



Mobility-Aware Computation Offloading in Edge Computing using Machine Learning

Erfan Farhangi Maleki, *Student member, IEEE*, Lena Mashayekhy , *Senior Member, IEEE*, and Seyed Morteza Nabavinejad 

Abstract—Cloudlets are resource-rich computing infrastructures of edge computing that are located at physical proximity of users to provide one-hop, high-bandwidth wireless access to additional computational resources. They enable computation offloading for user applications, which compensates for the resource limitation of user devices by providing ultra-low latency processing for their applications. Although the computation capability of user devices is dramatically augmented by offloading, spatio-temporal uncertainties due to user mobility and changes in application specifications bring the most challenging obstacles in deciding where to offload to provide minimum latency. In this paper, we focus on these challenges by designing efficient offloading approaches that take into account these uncertainties and dynamics in order to minimize the turnaround time of the applications, which is constituted by offloading latency, migration delay, and execution time. We first formulate this NP-hard problem as an integer programming model to obtain optimal offloading decisions. We tackle its intractability by designing two novel offloading approaches, called S-OAMC and G-OAMC, that fully assign applications to cloudlets by considering their expected future locations and specifications predicted by Matrix Completion, a machine learning method. S-OAMC is a sampling-based approximation dynamic programming approach that enhances scalability and obtains near-optimal solutions. G-OAMC is a fast greedy-based approach for finding low-turnaround time offloading decisions. We conduct extensive experiments to assess the performance of our proposed approaches. The results show that S-OAMC and G-OAMC lead to near-optimal turnaround time in a reasonable time, and they both obtain low migration rates.

Index Terms—Edge Computing, Computation Offloading, Mobility, Sampling, Dynamic Programming

1 INTRODUCTION

THE number of mobile devices is expected to increase to 16.8 billion in 2023 [1]. These smart devices continuously generate unprecedented amounts of online data requiring data analysis to capture value. However, restrictions of mobile devices by weight, size, battery life, and heat dissipation impose a severe constraint on their computational resources such as processor speed, memory size, and disk capacity. Resource limitation is not just a temporary constraint but a fundamental limitation due to the convenience of mobility of these devices, and it hinders execution of many applications that have the potential to augment human cognition such as speech recognition, natural language processing, and augmented reality [2].

Edge computing (EC) is a promising distributed computing paradigm allowing computation to be performed at the edge of the network, where data is generated [3], [4]. Computation offloading in EC is the process of enhancing the capacity of mobile devices by migrating their computing tasks to EC [5]. As a result, data processing can be maintained closer to these devices to provide realtime and location-aware services, while reducing traffic to the cloud. Despite the advantages of computation offloading to EC in improving response time, battery life, and data safety and privacy [3], spatio-temporal uncertainties bring

the most challenging obstacles in providing these benefits, while users move.

The distribution of computing resources throughout EC is accomplished by having many small-sized heterogeneous clusters of servers referred to as “cloudlets” or “micro data centers” in the vicinity of mobile devices at the edge of the network [6]. Cloudlets help to mitigate the overload of mobile devices by accepting offloaded computation. However, due to the mobility of mobile users and dynamic changes (e.g., load of cloudlets), a primarily assigned cloudlet to a device might not be optimal over time. Therefore, the migration of computation between cloudlets is perceived as a necessary solution to resolve this concern. However, such frequent migration incurs additional data movement over the network and results in degraded performance, increased latency, and turnaround time of the offloaded applications.

In this paper, we address this problem by designing two efficient mobility-aware computation offloading approaches in order to minimize the turnaround time of mobile applications over their lifetime. Our proposed approaches utilize matrix completion, a suitable machine learning approach, to predict unsteady specifications of mobile applications in order to obtain smart offloading assignments that require less migration of computation in the future, and consequently, lead to low turnaround time. We first formulate an optimal integer programming (IP) model for this problem. Since the problem is NP-hard and intractable, we then design two efficient online offloading approaches, called S-OAMC and G-OAMC, to solve the problem in a reasonable time to obtain close to optimal turnaround time. S-OAMC is a sampling-based dynamic programming solution, and G-

- E. Farhangi Maleki and L. Mashayekhy are with the Department of Computer and Information Sciences, University of Delaware, Newark, DE, USA 19716
- S.M. Nabavinejad is with the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran 19395
Email: erfanf@udel.edu, mlена@udel.edu, nabavinejad@ipm.ir

OAMC is a greedy-based solution. The main contributions of this paper include the following:

- The feasibility of migration between cloudlets is considered as a technique for mobility management for EC.
- A machine learning-based method, named Matrix Completion, is designed to predict the specifications of mobile applications in the future (e.g., locations, bandwidth, processing speed requirements) to improve offloading decisions.
- Two novel computation offloading approaches, S-OAMC and G-OAMC, are proposed to minimize the turnaround time of the mobile applications, while reducing the number of migrations.
- S-OAMC is a sampling-based approximation dynamic programming algorithm that achieves higher scalability than DP-based solutions. S-OAMC achieves better load balancing for cloudlets by distributing the computation among them, and it guarantees near-optimal solutions. G-OAMC is a practical greedy-based approach that ensures quick convergence speed with reasonable results.
- We evaluate the performance of the proposed approaches showing that they are able to find near-optimal turnaround time with low migration rates.

The rest of the paper is organized as follows. In the next section, we provide an overview of existing work in this domain. We then formulate our problem in Section 3. We describe our proposed approaches, S-OAMC in Section 4 and G-OAMC in Section 5. In Section 6, we evaluate the properties of our proposed approaches by extensive experiments. Finally, we summarize our results and present possible directions for future research.

2 RELATED WORK

This paper summarizes the most relevant studies in the literature based on their research directions in EC.

Minimizing Latency. A group of studies has focused on the latency minimization in EC. Liu *et al.* [7] proposed an efficient one-dimensional search algorithm to find an optimal stochastic computation offloading policy. A system called LAVEA [8] is built on top of an EC platform to offload computation between users and edge nodes (cloudlets) to provide low-latency video analytics. Sharghivand *et al.* [9] proposed efficient QoS-aware, two-sided matching solutions to match cloudlets to user applications. They also determined dynamic pricing of the edge services based on preferences and incentives of cloudlets and users. Yang *et al.* [10] studied the joint computation partitioning and resource allocation problem for service deployment of latency-sensitive applications in EC. They proposed a heuristic approach that finds when a violation on any of the resource constraints happens and adjusts the user’s partitioning to avoid the violation. This searching and adjustment procedure is done until the initial partitioning converges to a feasible solution. Ma *et al.* [11] designed a game theoretic solution for offloading tasks among a swarm of Unmanned Aerial Vehicles (UAVs) acting as aerial capacitated cloudlets pooling their computational resources to execute tasks offloaded from mobile devices in their coverage. UAVs’ goal is to reduce their energy consumption while guaranteeing QoS for users. Li *et al.* [12] proposed a task offloading with

Study	Turnaround Time									
	Latency	Mobility	Het. Cloudlets	Het. Devices	Capacity Const.	Real Dataset	Approximation	Sampling	Multi-User	Prediction
Liu <i>et al.</i> [7]	✓			✓	✓					
Yi <i>et al.</i> [8]	✓					✓				✓
Sharghivand <i>et al.</i> [9]	✓		✓	✓	✓				✓	✓
Yang <i>et al.</i> [10]	✓			✓	✓				✓	
Ma <i>et al.</i> [11]	✓		✓	✓	✓				✓	
Li <i>et al.</i> [12]	✓		✓	✓	✓	✓		✓	✓	
Wang <i>et al.</i> [13]	✓		✓	✓	✓				✓	
Ma <i>et al.</i> [14]	✓		✓	✓	✓				✓	
Bahreini <i>et al.</i> [15]		✓	✓							
Wang <i>et al.</i> [16]	✓	✓	✓		✓	✓	✓	✓		
Bittencourt <i>et al.</i> [17]	✓	✓	✓		✓				✓	
Zhang <i>et al.</i> [18]	✓	✓						✓		✓
Ouyang <i>et al.</i> [19]		✓	✓	✓	✓		✓		✓	
Our Study	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

TABLE 1: Comparison with existing research

a probabilistic QoS guarantee. They formulated the problem as a mixed integer non-Linear programming problem with the statistical latency constraint, and they proposed an approach using convex optimization theory and Gibbs sampling method to solve the problem. Wang *et al.* [13] proposed a collaborative task offloading approach based on Hungarian algorithm to allocate subtasks to cloudlets in order to minimize energy consumption while satisfying task’s latency. Ma and Mashayekhy [14] proposed a truthful mechanism for computation offloading that assigns a proper pair of wireless access point and cloudlet along with a service price for joining users satisfying their QoS.

Mobility. Mobility is a significant challenge in EC, and a few studies take into account user mobility while offloading. Bahreini *et al.* [15] proposed an offline integer programming model and an online heuristic algorithm for component placement of one application to multiple cloudlets considering the dynamic distances between the user and cloudlets. Wang *et al.* [16] proposed a Markov decision process (MDP) to formulate the real-time (live) migration of an edge application (service) of a single user considering the distances between the user and the cloudlets before a possible migration. Bittencourt *et al.* [17] considered multiple application classes and proposed resource management policies to allocate resources between cloudlets and cloud to handle variable demand due to users mobility. Zhang *et al.* [18] proposed a deep reinforcement learning approach to migrate tasks according to users’ mobility. However, their approach only considers one user moving from one place to another, and also it does not include any guarantee on learning time and running time. Ouyang *et al.* [19] studied the requirement of migrating services dynamically among multiple edge servers due to user mobility in order to maintain satisfactory user experience. However, service migration is modeled in terms of cost instead of a delay, which leads to modeling the problem to optimize the long-term performance under the predefined long-term cost budget constraint for the migration. They proposed two heuristic methods based on the Markov approximation and best response updates.

