Energy-Aware Application Placement in Mobile Edge Computing: A Stochastic Optimization Approach

Hossein Badri, *Student Member, IEEE,* Tayebeh Bahreini, *Student Member, IEEE,* Daniel Grosu, *Senior Member, IEEE,* and Kai Yang

Abstract—The Quality of Service (QoS) in Mobile Edge Computing (MEC) systems is significantly dependent on the application offloading and placement decisions. Due to the movement of users in MEC networks, an optimal application placement might turn into the least efficient placement in few minutes. Thus, it is crucial to take the dynamics of the system into account when designing application placement mechanisms. On the other hand, energy consumption of servers is a significant component of the cost of services in MEC systems and must also be considered in the design of the mechanisms. In this paper, we model the problem of energy-aware application placement in edge computing systems as a multi-stage stochastic program. The objective is to maximize the QoS of the system while taking into account the limited energy budget of the edge servers. To solve the problem, we design a novel parallel Sample Average Approximation (SAA) algorithm. We conduct an extensive experimental analysis to evaluate the performance of the proposed algorithm using real-world trace data.

Index Terms—Mobile edge computing, energy-aware application placement, quality of service, multi-stage stochastic programming, sample average approximation, parallel algorithms.

1 Introduction

Today, mobile devices are the primary tools for many applications in several domains such as communication, commerce, health, and entertainment. The increasing utilization of mobile devices necessitates coping with many of their limitations with respect to the processing power, memory size, disk capacity, and battery life [1]. Among all the technologies that were developed to alleviate these limitations of mobile devices, Mobile Cloud Computing (MCC) has received a significant attention [2], [3]. MCC enables the users of mobile devices to run some of their applications on the resource-rich servers available from clouds and datacenters. In MCC, the response time is a very important performance measure which is negatively impacted by the distance between mobile users and the cloud servers.

Mobile Edge Computing (MEC) [4], [5], [6], [7] has been recently introduced with the aim of reducing the response time of mobile applications. In MEC, some servers are located at the edge of mobile networks providing services to mobile users with a lower latency than in the case of the servers located in data centers. This has made MEC an attractive infrastructure for many applications, such as machine learning [8], [9], [10], vehicular computing [11], and Internet of Things (IoT) [12], [13].

One of the major challenges in MEC systems is how to efficiently place the mobile applications on edge servers. Due to the mobility of users, inefficient application placement might result in poor Quality of Service (QoS).

• Hossein Badri and Kai Yang are with the Dept. of Industrial & Systems Engineering, and Tayebeh Bahreini and Daniel Grosu are with the Dept. of Computer Science, Wayne State University, Detroit, MI, 48202.
E-mail: hossein.badri@wayne.edu, tayebeh.bahreini@wayne.edu, dgrosu@wayne.edu, kai.yang@wayne.edu.

In the last decade, the increased awareness of the environmental impacts of the energy consumption has motivated researchers to address energy consumption in distributed systems from different perspectives [11], [14], [15], [16]. In MEC systems, energy consumption of edge servers is an extremely important factor that has to be taken into account when placing applications on servers. Edge servers are expected to operate at a higher operating cost compared to the cloud servers, due to the fact that the cost per operation is highly dependent on the scale. It might not be quite easy to reduce the investment costs, but given the fact that energy consumption accounts for about 25% of the operating costs of cloud data centers [17], optimizing the energy consumption would be a promising way to reduce the operating costs. The share of the energy costs in the operating costs of the edge servers is expected to be higher than that of cloud data centers.

In order to reduce the energy consumption in a computing system, physical resources must be utilized efficiently. Therefore, the efficiency of application placement is a determinant factor in managing the energy consumption, and as a result the operating cost. Therefore, the application placement problem can be defined as follows. Given a set of heterogeneous edge servers and a set of user requests for executing applications, determine an assignment of applications to servers that maximizes the QoS for all users and takes into account the mobility of users, the energy budget and the availability of computational resources at the servers.

When it comes to developing efficient mechanisms for the application placement problem, it is crucial to take the mobility of users into account. An optimal placement based on the current location of users might turn into an extremely inefficient placement in few minutes due to the mobility of users across the network. Also, it might not be efficient to frequently relocate an application to a different server based on the location of the user. Since the future location of users is a nondeterministic parameter, an efficient application placement mechanism should be capable of taking this uncertainty into account when placing applications on servers.

In this paper, we develop a multi-stage stochastic programming method for the energy-aware application placement problem in MEC systems. Our objective is to maximize the total QoS of the system, while taking the energy budget of the edge servers into account. Our proposed method also takes the dynamics of the network into account when making placement decisions. To solve the problem efficiently, we design a parallel Sample Average Approximation (SAA) algorithm and perform an extensive experimental analysis using real-world data to evaluate the performance of the proposed method.

1.1 Our Contributions

This paper is an extension of our previous work on stochastic optimization of task placement in MEC systems [18]. In our previous work, we developed a multi-stage stochastic programming method for the application placement problem in MEC systems where the objective is to minimize the total cost, including, computation, communication, and relocation cost. In this paper, our aim is to maximize the total QoS of the system, i.e., the sum of the QoS of individual users who receive service. Here, the QoS of a user is defined based on the latency, that is, the QoS is inversely proportional to the distance between the user and the server that provides the service, and directly proportional to the size of the served request. Furthermore, we consider the limited energy budget of the edge servers, while in our previous work we took into account only the capacities of the edge servers' computational resources. We model the energy-aware application placement problem in edge computing systems as a multi-stage stochastic program, where each stage represents a time slot, and the location of users might change between time slots. The stochastic programming approach allows us to take the dynamics of the location of users into account when making application placement decisions. This is a significant issue in the design of efficient application placement methods in edge computing systems, and to the best of our knowledge, this is the first work on energy-aware resource allocation in MEC systems that takes into account the dynamics of the network in the decision making process.

In this paper, we consider two metrics in the objective of our proposed multi-stage stochastic model. We aim at maximizing the total QoS of the system, while taking the total relocation cost into account. Therefore, the first stage objective of the proposed model only considers the total QoS of the system, while the recourse function incorporates both the total QoS and the relocation cost.

As a constraint, we consider the energy budget of servers, that is, a server cannot be loaded beyond its energy budget. Exact evaluation of the recourse function (i.e., the cost of execution in the future time slots) in multi-stage stochastic optimization problems is very complex due to the large number of scenarios that have to be considered [19], [20].

To solve the multi-stage stochastic application placement problem, we use the Sample Average Approximation (SAA) method which is a promising approach for tackling the complexity caused by the number of scenarios. Having to solve multiple subproblems in the SAA method motivated us to leverage the computational power of multi-core systems and design a parallel SAA-based algorithm which obtains solutions to the multi-stage stochastic application placement problem in a reasonable amount of time. We also design a greedy algorithm to solve the integer optimization problem that needs to be solved in each of the phases of the proposed SAA-based parallel algorithm. This allows solving the problem in a reasonable amount of time. To evaluate the performance of the proposed algorithm, we conduct an extensive experimental analysis using real-world data.

1.2 Organization

The rest of the paper is organized as follows. In Section 2, we review the related work. In Section 3, we define the energy-aware application placement problem and formulate it as a multi-stage stochastic program. In Section 4, we present a parallel greedy SAA-based algorithm for solving the problem. In Section 5, we describe the experimental setup and discuss the experimental results. In Section 6, we conclude the paper and suggest possible directions for future work.

