Mobility-aware computation offloading in edge computing using prediction

Erfan Farhangi Maleki
Department of Computer and Information Sciences
University of Delaware
Newark, Delaware, USA
Email: erfanf@udel.edu

Lena Mashayekhy
Department of Computer and Information Sciences
University of Delaware
Newark, Delaware, USA
Email: mlena@udel.edu

***Abstract*—A key use case of edge computing is computation offloading that augments the capabilities of resource-constrained mobile devices by conserving their energy consumption and reducing latency of their applications. Edge computing resources, called cloudlets, are resource-rich computing infrastructures nearby users that aim at mitigating the overload of mobile devices and providing low-latency services. A main challenge in computation offloading to cloudlets is how to assign mobile applications to cloudlets efficiently such that the assignment captures the mobility inherent of mobile devices and leads to minimum latency during runtime of the applications. We address this problem by proposing a novel offloading approach that considers dynamics of mobile applications including mobility and changing specifications, and fully assigns applications to cloudlets, while minimizing their turnaround time (latency and execution time). We first formulate the problem as an integer programming model to minimize the turnaround time of mobile applications. This problem is an NP-hard problem. To tackle the intractability, we design a computation offloading algorithm, called OAMC, utilizing future specifications of mobile applications to obtain smart mobility-aware offloading decisions based on our prediction models. We conduct several experiments to evaluate the performance of our proposed approach. The results reveal that OAMC leads to near-optimal turnaround time in a reasonable running time.**

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

I. Introduction

The 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 mobile 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 [1].

Edge computing (EC) is a promising distributed paradigm allowing computation to be performed at the edge of the network [2]. Computation offloading in edge computing is the process of enhancing the capacity of mobile devices by migrating their computing tasks to EC [3]. Despite the advantages of computation offloading to EC in addressing response time requirements, battery life constraints, bandwidth usage, as well as data safety and privacy [2], spatio-temporal uncertainties due to user mobility bring the most challenging obstacles in providing these benefits.

EC consists of several cloudlets, where a cloudlet is a mini data center in the vicinity of mobile devices at the edge of the network. 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 and increased latency and turnaround time of the offloaded applications.

In this paper, we address this problem by designing an efficient mobility-aware computation offloading in order to minimize the turnaround time of mobile applications over their lifetime. Our proposed approach utilizes the prediction of unsteady specifications of mobile applications to obtain a smart offloading assignment that requires less migration of computation in the future and consequently obtains low turnaround time. We first formulate an optimal integer programming (IP) model for this problem. We then design an efficient online offloading approach, called OAMC, to obtain near-optimal turnaround time in a reasonable running time. OAMC is a dynamic programming-based approach providing a bounded turnaround time. The main contributions of this paper include the following:

- The feasibility of migration between cloudlets is considered as a technique for mobility management of mobile devices.
- Prediction algorithms are proposed to predict the specifications of mobile applications in the future (e.g., locations, bandwidth, processing speed requirements) to make better offloading decisions.
- A novel computation offloading approach, called OAMC, is proposed to minimize the turnaround time of the mobile applications, while reducing the number of migrations and the overall data movement.
- We evaluate the performance of the proposed approach

showing that OAMC is able to find near optimal turnaround time with lower migration rate.

In the next section, we provide an overview of existing work in this domain. We then formulate our problem in Section III. We describe our proposed approach, OAMC, in Section IV, and evaluate its properties in Section V. Finally, we summarize our results and present possible directions for future research.

II. Related Work

A group of studies focuses on the latency minimization in edge computing. Liu *et al.* [4] proposed an efficient one-dimensional search algorithm to find the optimal stochastic computation offloading policy. Sharghivand *et al.* [5] proposed efficient two-sided matching solutions to assign user applications to cloudlets considering their QoS and to determine dynamic pricing of the edge services. Bhatta and Mashayekhy [6] proposed a cost-aware cloudlet placement approach to guarantee minimum latency.