Our work investigates how to make smart computation offloading decisions to reduce the turnaround time of applications considering user mobility and application specification changes. It exploits an efficient machine learning method to predict application specifications and user mobility in the future. It introduces a sampling-based approximation offloading approach and a greedy-based approach to find offloading decisions with smallest turnaround times in scalable and timely manner. Table 1 further presents a detailed comparison of our work with the existing state-of-art studies based on several criteria.

3 SYSTEM MODEL

We first describe the system model consisting of a set of mobile applications and a set of cloudlets. We assume a set of n mobile applications requires offloading and is represented by $U = \{u_1, u_2, \dots, u_n\}$. We consider a set of m cloudlets $C = \{c_1, c_2, \dots, c_m\}$ is available to offer edge services to users. Computation offloading happens over a time period that can be viewed as a sequence of time slots $1, \dots, T$.

Each application $u_i \in U$ at time t has the following specifications: $u_i^t = (x_i^t, y_i^t, md_i^t, id_i^t, \omega_i^t, p_i^t, b_i^t)$, where x_i^t and y_i^t indicate the location of the mobile application $u_i \in U$ at time t ; md_i^t indicates the code and the metadata of the application u_i that is either offloaded directly from the device or migrated from a cloudlet at time t ; id_i^t is the size of the intermediate data that is sent to a cloudlet in the middle of u_i 's execution at time t ; ω_i^t is the computation requirement of u_i in terms of number of instructions (or cycles) needed at time t ; p_i^t represents the processing speed requirement of u_i at time t ; and b_i^t indicates the bandwidth requirement of u_i at time t . For simplicity of formulation, we consider $id_i^t = md_i^t$ and only use md_i^t when a migration happens (i.e., $t \geq 2$).

Each cloudlet $c_j \in C$ has the following specifications: $c_j = (x_j, y_j, \rho_j, \beta_j)$, where x_j and y_j indicate the location of the cloudlet; ρ_j represents the total processing speed of the cloudlet, and β_j is its total bandwidth.

Our objective is to minimize the overall turnaround time of computation offloading. The turnaround time includes offloading time, migration time, and computation time. Offloading and migration time are captured as a latency of communicating with a cloudlet to send the application's code and data (from the application or another cloudlet). Computation time is execution time of completing the application on a cloudlet. We define latency as follows:

$$l = \frac{d}{\theta} + \frac{s}{\beta}, \quad (1)$$

where d represents the distance, θ is the propagation speed, s denotes the data size, and β represents the bandwidth. The first part of the formula computes the propagation time, and the second part computes the transmission time. We use the notation $f_d^t(u_i, c_j)$ to show the distance between application u_i and cloudlet c_j . We use the Euclidean metric to calculate the distance between coordinates in the Cartesian coordinate system:

$$f_d^t(u_i, c_j) = \sqrt{(x_i^t - x_j)^2 + (y_i^t - y_j)^2}.$$

Similarly, $f_d(c_j, c_k)$ is used for showing the distance between a pair of cloudlets.

The computation offloading problem is to find an optimal assignment of all mobile applications in U to existing cloudlets in C minimizing the turnaround time of all applications satisfying all constraints. To optimally model this problem, we first define the decision variables μ_{ij}^t and α_{ijk}^t as follows:

- $\mu_{ij}^t = 1$ signifies that application $u_i \in U$ is assigned to cloudlet $c_j \in C$ at time t , and
- $\alpha_{ijk}^t = 1$ signifies that application u_i migrates from cloudlet c_j to cloudlet c_k at time t .

We formulate the computation offloading problem as an integer program (IP) as follows:

$$\begin{aligned} \text{Minimize } & \sum_{i=1}^n \sum_{j=1}^m \sum_{t=1}^T \left(\frac{f_d^t(u_i, c_j)}{\theta} + \frac{id_i^t}{b_i^t} + \frac{\omega_i^t}{p_i^t} \right) \mu_{ij}^t + \\ & \sum_{i=1}^n \sum_{j=1}^m \sum_{k=1}^m \sum_{t=2}^T \left(\frac{f_d(c_j, c_k)}{\theta} + \frac{md_i^t}{b_i^t} \right) \alpha_{ijk}^t \end{aligned} \quad (2)$$

Subject to:

$$\sum_{j=1}^m \mu_{ij}^t = 1 \quad \forall u_i \in U, t \in T \quad (3)$$

$$\sum_{i=1}^n p_i^t \mu_{ij}^t \leq \rho_j \quad \forall c_j \in C, t \in T \quad (4)$$

$$\sum_{i=1}^n b_i^t \mu_{ij}^t \leq \beta_j \quad \forall c_j \in C, t \in T \quad (5)$$

$$\alpha_{ijk}^t \geq \mu_{ij}^{t-1} + \mu_{ik}^t - 1 \quad \forall u_i \in U, c_j, c_k \in C, t \in T \quad (6)$$

$$\mu_{ij}^t \in \{0, 1\} \quad \forall u_i \in U, c_j \in C, t \in T \quad (7)$$

$$\alpha_{ijk}^t \in \{0, 1\} \quad \forall u_i \in U, c_j, c_k \in C, t \in T \quad (8)$$

The objective function (Eq. (2)) minimizes the total turnaround time for all applications. The first term calculates both the offloading time (latency) and the computation time, and the second term calculates the migration time. If a migration happens, previous states (metadata) of the application along with its code will be transferred to the new cloudlet. Constraints (3) guarantee that each application is assigned to exactly one cloudlet at each time slot. Constraints (4) and (5) ensure that the total requested processing speed and bandwidth of the assigned applications to a cloudlet do not exceed the available processing speed and bandwidth of that cloudlet at each time slot. Constraints (6) ensure that α_{ijk}^t is one if μ_{ik}^t and μ_{ij}^{t-1} are one, which means application $u_i \in U$ migrates from cloudlet $c_j \in C$ to cloudlet $c_k \in C$ at time t ; otherwise, it is zero. Constraints (7) and (8) guarantee that the decision variables are binary.

Our proposed IP finds the optimal assignment of the applications to cloudlets minimizing the turnaround time (offloading, computation, and migration) considering full knowledge of future user mobility and application specifications. In addition, as this problem is an instance of the generalized assignment problem (GAP) [20], it is NP-hard.

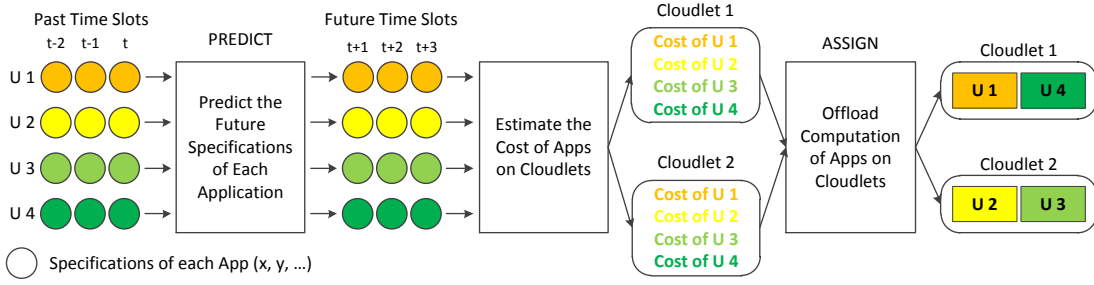


Fig. 1: Mobility-Aware Computation Offloading

We tackle these issues by designing efficient and scalable algorithms to find such assignments in a reasonable time. We propose two mobility-aware computation offloading approaches, S-OAMC and G-OAMC, presented in Sections 4 and 5, respectively. Fig. 1 illustrates the workflow of these approaches: First, future specifications of mobile applications are predicted based on the history of past time slots by calling PREDICT function that uses a machine learning method. Second, using the specifications of current and predicted future time slots, the expected offloading cost of applications to cloudlets is estimated. Third, ASSIGN function determines a smart decision on offloading based on the cost values.

4 SAMPLING-BASED OFFLOADING TO CLOUDLETS

We design a Sampling-based algorithm for Online Assignment of Mobile Applications to Cloudlets, called S-OAMC, which uses prediction to make the best offloading decisions. The pseudo code of our proposed algorithm is presented in Algorithm 1. S-OAMC assigns every application to a cloudlet at each time slot such that it leads to a reasonable total turnaround time in the lifetime of the applications.

S-OAMC receives ϵ, η , window size w , and discount factor γ as inputs (line 1), where $\epsilon \in [0, 1]$ and η are control parameters used in DP-APC, w indicates the number of future specifications taken into account for the offloading decisions, and $\gamma \in [0, 1]$ is to indicate the impact for different future time slots.

S-OAMC uses set \hat{U}_i^t to maintain the specifications' history of application u_i from the first time slot to current time slot t , i.e., $\hat{U}_i^t = \bigcup_{\delta=1}^t u_i^\delta$, where u_i^δ denotes the specification of application u_i at time slot δ . This set is initially an empty set (line 2). In addition, \hat{R}_i^t maintains the history of cloudlet assignments of application u_i to current time slot t , which is initially empty (line 3). For example, $\hat{R}_i^t = j$ means application u_i is assigned to cloudlet c_j at time t . These sets serve as the history of specifications and assignments. Note that for simplicity, we use notations with hat operator (e.g., \hat{U}) to show the history and bar operator (e.g., \bar{U}) to show the predicted values.

S-OAMC intends to minimize the total turnaround time by minimizing the number of migrations required in the lifetime of the applications considering their next w specifications (that include user mobility too). These future specifications are used to ensure the determined offloading assignments for the next w time slots lead to minimum total turnaround time. Therefore, S-OAMC uses predicted specifications of all applications $u_i \in U$ in next w time slots

based on specification histories ($\hat{U}_i^t, \forall u_i$) that have been collected to make the best offloading decisions. In doing so, S-OAMC first updates \hat{U}_i^t by adding previous specification histories and the new specification u_i^t at time t to include the most recent specification of application u_i (line 6). Then, it checks whether the current time slot is a multiple of w (line 7). This is to ensure that offloading decisions are carried out every w time slots (i.e., obtained assignments are valid for the next w time slots).

