each process at the edge server that is available and initially closest to the user. Once migration graphs are created for all processes in the system, migration plans are generated by optimizing over the migration graph. Unlike the linear integer programming approach, the Seq-Greedy approach generates one migration plan at a time, and thus does not require knowledge of the arrival and departure times of processes in the system. The shortest path is found along the migration graph and is set as the temporary migration plan. The plan is then broadcast to the central controller. If there are enough computation and bandwidth resources to support it, the necessary resources are reserved by the central controller. Otherwise, the migration graph is edited to remove the nodes and edges with no resources and the sequence is run again. Unlike the static optimization approach, Seq-Greedy is a **dynamic** algorithm that solves for jobs arriving in real time. This method is much more scalable than outright solving the migration optimization problem:

Proposition 3 (Complexity of Seq-Greedy)**.** *The number of edges and vertices in the migration graph grows as* $O\left(S^2T^2\right)$ *for large numbers of servers and time steps. The complexity is equal to* $O\left(D(S^2T^2)\right)$*, where* $D(x)$ *denotes the complexity of finding a shortest path in a graph with* x *vertices.*

This result implies that generating the migration graph incurs a cost that grows quadratically with the numbers of servers and time steps. Finding a migration plan given the migration graph, on the other hand, is simply equivalent to finding the shortest path for each process in sequence. Djikstra's algorithm, for instance, runs in $O\left(S^4T^4\right)$ time [42]. We further note that the shortest path algorithms can be implemented *distributedly* across the different vertices. Thus, the edge servers can solve the migration plan with only the resource information from the cloud controller. This property is particularly useful when users cross from one controller domain to another, as it removes the need for complex handoff mechanisms. The distributed method also leverages existing computation resources at the edge to compute migration plans.

Next, we introduce in Algorithm 1 the **batch method** that is designed to handle stochastic and individualized user movement. In lines 4-15, the batch method defines a series of time windows and creates one migration graph for each window. It iteratively generates plans for all users using Seq-Greedy for each window and executes the plan. We expect this approach will yield better plans, as the user mobility predictions are improved by conditioning on the user's location at the end of each time window, as seen in line 5. Although Algorithm 1 assumes that all users have equal sized batches generated at the same time step, variable window lengths across users are possible. Like Seq-Greedy, the batch method is also a **dynamic** algorithm that further reduces the time horizon for which the migration problem is solved. Although the batch method makes long-term migrations spanning time lengths greater than the windows impossible, it limits the complexity of the Seq-Greedy method by limiting the number of time slots considered within time window.

We note that **reducing the number of servers** provides an equivalent reduction in Seq-Greedy's complexity as reducing the number of time steps (Proposition 3). Indeed, if resources are not too constrained, then a server far away from a user is unlikely to be optimal due to high latency costs. Since Seq-Greedy considers each user individually, we can remove servers from migration graphs depending on individual users' movement predictions. We introduce a new parameter γ such

Algorithm 1: Batch method for all users in the system

```
Upon trigger batch_plan(jobs, users, num_batch);
batch_id ← 1;
while batch_id ≤ num_batch do
    for u ∈ users do
        usr_MC(u) ← update_user_MC(u);
        mig_graph(u) ←
          make_batch_graph(usr_MC(u), jobs(u));
        check(u) ← false;
        while check is false do
            mig_plan(u) ← make_plan(mig_graph(u));
            check(u) ← reserve_resource(mig_plan(u));
            mig_graph(u) ← update_resource()
        end
    end
    execute(mig_plan);
    batch_id ← batch_id + 1;
end
```

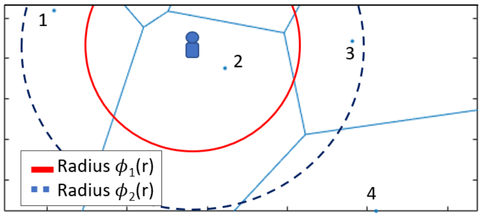


Fig. 4: The truncation method is used to disregard servers when generating the migration graph based on user movement distribution. We include servers that are within the user movement radius ($\phi_1(r)$), and those that are outside but have service areas overlap with the user movement area ($\phi_2(r)$).

that we only include servers s in the user's migration graph if the probability that a user lies in s's coverage area at time t is at least γ. We evaluate this truncation method's impact on the system complexity in terms of γ in Section VI.