2 Related work

There exists several approaches for solving the application placement problem in cloud computing [21], [22], but they are not directly applicable in the context of MEC. The application placement problem in MEC has to consider several issues that were not present in the data-center or cloud computing settings. In MEC, mobile users may move to different locations after the initial application placement. Thus, an optimal application placement decision made at the time of receiving a request may not remain optimal for the whole duration of user's application execution. In addition to this, the availability of servers' resources may change over time. Therefore, an efficient application placement algorithm must be adaptive to this dynamic setting.

When it comes to performance metrics, the cost of services [23], [24], [25], the energy consumption [26], [27], [28], [29], and the quality of service [30], [31] are the most important metrics. Researchers have proposed various approaches for energy-aware resource allocation in cloud computing systems [32], [33], [34]. Some researchers also proposed task/workload consolidation techniques to reduce the energy consumption in cloud systems [35], [36]. Minimizing the total number of active servers is also a strategy for optimizing the energy consumption [37]. Readers can refer to Beloglazov et al. [38] and Hameed et al. [39] for more details discussions on energy-efficient cloud computing systems.

Researchers have tackled the dynamic nature of the MEC systems, which stems mainly from the mobility of users, using different approaches. Some researchers develop methods which do not require any information from future mobility of users [40], [41]. They first divide the long-term optimization problem into a series of one-shot problems, and focus on developing efficient algorithms to solve one-shot problems.

There are many nondeterministic parameters in edge computing systems which need to be taken into account in the design of an efficient application placement mechanism. Due to the movement of users in MEC networks, an optimal application placement might turn into the least efficient placement in few minutes. Thus, it is crucial to take into account the dynamic nature of the system when designing application placement mechanisms.

Markov Decision Processes (MDP) have been used by several researchers to model uncertainties in application placement problems in MEC. Wang et al. [42] presented an online algorithm for application placement in the context of MEC. They modeled the problem as an MDP and reduced the state space of the problem by deriving a new MDP model in which states are defined only based on the distance between users and servers. The authors showed that the distance-based MDP is a good approximation to the original problem. For one-dimensional mobility spaces, their proposed algorithm provides the optimal solution, while for two-dimensional spaces it gives a near optimal solution. Adopting a similar approach, Urgaonkar et al. [43] modeled the application placement problem as an MDP. To reduce the state space of the problem, they converted the problem into two independent MDP problems with separate state spaces. Then, they designed an online algorithm for the new problem that is provably cost-optimal. None of the MDP-based solutions presented above consider the energy budget of the servers when making allocation decisions.

Most variants of the MDP problems are known to be P-complete, that is, it is not possible to design highly efficient parallel algorithms to solve them unless all problems in P have such highly efficient parallel solutions [44]. This fact hinders the design of efficient parallel algorithms for finding quality solutions for the application placement problem modelled as an MDP, and the possibility of having fast MDP-based algorithms for solving the application placement problem. In this paper, we design a novel algorithm for solving the application placement problem that is not based on MDPs and is fast and suitable for parallelization, being able to exploit the huge computing power offered by the current and future multi-core computer systems. Furthermore, our proposed algorithm takes into account the energy budget of the edge servers when making placement decisions.

3 The Energy-Aware Application Placement Problem in MEC: Multi-stage Stochastic Programming Formulation

The MEC system considered here consists of a set of M edge servers, $\mathcal{S} = \{S_1, S_2, \ldots, S_M\}$, which provides services to a set of N users, $\mathcal{U} = \{U_1, U_2, \ldots, U_N\}$. The network model for the MEC system is shown in Figure 1. Our model is analogous to the network model considered in [45], except that it does not include cloud servers. In our model, edge

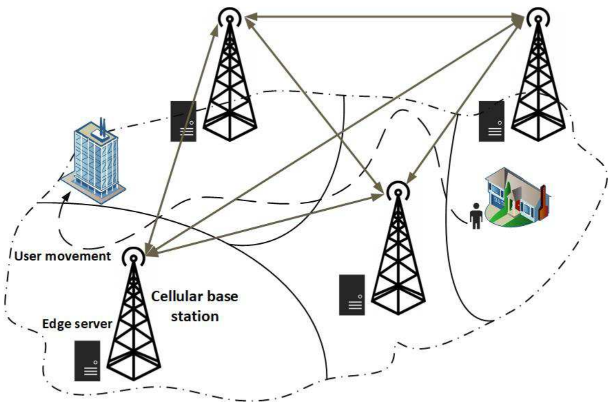


Fig. 1: MEC Network model.

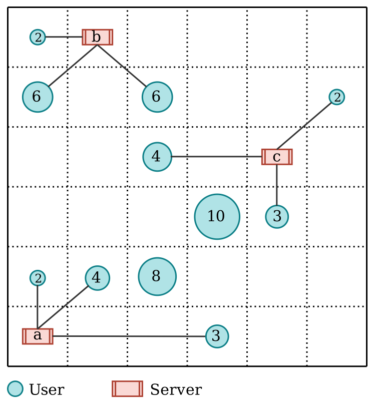
server S_j is co-located with a base station and is characterized by its computational capacity, Q_j, expressed in terms of the maximum number of unit-size containers that the server can host. The unit-size containers have a fixed resource configuration which is determined by the provider. User U_i requests to offload and execute an application (a single container) on the edge servers via 4G/5G/WiFi access networks. We assume that all towers are accessible to all users in the network. The size R_i of user U_i's requested container is expressed in terms of the equivalent number of unit-size containers needed to complete the application. User U_i also specifies the time τ_i needed to complete the execution of the application on a container of size R_i. We assume that each created container only serves the request from one user.

We consider a discrete time slotted system and assume that the placement decisions are made at periodic intervals that are called time slots. The location of a user is specified by its coordinates in a two-dimensional grid of cells. The locations of users do not change during one time slot, but a user can change its location between two time slots, that is, it can move into any of the neighboring cells or stay in the same cell. Our aim is to maximize the total QoS of the system while taking into account the energy budget and the computational capacities of the edge servers. We formulate the problem as a multi-stage integer stochastic program. The notation we use in our formulation is provided in Table 1.

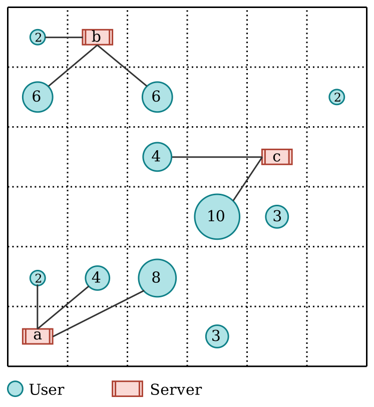
The objective of the multi-stage stochastic application placement problem is to maximize the quality of service of the system which is defined as the sum of the QoS of individual users who receive services while taking the relocation cost into account. Here we define the QoS, based on two important factors that determine the latency, the distance between user and server, and the size of the request. A user whose request is assigned to a nearby edge server is expected to have low latency, therefore experiencing a high QoS. Also, a large size request results in higher communication latency. Thus, the quality of service that user U_i receives from server S_j in time slot t is defined as,

$$QoS_{ij}^t = \frac{\gamma \cdot R_i}{d_{ij}^t} \tag{1}$$

where, d_{ij}^t is the distance between user U_i and server S_j in time slot t, and γ is a regulatory parameter. In other words, the *quality of service* QoS_{ij}^t that user U_i receives



(a) (AEU = 0.71, QoS = 190)



(b) (AEU = 0.93, QoS = 206.67)

Fig. 2: Energy-aware application placement: two placements with different values for AEU and QoS.

from server S_j in time slot t is inversely proportional to the distance between the user and the server that provides the service, and directly proportional with the size of the request. Here, we assume that the distance between users and servers is the Manhattan distance. Therefore, if user U_i is located in cell (a_u, b_u) and server S_j is located in cell (a_s, b_s) in time slot t, then the distance between the user and the sever is given by, $|a_u - a_s| + |b_u - b_s|$.