S-OAMC calls PREDICT function, described in subsection 4.1, to obtain the predicted specifications of the applications in next w time slots (line 8). Then, S-OAMC computes the expected cost (i.e., the expected turnaround time considering the discount factor) of offloading each application for the next w time-slot period (lines 9-15). The value of v_{ij}^t represents this cost (i.e., assigning application u_i to cloudlet c_j at time slot t for the next w period), and it is initially set to infinity, where $V_i^t = \bigcup_{j=1}^m v_{i,j}^t$ (line 9). This value is only updated if a cloudlet is nearby the user or in the area of the user. The expected cost covers offloading and computation time (lines 11-12) and migration time (line 15). The offloading and computation time is the summation of offloading and computation time at the current time slot (line 11) and next w time slots (line 12). Since predicted values in different time slots should not have the same impacts on the offloading decision, the impact of further time slots is decreased by using γ . Therefore, more weight is given to the closest time slots, and less weight is given to the furthest time slots. For example, the weight of the first future time slot is γ^1 while the weight of the last future time slot is γ^w . In addition, S-OAMC checks whether cloudlet c_j is different from the currently assigned cloudlet (at time slot $t-1$) for the application u_i to add the migration cost (lines 13-15); otherwise, there is no migration cost.

When the current time slot is a multiple of w (line 16), based on the calculated expected costs and predicted future specifications, S-OAMC finds the assignment of applications to cloudlets to minimize the total turnaround time. In doing so, it calls ASSIGN function (line 18), presented in Algorithm 2, with V^t, U^t , and C as input parameters, where V^t is a matrix corresponding to all cost $v_{i,j}^t$, i.e., $V^t = \bigcup_{i=1}^n \bigcup_{j=1}^m v_{i,j}^t$, U^t is a matrix corresponding to all specifications u_i^t , i.e., $U^t = \bigcup_{i=1}^n u_i^t$, and C is a matrix corresponding to all cloudlet specifications c_j , i.e., $C = \bigcup_{j=1}^m c_j$. ASSIGN returns S^t , a set of offloading decisions, that determines the assignment of applications to cloudlets. For example, given 2 cloudlets and 2 applications $U = \{u_1, u_2\}$, $S^t = (\{\}, \{u_1, u_2\})$ indicates no application

Algorithm 1 Online Assignment of Mobile applications to Cloudlets (S-OAMC)

```

1: Input:  $\epsilon, \eta, w$ : window size,  $\gamma$ : discount factor
2:  $\hat{U}_i^0 = \emptyset, \forall u_i \in U$ 
3:  $\hat{R}_i^0 = \emptyset, \forall u_i \in U$ 
4: for all  $t \in T$  do
5:   for all  $u_i \in U$  do
6:      $\hat{U}_i^t = \hat{U}_i^{t-1} \cup u_i^t$ 
7:     if  $t \bmod w = 0$  then
8:        $\{\bar{u}_i^{t+1}, \dots, \bar{u}_i^{t+w}\} = \text{PREDICT}(\hat{U}_i^t, t, w)$ 
9:        $V_i^t \leftarrow \emptyset$ 
10:      for all  $c_j \in C$  adjacent to  $u_i$  do
11:         $v_{ij}^t = \frac{f_d^t(u_i, c_j)}{\theta} + \frac{id_i^t}{b_i^t} + \frac{\omega_i^t}{p_i^t}$ 
12:         $v_{ij}^{t+\tau} = \sum_{\tau=1}^w \gamma^\tau \left( \frac{f_d^{t+\tau}(\bar{u}_i, c_j)}{\theta} + \frac{\bar{id}_i^{t+\tau}}{b_i^{t+\tau}} + \frac{\bar{\omega}_i^{t+\tau}}{p_i^{t+\tau}} \right)$ 
13:        if  $t \geq 2$  and  $j \neq \hat{R}_i^{t-1}$  then
14:           $k = \hat{R}_i^{t-1}$ 
15:           $v_{ij}^{t+\tau} = \frac{f_d^{t+\tau}(c_j, c_k)}{\theta} + \frac{md_i^t}{b_i^t}$ 
16:      if  $t \bmod w = 0$  then
17:        do
18:           $S^t = \text{ASSIGN}(V^t, U^t, C)$ 
19:           $\hat{R}^t \leftarrow S^t$ 
20:          Update  $V^t, U^t, C$ 
21:      while  $\exists \hat{R}_i^t \in \hat{R}^t$  which is not assigned
22:    else
23:      for all  $u_i \in U$  do
24:         $\hat{R}_i^t = \hat{R}_i^{t-1}$ 
25: Return:  $\bigcup_{t=1}^T \hat{R}^t$ 

```

is assigned to cloudlet 1, while both applications u_1 and u_2 are assigned to cloudlet 2.

S-OAMC converts S^t to \hat{R}^t , where \hat{R}^t is a set corresponding to all \hat{R}_i^t (i.e., $\hat{R}^t = \bigcup_{i=1}^n \hat{R}_i^t$) to maintain the assigned cloudlets to all applications (line 19). S-OAMC then removes the specifications of the assigned applications from U^t and also updates the specifications of the cloudlets to reflect their available resources (ρ and β). Similarly, it updates the cost matrix (V^t) by eliminating the rows in the matrix that refer to the assigned applications (line 20). This procedure (lines 17-21) is repeated until there is no unassigned application.

If the current time slot is not a multiple of w , all applications remain assigned to their current cloudlets (lines 22-24). Finally, S-OAMC returns the offloading assignments in all time slots (line 25). More details about PREDICT and ASSIGN functions will be given in the next subsections.

4.1 Forecasting Method for Mobility-Aware Offloading

We use matrix completion method in PREDICT function of S-OAMC to predict the future specifications of the applications over the course of time. More specifically, PREDICT uses matrix completion to obtain the predicted specifications of application u_i for the next w time slots. The specification of the applications composed of x, y, md, id, ω, p , and b . For each specification of the application (e.g., b), we

construct a separate matrix, where the rows of the matrix are the applications and the columns are the time slots.

Matrix completion method [21], [22] is an ML approach to predict the missing entries of a partially observed matrix. To recover the missing entries of a matrix, matrix completion uses Singular Value Decomposition (SVD) to find similarity between rows and columns and reduce the dimensions of the matrix. For example, recommendation systems use SVD to extract similarities between users and items [23]. This is a robust approach to missing entries and imposes relaxed sparsity constraints to provide accuracy guarantees [24].

We also need additional information about a matrix such as its rank. A matrix has rank r if its rows or columns span an r -dimensional space. This rank represents the number of similarity concepts identified by SVD.

Feeding SVD with a matrix of rank r , we will have a factorization of the form $U \times \Sigma \times V^T$, where U and V are the left and right singular vectors matrices representing the strength of the row-to-concept similarity and the concept-to-column similarity, respectively, and Σ represents the matrix of singular values (similarity concepts are represented by σ_i). For example, for the bandwidth requests of applications, matrix B with n_1 rows for applications and n_2 columns for time slots is formed. Therefore,

$$B = U \times \Sigma \times V^T,$$

where

$$U_{n_1 \times r} = \begin{bmatrix} \mathbf{u}_{11} & \dots & \mathbf{u}_{1r} \\ \mathbf{u}_{21} & \dots & \mathbf{u}_{2r} \\ \vdots & \ddots & \vdots \\ \mathbf{u}_{n_1 1} & \dots & \mathbf{u}_{n_1 r} \end{bmatrix}, V_{n_2 \times r} = \begin{bmatrix} \mathbf{v}_{11} & \dots & \mathbf{v}_{1r} \\ \mathbf{v}_{21} & \dots & \mathbf{v}_{2r} \\ \vdots & \ddots & \vdots \\ \mathbf{v}_{n_2 1} & \dots & \mathbf{v}_{n_2 r} \end{bmatrix},$$

and

$$\Sigma_{r \times r} = \begin{bmatrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_r \end{bmatrix}.$$

Having matrices U , Σ , and V , we apply PQ-reconstruction method [25], [26] to obtain matrix X that predicts the missing values of the matrix (i.e., matrix X is an approximate of matrix B). Estimating the missing values in matrix X can be done via different approaches such as convex optimization [21], gradient descent [27], and alternating least squares minimization [28]. In matrix X , for the observed entries of matrix B , we have: $X_{ij} = B_{ij}$ and rank X is minimized. We use convex optimization by TFOCS (Templates for First-Order Conic Solvers) [29] tool to estimate matrix X . Figure 2 shows the overall flow of the proposed approach.

To create a matrix for each specification, we first select several representative applications that can present the patterns of each specification over past time slots. These applications build our training data to help the matrix completion method learn the patterns of the applications. For new applications that S-OAMC needs to predict their specifications, they are inserted at the end of each matrix in the form of new rows. For these applications, S-OAMC knows their specifications over previous time slots until current time slot t . Note that our approach for using matrix completion is different from the proposed ones in [30] that

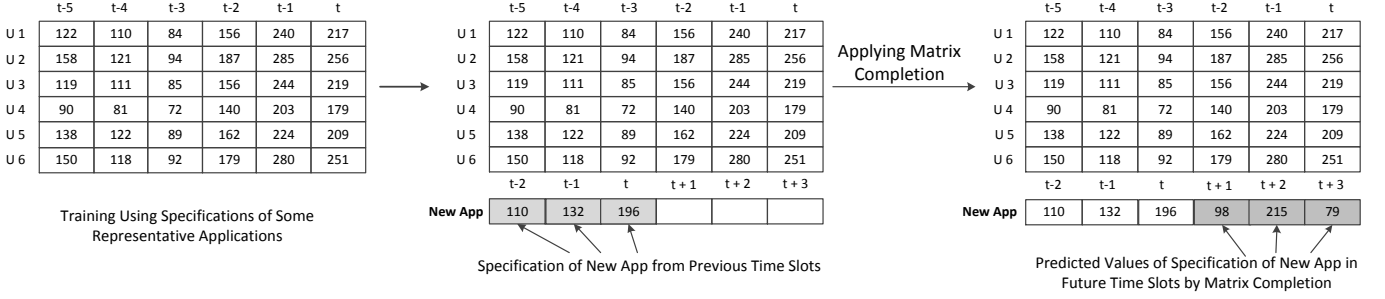


Fig. 2: Flow of Matrix Completion for Predicting the Specification of a Mobile Application

considered only one application. In [30], the future values for that application are predicted based on its own previous data using SVD and a neural network. However, in our approach, we use the past data of other representative applications. Their future specifications (i.e., $t + 1$ to $t + w$) are the missing values in each matrix. Matrix completion method predicts these values and returns them to S-OAMC to be used in ASSIGN function.