Deciding which servers to truncate then lies in determining the probability that the user will lie in each server's coverage area at a given time. To do so, we define X_u^t as user u's location after t time steps, given a stochastic mobility model. Using [43]'s results on two-dimensional random walks, we can find the cumulative distribution function $F_{X_u^t}$ for the magnitude of travel for user u at time t. Thus, with probability γ, user u is located within a circle of radius $F^{-1}_{X_u^t}(\gamma)$.

We finally estimate the reduction in complexity due to truncation. Let $\phi(r)$ denote the expected number of servers whose coverage areas intersect a circle of radius r. As seen in Figure 4, if a server's coverage area intersects a circle of radius $r = F^{-1}_{X_u^t}(\gamma)$, then it must include at least one point on the circumference of this circle. We can then define the probability distribution of the expected distance from each point z to its closest server, $F_l = \min_{s\in[S]}\{\|l_s - z\|_2\}$ where l_s denotes the location of server s and is uniformly distributed over the region. Then with probability $F_l(\rho)$, all servers included in our truncation lie within a larger circle of radius $\rho+F^{-1}_{X_u^t}(\gamma)$ around the user's location, i.e., $\mathbb{E}\left[\phi\left(F^{-1}_{X_u^t}(\gamma)\right)\right] \leq \pi\left(F^{-1}_{X_u^t}(\gamma)+\rho\right)^2\frac{S}{A}$ with probability $F_d(\rho)$, if the servers are uniformly distributed throughout the service region. Here, A represents the total physical area of consideration. In Section VI, we show that taking $\gamma = 0.9$ leads to 25% fewer edges in the user migration graphs, significantly reducing Seq-Greedy's complexity with little increase in cost.

C. Optimality of Our Seq-Greedy Heuristic

We assess the optimality of our heuristic, focusing mainly on how the presence of resource constraints and unknown user mobility change the optimal migration plans and make finding the optimal plans more difficult.

Effect of resource constraints. We first consider an alternative method for solving (6): relaxing the integer linear program and rounding the solution to an integer solution:

Proposition 4 (Optimality of the relaxed problem)**.** *Suppose all bandwidth costs equal zero ($C_{B_s}+C_{B_m}=0$), and network resource constraints do not exist. Then if all processes have the same size $\vec{w}_u = \vec{w}$ at every time step and the server capacities R_s are integer multiples of $\vec{w}$, the optimal solution of (6) is the same as the optimal solution to the relaxed version of (6) where we let $h^{u,t_1,t_2}_{s_1,s_2} \in [0,1]$.*

Proof: We show that the solution to the relaxed problem is integral, and thus solves the original problem (6). The key step is to recognize that the cost of any fractional solution can be reduced by shifting processes between servers. ■

The assumption that all processes have the same size may hold if we consider a specific application from multiple users, e.g., small VMs that store machine learning models occasionally called by the application. In general, however, we may consider heterogeneous applications, as in Section VI's evaluation. Thus, we next analyze Seq-Greedy. Proposition 1 suggests that the edge servers' capacity constraints significantly contribute to the complexity of solving the optimization problem. We verify that intuition by showing that Seq-Greedy is optimal with no resource constraints:

Proposition 5 (Seq-Greedy optimality with sufficient resources)**.** *Given enough resources to serve all users simultaneously, i.e., $\sum_{u=1}^{U} 2\vec{w}_u \leq R_s; \forall s$, $\sum_{u=1}^{U} B_s^{u,t} + B_m^{u,t} \leq B; \forall t$, Seq-Greedy converges to the optimal solution of (6).*

Proof: The assumption of sufficient resources allows us to ignore the resource constraints; thus, (6) reduces to finding the minimum cost migration path for each process. Since the objective is additively separable across users, it decomposes into minimizing each user's cost, independent of the other users. This is exactly our heuristic. ■

When resource constraints are effective, we do not expect Seq-Greedy to generally find the optimal solution. However, we can show that it out-performs a baseline algorithm that does not take mobility into account:

Proposition 6 (Comparison with a naïve baseline)**.** *Suppose $S = 2$ and that $\vec{c}_1 = \vec{c}_2$. Then if Seq-Greedy finds migration paths for users in descending order of W_u, the resulting total cost is no greater than that incurred without migrations.*

Proof: It suffices to consider only those users whose server assignment deviates from the optimal migration path without constraints. The result follows on observing that the cost incurred in the timeslots with such a deviation is no larger than that incurred when all users remain stationary. ■

Thus, at least when there are few servers present, Seq-Greedy out-performs a naïve static baseline, for any number of users. The assumption that $S = 2$ is reasonable if users have limited mobility, e.g., among students who stay on a college campus; or if edge servers serve large areas, e.g., mini-datacenters serving city neighborhoods. We numerically show that this result still holds for $S > 2$ servers in Section VI.