The *relocation cost*, denoted by $\rho_{j'j}$ is associated with changing the assignment of a user's request from server $S_{j'}$ to server S_j during the execution. Therefore, if the user's request is completely fulfilled by a single server, no relocation cost is incurred. The relocation cost is proportional to the distance between servers, which is the Manhattan distance. The locations of users in the current slot, t, are known at the beginning of the slot, while the locations of users in the future time slots are not known.

We denote the energy budget of server S_j by E_j. The amount of energy utilized by the request from user U_i when it runs on server S_j is denoted by ϵ_{ij} and is given by,

$$\epsilon_{ij} = \frac{\sigma \cdot R_i}{Q_j} \tag{2}$$

where σ is a regulatory constant coefficient. According to Equation (2), the amount of energy consumed by each server is proportional to the utilization of computational resources (i.e., R_i/Q_j). The servers are heterogeneous in terms of their energy budget and the computation capacity.

Figures (2a) and (2b) show an edge system with two different task placements. In these figures, users are represented by circles, where the radius of circles is proportional to the size of their requests, R_i. Also, there are three edge servers which are represented by rectangles. A line connecting a user to an edge server represents the placement of that user's application on the edge server. In this example, for the sake of simplicity, servers are assumed to be identical in terms of their energy budgets as well as their computation capacities. Let's assume that each server has an energy budget of 10 units, and a capacity of 30 containers. The requests of users range from 2 to 10 containers. We also set $\gamma = 10$ and $\sigma = 20$. We define the Average Energy Utilization (AEU) of the system as,

$$\text{AEU} = \frac{\sum_{i \in \mathcal{U}} \sum_{j \in \mathcal{S}} \frac{\epsilon_{ij}}{E_j} \cdot x_{ij}}{M} \tag{3}$$

where, x_{ij} is a binary variable which is 1 if the request of user U_i is assigned to server S_j, and 0, otherwise.

In Figure (2a), the amount of energy utilized by Server a is given by, $\frac{20}{30}(2 + 4 + 3) = 6$. Since the energy budget of this server is 10, its energy utilization is 0.6. We can similarly compute the energy utilization of the other two servers, which is 0.93 and 0.6 for Server b and c, respectively. Therefore, the average energy utilization corresponding to this task placement is 0.71. For this placement, the QoS of Server a is, $10(\frac{2}{1} + \frac{4}{2} + \frac{3}{3}) = 50$. The QoS of Server b and c is 80 and 60, respectively. Therefore, the total QoS of the application placement in Figure (2a) is 190. The task placement of Figure (2b) is more efficient in terms of

TABLE 1: Notation

$\mathcal{U}$	Set of users, $\{U_1, U_2, \ldots, U_N\}$.
$\mathcal{S}$	Set of servers, $\{S_1, S_2, \ldots, S_M\}$.
$\mathcal{T}$	Set of time slots, $\{1, 2, \ldots, \tau\}$.
ξ	Set of scenarios on the movement of users.
AEU	Average energy utilization of the system.
QoS^t_{ij}	Quality of service that user U_i receives from server S_j in time slot t.
E_j	Energy budget of server S_j.
ϵ_{ij}	The amount of energy utilized by the request from user U_i when it runs on server S_j.
σ	Constant coefficient in the calculation of ϵ_{ij}.
Q_j	Computational capacity of server S_j.
R_i	Size of container requested by user U_i.
τ_i	Time to complete U_i's application on a container of size R_i.
d^t_{ij}	Distance between user U_i and server S_j in time slot t.
$\rho_{j'j}$	Relocation cost associated with changing the assignment of a user's request from server $S_{j'}$ to server S_j.
γ	Constant coefficient in the calculation of QoS^t_{ij}.
$\mathbb{E}_\xi[.]$	Expected value of the recourse function over scenarios on the movement of users.
x_{ij}	Binary variable, it is 1 if the request of user U_i is assigned to server S_j, 0 otherwise.
X^t	Vector of binary decision variables x^t_{ij}.
$y^t_{ij'j}$	Binary variable, it is 1 if the application of user U_i is relocated from server $S_{j'}$ to server S_j in time slot t, 0 otherwise.
Y^t	Vector of binary decision variables $y^t_{ij'j}$.
$c_t(X^t, Y^t)$	Objective function for time slot t.

both AEU and QoS with the values of 0.93 and 206.67, respectively.

In this paper, we develop a multi-stage stochastic programming model of the energy-aware application placement problem in MEC. In this model, the variables giving the assignments of users' requests in the current stage are the first stage variables which are decided before the realization of the uncertain parameters (i.e., location of users in the future stages) becomes known. The objective is to make the first stage decisions in a way that the sum of QoS at the first stage and the expected objective function value of the recourse costs (i.e., QoS minus relocation costs in future stages) is maximized [20]. The objective function of the application placement problem for time slot t is given by:

$$c_t(X^t, Y^t) := \sum_{i \in \mathcal{U}} \sum_{j \in \mathcal{S}} \left(\frac{\gamma \cdot R_i}{d_{ij}^t} \cdot x_{ij}^t - \sum_{j' \in \mathcal{S}} \rho_{j'j} \cdot y_{ij'j}^t \right) \quad (4)$$

where X^t and Y^t are the vectors of binary decision variables x_{ij}^t and $y_{ij'j}^t$, respectively. The decision variables are defined as follows: (i) x_{ij}^t is 1 if the request of user U_i is assigned to server S_j in time slot t, and 0, otherwise; and (ii) $y_{ij'j}^t$ is 1 if the application of user U_i is relocated from server $S_{j'}$ to server S_j in time slot t, and 0, otherwise. Therefore, the multi-stage stochastic application placement problem for time slot t is formulated as the following stochastic program:

$$\max_{X,Y} \quad \sum_{i \in \mathcal{U}} \sum_{j \in \mathcal{S}} \frac{\gamma \cdot R_i}{d_{ij}^t} \cdot x_{ij}^t + \mathbb{E}_\xi [\sum_{t'=t+1}^{\tau} c_{t'}(X^{t'}, Y^{t'})] \quad (5)$$

subject to:

$$\sum_{i \in \mathcal{U}} \frac{\sigma \cdot R_i}{Q_j} \cdot x_{ij}^t \le E_j \quad \forall j \in \mathcal{S} \quad (6)$$

$$\sum_{j \in \mathcal{S}} x_{ij}^{t'} \le 1 \quad \forall i \in \mathcal{U}, t' \in \{t, \dots, \tau\} \quad (7)$$

$$x_{ij}^t \le x_{ij}^{t'} \quad \forall i \in \mathcal{U}, j \in \mathcal{S}, t' \in \{t+1, \dots, \tau\} \quad (8)$$

$$x_{ij}^{t'} + x_{ij'}^{t'-1} - 1 \le y_{ij'j}^{t'} \quad \forall i \in \mathcal{U}, j \in \mathcal{S}, j' \in \mathcal{S}, t' \in \{t+1, \dots, \tau\};\ j \ne j' \quad (9)$$

$$x_{ij}^t, y_{ij'j}^t \in \{0,1\} \quad \forall i \in \mathcal{U}, j \in \mathcal{S}, j' \in \mathcal{S}, t \in \mathcal{T} \quad (10)$$

The objective function (5) is to maximize the total quality of service in the first stage, and the recourse cost which is the expected value of the total quality of service minus the relocation cost in the future stages. Here, $\mathbb{E}_\xi[.]$ is the expected value of the recourse function over scenarios on the movement of users (ξ), and τ is the time horizon for which the objective function is defined. Here, $\tau = \min_{i \in \mathcal{U}} \tau_i$. Constraint (6) prevents loading a server beyond its energy budget. Based on this constraint, the capacity of the servers is controlled by their available energy budget, therefore we do not need a separate capacity constraint for it. Constraint (7) ensures that the request of a user is assigned to at most one server during each time slot. Constraint (8) guarantees that processing of a user's request is not interrupted in the system. Constraint (9) ensures that when a relocation happens the decision variables are set accordingly. Constraint (10) ensures that the decision variables are binary.