PREDICT function with (\hat{U}_i^t, t, w) as input parameters uses this approach and returns $\bigcup_{h=1}^w \bar{u}_i^{t+h}$, where $\bar{u}_i^{t+h} = (\bar{x}_i^{t+h}, \bar{y}_i^{t+h}, \bar{m}d_i^{t+h}, \bar{i}d_i^{t+h}, \bar{\omega}_i^{t+h}, \bar{p}_i^{t+h}, \bar{b}_i^{t+h})$.

4.2 Sampling-based Assignment

S-OAMC calls ASSIGN function, presented in Algorithm 2, to find the offloading assignment of the applications to the cloudlets. This function needs to solve an instance of the Generalized Assignment Problem (GAP), which is a well-known NP-hard problem. Approximate algorithms have been proposed to solve GAP, however, they mostly require exponential preprocessing time and complicated rounding techniques [31]–[33]. ASSIGN uses our sampling-based local-ratio approximation technique to provide near-optimal solutions and a faster running time by tackling the slow execution of dynamic programming due to curse of dimensionality [34]. It receives the cost functions (V^t) , specifications of the applications (U^t) , and specifications of the cloudlets (C) , and it returns S as a set of offloading assignments.

ASSIGN is an iterative approach, where each iteration starts with a new cloudlet (i.e., c_j) and uses two matrices G^t and N^t with an initial size of $n \times m$ (n is the number of applications and m is the number of cloudlets) to determine the elements of the cost function for the next iteration (i.e., cloudlet c_{j+1}). Similarly, V^t is initially an $n \times m$ matrix. The dimension of these matrices is dynamically changed by removing the reviewed cloudlets.

ASSIGN calls function DP-APC, presented in Algorithm 3, to obtain a set \tilde{S}_j of assigned applications to cloudlet c_j (line 3). Then, it fills the elements of matrix G^t by the values related to cloudlet c_j and the applications in \tilde{S}_j , and it sets all other elements to zero (lines 4-9). ASSIGN then computes the values of matrix N^t by subtracting G^t from V^t (line 10). Finally, it removes the column of the current cloudlet from N^t and sets the modified matrix to V^t (line 11). Subsequently, it performs this procedure iteratively for each cloudlet.

A set of assigned applications to each cloudlet is determined (\tilde{S}_j for cloudlet c_j). However, ASSIGN performs a procedure to ensure that no application can be assigned to two or more cloudlets (lines 13-15). In doing so, S_j determines the applications exclusively assigned to c_j by excluding the common applications (that are already assigned to other cloudlets). It is equivalent to $\tilde{S}_j \setminus \bigcup_{k=j+1}^m S_k$. Finally, ASSIGN returns the application assignments S to all cloudlets (line 16).

DP-APC function, presented in Algorithm 3, is a sampling-based approximation Dynamic Programming approach to find the Assignment Per single Cloudlet. DP-APC is given the cost values (V_j^t) associated with cloudlet c_j , the specifications of the applications (U^t) , and the specification of cloudlet c_j (including ρ_j and β_j representing its processing speed and total bandwidth, respectively). Note that all cost values and specifications are positive. DP-APC has ϵ and η as input parameters, where ϵ indicates the accuracy of results, runtime, and size of the rounding, and η indicates the size of the sampling. DP-APC uses sampling to reduce the number of applications that is possible to assign to the cloudlet. The goal of DP-APC is to find a subset of applications $I \subseteq U^t$ such that these applications exploit the available capacity of cloudlet c_j as much as possible, while it also leads to a minimum expected cost (turnaround time).

Algorithm 2 ASSIGN(V^t, U^t, C)

- 1: Initialize matrix G^t and N^t
 - 2: **for all** $c_j \in C$ **do**
 - 3: $\tilde{S}_j = \text{DP-APC}(V_j^t, U^t, c_j)$
 - 4: **for all** $u_i \in U$ **do**
 - 5: **for all** unreviewed $c_k \in C$ **do**
 - 6: **if** ($u_i \in \tilde{S}_j$) or ($c_k = c_j$) **then**
 - 7: $g_{ik}^t = v_{ij}^t$
 - 8: **else**
 - 9: $g_{ik}^t = 0$
 - 10: $N^t = V^t - G^t$
 - 11: $V^t \leftarrow N^t$ without column of c_j
 - 12: c_j 's status is changed to *reviewed*
 - 13: $S_m = \tilde{S}_m$
 - 14: **for** $j = m - 1$ **to** 1 **do**
 - 15: $S_j = \tilde{S}_j \setminus \bigcup_{k=j+1}^m S_k$
 - 16: **Return:** $S = \bigcup_{j=1}^m S_j$
-

Algorithm 3 DP-APC(V_j^t, U^t, c_j)

```

1: Input:  $\epsilon, \eta$ 
2:  $\check{U} \leftarrow$  sort all applications in ascending order of  $v_{ij}^t$ 
3:  $\ddot{U} \leftarrow$  select  $\eta$  number  $u_i \in \check{U}$  with  $v_{ij}^t \neq \infty$ 
4:  $M \leftarrow \max_{u_i \in \ddot{U}} v_{ij}^t$ 
5: for all  $u_i \in \ddot{U}$  do
6:    $\check{v}_{ij}^t \leftarrow M + 1 - v_{ij}^t$ 
7:  $\check{M} \leftarrow \max_{u_i \in \ddot{U}} \check{v}_{ij}^t$ 
8:  $\lambda \leftarrow \epsilon \check{M} / |\check{U}|$ 
9: for all  $u_i \in \ddot{U}$  do
10:  if  $\lambda > 1$  then
11:     $r_{ij}^t \leftarrow \lfloor \check{v}_{ij}^t / \lambda \rfloor$ 
12:  else
13:     $r_{ij}^t \leftarrow \check{v}_{ij}^t$ 
14:  $r^* \leftarrow \max_{u_i \in \ddot{U}} r_{ij}^t$ 
15: for all  $k = 1$  to  $|\check{U}| \cdot r^*$  do
16:    $A[k].p = A[k].b = \infty$ 
17:  $A[0].p = A[0].b = 0$ 
18: for  $i = 1$  to  $|\check{U}|$  do
19:   for  $k = (i - 1) \cdot r^*$  down to  $0$  do
20:     $A[k + r_{ij}].b = \min(A[k + r_{ij}].b, A[k].b + b_i^t)$ 
21:     $A[k + r_{ij}].p = \min(A[k + r_{ij}].p, A[k].p + p_i^t)$ 
22:  $\tilde{k} \leftarrow \arg \max_{k \in \{1, \dots, |\check{U}| \cdot r^*\}} A[k].p \leq \rho_j$  and  $A[k].b \leq \beta_j$ 
23: Find a subset  $I \subseteq \ddot{U}$  such that  $\sum_{i \in I} p_i^t = A[\tilde{k}].p$ ,
    $\sum_{i \in I} b_i^t = A[\tilde{k}].b$ 
24: Return:  $I$ 

```

DP-APC sorts all applications in ascending order of v_{ij}^t (line 2) and further reduces the complexity of dynamic programming by only considering a maximum of η sorted applications in the area of the cloudlet, which have non-infinity values of v_{ij}^t (line 3). The size of the sampling is η and only if the number of applications with non-infinity values of v_{ij}^t is less than η , the size of sampling is equal to the number of non-infinity values. DP-APC uses a rounding down approach for solving the assignment problem approximately for each cloudlet. It first computes the maximum value associated with cloudlet c_j excluding applications with infinity values and assigns this maximum value to M (line 4). DP-APC then calculates benefit values \check{v}_{ij}^t from the cost values v_{ij}^t by subtracting each from $M + 1$ (lines 5-6). By using the benefit values, the problem becomes a maximization problem. DP-APC then finds the maximum benefit value of applications in \ddot{U} , that is \check{M} (line 7). Then, it computes λ based on \check{M} , $|\check{U}|$, and ϵ , where ϵ determines the accuracy of results and size of the rounding (line 8). DP-APC then converts each benefit value \check{v}_{ij}^t to the multiple of λ only if $\lambda > 1$ (lines 9-13). DP-APC uses these new rounded benefit values r_{ij}^t to find subset I of applications in \ddot{U} that satisfies the constraints of cloudlet c_j and has maximum benefit (minimum expected turnaround time).