Effect of movement uncertainty. We now consider our algorithm performance in the context of our second challenge, uncertain user mobility. While we might expect that a stochastic formulation would help the migration plan better track user movement, in some cases uncertainty can actually hurt:

Proposition 7 (Migrations with uncertain mobility)**.** *If user movements are Markovian, then for T sufficiently large there exists a time $t_u < T$ for each user u such that for $t \geq t_u$, the optimal migration plan does not migrate u's VM.*

Proof: The result follows from the convergence of the distribution of user locations to a steady state. ■

In essence, under a stochastic mobility model users' movement is eventually so uncertain that there is no value to migrating. Thus, even when Proposition 6's conditions hold and the optimal migration plan should outperform the stationary solution with known mobility, when mobility uncertainty is introduced into the model the stationary solution becomes optimal. Our batch method avoids this result by *re-optimizing* the migration every few time slots, and we show in Section VI that it indeed outperforms Seq-Greedy given stochastic mobility.

VI. Evaluation

In this section, we numerically evaluate MoDEMS, validating, and going beyond Section V's results. Specifically, we aim to show that we have solved the primary research challenges introduced in Section I: designing a *feasible* migration algorithm that (i) scales to realistic edge computing systems, (ii) respects the lack of resources at edge servers and links, and (iii) optimizes over uncertain user movement. After describing our experimental setup, we examine the achieved scalability and cost of Seq-Greedy and our proposed variants compared to baseline algorithms, under different resource constraints and mobility patterns. We finally evaluate the improvement in edge user experience with Seq-Greedy in a realistic network environment simulated by `ns-3` [44] and a LTE testbed.

A. Numerical Analysis Setup

We use synthetic server locations spread out evenly within an area of 5 miles by 5 miles. We consider edge servers with limited resources, aggregation servers with more resources, and a cloud server with high resources and latency. All edge servers are connected to the closest aggregation server, and all aggregation servers are connected to the cloud server. Initial locations of users are drawn from a uniform distribution and Markovian user movements are estimated from the Yonsei/Lifemap mobility dataset [17]. The size of the simulation space is set based on the area of downtown Seoul, South Korea (where the traces are from), an urban area typical of edge computing deployments [45]. Time steps are five minutes long unless otherwise stated.

To evaluate the effects of resource limitations, the simulations are run with either *limited* or *ample* resources. In the limited resource setting, resource capacities are drawn from uniform distributions such that edge servers on average can service 2.5 processes simultaneously and each link can migrate six processes in a single time step. Resource constraints do not affect migration decisions with ample resources. Servers provide three resources: CPU cores, RAM, and storage, with prices per five-minute time step of \$0.02 per CPU core, \$0.01 per GB of RAM, and \$0.02 per GB of storage, following current cloud prices [46]. Process sizes are chosen to simulate VR, AR, and personal assistant applications as measured in [47], [48]. To conserve space, we do not separately examine the effects of limited edge server and link capacity resources.