The main challenge in solving multi-stage stochastic programs is to obtain the expected value of the recourse costs. In discrete optimization problems, the complexity of the multi-stage stochastic programs is highly dependent on the number of considered scenarios. It might be possible to obtain the optimal solution of the problem when there is a limited number of scenarios. But with a large number of scenarios it is practically impossible to solve the problem in a reasonable amount of time. Simulation-based methods have been widely used as efficient methods for estimating the expected value of the recourse costs in multi-stage stochastic programs. In this paper, we employ the Sample Average Approximation (SAA) method [19], [20] which is a Monte Carlo simulation-based approach to obtain a reliable estimation of the expected value of the recourse cost function.

4 Parallel SAA-based Application Placement Algorithm

The proposed model for the energy-aware application placement problem is a multi-stage stochastic program with an integer recourse function. The most critical part of multi-stage stochastic programs is the way that recourse costs are exactly evaluated or approximately estimated. Obviously, due to its high computational complexity, multi-dimensional integration of the recourse function is not a feasible option for an exact evaluation of recourse costs in programs with integer recourse. On the other hand, in multi-stage stochastic programs with integer recourse, optimizing the expected recourse costs is of high complexity, because the functions are highly non-convex and not continuous [20].

The SAA method has received a considerable attention as an efficient method for solving multi-stage stochastic programs [19], [20], [46], [47], [48]. SAA is a Monte Carlo simulation-based approach to solving stochastic discrete optimization problems. The basic idea of the SAA algorithm is to approximate the recourse function by the sample average function, where the samples are *independent and identically distributed* (iid) and include a constant number of scenarios.

The SAA procedure is given in Algorithm 1. This procedure is applied in each stage (time slot) t. All the pa-

Algorithm 1 SAA algorithm

```
   {Executed every time slot $t$}
1: Generate $H$ independent scenario samples, each of size $L$,
      ($k \in \{1, \dots, H\}$)
2: for $k = 1$ to $H$ do
3:     Solve the deterministic equivalent problem
         corresponding to each sample.
4: end for
5: Generate a sufficiently large sample of size $L'$, $L' \gg L$.
6: for $k = 1$ to $H$ do
7:     Evaluate the candidate solutions obtained in line (3)
         by solving the deterministic equivalent problem
         corresponding to the sample of size $L'$, while
         the current stage variables are fixed to their
         values obtained in line (3).
8: end for
9: Out of $H$ candidate solutions, choose the one that has
     the largest estimated objective value.
```

Algorithm 2 PG-SAA: Parallel Greedy SAA-based application placement algorithm

```
    {Executed every time slot $t$}
 1: Generate $H$ independent scenario samples
       ($k \in \{1, \dots, H\}$), each of size $L$.
 2: for $k = 1$ to $H$ do in parallel
 3:     for $i = 1$ to $N$ do
 4:         for $\ell = 1$ to $L$ do
 5:             Create graph $G_i(V, E)$ for user $U_i$ under
                  scenario $\ell$.
 6:             Use Dijkstra's algorithm to find the shortest
                  paths between vertices $S_j^t$ and vertex $D$ in
                  graph $G_i(V, E)$ for scenario $\ell$.
 7:             $w_{ij}^{k,\ell} \leftarrow$ cost of the shortest paths.
 8:         end for
 9:         for $j = 1$ to $M$ do
10:             $\bar{w}_{ij}^k = \frac{1}{L}\sum_{\ell=1}^{L} w_{ij}^{k,\ell}$
11:         end for
12:     end for
13:     Call G-MAP ($\bar{W}^k$) to obtain the placement.
14:     Record the optimal solution as $(\bar{X}^{k,t}, \bar{Y}^{k,t})$.
15: end for
16: Generate a sufficiently large sample of size $L'$, $L' \gg L$.
17: for $k = 1$ to $H$ do in parallel
18:     for $i = 1$ to $N$ do
19:         for $\ell' = 1$ to $L'$ do
20:             Create graph $G'_i(V, E)$ for user $U_i$ under
                  scenario $\ell'$.
21:             Use Dijkstra's algorithm to find the shortest
                  paths between vertex $S_{j_i^*}$ and dummy
                  vertex $D$ in graph $G'_i(V, E)$ for scenario $\ell'$.
22:             $w_{ij}^{k,\ell'} \leftarrow$ cost of the shortest paths.
23:         end for
24:         for $j = 1$ to $M$ do
25:             $\bar{w}_{ij}^k = \frac{1}{L'}\sum_{\ell'=1}^{L'} w_{ij}^{k,\ell'}$
26:         end for
27:     end for
28:     $Z^k \leftarrow$ objective value corresponding to sample $k$.
29: end for
30: $Z^* \leftarrow \max_{k \in \{1,\dots,H\}} Z^k$
31: Allocate users' requests to servers according to $\bar{X}^{k,t}$ corre-
    sponding to $Z^*$.
```

rameters for the current stage are known, while some of the parameters in future stages are not known but follow a certain probability distribution. The first step of the SAA algorithm is to generate H samples, each including L independent scenarios. We denote by $\xi_{[\tau]} := (\xi_t, \dots, \xi_\tau)$ the history of the scenarios for the location of users from time slot t up to time slot τ. These samples will be used to estimate the recourse function. The next step (lines 2-4) is to solve the deterministic equivalent problem for each sample out of the H samples, resulting in H candidate solutions. Then, a sample with L' scenarios ($L' \gg L$) is generated to evaluate the candidate solutions (line 5). In the next step (lines 6-8), candidate solutions are evaluated using the generated sample. Finally, the selected solution is the solution with the largest objective value among the evaluated candidates (line 9).

In multi-stage stochastic programs, the total number of scenarios grows exponentially with the number of stages [46]. Therefore, the efficiency of SAA requiring a large number of scenarios for accurately estimating the recourse function is negatively affected as the number of stages in-

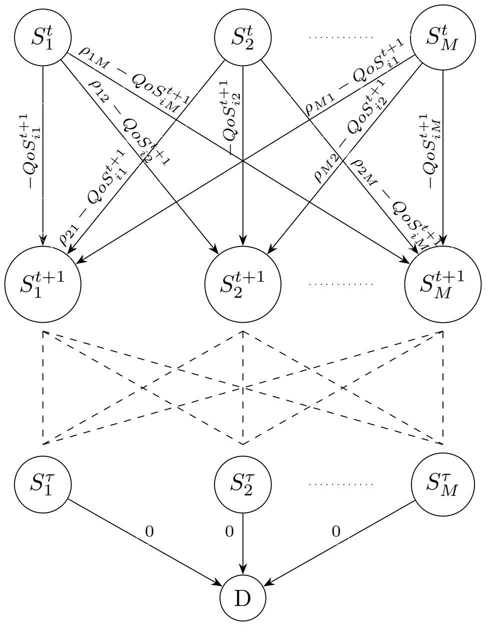

S_1^t
S_2^t
S_M^t
S_1^{t+1}
S_2^{t+1}
S_M^{t+1}
S_1^τ
S_2^τ
S_M^τ
D
$-QoS_{i1}^{t+1}$
$\rho_{1M} - QoS_{iM}^{t+1}$
$\rho_{12} - QoS_{i2}^{t+1}$
$\rho_{21} - QoS_{i1}^{t+1}$
$-QoS_{i2}^{t+1}$
$\rho_{M1} - QoS_{i1}^{t+1}$
$\rho_{M2} - QoS_{i2}^{t+1}$
$\rho_{2M} - QoS_{iM}^{t+1}$
$-QoS_{iM}^{t+1}$
0
0
0

Fig. 3: Graph $G_i(V, E)$ used for computing the assignment of user U_i.

creases. Shapiro [46] proved bounds on the required number of scenarios in SAA for multi-stage stochastic programs. His analysis is based on the assumption that distribution of the stochastic parameters in different stages are independent. In the application placement problem in MEC, it is crucial to obtain a quality solution very fast. The SAA algorithm for this problem requires solving optimally multiple Integer Programs (IP) using commercial optimization solvers which might take a considerable amount of time. In this paper, we *design a fast parallel SAA-based greedy algorithm* to solve the multi-stage stochastic program corresponding to the energy-aware application placement problem.