DP-APC assigns the maximum rounded value of r_{ij}^t to r^* (line 14). Note that r^* is bounded to $|\check{U}|/\epsilon$. Then, it uses a dynamic programming array A with $|\check{U}| \cdot r^* + 1$

Algorithm 4 Greedy Online Assignment of Mobile applications to Cloudlets (G-OAMC)

```

1: Input:  $w$ : window size
2:  $\hat{U}_i^0 = \emptyset, \forall u_i \in U$ 
3:  $\hat{R}_i^0 = \emptyset, \forall u_i \in U$ 
4: for all  $t \in T$  do
5:   for all  $u_i \in U$  do
6:      $\hat{U}_i^t = \hat{U}_i^{t-1} \cup u_i^t$ 
7:     if  $t \bmod w = 0$  then
8:        $\{\bar{u}_i^{t+1}, \dots, \bar{u}_i^{t+w}\} = PREDICT(\hat{U}_i^t, t, w)$ 
9:        $b_i^{\max} = \text{maximum in } \{\bar{b}_i^{t+1}, \dots, \bar{b}_i^{t+w}\}$ 
10:       $p_i^{\max} = \text{maximum in } \{\bar{p}_i^{t+1}, \dots, \bar{p}_i^{t+w}\}$ 
11:      for all  $c_j \in C$  do
12:         $v_{ij}^t = \frac{f_d^t(u_i, c_j)}{\theta} + \frac{id_i^t}{b_i^t} + \frac{\omega_i^t}{p_i^t}$ 
13:         $v_{ij}^t += \sum_{\tau=1}^w \gamma^\tau \left( \frac{f_d^{t+\tau}(\bar{u}_i, c_j)}{\theta} + \frac{i\bar{d}_i^{t+\tau}}{b_i^{t+\tau}} + \frac{\bar{\omega}_i^{t+\tau}}{p_i^{t+\tau}} \right)$ 
14:        if  $t \geq 2$  and  $j \neq \hat{R}_i^{t-1}$  then
15:           $k = \hat{R}_i^{t-1}$ 
16:           $v_{ij}^t += \frac{f_d^t(c_j, c_k)}{\theta} + \frac{md_i^t}{b_i^t}$ 
17:        if  $t \bmod w = 0$  then
18:          for all  $c_j \in C$  do
19:             $\check{U}_j =$  Sort applications in ascending order of
               $\frac{v_{ij}^t}{\frac{p_i^{\max}}{\rho_j} + \frac{b_i^{\max}}{\beta_j}}$ 
20:            for all  $u_i \in \check{U}_j$  do
21:              if  $p_i^{\max} \leq \rho_j$  and  $b_i^{\max} \leq \beta_j$  and  $u_i$  is not
                assigned then
22:                 $\hat{R}_i^t = j$ 
23:                Update  $c_j$ 
24:              else
25:                for all  $u_i \in U$  do
26:                   $\hat{R}_i^t = \hat{R}_i^{t-1}$ 
27: Return:  $\bigcup_{t=1}^T \hat{R}^t$ 

```

cells, where each cell $A[k]$ ($k \in \{0, \dots, |\check{U}| \cdot r^*\}$) has two values $A[k].p$ and $A[k].b$ computed iteratively. The former is the minimum processing speed to achieve a rounded benefit value of k , and the latter is the minimum bandwidth to achieve that. Initially, $A[k].p$ and $A[k].b$ are set to infinity for all $k \geq 1$, and $A[0].p$ and $A[0].b$ are initialized to 0 (lines 15-17). For the remaining cells, DP-APC checks to include or not include a new application i in the optimal subset (lines 18-21).

Finally, DP-APC finds \tilde{k} , the maximum value of k such that it satisfies the capacity constraints of cloudlet c_j (line 22) and builds subset I based on that (line 23).

5 GREEDY OFFLOADING TO CLOUDLETS

In this section, we propose another approach for Online Assignment of Mobile Applications to Cloudlets using a Greedy approach, called G-OAMC. G-OAMC, contrarily to S-OAMC, does not need to call a computationally-intensive dynamic programming approach, and it has the advantage of a quicker convergence speed over S-OAMC. However,

it cannot guarantee near-optimal results of DP-based approaches (OAMC [35] and S-OAMC). G-OAMC reduces the number of used cloudlets by attempting to assign more applications to the already used cloudlets. The pseudo-code of G-OAMC is presented in Algorithm 4.

G-OAMC, similar to S-OAMC, receives window size w and uses \hat{U}_i^t and \hat{R}_i^t as described in Algorithm 1 (lines 1-3). Function PREDICT determines predicted specifications of the applications in the next w time slots (line 8).

G-OAMC finds the maximum amount of processing speed, p_i^{\max} , and bandwidth, b_i^{\max} , required by application u_i in the next w time slots (lines 9-10). These maximum values will be used to guarantee that the new assigned cloudlet to each application has enough capacity to satisfy the application requirements during its execution on that cloudlet. They will also be used in our density metric to prioritize applications with higher expected demands. Similar to S-OAMC, G-OAMC computes the expected cost of assigning application u_i to cloudlet c_j at time slot t , represented by v_{ij}^t , where it covers offloading and computation time (lines 12-13) and migration time (line 16).

The decision about the assignment is determined every w time slots, and it is valid for that duration (line 17). G-OAMC sorts applications in ascending order according to a density metric defined as follows (line 19):

$$\frac{v_{ij}^t}{\frac{p_i^{\max}}{\rho_j} + \frac{b_i^{\max}}{\beta_j}}, \quad (9)$$

where applications with smaller cost (v_{ij}^t) and more required resources (p_i^{\max} and b_i^{\max}) have higher priorities to be assigned to cloudlet $c_j \in C$. G-OAMC checks the availability of available resources in cloudlet c_j for all the unassigned applications in the sorted list \check{U}_j . If there exists enough capacity, the application is assigned to cloudlet c_j . Subsequently, G-OAMC updates the capacity of the assigned cloudlet to reflect the available resources in that cloudlet (lines 20-23). However, if the current time slot is not a multiple of w , all applications are retained assigned to their current cloudlets (lines 24-26). Finally, G-OAMC returns the final assignments.

6 EXPERIMENTAL EVALUATION

We conduct a set of experiments to evaluate the effectiveness of our proposed computation offloading approaches when users are mobile. In this section, we describe the experimental setup and analyze the experimental results.

6.1 Experimental Setup

We compare the performance of S-OAMC and G-OAMC with the following approaches to assess their efficiency.

- IP: We use our IP model, presented in equations (2-8), as a benchmark. This IP is implemented using IBM ILOG Concert Technology API for C++ [36] to provide optimal solutions when possible.
- Best Fit Decreasing (BFD): This approach first sorts the applications in decreasing order of required resources including processing speed and bandwidth. Then, each application is offloaded to the nearest cloudlet whose capacity is sufficient.

TABLE 2: Experiment Scenarios

Exp.	# application	# cloudlets	# T	# w
1	60	12	20	5
2	90	15	20	5
3	120	20	20	5
4	150	25	20	5
5	210	35	20	5

- Online Assignment of Mobile applications to Cloudlets (OAMC): This approach [35] is an earlier version of S-OAMC, but it is not a sampling-based approximation method, and it does not use ML for prediction.
- Sampling-based Online Assignment of Mobile applications to Cloudlets Without using Prediction (S-OAMC-WP): This approach uses an idea similar to S-OAMC; however, the concept of prediction is not incorporated in S-OAMC-WP.

The algorithms are implemented in C++, and the experiments are conducted on a desktop PC with 2.67 GHz Intel Core i5-480M with 4GB RAM.

For mobile applications, to determine the coordinates of users' locations at each time slot, we use three different datasets with several mobility patterns, including driving, walking, and riding on subway trains, trolleys, and buses, to evaluate the performance of our proposed methods over different mobility patterns. The datasets are: 1) *DACT* [37], which contains about 13 hours of driving records; 2) The *Disney World (Orlando)* traces [38], which includes mainly walking and occasionally riding trolleys obtained from volunteers in Disney World, Florida; and 3) The *New York City* traces [38], where obtained from volunteers living in Manhattan or its vicinity that their means of travel include riding subway trains, buses, and walking. Each dataset constitutes one third of the total coordinates of the applications. Since each represents different geographical locations, we project coordinates of Disney World (Orlando) and The New York City traces to the same geographic area as DACT for consistency. Some coordinates were in Geodetic Coordinate system and were converted to the Cartesian to achieve better prediction accuracy.

We model the processing speed of applications using Microsoft Azure VM workload [39], where the CPU utilization traces of VMs are available across multiple time slots. We then scale these values appropriately for the computation requirements of the applications at each time slot (ω_i^t).

As for bandwidth requirements of applications (b_i^t), we use bandwidth traces measurements in 4G networks along several city routes [40]. The measurement guarantees appropriate download speeds through a 100Mbps Ethernet connection. We scale these trace values for bandwidth requirements of applications in edge computing. We also generate the code and the metadata of the applications at each time slot (md_i^t) based on the bandwidth values.

To model intermediate data of applications at each time slot (id_i^t), we use *GWA-T-12* trace [41] that contains the performance metrics of 1750 VMs of a mid-size datacenter managed by Bitbrains, which is a service provider. We use the Memory usage specification of the trace as id_i^t .

For cloudlets, each is located randomly within all posi-

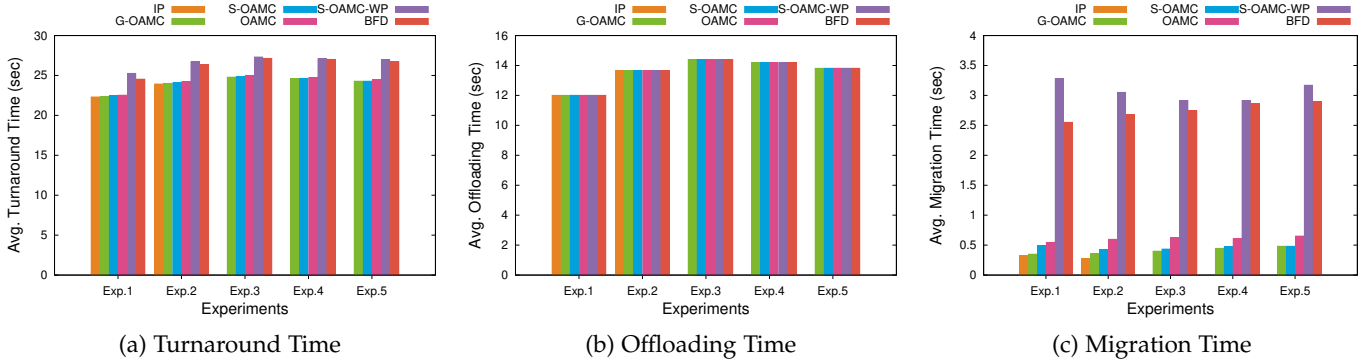


Fig. 3: Performance Analysis - Turnaround Time Components (*IP was not able to determine any solution for Exp. 3-5 in feasible time, and thus, there are no bars in the plots for those cases)