Several studies concentrate on setting a tradeoff between a mixture of objectives for computation offloading. Chen *et al.* [7] considered both energy consumption and latency, and proposed a game-theoretic approach for the computation offloading decision-making problem among multiple mobile devices. De Maio *et al.* [8] proposed ECHO, Edge Cloud Heuristic Offloading, to find a tradeoff solution between running time, user cost, and battery lifetime according to the user's preferences. Ma *et al.* [9] designed a game-theoretic solution for offloading tasks among a swarm of Unmanned Aerial Vehicles (UAVs) to reduce their energy consumption while guaranteeing QoS for users.

Mobility is a significant challenge in edge computing, and a few studies take into account user mobility while offloading. Bahreini *et al.* [10] proposed an offline IP 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.* [11] 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 possible migration. Bittencourt *et al.* [12] considered multiple application classes and proposed resource management policies to allocate resources between cloudlets and cloud to handle variable demand due to users mobility. Goncalves *et al.* [13] modeled an optimal VM placement and migration based on mobility prediction. However, as their model represents an NP-hard problem, they did not design any algorithm. Zhang *et al.* [14] 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.

To the best of our knowledge, our work is the only one that considers a general realistic case with multiple users and exploits efficient prediction methods to provide smart decisions on the computation offloading in order to reduce turnaround time of the applications considering user mobility.

III. System Model

The system model consists of a set of mobile applications and a set of cloudlets. A set of n mobile applications represented by $U = \{u_1, u_2, \ldots, u_n\}$ requires offloading, and a set of m cloudlets $C = \{c_1, c_2, \ldots, c_m\}$ 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 $1, \ldots, T$.

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, ω_i^t, p_i^t, b_i^t), where x_i^t and y_i^t indicate the location (latitude and longitude coordinates) of the mobile device $u_i \in U$ at time t; md_i^t indicates the code and the metadata of the application u_i that is either assigned or migrated to 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 $id_i^1 = md_i^1$ 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, ρ_j, β_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 the total bandwidth of the cloudlet.

Our objective is to minimize the overall turnaround time of computation offloading. The turnaround time consists of latency of communicating with a cloudlet when sending the application's code and data, migration time if needed, and execution time of the application on a cloudlet. Generally, latency is defined as follows:

$$l = \frac{d}{\theta} + \frac{s}{\beta}, \tag{1}$$

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 Haversine formula to calculate the distance as follows:

$$f_d^t(u_i, c_j) = \\ 2r_e \arcsin \sqrt{\sin^2(\tfrac{x_i^t - x_j}{2}) + \cos(x_i^t)\cos(x_j)\sin^2(\tfrac{y_i^t - y_j}{2})},$$

where r_e is the Earth radius (6,371 km). 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 considering their specifications. To optimally model this problem, we first define decision variables μ_{ij}^t and α_{ijk}^t, where $\mu_{ij}^t = 1$ 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:

$$\text{Minimize} \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{t=1}^{T}(\frac{f_d^t(u_i,c_j)}{\theta}+\frac{id_i^t}{b_i^t}+\frac{\omega_i^t}{p_i^t})\mu_{ij}^t+$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{m}\sum_{t=2}^{T}(\frac{f_d(c_j,c_k)}{\theta}+\frac{md_i^t}{b_i^t})\alpha_{ijk}^t \quad (2)$$

Subject to:

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

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

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

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

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

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

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 offloading assignment of the applications to cloudlets minimizing the turnaround time (offloading, computation, and migration). Our goal is to design an efficient algorithm to find such assignments on the fly.

IV. Online Offloading to Cloudlets

We design a prediction-based algorithm for Online Assignment of Mobile Applications to Cloudlets, called OAMC. The pseudo code of our proposed algorithm is presented in Algorithm 1. OAMC assigns every application to a cloudlet at each time slot such that it leads to a low turnaround time during the application lifetime (note that turnaround time is comprised of computation, offloading, and migration time). In doing so, it uses the near-future predicted specifications of the applications to make their offloading decisions.