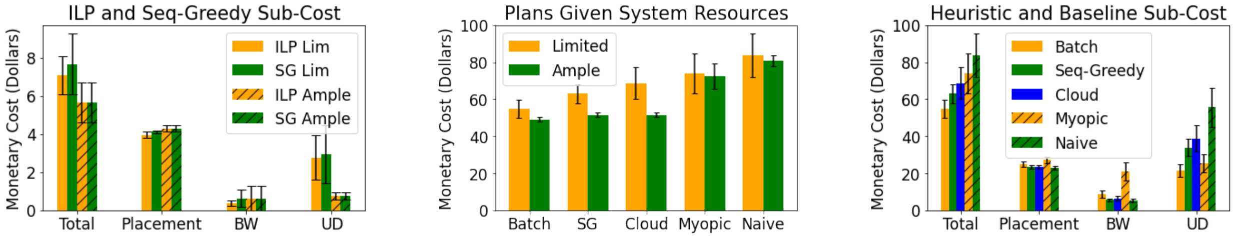
We compare our proposed **Seq-Greedy** approach and its batch and truncation extensions to the **optimization** approach and three baselines. The **naïve** approach minimizes the cost with no migrations while choosing the closest server available, as in SDN/NFV placement optimization [36], [37]. The **myopic** approach migrates processes to the closest feasible server at every time step, as in reactive migration frameworks [20]; this comparison shows the value of predicting individual user mobility. The **cloud** approach generates migrations that minimize user costs without considering resource constraints, as in [27]. The cloud then serves processes violating resource constraints, showing the value of accounting for these constraints in the optimization itself.

Unless otherwise stated, we show the average and standard deviation of results over 5 to 10 trials.

B. Comparing the Different Migration Plan Methods

We first compare Seq-Greedy and the batch method to the optimal migration solution and our three baselines, under our two resource scenarios. We then show how an operator might choose the batch length and truncation parameters before evaluating the effect of different user mobility patterns on Seq-Greedy's cost and recommended migrations.

Comparison to optimal approach. We compare the cost achieved by the optimization and Seq-Greedy approaches under limited and ample resources in Figure 5a. The optimization has a slightly lower cost, since the Seq-Greedy method places processes sequentially. Under ample resources, their performance is equivalent (Proposition 5). As we would expect from Section V-A's complexity analysis (Propositions 1–3), the number of edges in the migration graph grows approximately

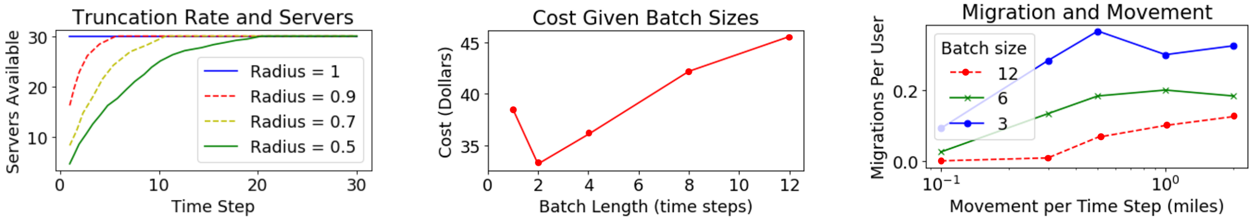


(a) Sub-cost comparison between optimization (ILP) and Seq-Greedy approach (6 users, 5 servers, 5 time steps).

(b) Cost incurred by different plan methods for limited and ample resources (20 users, 10 servers, 10 time steps).

(c) Sub-cost of the heuristic and baseline methods presented (20 users, 10 servers, 10 time steps, limited resources).

Fig. 5: Our proposed Seq-Greedy and batch methods out-perform the cloud, myopic and naïve baselines, though they are not optimal. BW represents bandwidth cost, while UD represents user dissatisfaction (latency) costs.



(a) Number of servers available at each time step for migration during truncation of Seq-Greedy method (30 total servers).

(b) The cost generally decreases with batch size (7 servers, 10 users, 12 time steps, and limited resources).

(c) Migrations occur more with faster travel and smaller batches (20 users, 8 servers, 12 time steps, ample resources).

Fig. 6: Effect of system settings such as batch length, truncation rates, and user speed in plan generation.

			Servers			
			5	10	15	20
Probability (γ)	1.0	TS 5	172	576	1305	2347
		TS 20	2071	9166	16736	27624
	0.9	TS 5	109	361	687	1198
		TS 20	1628	5434	11420	19098
	0.7	TS 5	87	224	509	696
		TS 20	1554	4834	11202	15835

TABLE II: Average edge counts per user given 30 users for Seq-Greedy migration graph as a function of radius truncation probability levels and the numbers of time steps and servers.

quadratically with respect to the number of servers and time steps (Table II), leading to a runtime for Seq-Greedy that is two order of magnitude shorter than the optimization approach.