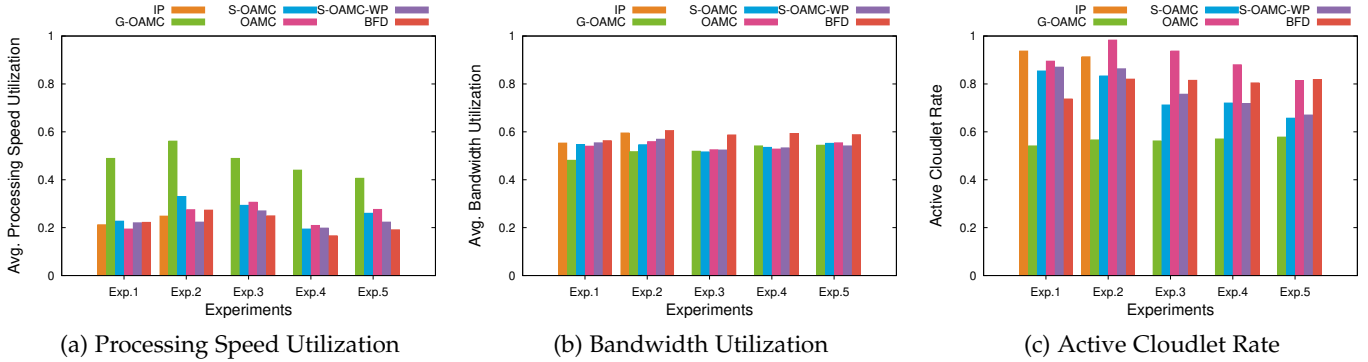


Fig. 4: System Utilization (*IP was not able to determine any solution for Exp. 3-5 in feasible time, and thus, there are no bars in the plots for those cases)

tions of the mobile applications. We use uniform distribution to obtain the coordinates of their locations. We assume each cloudlet has a total processing speed equivalent to the cumulative processing speed of $U(1,20)$ number of VMs, where each VM possesses $U(25000, 50000)$ MHz processing speed and has $U(5000, 10000)$ Mbps bandwidth.

We consider five scenarios for the experiments. Table 2 summarizes these scenarios. The values of γ and θ are set to 0.3 and 3×10^8 , respectively. We set the value of ϵ to 0.9 and η to 10 in all experiments, unless otherwise noted.

For matrix completion, we consider 20 applications as the training set. These applications are in addition to the number of applications in each experiment shown in Table 2. For these applications, we have their past specifications data over 20 time slots that are used to predict the specifications of other similar applications in the experiment. In Section 6.3, we evaluate the accuracy of matrix completion, and we consider the applications in each experiment as the test set to compare the predicted values by matrix completion against their real values and measure the accuracy of matrix completion accordingly.

6.2 Comparative Analysis

We use several metrics such as turnaround, offloading time, migration time, migration rate, runtime, and mobility patterns to compare our proposed approaches S-OAMC and G-OAMC with IP, BFD, OAMC, and S-OAMC-WP. Since the problem is NP-hard, solving the IP optimally is intractable. We set 60 minutes as a maximum feasible time to get the IP

results by the solver [36]. However, the solver cannot reach optimal results within 60 minutes for most cases.

Comparative Analysis on Turnaround Time. Our optimization criterion is to minimize the turnaround time of mobile applications. The average turnaround time of applications per time slot is presented in Fig. 3a, which is measured in seconds. The results show that S-OAMC, G-OAMC, and OAMC obtain turnaround time close to the optimal results obtained by IP, and much lower than those of S-OAMC-WP and BFD in all the experiments. Since S-OAMC-WP and BFD are not prediction-based methods, they have the worst turnaround time for the applications. BFD obtains slightly better results compare to S-OAMC-WP due to its assignment strategy to the nearest cloudlet, which leads to lower number of migrations in consecutive time slots. The results obtained by S-OAMC, G-OAMC, and OAMC are very close. Note that IP cannot solve the problem for experiments 3, 4, and 5, which shows intractability of the problem and the scalability issue of IP. We now investigate each component of the turnaround time to better analyze the performance of each approach.

Comparative Analysis on Offloading Time and Computation Time. Fig. 3b shows the average offloading time of applications per time slot. The results do not show a notable difference in offloading time among all the approaches for each experiment. This is due to the fact that the offloading time depends on id , b , θ , and f_d^t , where id and b are specifications of the applications and do not depend on the

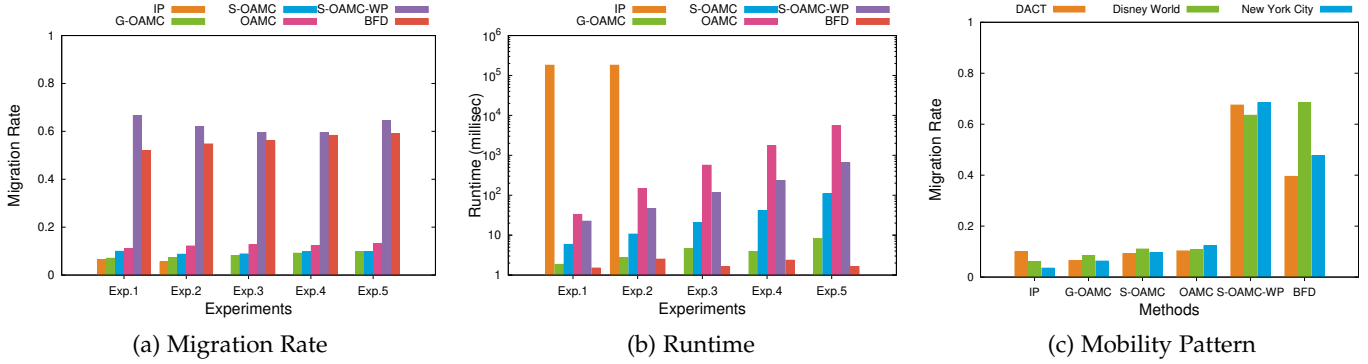


Fig. 5: Performance Analysis (*IP was not able to determine any solution for Exp. 3-5 in feasible time, and thus, there are no bars in the plots for those cases)

type of approaches, while θ is a constant value. Therefore, the difference in the offloading time comes only from f_d^t , the distances of applications and cloudlets. Since all approaches choose nearby cloudlets, their impacts on the offloading time is not significant. The difference in the specifications and distances in each experiment leads to different offloading time for the experiments.

Similarly, the computation time depends on parameters ω and p . Since these parameters belong to application specifications, the computation time results are the same for all approaches when using any of the approaches.

Despite the same computation time and offloading time obtained by the approaches, the assignments obtained by them are different. Therefore, we present the utilization of processing speed, bandwidth, and active cloudlets per time slot in Figs. 4a, 4b, and 4c, respectively. The utilizations are obtained based on the ratio of utilized capacity (e.g., allocated processing speed) to the full capacity for each cloudlet. The active cloudlet ratio shows the ratio of non-idle cloudlets to all cloudlets at each time slot. These results show that the selection of the applications for the assignment to cloudlets is different. G-OAMC has the lowest active cloudlet rate because it tries to put more applications in the current cloudlet if possible, otherwise it chooses the next cloudlet. Therefore, a lower number of cloudlets become active. OAMC has a high active cloudlet rate compared to S-OAMC. This is due to the fact that OAMC does not restrict the application set by sampling, and thus more cloudlets become active. OAMC, S-OAMC, and S-OAMC-WP achieve close utilization for processing speed and bandwidth because they utilize similar dynamic programming approaches. The slight differences arise from sampling and prediction.

Comparative Analysis on Migration Time. Fig. 3c shows the average migration time of applications per time slot. Since IP provides the optimal solution in terms of turnaround time, it has the minimum migration time too considering that offloading time and computation time are almost the same in all approaches. G-OAMC and S-OAMC obtain migration time very close to that of IP. G-OAMC chooses a cloudlet and assigns applications until it reaches the cloudlet capacity, and thus, it uses fewer cloudlets. Therefore, it requires the least number of migration, and consequently, it obtains low migration time compared to

other non-optimal approaches. OAMC results in higher migration time than S-OAMC since S-OAMC uses sampling in DP-APC. Sampling uses sorting of the applications based on the expected cost of assignment (v_{ij}^t) to select a collection of potential applications to be assigned to a cloudlet. This empowers S-OAMC to obtain better assignments that lead to lower migration time than OAMC.

S-OAMC-WP performs the worst among all approaches since it does not utilize prediction, and it determines the best cloudlet for offloading at each time slot, irrespective of future dynamics. BFD is a greedy approach, and similar to S-OAMC-WP, does not consider uncertainties in future demands, while it assigns applications to their near cloudlet, which leads to less migration requirement in consecutive time slots. As a result, it achieves a better migration time than S-OAMC-WP.

Comparative Analysis on Migration Rate. We compare the migration rate of the applications per time slot obtained by each method. The results are presented in Fig. 5a and show that IP obtains lower migration rates, even though minimizing the migration rate is not a direct objective of our IP. However, we believe that approaches with lower migration rates will result in closer to optimal turnaround time due to the decrease in their migration time. G-OAMC and S-OAMC lead to comparable low migration rates compared to the results of IP and lower migration rates compared to other approaches. OAMC obtains a slightly higher migration rate than S-OAMC since it does not utilize sampling. As expected, S-OAMC-WP and BFD obtain significantly higher migration rates as they are not prediction-based approaches.

Comparative Analysis on Runtime. Fig. 5b shows the runtime of the approaches on a logarithmic scale. The runtime is measured in milliseconds, and shows the average time needed to find a solution using an approach. This is to evaluate the time complexity of the approaches and their scalability. The results show that BFD performs the fastest compared to other approaches since it is a trivial algorithm with a lower time complexity than others. IP fulfills its time limit in Experiments 1, 2, and obtains the worst running time as we expect due to the intractability of our NP-hard problem. G-OAMC is a greedy approach, and it is the fastest approach after BFD.