The basic idea in the proposed parallel SAA-based algorithm for solving the energy-aware application placement problem is to evaluate the candidate solutions in parallel and use a greedy procedure to solve the deterministic equivalent problems corresponding to each sample. The parallel greedy SAA-based application placement algorithm, called PG-SAA, is given in Algorithm 2.

First, PG-SAA generates H samples, each including L independent scenarios for the location of users in the upcoming stages (line 1). The next step (lines 2-15) is to solve the deterministic equivalent problem corresponding to each sample. This step is executed in parallel and the output consists of H candidate solutions for the application placement in the current stage. The objective function of the *deterministic equivalent problem* corresponding to each sample is as follows:

$$z_L^{k,t} = \max_{X^{k,t}, Y^{k,t}} \{ \sum_{i \in \mathcal{U}} \sum_{j \in \mathcal{S}} \frac{\gamma \cdot R_i}{d_{ij}^t} \cdot x_{ij}^{k,t} + \frac{1}{L} \sum_{l=1}^{L} \sum_{t'=t+1}^{\tau} c_{t'}(X^{k,t'}, Y^{k,t'}, \xi_{[t']}^l) \} \quad (11)$$

where, $X^{k,t}$ and $Y^{k,t}$ are the vectors of decision variables.

To solve these IP problems approximately, we design a greedy heuristic which is based on the idea of evaluating the expected recourse function of each user independently. The first step of this procedure (line 5) creates a graph $G_i(V, E)$ for each user U_i, where V is the set of vertices composed of vertices corresponding to the set of servers replicated for each stage up to stage τ, and E is the set of edges. Figure 3 shows the graph $G_i(V, E)$ for the assignment of user U_i's request. We denote the vertex in G_i corresponding to server S_j in stage t, by S_j^t, and the dummy vertex by D. The weight of the edge between server S_j^t and $S_{j'}^{t+1}$ represents the expected relocation cost from server j to j' minus $QoS_{ijj'}^{s_{t+1}}$ when the request of user U_i is executed on server S_j in stage t and then it is executed on server $S_{j'}$ in stage $t+1$. There is no relocation cost if the application of a user is executed on the same server in two consecutive time slots. Also, the weights of edges to the dummy vertex D are zero. Let Γ_{ij} denote the expected value of the shortest path from vertices S_j^t (current stage) to the dummy vertex D over all the considered scenarios. Then we have,

$$\Gamma_{ij} = -\frac{1}{L}\sum_{l=1}^{L}\sum_{t'=t+1}^{\tau} c_{t'}(X^{k,t'}, Y^{k,t'}, \xi_{[t']}^{l}). \tag{12}$$

Therefore, the expected recourse functions are obtained by finding the M shortest paths from vertices S_j^t to vertex D and selecting the minimum one among them. The shortest paths are computed using Dijkstra's algorithm (line 6 of PG-SAA). Under unlimited energy budget, the request of user U_i will be placed on the server which has the minimum value of the shortest path among all the servers. Due to the limited energy budget of servers, it might not be feasible to assign each user to a server as determined by the shortest paths. For this reason, we need to find a refined assignment that takes into account the energy budget of each server. To do that, we consider solving an associated assignment problem for sample k, where the objective is to maximize the total assignment weights. This corresponds to the objective of the deterministic equivalent problem given in Equation (11). We define the expected value of the assignment weights associated with executing the request of user U_i on server S_j for sample k as,

$$\bar{w}_{ij}^{k} = \frac{\gamma \cdot R_i}{d_{ij}^{t}} - \Gamma_{ij}. \tag{13}$$

Therefore, the associated Maximal Assignment Problem (MAP) for sample k is formulated as follows,

$$\max_{X} \quad \sum_{i\in\mathcal{U}}\sum_{j\in\mathcal{S}} \bar{w}_{ij}^{k} x_{ij}^{k} \tag{14}$$

subject to:

$$\sum_{i\in\mathcal{U}} \frac{\sigma \cdot R_i}{Q_j} \cdot x_{ij} \le E_j \quad \forall j \in \mathcal{S} \tag{15}$$

$$\sum_{j\in\mathcal{S}} x_{ij}^{k} \le 1 \quad \forall i \in \mathcal{U} \tag{16}$$

$$x_{ij}^{k} \in \{0,1\} \quad \forall i \in \mathcal{U}, j \in \mathcal{S} \tag{17}$$

The objective function (14) of this model is to maximize the total assignment weights. Constraint (15) ensures that the energy budget of each server is not violated, constraint (16) guarantees that each user is assigned to at most one server, and constraint (17) defines the binary variables.

The MAP problem is an NP-hard problem [49], that is, no algorithm is able to find the optimal solution in polynomial time, unless P=NP.

Theorem 1. MAP is NP-hard in the strong sense.

Proof: First, let us ignore the index k of samples, because the problems associated with samples are solved independently. Let us also consider a special case of MAP called MAP-I in which for each user, the weights are the same for all the servers, i.e., $\bar{w}_i = \bar{w}_{i1} = \cdots = \bar{w}_{iM}$. We define $C_j = \frac{Q_j \cdot E_j}{\sigma}$. Therefore, the MAP-I problem is,

$$\max_{X} \quad \sum_{i\in\mathcal{U}}\sum_{j\in\mathcal{S}} \bar{w}_i x_{ij} \tag{18}$$

subject to:

$$\sum_{i\in\mathcal{U}} R_i \cdot x_{ij} \le C_j \quad \forall j \in \mathcal{S} \tag{19}$$

$$\sum_{j\in\mathcal{S}} x_{ij} \le 1 \quad \forall i \in \mathcal{U} \tag{20}$$

$$x_{ij} \in \{0,1\} \quad \forall i \in \mathcal{U}, j \in \mathcal{S} \tag{21}$$

MAP-I is equivalent to the 0-1 Multiple Knapsack Problem MKP, where $\bar{w}_i$ is the profit of assigning item i to knapsack j (executing request of user U_i on server S_j). Also, C_j is equivalent to the capacity of knapsack j. The objective of MKP is to assign items (requests) to knapsacks (servers) such that the total profit (weights) is maximized, while the capacity constraint is not violated. MKP is NP-hard in the strong sense [50]. Therefore, MAP-I is strongly NP-hard, and since MAP-I is a special case of MAP, we conclude that MAP is NP-hard in the strong sense. □

Existing algorithms for MKP are not directly applicable to MAP, because in MKP the profit of items ($\bar{w}_i$) is independent of the knapsacks. In contrast, we have $\bar{w}_{ij}^{k}$ in MAP. Therefore, to solve the MAP problem in a reasonable amount of time, we develop a fast greedy algorithm. We call this algorithm G-MAP and is presented in Algorithm 3. We will briefly describe the details of this algorithm later in this section.