Comparison to heuristic baselines. As seen in Figure 5b, all plan generation methods induce lower cost with ample compared to limited resources, as low latency placements are possible for every process. Seq-Greedy and the batch method significantly outperform the naïve and myopic baseline algorithms due to less frequent misplaced VM migrations compared to user location. Most notably, the batch method saves 35% in cost compared to the naïve method and 26% compared to the myopic method under limited resources. Under limited resources, the Seq-Greedy method outperforms the cloud method due to lower latencies as processes are not necessarily placed on the cloud given resource constraints, while their performances are equivalent given ample resources as no processes are sent to the cloud. Figure 5c shows the sub-costs for the Seq-Greedy and batch heuristics without truncation, as well as the three baseline algorithms. The heuristic approaches outperform the baselines largely due to closer process placements and less user dissatisfaction. The naïve approach does not perform migrations and suffers as users move away from their original position, verifying that Proposition 6 holds for more general scenarios. The myopic method has a lower latency cost incurred than both the Seq-Greedy and the cloud method due to frequent migrations, but incurs higher placement and bandwidth usage cost in the process. The batch approach outperforms the Seq-Greedy and cloud methods due to its superior predictions of user mobility by updating its conditioning on the user Markov chain.

Effect of truncation and batch length parameters. We next examine the truncation technique. Figure 6a shows that as the truncation probability increases, more servers are included for the migration graph. As we would expect, the number of servers included increases over time for each fixed truncation probability, as users move further distances from their current locations. Even a high truncation probability of 0.9, however, reduces the optimization complexity by 25% (Table II), indicating that truncation is an effective way to decrease complexity without significantly increasing cost.

The choice of batch length also affects the cost and complexity. As we increase the number of batches from 1 (Seq-Greedy) to higher values, the migrations per user increases, as seen in Figure 6c, as there is more certainty in user locations. Figure 6b similarly shows that the cost falls as the number of batches increases (which also reduces the algorithm complexity, as in Table II). When there are more batches present in the system due to shorter batch lengths, more frequent updates to user movement predictions allow for better migrations. However, because shorter batches prevent longer migrations that cross batch windows, the cost increases again when the batch length becomes too small, indicating that it should be carefully chosen to balance the cost effects, given the uncertainty present in user mobility patterns.

Effect of user mobility. To observe the impact of user movement on migration plan generation, Figure 6c shows the number of migrations a typical user undergoes for the lifetime of the requested service against the average speed per time step of users drawn from an exponential distribution. Users with higher average speeds incur more migrations, since the closest server to the user changes more frequently. Thus, MoDEMS adapts to different mobility characteristics in different areas.

C. Network Simulator Experiments

We validate MoDEMS' results by running our migration plans in a `ns-3` simulator that mimics realistic network delays. We simulate 10 edge servers, each connected to a base station (e.g., eNB) through a point-to-point connection. When a user sends a packet, it is received by the virtual device and then forwarded by IP forwarding to the edge server connected to the eNB, and then on to another edge server if needed. All eNBs use the LTE socket with Proportional Fair scheduling [49] to forward packets to users. The users' transmission mode is set to MIMO Spatial Multiplexity (2 layers). There are 20 users over 10 discrete time steps of 200s. We compare the performance of the batch (with batches of 2 time steps each), myopic, and naïve methods when transferring 1MB of data, which could represent computation results from the edge, from the VM to the UE per time step. Low throughput levels between eNBs simulate heavy traffic.

Figure 7a shows the resulting cumulative distribution of the average request completion times of all 20 users for each plan generation scheme, after removing outliers. As is consistent with Figure 5c, the average transmission times are the shortest for the batch method (31% less than the naïve method), followed by the myopic, Seq-Greedy, and naïve methods.

D. LTE Testbed Experiments

The importance of preemptive migrations is demonstrated with LTE testbed experiments (Figure 7b). The testbed has two eNodeBs (eNBs), UEs (user equipment) in a shield box, a signal attenuator, and edge servers. The Evolved Packet Core (EPC, not shown for simplicity) manages the network, including connecting to the Internet. We use two commercial indoor LTE small cell products, Juni JL620 [50], each connected to an edge server. UEs inside the shield box communicate via antennas connecting the eNBs and shield box. To emulate various RF situations, including handover between two eNBs, we install a signal attenuator between the eNBs and feed its output to the shield box. By changing the input power of each eNB, we can emulate a handover where a UE connects to an adjacent eNB of more robust signals. The wired latency between the two eNBs is set at 40ms, while wireless latency between an eNB and UE is approximately 60ms.