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

1: **Input:** w: window size, γ : 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: $\{\bar{u_i}^{t+1}, .., \bar{u_i}^{t+w}\}$ =PREDICT$(\hat{U}_i^t, t, w)$
9: $V_i^t \leftarrow \infty$
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+=$\sum_{\tau=1}^{w}\gamma^\tau(\frac{f_d^{t+\tau}(\bar{u_i},c_j)}{\theta} + \frac{i\bar{d}_i^{t+\tau}}{\bar{b_i}^{t+\tau}} + \frac{\bar{\omega_i}^{t+\tau}}{\bar{p_i}^{t+\tau}})$
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+=$\frac{f_d^t(c_j,c_k)}{\theta} + \frac{md_i^t}{b_i^t}$
16: **if** $t \mod w = 0$ **then**
17: **do**
18: S^t = 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:** $\bigcup_{t=1}^{T}\hat{R}^t$

OAMC receives window size w and discount factor γ as inputs (line 1), where w indicates the number of future time slots taken into account, meaning that next predicted w specifications will be used for the offloading decisions. To reduce the impact of the later time slots on the offloading decision and decrease the prediction inaccuracy, OAMC uses the notion of γ. This is set to a value between 0 and 1, and a future time slot τ $(1 \leq \tau \leq w)$ has the impact of γ^τ. Therefor, the impact decreases by increasing the value of τ.

OAMC uses set $\hat{U}_i^t$ to maintain the specifications of application u_i from the first time slot to time slot t, which is initially set to the empty set (line 2), 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. Similarly, $\hat{R}_i^t$ maintains the cloudlet assigned to application u_i at time slot t, which is initially set to the empty set (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. For simplicity, we use notations with the hat operator (e.g., $\hat{U}$) to show the history and the bar operator (e.g., $\bar{U}$) to show the predicted values.

OAMC intends to minimize the total turnaround time by minimizing computation and offloading time when deciding on

the assignment of applications to cloudlets and by minimizing the total number of migrations required in the lifetime of the applications considering the future specifications of the applications. In doing so, OAMC uses the next w specifications and determines offloading assignments that do not require frequent reassignments and migrations leading to minimum total turnaround time. Therefore, OAMC uses predicted specifications of all $u_i \in U$ in next w time slots based on a history of specifications ($\hat{U}_i^t, \forall\, u_i$) that has been collected. It first updates $\hat{U}_i^t$ by adding u_i^t at time t to the set to include the most recent specification of application u_i (line 6). Then, it checks whether or not the current time slot is a multiple of w (line 7). That is done to ensure that the decision about the assignment is carried out every w time slots and the obtained assignment is valid for w time slots. Subsequently, function PREDICT, presented in Algorithm 2, outputs the predicted specifications of the applications in next w time slots (line 8).

In the next step, OAMC computes the expected cost (the expected turnaround time considering the discount factor) of assigning applications to cloudlets in a period of time that starts from the current time slot and continuous to next w time slots. It initializes all the cost values to infinity (line 9) and only updates the cost for cloudlets that are close to (or in the area of) the user. The cost is computed based on the actual specifications in the current time slot and the predicted ones in next w time slots assuming applications are assigned to same cloudlets during this period, and migration can only happen at time $t \geq 2$. The expected cost of assigning application u_i to cloudlet c_j at time slot t is represented by v_{ij}^t, which 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). By using γ, the impact of further time slots is decreased. 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. Additionally, OAMC checks whether or not cloudlet c_j is different from the currently assigned cloudlet to the application u_i. If it is different, OAMC adds up the migration cost to v_{ij}^t as well (lines 13-15); otherwise, there is no migration cost.

When the current time slot is a multiple of w (line 16), having the predicted future specifications and all the calculated expected costs, the next step of OAMC is to assign applications to cloudlets in order to minimize the total turnaround time. Hence, OAMC calls ASSIGN function (line 18) using 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, which is a set of offloading decisions that determines the assignment of applications to each cloudlet. For example, given 2 cloudlets and 2 applications $U = \{u_1, u_2\}$, $S^t = (\{\}, \{u_1, u_2\})$ indicates no application is assigned to cloudlet c_1, while both applications u_1 and u_2 are assigned to cloudlet c_2.