S-OAMC obtains a polynomially scalable running time, and it finds the results in less than 100 milliseconds. Com-

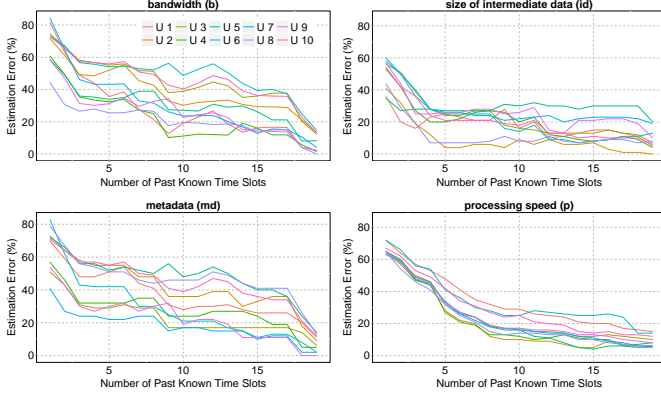


Fig. 6: Accuracy of Matrix Completion

pared to OAMC, it has more than one order of magnitude time reduction (except for Experiment 1). OAMC achieves the highest running time after IP. The runtime of OAMC is about 5500 milliseconds for Experiment 5. This shows the impact of our sampling approach in S-OAMC. The runtime of S-OAMC-WP is about 600 milliseconds for the largest experiment (Experiment 5) since it does not use any predictions, and it has to find assignments at each time slot.

Comparative Analysis on Mobility Patterns. We now evaluate the migration rates for different mobility patterns in our datasets, DACT, Disney World, and New York City. We analyze the results of all approaches for each dataset in Experiment 1. As shown in Fig. 5c, migration rates differ in each dataset and across different approaches. This is due to the fact that each approach obtains a unique assignment, while the mobility pattern of users in each time slot is different. Even though IP obtains the minimum overall migration rate for Experiment 1 (Fig 5a), it results in a higher migration rate for DACT dataset than its closest rival, G-OAMC. This is due to the fact that DACT dataset stores the driving records, and users move much faster with significant changes in their locations. Therefore, IP minimizes the turnaround time, while uses more migration in DACT, compared to G-OAMC. IP obtains a fewer migration rates for both Disney World and New York City datasets as walking is the users' main means of travel. As expected, S-OAMC-WP and BFD obtain the worst results for all datasets.

6.3 Precision Analysis

We now show the performance of our prediction method and its accuracy in estimating various specifications. In the experiments, we change the number of past known time slots from one to 19 in increments of one. This helps us to evaluate the impact of history (previous known time slots) on the estimation accuracy of matrix completion. Since we cannot show the results for all the applications and all the specifications due to lack of space, we present the results for a few selected ones. In Fig. 6, the results for ten representative applications over four specifications are shown. As can be seen, when the number of past known time slots is low, the accuracy of matrix completion is also low. However, with the increase in the number of known time slots, we see significant improvement in the estimation

accuracy. Moreover, the amount of accuracy changes from specification to specification. It indicates the direct impact of specifications' patterns on the performance of the matrix completion.

6.4 Sensitivity Analysis

In this section, we present sensitivity analysis with respect to window size (w), sample size (η), and ϵ to show their impacts on the results. In each analysis, we fix the other parameters to study the changes caused by a single parameter and analyze how sensitive the approaches are to such a change.

Sensitivity Analysis on ϵ . In DP-APC, ϵ is the parameter that determines the accuracy of results and size of the rounding. We conduct two sensitivity analysis on ϵ for runtime and migration time considering 210 mobile applications, 35 cloudlets, $\eta = 50$, and $T = 20$. The results are presented in Fig. 7a and 7b, respectively. Note that only DP-based approaches are sensitive to the value of ϵ . We only show the results of IP, BFD, and G-OAMC to present their obtained values.

Fig. 7a shows as ϵ increases, the runtime decreases. This is due to the fact that as ϵ increases, the value of λ increases in DP-APC, leading to a decrease in the maximum rounded value (r^*) and, consequently, reducing the size of the dynamic programming array A . Therefore, the runtime decreases for all DP-based approaches (S-OAMC, OAMC, and S-OAMC-WP). However, changing the value of ϵ does not have a visible impact on migration time. This is due to the fact that the migration rate is not directly correlated with the value of ϵ . Therefore, choosing a higher value of ϵ is preferable since it leads to a lower running time without increasing the migration time.

Sensitivity Analysis on Sample Size (η). For this experiment, we consider 210 mobile applications, 35 cloudlets, $\epsilon = 0.9$, and $T = 20$. We perform sensitivity analysis on η for runtime and migration time and present the results in Fig. 7c and 7d, respectively. Only S-OAMC and S-OAMC-WP react to this sample size change, and we present the fixed results of other approaches too.

Fig. 7c shows as the sample size η increases, the runtime of both S-OAMC and S-OAMC-WP increases. By increasing the sample size, the size of the dynamic programming array A becomes larger in DP-APC. This leads to an increase in runtime of these approaches. As shown in Fig. 7d, increasing the sample size means the collection of potential applications that can be assigned to the cloudlet grows, and therefore, more cloudlets are active. With a higher number of active cloudlets, migration rate and migration time increase, consequently.

Sensitivity Analysis on Window Size (w). Finally, we conduct sensitivity analysis on the window size for turnaround time and migration rate considering 60 mobile applications, 12 cloudlets, and $T = 20$, $\epsilon = 0.9$, and $\eta = 10$. The results are presented in Fig. 7e and 7f, respectively.

Fig. 7e shows that with an increase in the window size, the turnaround time obtained by S-OAMC, G-OAMC, and OAMC is decreased. This is due to the fact that: first, using higher value of w , these approaches have more information

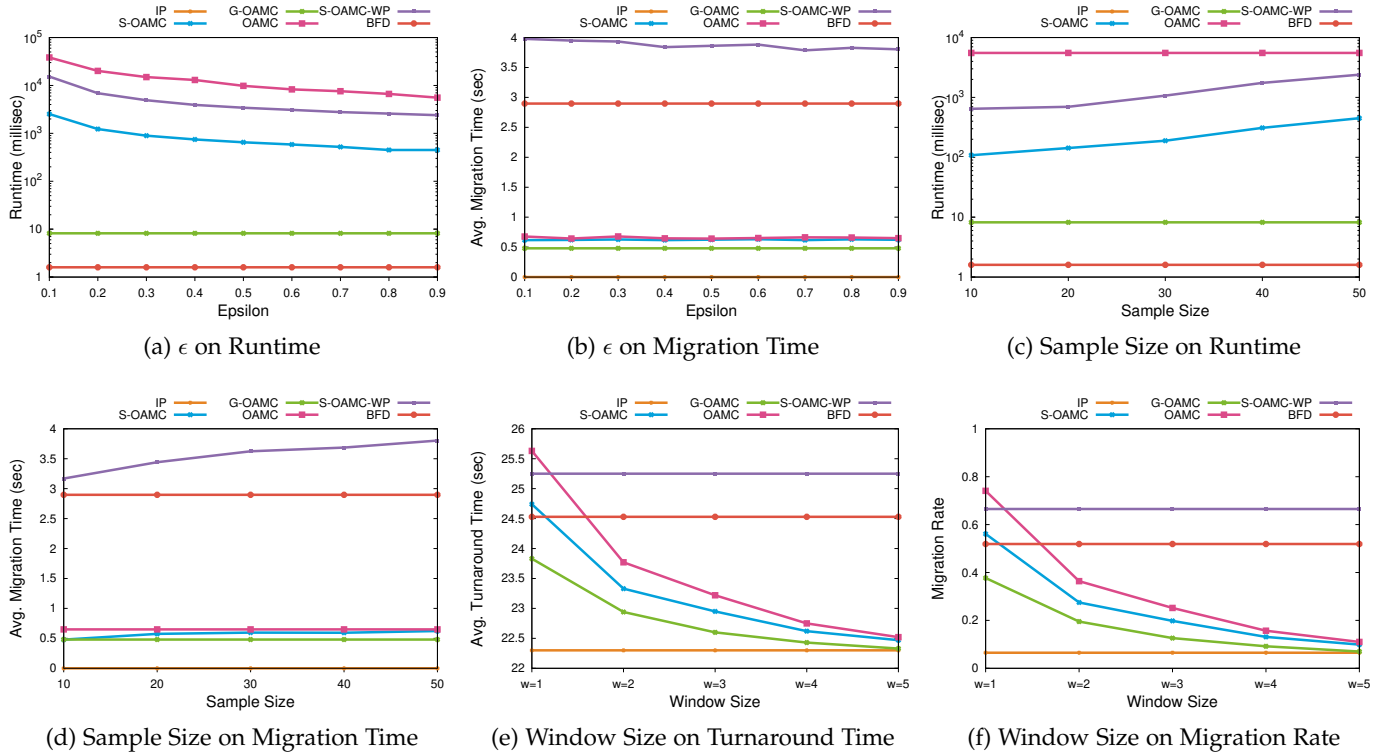


Fig. 7: Sensitivity Analysis

about future time slots, and thus, they can make better offloading decisions, which lead to reduced turnaround time; second, the applications require less migration rate using higher value of w , and consequently, less migration time. These results are shown in 7f. The results of IP, BFD, and S-OAMC-WP do not change as they do not use the concept of w .

To sum up, the results show that our proposed approaches S-OAMC and G-OAMC are efficient in finding offloading decisions fast and close to the optimal turnaround time. They also obtain lower migration rates than OAMC, S-OAMC-WP, and BFD.