We monitor round trip times (RTT) between the UE and servicing VM over 120 seconds. The VM migration from edge server 1 to server 2 always starts at time $t = 0$s, and completes around $t = 80$s subject to network conditions. The UE moves from base station 1 to base station 2 at $t = \{0\text{s}, 40\text{s}, 80\text{s}\}$. User movement at $t = 0$s represents a reactive migration scheme, such as the myopic baseline, as the VM migration only begins after the user has moved. The $t = \{40\text{s}, 80\text{s}\}$ cases represent preemptive migrations, such as the batch method.

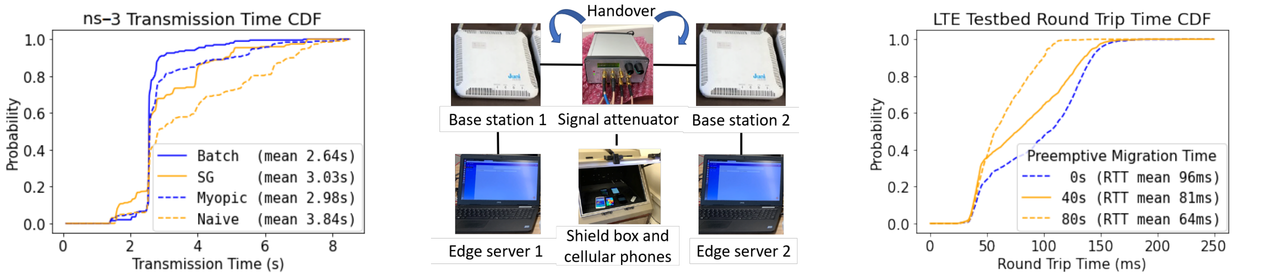
Figure 7c shows the resulting cumulative distribution of the round trip times between the UE and the servicing VM given different migration times. Consistent with the user dissatisfaction cost in Figure 5c and the service completion times of Figure 7a, migration schemes that have preemptive migrations (e.g. batch method) incur overall lower round trip times than migration schemes with reactive migrations (e.g. myopic method) by approximately 33%.

VII. Conclusion

While the use of cloud computing has grown in recent years, the distance between the cloud and the user presents issues of long latencies and limited bandwidth. Edge and fog computing mitigate those issues but require the strategic placement and migration of processes due to user movement. In this paper, we introduce MoDEMS, the first system to optimize edge computing deployments according to user mobility with a *theoretical framework* that jointly addresses practical deployment challenges, and *experimental validation* on real mobility traces and an LTE testbed. We formulate a linear integer programming problem and the Seq-Greedy heuristic used to generate migration plans that minimize system cost and user latency. Seq-Greedy saves orders of magnitude in terms of overhead compared to the optimization approach. Compared to a naïve approach that does not migrate processes, a myopic migration approach that does not attempt to predict user movement, or a cloud-based approach that does not account for resource constraints, we can save significant system cost and improve user experience. Moving forward, we can examine how migration plans can be generated for processes that serve many users at once on multiple edge nodes.

Acknowledgements

This work was partially supported by the US Army Research Office grant W911NF1910036, NSF CNS-2103024, Cisco Systems grant 1368170, IITP grant No.2021-0-01817 funded by MSIT, and the NSFC grant No.62102460.



(a) CDF of each user's average service completion times in the `ns-3` simulator.

(b) Setup of the LTE base station experiments with handover.

(c) CDF of user-VM round trip times for different preemptive migration times.

Fig. 7: Setup and cumulative distribution functions (CDFs) of measurement results for `ns-3` and LTE testbed experiments.

References

[1] "Cisco global cloud index: Forecast and methodology, 2016–2021 white paper," 2018.

[2] GSMA, "Cloud ar/vr streaming:accelerate mass adoption and improve quality of experience of ar/vr using 5g and edge cloud." https://www.gsma.com/futurenetworks/wp-content/uploads/2019/03/Cloud-ARVR-booklet-for-MWC19.pdf, Mar 2019.

[3] 3GPP, "5G; System architecture for the 5G System (5GS)," 2019. http://www.3gpp.org/dynareport/23501.htm.