Algorithm 2 PREDICT($\hat{U}_i^t, t, w$)

1: **for all** $h = 1$ to w **do**
2: $\bar{x_i}^{t+h} = x_i^t + \frac{h}{t-1}(x_i^t - x_i^1)$
3: $\bar{id_i}^{t+h} = \frac{t \times id_i^t + (t-1) \times id_i^{t-1} + \ldots + id_i^1}{t + (t-1) + \ldots + 1}$
4: Calculate $\bar{y_i}^{t+h}$ similar to line 2
5: Calculate $\bar{md_i}^{t+h}$, $\bar{\omega_i}^{t+h}$, $\bar{p_i}^{t+h}$, $\bar{b_i}^{t+h}$ similar to line 3
6: $\bar{u_i}^{t+h} = (\bar{x_i}^{t+h}, \bar{y_i}^{t+h}, \bar{md_i}^{t+h}, \bar{id_i}^{t+h}, \bar{\omega_i}^{t+h}, \bar{p_i}^{t+h}, \bar{b_i}^{t+h})$
7: **Return:** $\bigcup_{h=1}^{w} \bar{u_i}^{t+h}$

OAMC converts S^t to $\hat{R}^t$, where $\hat{R}^t$ is a set corresponding to all $\hat{R}_i^t$ (i.e., $\hat{R}^t = \bigcup_{i=1}^{n} \hat{R}_i^t$), and it maintains the assigned cloudlets to all applications (line 19). Moreover, the algorithm removes the specifications of assigned applications from U^t and also updates the specification of the cloudlets in order 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 loop (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, OAMC returns the assignments of cloudlets to the applications in all time slots (line 25). More details about PREDICT and ASSIGN functions will be given next.

1) Forecasting Methods of OAMC: To predict future specifications of application u_i for the next w time slots, we propose PREDICT function, presented in Algorithm 2. The specification of the applications composed of x, y, md, id, ω, p, and b. Due to user mobility, x and y values are expected to change over time. However, data and processing requirements of applications are expected to remain close to their recent requirements, and the critical factor in determining the requirements is the type of applications. Therefore, we use two forecasting methods, Drift method to predict x and y, and Weighted Moving Average method to predict other parameters.

Drift method [15] allows changes (decrease or increase) over time in the amount of the forecasting parameter, where the amount of changes depends on the average change in the historical data. Therefore, it calculates the average change between the first and last observations, and then assumes every single future observation will be changed as much as that amount compared to the last observation. The forecasting value for x axis of application u_i is defined as follows:

$$\bar{x}_i^{t+h} = x_i^t + \frac{h}{t-1} \sum_{t'=2}^{t} (x_i^{t'} - x_i^{t'-1}) = x_i^t + \frac{h}{t-1}(x_i^t - x_i^1), \quad (9)$$

where $\bar{x}_i^{t+h}$ represents the value of the forecasting parameter at time $t+h$, assuming h is the number of time slots ahead, t is the current time slot, and the real values of forecasting parameter are known up to time slot t (i.e., x_i^1, x_i^2, ..., x_i^t) (line 2).

Weighted Moving Average [15] considers the recent observations more valuable and the old observations less valuable

by considering greater coefficients for the recent observations and smaller coefficients for the old observations. It assumes the value of a forecasting parameter is likely to be close to its recent values. Using this approach, the forecasting value for the id parameter of application u_i is presented in line 3. The forecasting value is computed based on t recent observations, such that the most recent observation is id_i^t, the second recent observation is id_i^{t-1} and so on.

The predicted values of other specifications of application u_i in next w time slots are calculated, and PREDICT function returns these values.

2) OAMC Assignment: OAMC calls ASSIGN function 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. In this paper, ASSIGN uses the local-ratio technique, adopted from [16] to provide $2+\epsilon$-approximate with a better running time. It receives the cost functions (V^t), specifications of the applications (U^t), specifications of the cloudlets (C) and returns $S = \bigcup_{j=1}^{m} S_j$ as a set of offloading assignments, where S_j is the set of assigned applications to cloudlet c_j.

V. Experimental Evaluation

We conduct a set of experiments to evaluate the effectiveness of our proposed approach in computation offloading of mobile applications.