PG-SAA, calls G-MAP to obtain a feasible assignment for the current stage (line 13), where $\bar{W}^k$ is the vector of $\bar{w}_{ij}^{k}$, that is, the expected value of the assignment weights for sample k. Next, PG-SAA records the candidate solutions obtained for each sample k as $(\bar{X}^{k,t}, \bar{Y}^{k,t})$.

Next, PG-SAA generates a sample with L' scenarios ($L' \gg L$) to evaluate the candidate solutions (line 16). Then, the candidate solutions are evaluated using the generated sample (lines 17-29). The evaluation models are also solved in parallel for the H samples. The objective function of the deterministic equivalent of the evaluation model is formulated as follows:

$$\hat{z}_{L'}^{k,t} = c_t(\bar{X}^{k,t}, \bar{Y}^{k,t}) + \frac{1}{L'}\sum_{l=1}^{L'}\sum_{t'=t+1}^{\tau} c_{t'}(X^{k,t'}, Y^{k,t'}, \xi_{[t']}^{l}) \tag{22}$$

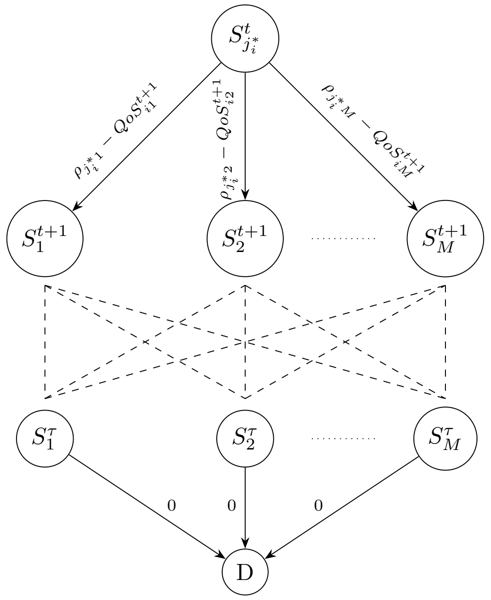

$S^t_{j^*_i}$
$\rho_{j^*_i 1} - QoS^{t+1}_{i1}$
$\rho_{j^*_i 2} - QoS^{t+1}_{i2}$
$\rho_{j^*_i M} - QoS^{t+1}_{iM}$
S^{t+1}_1
S^{t+1}_2
S^{t+1}_M
S^τ_1
S^τ_2
S^τ_M
0
0
0
D

Fig. 4: Graph $G'_i(V, E)$ used when evaluating the candidate solutions.

where, $\bar{X}^{k,t}$ and $\bar{Y}^{k,t}$ are the vectors of solutions recorded in line 14. For the evaluation step (lines 17-29), PG-SAA uses a similar procedure to that used for obtaining the candidate solutions with some minor changes. The main difference between the two procedures is that for the evaluation step, we do not need to calculate the shortest paths for all users-servers combinations. Here, we create a new graph for user U_i denoted by $G'_i(V, E)$. As shown in Figure 4, there is only one node in the first stage which represents the server that was assigned to user U_i at time slot t based on the candidate solution. Therefore, PG-SAA executes Dijkstra's algorithm for the server that is assigned to each user in the current stage ($\bar{x}^{k,t}_{ij} = 1$), where $S_{j^*_i}$ is the server assigned to user U_i according to the candidate solution recorded in line 14. It should be noted that the feasibility of this assignment was already guaranteed because this solution is obtained by the G-MAP algorithm. Finally, the best solution is the solution with the maximum value among the evaluated candidates (lines 30-31):

$$(\bar{X}^*, \bar{Y}^*) \in \text{argmax}_{k\in\{1,\ldots,H\}}\{\hat{z}^{k,t}_{L'}(\bar{X}^{k,t}, \bar{Y}^{k,t})\}. \quad (23)$$

We now describe briefly the G-MAP algorithm given in Algorithm 3. The aim of this algorithm is to find a feasible assignment of users' requests to servers taking the energy budget into account. The basic idea of this algorithm is to prioritize users based on their expected contributions to the objective and the amount of energy consumed to satisfy their requests. For this purpose, we define Φ_i metric as,

$$\Phi_i = \max_{j\in F} \frac{\bar{w}_{ij}}{\sigma \cdot R_i} \quad (24)$$

where, F is the feasibility set of servers based on their available energy budget. We also define a new variable for the energy budget of servers denoted by E'_j. First, G-MAP sets the value of E'_j to the total available energy budget E_j for server S_j. Here, we denote the set of unvisited users by $\bar{U}$. Also, we put all the servers in the feasibility set (F), due to the assumption that all servers are able to provide services according to their free capacity. The initial value of vector X is also set to zero. G-MAP repeats the following steps (lines 6-26) until all the users are visited. In lines 7-17, the algorithm finds the user with the maximum value of Φ_i, taking the energy requirement of its request into account. Then, it assigns the user with the maximum value of Φ_i to the associated server ($j_{\hat{i}}$) and updates the energy budget of that server (lines 19-20). In lines 21-23, if the server exceeds its energy budget, then it is excluded from the feasibility set. Also, the visited user is removed from the set of unvisited users (line 25).

Algorithm 3 G-MAP algorithm

```
1: Input: $\bar{w}_{ij}$, $E_j$, $Q_j$, $R_i$, $\sigma$.
2: $E'_j \leftarrow E_j \quad \forall j \in \{1,\ldots,M\}$
3: $\bar{U} \leftarrow \{1,\ldots,N\}$
4: $F = \{1,\ldots M\}$
5: $X \leftarrow 0$
6: while $\bar{U} \neq \emptyset$ do
7:     for all $i$ in $\bar{U}$ do
8:         $j_i \leftarrow -1$
9:         $\Phi_i \leftarrow 0$
10:        for all $j$ in $F$ do
11:            if $\frac{\bar{w}_{ij}}{\sigma \cdot R_i} \geq \Phi_i$ and $E'_j > \sigma\frac{R_i}{Q_j}$ then
12:                $j_i \leftarrow j$
13:                $\Phi_i \leftarrow \frac{\bar{w}_{ij}}{\sigma \cdot R_i}$
14:            end if
15:        end for
16:    end for
17:    $\hat{i} = \arg\max_i\{\Phi_i\}$
18:    if $j_{\hat{i}} \neq -1$ then
19:        $x_{\hat{i}j_{\hat{i}}} \leftarrow 1$
20:        $E'_{j_{\hat{i}}} \leftarrow E'_{j_{\hat{i}}} - \sigma\frac{R_{\hat{i}}}{Q_{j_{\hat{i}}}}$
21:        if $E'_{j_{\hat{i}}} \leq 0$ then
22:            $F \leftarrow F \setminus \{j_{\hat{i}}\}$
23:        end if
24:    end if
25:    $\bar{U} = \bar{U} \setminus \{\hat{i}\}$
26: end while
```

Now, we determine the runtime complexity of PG-SAA. Determining the assignments for each user under each scenario using Dijkstra's algorithm (Algorithm 2, lines 5 to 7) takes $O(M^2\tau^2)$. The G-MAP algorithm takes $O(MN^2)$. Evaluating each of the H solutions on the new larger sample (L') for each user under each scenario (Algorithm 2, lines 20 to 22) takes $O(M^2\tau^2)$. Therefore, the total runtime complexity assuming a parallel computing system with H processors, is $O(NLM^2\tau^2 + MN^2 + NL'M^2\tau^2)$. Since $L' >> L$, the complexity of the proposed algorithm with H processors is $O(MN^2 + NL'M^2\tau^2)$.