[4] ETSI GS MEC 003 V2.1.1, "Mobile Edge Computing (MEC); Framework and Reference Architecture," 2019. https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/02.01.01_60/gs_MEC003v020101p.pdf.

[5] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. Leung, "Cache in the air: Exploiting content caching and delivery techniques for 5g systems," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 131–139, 2014.

[6] J. Lee, S. Moon, B. Bae, and J. Lee, "Local area data network for 5g system architecture," in *2018 IEEE 5G World Forum (5GWF)*, 2018.

[7] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 68–73, Dec. 2008.

[8] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, (New York, NY, USA), pp. 13–16, ACM, 2012.

[9] C. Martín Fernández, M. Díaz Rodríguez, and B. Rubio Muñoz, "An edge computing architecture in the internet of things," in *IEEE 21st International Symp. on Real-Time Distributed Computing*, pp. 99–102, May 2018.

[10] S. Dustdar, C. Avasalcai, and I. Murturi, "Invited paper: Edge and fog computing: Vision and research challenges," in *IEEE International Conf. on Service-Oriented System Engineering*, pp. 96–9609, April 2019.

[11] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, (New York, NY, USA), pp. 301–314, ACM, 2011.

[12] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, pp. 637–646, Oct 2016.

[13] Y. Yu, "Mobile edge computing towards 5g: Vision, recent progress, and open challenges," *China Communications*, vol. 13, pp. 89–99, N 2016.

[14] S. Wang, J. Xu, N. Zhang, and Y. Liu, "A survey on service migration in mobile edge computing," *IEEE Access*, vol. 6, pp. 23511–23528, 2018.

[15] O-RAN Alliance, "O-ran: Towards an open andsmart ran." https://www.o-ran.org/, 2018.

[16] H. Gebrie, H. Farooq, and A. Imran, "What machine learning predictor performs best for mobility prediction in cellular networks?," in *2019 IEEE ICC Workshops*, pp. 1–6, IEEE, 2019.

[17] Y. Chon, E. Talipov, H. Shin, and H. Cha, "Crawdad dataset yonsei/lifemap (v. 2012-01-03)," Jan 2012.

[18] T. Kim, S. Chen, Y. Im, X. Zhang, S. Ha, and C. Joe-Wong, "Modems: Optimizing edge computing migrations for user mobility." https://research.ece.cmu.edu/lions/Papers/MoDEMS_INFOCOM.pdf, 2021.

[19] X. Sun and N. Ansari, "EdgeIoT: mobile edge computing for the internet of things," *IEEE Comm. Magazine*, vol. 54, no. 12, pp. 22–29, 2016.

[20] Z. Rejiba, X. Masip-Bruin, and E. Marín-Tordera, "A survey on mobility-induced service migration in the fog, edge, and related computing paradigms," *ACM Comput. Surv.*, vol. 52, Sept. 2019.

[21] Z. Liang, Y. Liu, T.-M. Lok, and K. Huang, "Multi-cell mobile edge computing: Joint service migration and resource allocation," arXiv:2102.03036 [cs.IT], 2021.

[22] M. V. Ngo, T. Luo, H. T. Hoang, and T. Q. S. Quek, "Coordinated container migration and base station handover in mobile edge computing," arXiv:2009.05682 [cs.NI], 2020.

[23] L. Ma, S. Yi, N. Carter, and Q. Li, "Efficient live migration of edge services leveraging container layered storage," *IEEE Transactions on Mobile Computing*, vol. 18, no. 9, pp. 2020–2033, 2019.

[24] C.-F. Liu, M. Bennis, M. Debbah, and H. V. Poor, "Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing," *IEEE Transactions on Communications*, vol. 67, p. 4132–4150, Jun 2019.

[25] J. Wang, J. Hu, and G. Min, "Online service migration in edge computing with incomplete information: A deep recurrent actor-critic method," arXiv:2012.08679 [cs.NI], 2020.

[26] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge computing based on markov decision process," *IEEE/ACM Transactions on Networking*, 2019.

[27] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. S. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2019.

[28] I. Labriji, F. Meneghello, D. Cecchinato, S. Sesia, E. Perraud, E. C. Strinati, and M. Rossi, "Mobility aware and dynamic migration of mec services for the internet of vehicles," *IEEE Trans. on Netw. and Serv. Manag.*, vol. 18, p. 570–584, mar 2021.