Experimental Setup. We compare the performance of OAMC with the following methods to assess its 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++ [17].
- Best Fit Decreasing (BFD): This method first sorts the applications in decreasing order of required resources including processing speed and bandwidth. In fact, applications requiring higher processing speed and bandwidth come before other applications in the sorted list. Then, each application is offloaded to a cloudlet that has the smallest resources (processing speed and bandwidth), but enough for that application.
- OAMC Without using Prediction (OAMC-WP): This method uses the same assignment function as in OAMC. However, the concept of prediction is not incorporated in 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 the mobile applications, we use mobility traces of taxi cabs in San Francisco [18] to determine the coordinates of users' locations at each time slot. For the cloudlets, each is located randomly in a position within all positions of the mobile applications. We use uniform distribution to obtain the coordinates of their locations. For other specifications of the mobile applications at each time slot and cloudlet, we use a dataset [19] that contains the performance metrics of 1750 virtual machines of a mid-size datacenter managed by Bitbrains, which is a service provider. We assume each cloudlet has a total processing speed equivalent to the cumulative processing speed of *U(1,20)* number of virtual machines, where the statistics of the virtual machines are presented in [19]. Similarly, we use the processing speed usage statistic in the dataset as processing speed requirements of the applications. Other specifications of the mobile applications at each time slot and cloudlet are represented in Table I.

TABLE I: Statistics of each user u_i and cloudlet c_j

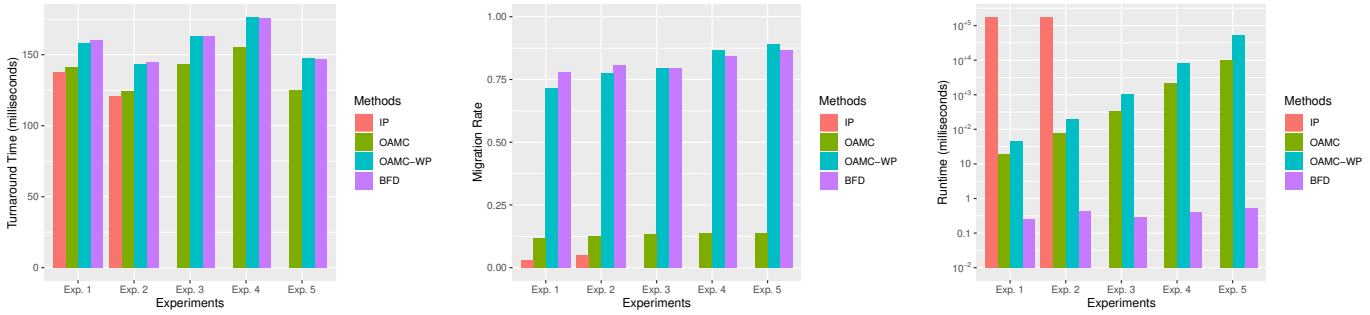
Spec.	**Mean**	**SDev**	**Dist.**
md_i^t (Mb)	$\mu =$ *U(8,48)*	$\sigma =$ *U(0,8)*	$\mathcal{N}(\mu, \sigma^2)$
id_i^t (Mb)	$\mu =$ *U(8,160)*	$\sigma =$ *U(0,16)*	$\mathcal{N}(\mu, \sigma^2)$
ω_i^t (M)	$\mu =$ *U(1,80)*	$\sigma =$ *U(0,20)*	$\mathcal{N}(\mu, \sigma^2)$
b_i^t (Mbps)	$\mu =$ *U(750,1500)*	$\sigma =$ *U(100,200)*	$\mathcal{N}(\mu, \sigma^2)$
β_j (Mbps)	-	-	*U(7500,15000)*

For the experiments, we consider five scenarios in 20 time slots and window size of 5, Exp 1 with 50 applications and 10 cloudlets, Exp 2 with 75 applications and 12 cloudlets, Exp 3 with 100 applications and 15 cloudlets, Exp 4 with 150 applications and 20 cloudlets, and Exp 5 with 200 applications and 30 cloudlets. The values of γ, θ, and ϵ are set to 0.3, 3×10^8, 0.5, respectively.

Comparative Analysis. We use turnaround, migration rate, and runtime as metrics to compare our proposed approach OAMC with IP, BFD, and OAMC-WP. Note that in some cases, IP does not have any results as it could not obtain a solution in a feasible time (60 minutes).