5 Experimental Analysis

To evaluate the performance of the proposed application placement algorithm, we perform an extensive experimental analysis using real-world trace data. We aim at evaluating the quality of solutions and the running times of the proposed PG-SAA algorithm for problem instances of different sizes.

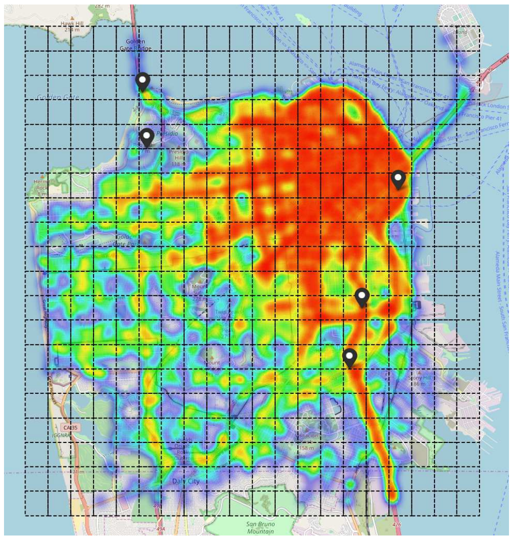


Fig. 5: Distribution of users and servers in the network

5.1 Experimental setup

We perform our analysis using a real-world data set of cab mobility traces collected in San Francisco [51] to capture distribution of demands as well as the mobility of users across the network. Figure 5 shows the map of the area partitioned in a two-dimensional grid of 20 × 20 cells in an area of San Francisco with the highest number of trace records. The map is created based on the trace data of 20 taxi cabs. Darker (red in color) regions represent areas with a high number of trace records. For our experiments, we only consider the dense areas, that is, areas with a small number of trace records are excluded from our analysis.

In our analysis, we consider that the edge servers are co-located with antenna towers. The locations of the towers are taken from [52]. We choose five base-stations registered with Verizon Wireless Co. that are shown by pointers in Figure 5. We choose Dell PowerEdge R740 Rock Server as the edge servers. The configuration of the servers are shown in Table 2.

In order to ensure high quality services, we define the unit-size container according to the computational capacity of a single core, and at most one container is allocated to a core. Therefore, the computation capacity, Q_j, of a server is equivalent to the number of CPU cores. According to a survey on data centers energy consumption [53], the idle power of an edge server accounts for 60% of the full state power. Thus, in our setting, the energy budget of a server (in Joules) for a time slot is: $E_j = 0.4 \times 495\text{W} \times 120\text{s} = 23,760\,\text{J}$, where 120s represents the length of the time slots considered in the experiments.

We assume that if a core is utilized, it requires full power. Therefore, the power of each core is obtained by dividing the total power of a server by the number of cores. Based on this assumption the power of each core of the PowerEdge R740 Rock servers is 20.625 W. We use the dataset in [54] on smartphones, to set the size of the containers requested by users. This dataset has about 1 million records, and we draw a random sample from it to obtain the users' requests. A sample of this dataset is shown in Table 3. In the table,

TABLE 2: Configuration of servers

Model		Dell PowerEdge R740 Rack Server
CPU	Model	Intel® Xeon® Gold 6240
	Frequency	2.4GHz
	Cache	35.75M
	Cores	24
RAM		32GB
Power		495W

TABLE 3: Mobile applications sample data

UUID	CPU freq. (GHz)	Total_CPU (%)	#Cores
1461792776014	2.4	60.029762	3
1461792780433	2.4	55.32268	3
1461792785058	2.4	4.946742	1
1461792790505	2.4	14.84594	1

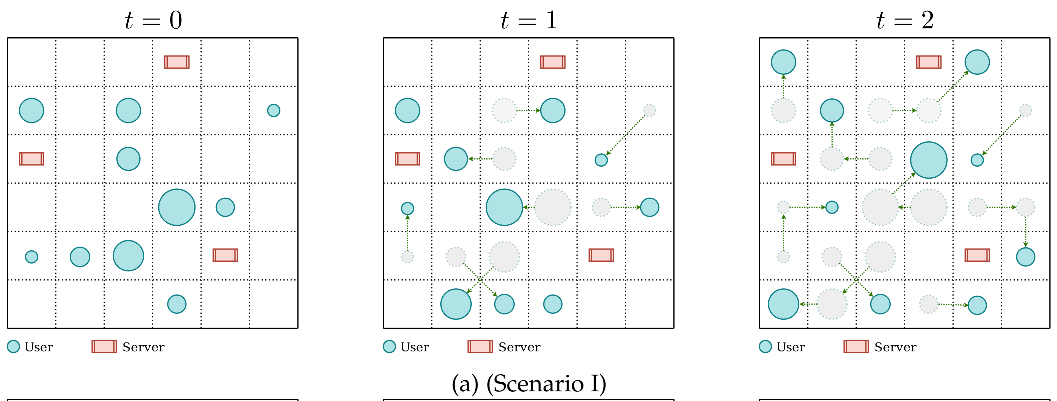
Total_CPU represents the average utilization among four cores. Because the dataset gives the utilization with respect to four cores, we determine the number of cores necessary to host a container associated with the request of a user, by multiplying the Total_CPU by 4. As an example, for the request with UUID: 1461792776014, the number of cores is $60.029762\% \times 4 \simeq 3$.

Here, we assume that when a container is allocated to multiple cores, all the cores are fully utilized. Therefore, the utilization rate of a server can be simply obtained by dividing the allocated cores by the total number of cores.

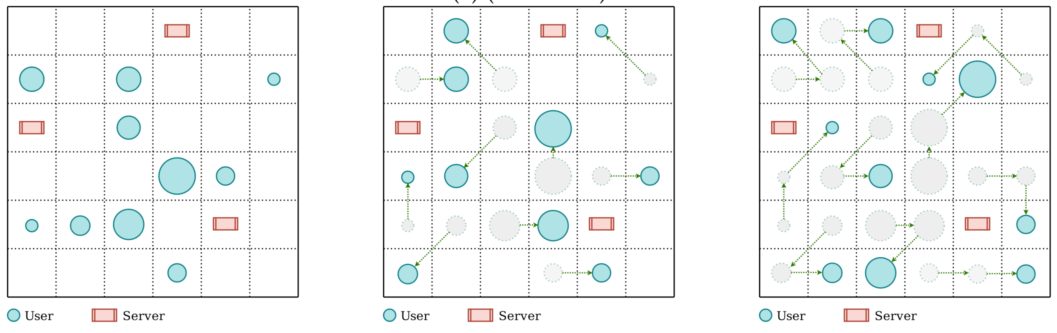
The relocation costs are computed as $\beta d_{jj'}$, where β is a constant factor and $d_{jj'}$ is the distance between server S_j and server $S_{j'}$. We use the haversine formula to calculate the distance between two points in the network using their longitudes and latitudes. We use the center point of each cell for the location of users in that cell. Here, we set β to 10. We also set $\gamma = 100$ and $\sigma = 23,760$. We set the sample size and the number of scenarios in the SAA algorithm to $H = 10$, $L = 20$, and $L' = 50$. All programs implementing the algorithms are compiled using GCC version 4.8.5 and executed on a 64-core 2.4 GHz dual-processor AMD Opteron system with 512 GB of RAM and Red Hat Enterprise Linux Server as the operating system. The proposed parallel algorithm is implemented using OpenMP. For the experiments involving CPLEX we use the CPLEX 12 solver provided by IBM ILOG CPLEX optimization studio for academics initiative [55]. Each experiment is performed five times and the analysis is performed based on the average value of the metrics. For the execution time analysis, we use the error bar histograms showing the standard deviation of the execution time among the five runs.