[29] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE J.Sel. A. Commun.*, vol. 36, p. 2333–2345, oct 2018.

[30] T. Cao, Z. Qian, K. Wu, M. Zhou, and Y. Jin, "Service placement and bandwidth allocation for mec-enabled mobile cloud gaming," in *2021 IEEE 22nd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pp. 179–188, 2021.

[31] B. Yang, W. K. Chai, Z. Xu, K. V. Katsaros, and G. Pavlou, "Cost-efficient nfv-enabled mobile edge-cloud for low latency mobile applications," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 475–488, 2018.

[32] A. Nadembega, A. S. Hafid, and R. Brisebois, "Mobility prediction model-based service migration procedure for follow me cloud to support qos and qoe," in *2016 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2016.

[33] B. Hu and W. Hu, "Linkshare: Device-centric control for concurrent and continuous mobile-cloud interactions," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, SEC '19, (New York, NY, USA), p. 15–29, Association for Computing Machinery, 2019.

[34] B. Ottenwälder, B. Koldehofe, K. Rothermel, and U. Ramachandran, "Migcep: Operator migration for mobility driven distributed complex event processing," in *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, (New York, NY, USA), pp. 183–194, ACM, 2013.

[35] B. Ottenwälder, B. Koldehofe, K. Rothermel, K. Hong, D. Lillethun, and U. Ramachandran, "Mcep: A mobility-aware complex event processing system," *ACM Trans. Internet Technol.*, vol. 14, pp. 6:1–6:24, Aug. 2014.

[36] Y. Jia, C. Wu, Z. Li, F. Le, and A. Liu, "Online scaling of nfv service chains across geo-distributed datacenters," *IEEE/ACM Transactions on Networking*, vol. 26, pp. 699–710, April 2018.

[37] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Trans. on Network and Service Management*, vol. 13, pp. 725–739, Dec 2016.

[38] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1002–1016, 2016.

[39] T. Kim, S. Chen, Y. Im, X. Zhang, S. Ha, and C. Joe-Wong, "Modems: Optimizing edge computing migrations for user mobility," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, pp. 1–2, 2021.

[40] L. Wang, L. Jiao, J. Li, and M. Mühlhäuser, "Online resource allocation for arbitrary user mobility in distributed edge clouds," in *Proceedings of the 37th IEEE ICDCS*, pp. 1281–1290, IEEE, 2017.

[41] L. Fleischer, M. X. Goemans, V. S. Mirrokni, and M. Sviridenko, "Tight approximation algorithms for maximum general assignment problems," in *Proceedings of the 17th ACM-SIAM Symp. on Discrete Algorithms*, pp. 611–620, Society for Industrial and Applied Mathematics, 2006.

[42] M. Barbehenn, "A note on the complexity of dijkstra's algorithm for graphs with weighted vertices," *Computers, IEEE Transactions on*, vol. 47, p. 263, 03 1998.

[43] W. Stadje, "The exact probability distribution of a two-dimensional random walk," *J. of Statistical Physics*, vol. 46, pp. 207–216, Jan 1987.

[44] "ns-3 network simulator," 2020. https://www.nsnam.org/.

[45] X. e. a. Liang, "Unraveling the origin of exponential law in intra-urban human mobility," vol. 3, no. 2983, 18 Oct. 2013.

[46] J. B. Gilmour, A. W. Lui, and D. C. Briggs, "Emr," 1986.

[47] C. Zhou, Z. Li, and Y. Liu, "A measurement study of oculus 360 degree video streaming," in *Proceedings of the 8th ACM on Multimedia Systems Conference*, pp. 27–37, 2017.

[48] L. Ma, S. Yi, N. Carter, and Q. Li, "Efficient live migration of edge services leveraging container layered storage," *IEEE Transactions on Mobile Computing*, vol. 18, no. 9, pp. 2020–2033, 2018.

[49] S. Sesia, I. Toufik, and M. Baker, *LTE-the UMTS long term evolution: from theory to practice*. John Wiley & Sons, 2011.

[50] Juni, "Enterprise Small Cell JL620." http://www.juniglobal.com/product/jl-620fdd-jlt-621tdd/, 2017.