7 CONCLUSION

Computation offloading to cloudlets augments the capabilities of mobile devices and ensures lower-latency services. However, a decision on computation offloading concerning uncertainties such as user mobility and dynamic changes is significantly challenging. In this paper, we addressed this challenge by proposing two novel computation offloading approaches; a sampling-based dynamic programming solution called S-OAMC; and a fast greedy-based solution called G-OAMC. Both approaches utilize machine learning-based predictions generated by the matrix completion method to obtain smart decisions on offloading in order to minimize the turnaround time of the offloaded applications. Experimental evaluations show that our proposed approaches find near-optimal turnaround time in a reasonable time, while they are highly scalable. Moreover, they reduce the number of migrations significantly compared to other non-optimal approaches. For future work, we plan to investigate the energy utilization of devices and cloudlets and propose efficient energy-aware offloading methods while guaranteeing low turnaround time.

ACKNOWLEDGMENTS

This research was supported in part by NSF grant CNS-1755913.

REFERENCES

- [1] Statista, "Forecast number of mobile devices worldwide from 2019 to 2023." Available: <https://www.statista.com/statistics/245501/multiple-mobile-device-ownership-worldwide/>, 2020. Accessed: 2020-03-02.
- [2] M. Satyanarayanan, V. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, 2009.
- [3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.
- [4] J. L. D. Neto, S.-Y. Yu, D. F. Macedo, J. M. S. Nogueira, R. Langar, and S. Secci, "ULOOF: A user level online offloading framework for mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 17, no. 11, pp. 2660–2674, 2018.
- [5] M. Chen, S. Guo, K. Liu, X. Liao, and B. Xiao, "Robust computation offloading and resource scheduling in cloudlet-based mobile cloud computing," *IEEE Transactions on Mobile Computing*, vol. 20, no. 05, pp. 2025–2040, 2021.
- [6] D. Bhatta and L. Mashayekhy, "Generalized cost-aware cloudlet placement for vehicular edge computing systems," in *Proc. of the 11th IEEE Intl. Conf. on Cloud Computing Technology and Science*, pp. 159–166, 2019.
- [7] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *Proc. of the IEEE Intl. Symp. on Info. Theory*, pp. 1451–1455, 2016.
- [8] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *Proc. of the 2nd ACM/IEEE Symp. on Edge Computing*, pp. 1–13, 2017.
- [9] N. Sharghivand, F. Derakhshan, L. Mashayekhy, and L. Mohammadkhanli, "An edge computing matching framework with guaranteed quality of service," *IEEE Transactions on Cloud Computing (TCC)*, 2020 (in press), pp. 1–14, 2020.
- [10] L. Yang, B. Liu, J. Cao, Y. Sahni, and Z. Wang, "Joint computation partitioning and resource allocation for latency sensitive applications in mobile edge clouds," *IEEE Transactions on Services Computing*, 2019.

- [11] W. Ma, X. Liu, and L. Mashayekhy, "A strategic game for task offloading among capacitated uav-mounted cloudlets," in *Proc. of the IEEE Intl. Congress on Internet of Things*, pp. 61–68, 2019.
- [12] Q. Li, S. Wang, A. Zhou, X. Ma, A. X. Liu, et al., "Qos driven task offloading with statistical guarantee in mobile edge computing," *IEEE Transactions on Mobile Computing*, 2020.
- [13] J. Wang, W. Wu, Z. Liao, A. K. Sangaiah, and R. S. Sherratt, "An energy-efficient off-loading scheme for low latency in collaborative edge computing," *IEEE Access*, vol. 7, pp. 149182–149190, 2019.
- [14] W. Ma and L. Mashayekhy, "Truthful computation offloading mechanisms for edge computing," in *Proc. of the 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom 2020)*, pp. 199–206, 2020.
- [15] T. Bahreini and D. Grosu, "Efficient placement of multi-component applications in edge computing systems," in *Proc. of the 2nd ACM/IEEE Symp. on Edge Computing*, pp. 1–11, 2017.
- [16] 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*, vol. 27, no. 3, pp. 1272–1288, 2019.
- [17] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 26–35, 2017.
- [18] C. Zhang and Z. Zheng, "Task migration for mobile edge computing using deep reinforcement learning," *Future Generation Computer Systems*, vol. 96, pp. 111–118, 2019.
- [19] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2333–2345, 2018.
- [20] L. Özbakir, A. Baykasoğlu, and P. Tapkan, "Bees algorithm for generalized assignment problem," *Applied Mathematics and Computation*, vol. 215, no. 11, pp. 3782–3795, 2010.
- [21] E. Candes and B. Recht, "Exact matrix completion via convex optimization," *Communications of the ACM*, vol. 55, no. 6, pp. 111–119, 2012.
- [22] T. Hastie, R. Mazumder, J. D. Lee, and R. Zadeh, "Matrix completion and low-rank SVD via fast alternating least squares," *The Journal of Machine Learning Research*, vol. 16, no. 1, pp. 3367–3402, 2015.
- [23] R. M. Bell, Y. Koren, and C. Volinsky, "The bellkor 2008 solution to the netflix prize," *Statistics Research Department at AT&T Research*, 2008.
- [24] S. M. Nabavinejad, L. Mashayekhy, and S. Reda, "ApproxDNN: Incentivizing DNN approximation in cloud," in *Proc. of the 20th IEEE/ACM Intl. Symp. on Cluster, Cloud and Internet Computing (in press)*, pp. 639–648, 2020.
- [25] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc. of the Intl. Conf. on Computational Statistics*, pp. 177–186, Springer, 2010.
- [26] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.
- [27] R. H. Keshavan, S. Oh, and A. Montanari, "Matrix completion from a few entries," in *Proc. of the IEEE Intl. Symp. on Information Theory*, pp. 324–328, 2009.
- [28] P. Jain, P. Netrapalli, and S. Sanghavi, "Low-rank matrix completion using alternating minimization," in *Proc. of the ACM Symp. on Theory of computing*, pp. 665–674, 2013.
- [29] S. R. Becker, E. J. Candès, and M. C. Grant, "Templates for convex cone problems with applications to sparse signal recovery," *Mathematical programming computation*, vol. 3, no. 3, p. 165, 2011.
- [30] P. Kanjilal and S. Palit, "Modelling and prediction of time series using singular value decomposition and neural networks," *Computers & electrical engineering*, vol. 21, no. 5, pp. 299–309, 1995.
- [31] L. Fleischer, M. X. Goemans, V. S. Mirrokni, and M. Sviridenko, "Tight approximation algorithms for maximum general assignment problems," in *Proc. of the 17th annual ACM-SIAM Symp. on Discrete Algorithm*, pp. 611–620, 2006.
- [32] U. Feige and J. Vondrak, "Approximation algorithms for allocation problems: Improving the factor of $1-1/e$," in *Proc. of the 47th Annual IEEE Symp. on Foundations of Computer Science*, pp. 667–676, 2006.
- [33] D. P. Williamson and D. B. Shmoys, *The design of approximation algorithms*. Cambridge university press, 2011.
- [34] W. B. Powell, *Approximate Dynamic Programming: Solving the curses of dimensionality*, vol. 703. John Wiley & Sons, 2007.
- [35] E. Farhangi Maleki and L. Mashayekhy, "Mobility-aware computation offloading in edge computing using prediction," in *Proc. of the 4th IEEE Intl. Conf. on Fog and Edge Computing*, pp. 69–74, 2020.
- [36] IBM, "Concert Technology version 12.1 C++ API Reference Manual." Available: <ftp://public.dhe.ibm.com/software/websphere/ilog/docs/optimization/cplex/refcpcplex.pdf>, 2009. Accessed: 2019-05-25.
- [37] S. Moosavi, B. Omidvar-Tehrani, and R. Ramnath, "Trajectory annotation by discovering driving patterns," in *Proc. of the 3rd ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics*, pp. 1–4, 2017.
- [38] I. Rhee, M. Shin, S. Hong, K. Lee, S. J. Kim, and S. Chong, "On the levy-walk nature of human mobility," *IEEE/ACM transactions on networking*, vol. 19, no. 3, pp. 630–643, 2011.
- [39] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *Proc. of the 26th Symposium on Operating Systems Principles*, pp. 153–167, 2017.
- [40] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alfance, T. Bostoen, and F. De Turck, "HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks," *IEEE Communications Letters*, vol. 20, no. 11, pp. 2177–2180, 2016.
- [41] S. Shen, V. van Beek, and A. Iosup, "Statistical characterization of business-critical workloads hosted in cloud datacenters," in *Proc. of the 15th IEEE/ACM Intl. Symp. on Cluster, Cloud and Grid Computing*, pp. 465–474, 2015.

BIOGRAPHIES



Erfan Farhangi Maleki is currently a Ph.D. student in the Department of Computer and Information Sciences at the University of Delaware. He received his B.S. degree in computer engineering-software from Shahid Beheshti University in 2015 and his M.Sc. degree in computer engineering-software from Isfahan University of Technology in 2018. His research interests include edge computing, cloud computing, and distributed systems. He is a student member of the IEEE.



Lena Mashayekhy is an assistant professor in the Department of Computer and Information Sciences at the University of Delaware. Her research interests include edge/cloud computing, data-intensive computing, Internet of Things, and algorithmic game theory. Her doctoral dissertation received the 2016 IEEE TCSC Outstanding PhD Dissertation Award. She is also a recipient of the 2017 IEEE TCSC Award for Excellence in Scalable Computing for Early Career Researchers. She has published more than forty

peer-reviewed papers in venues such as IEEE Transactions on Parallel and Distributed Systems and IEEE Transactions on Cloud Computing. She is a member of the IEEE and ACM.



Seyed Morteza Nabavinejad is a Post-Doctoral Research Fellow at the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran. He received the B.Sc. degree in Computer Engineering from Ferdowsi University of Mashhad, and the M.Sc. and Ph.D. degrees from Sharif University of Technology in 2011, 2013, and 2018, respectively. He has published several peer-reviewed papers in venues such as IEEE Transactions on Cloud Computing, IEEE Transactions on Big Data, CAL, DATE, and

SAC. His research interests include big data processing, cloud computing, green computing, and approximate computing.