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. 1a, which is measured in milliseconds. The results show that OAMC obtains turnaround time close to the optimal results obtained by IP. Moreover, OAMC obtains a lower turnaround time than OAMC-WP and BFD in all the experiments. OAMC-WP and BFD that are not prediction-based methods have the worst turnaround time for the applications, and they lead to similar results. Note that IP is unable to solve the problem for experiments 3, 4, and 5, which shows IP lacks scalability.

We compare the migration rate (per application per time slot) obtained by each method. The results are presented in Fig. 1b. The results show that IP obtains lower migration rates, even though minimizing the migration rate is not the direct objective of our IP. However, we believe that methods with lower migration rates will result in closer to optimal turnaround time due to the decrease in their migration cost. OAMC leads to comparable low migration rates compared to IP and significantly lower migration rates compared to other approaches. OAMC-WP and BFD obtain high migration rates as they are not prediction-based methods.



(a) Turnaround Time (b) Migration Rate (c) Runtime

Fig. 1: Performance Analysis of OAMC

Fig. 1c shows the average runtime of the methods per time slot on a logarithmic scale. The runtime is measured in milliseconds. The results show that BFD performs the fastest compared to other methods since it is a trivial algorithm with a lower time complexity than others. IP obtains the worse running time as we expect due to the intractability of our NP-hard problem. OAMC obtains a polynomially scalable running time, and it finds the results in less than 10 sec for the largest experiment and below 0.1 sec for the smallest experiment. Moreover, the average runtime of OAMC-WP is about 52 sec for experiment 5 since it does not use any predictions, and it needs to find assignments in each time slot.

To sum up, the results show our proposed method OAMC results in close to optimal turnaround time, and it is polynomial in the size of the input. It also obtains significantly lower migration rates than OAMC-WP and BFD.

VI. Conclusion and Future Work

Computation offloading to cloudlets is a beneficial approach to augment the capabilities of mobile devices. However, a decision on computation offloading concerning mobility inherent of mobile devices is a significant challenge. In this paper, we addressed this challenge by proposing a novel computation offloading approach called OAMC, which utilizes dynamic predictions to obtain smart decisions on offloading. Experimental evaluations show that our approach finds close to optimal turnaround time in a reasonable running time. Moreover, it reduces the number of migrations significantly compared to other approaches. For the future work, we plan to enhance the running time and scalability of our approach.

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

References

[1] M. Satyanarayanan, V. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE pervasive Computing*, 2009.

[2] 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.

[3] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.

[4] 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 Information Theory*, pp. 1451–1455, 2016.

[5] N. Sharghivand, F. Derakhshan, and L. Mashayekhy, "Qos-aware matching of edge computing services to Internet of Things," in *Proc. of the IEEE 37th Intl. Performance Computing and Communications Conf.*, pp. 1–8, 2018.

[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. 1–8, 2019.

[7] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2015.

[8] V. De Maio and I. Brandic, "First hop mobile offloading of dag computations," in *Proc. of the 18th IEEE/ACM Intl. Symp. on Cluster, Cloud and Grid Computing*, pp. 83–92, 2018.

[9] 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.

[10] 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*, p. 5, 2017.

[11] 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.

[12] 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.

[13] D. Gonçalves, K. Velasquez, M. Curado, L. Bittencourt, and E. Madeira, "Proactive virtual machine migration in fog environments," in *Proc. of the IEEE Symp. on Computers and Communications*, pp. 742–745, 2018.

[14] 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.

[15] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*. OTexts, 2018.

[16] R. Cohen, L. Katzir, and D. Raz, "An efficient approximation for the generalized assignment problem," *Information Processing Letters*, vol. 100, no. 4, pp. 162–166, 2006.

[17] IBM, "Concert Technology version 12.1 C++ API Reference Manual." Available: ftp://public.dhe.ibm.com/software/websphere/ilog/docs/optimization/cplex/refcppcplex.pdf, 2009. Accessed: 2019-05-25.

[18] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser, "CRAWDAD dataset." Available: https://crawdad.org/epfl/mobility/20090224.

[19] 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.