5.2 Scenario generation model

In our scenario generation model, we use the real-world data set of cab mobility traces in San Francisco. We use this data to extract the parameters necessary to generate the mobility scenarios of users. We develop our scenario generation model using the mobility traces of 20 taxi cabs. We record the location of each taxi cab every two minutes, which represents the length of the time slot in our analysis. For our experiments, we consider that users are located within a two-dimensional grid of 20 × 20 cells in an area of San Francisco that has the highest number of records, that is, areas with small number of records are excluded



(a) (Scenario I)



(b) (Scenario II)

Fig. 6: Scenario generation model: an illustrative example

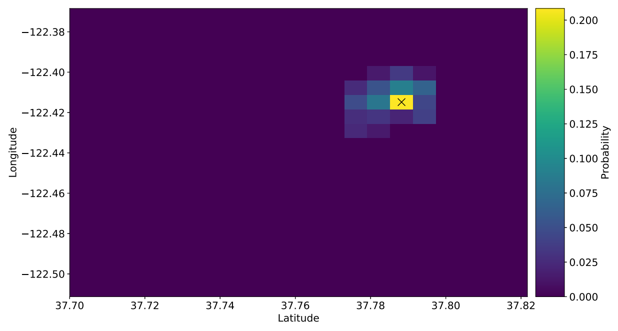


Fig. 7: An example of users' mobility model

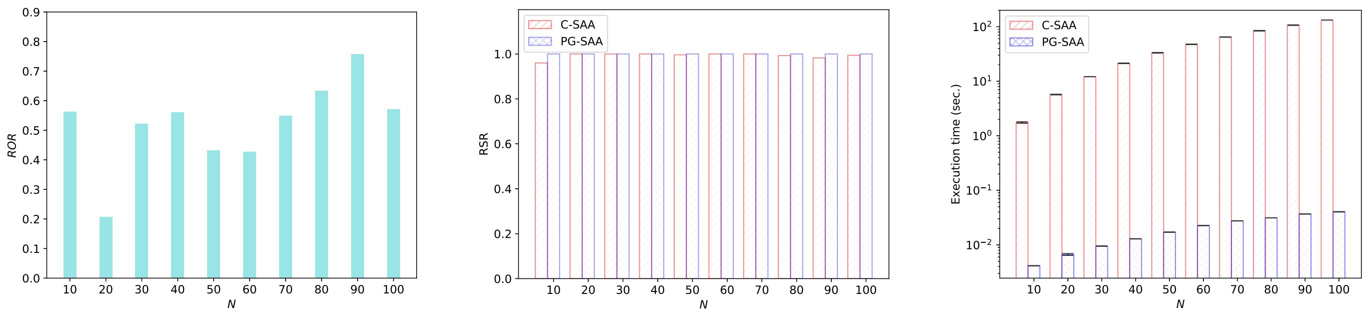
from our analysis. The frequency of records in a cell is used as a probabilistic parameter to distribute users across the network in the first time slot. To determine the probability that a user moves from cell A to cell B within a time slot, we consider the number of movements from cell A to cell B divided by the total number of departures from cell A. Figure 7 shows the movement probability from the cell marked with X to other cells as a heat map. We observe in this example, that with the highest probability a user will stay in the same cell in the next time slot, and that is more probable that the user goes to the top or left cells.

We denote by $\xi_{[\tau]} := (\xi_t, \dots, \xi_\tau)$ the history of the scenarios for the location of users from time slot t up to time slot τ.

Figures 6a and 6b show an example of our scenario generation method. In this example, there are three edge servers (represented as rectangles) and ten users (represented as circles) that are randomly distributed in a grid of 6×6 cells. These figures show two scenarios for three time slots. It is observed that the location of the servers is the same for all time slots. Also, the locations of users in the current time slot ($t = 0$) is the same for all the scenarios. The locations of users in the second time slot ($t = 2$) are dependent on their locations in the preceding time slot ($t = 1$) in the same scenario. In these figures, the radius of a circle is proportional to the size of the user's request. Since in our formulation the total QoS of the system is a function of the size of requests and the distance of users from the host server, it is crucial to consider the location of users during the planning time frame. The probability of moving from cell A to cell B is extracted from the cab mobility traces, that is, the number of movements from cell A to cell B divided by the total number of departures from cell A.

5.3 Experimental results

We perform the experimental analysis on two classes of instances. First, we present the experimental results for the class of small size instances where each instance consists of up to 100 users. Our aim is to evaluate the performance of our proposed algorithm by comparing it with that obtained by using the CPLEX solver to solve the optimization problems in the SAA algorithm. For this set of experiments, we run our algorithm using one core (i.e., serial). The second class of instances in our analysis consists of large instances with the number of users ranging from 1000 to 4000. CPLEX cannot solve problem instances of such sizes in a reasonable amount of time. Our aim for this class of instances is to analyze the scalability of our proposed parallel algorithm. In this set of experiments, we run our algorithm using multiple cores ranging from 2 to 64 and compare the execution time with that of the serial execution of the algorithm.



(a) Relative objective ratio (ROR) (b) Request satisfaction ratio (RSR) (c) Execution time

Fig. 8: PG-SAA vs. C-SAA performance for various numbers of users, N.

5.3.1 Results for small-scale instances

First, we compare the quality of solutions obtained by the PG-SAA to that of the solutions obtained by using the CPLEX solver to solve the optimization problems in the SAA algorithm.

Due to the large execution time required by CPLEX for solving the subproblems in the SAA-based algorithm, we consider here only small sizes for the application placement problem instances. To provide a fair comparison, we execute PG-SAA and the CPLEX-based SAA algorithm by considering the same samples. In the following, we denote by C-SAA the CPLEX-based SAA algorithm. To characterize the quality of the solutions we define the *relative objective ratio* (ROR) as follows:

$$\text{ROR} = \frac{Z^{C-SAA} - Z^{PG-SAA}}{Z^{C-SAA}} \tag{25}$$

where Z^{PG-SAA} is the objective value determined by the PG-SAA and Z^{C-SAA} is the objective value determined by the CPLEX-based SAA algorithm, C-SAA.

We also perform a set of experiments to investigate the demand satisfaction obtained by the two solution methods. In order to do this, we define the *Request Satisfaction Ratio* (RSR) as the percentage of users that receive services from the network, that is

$$\text{RSR} = \frac{\sum_{i \in \mathcal{U}} \sum_{j \in \mathcal{S}} x_{i,j}}{N} \tag{26}$$

where $x_{i,j}$ is the binary variable and is 1 if the request of user U_i is execute on server S_j, and 0 otherwise.

The objective ratio, request satisfaction ratio, and execution times obtained for the small instances are presented in Figure 8. Because of the small size of the instances considered here, we execute PG-SAA on only one core. This is because the benefits of parallelization are not observed for such small instances. We will execute PG-SAA on multiple cores for the next set of experiments involving large-scale instances of the application placement problem.

Figure 8a shows the objective ratio obtained for the small instances. The solutions obtained by PG-SAA for these types of instances are within the range of 0.2 and 0.8 compared to the solution obtained by the C-SAA algorithm which is an acceptable range. We do not observe any significant increase in the objective ratio with the increase in the number of users.

In Figure 8b, we observe that the request satisfaction ratio obtained by the PG-SAA method is very close to that of C-SAA for all sizes of instances. We observe that in some instances the request satisfaction ratio of PG-SAA is higher than that of C-SAA. This observation is justified by the fact that based on our definition and formulation, a higher RSR would not necessarily result in a higher QoS. Therefore, solutions obtained by C-SAA may not satisfy requests that would negatively affect QoS.

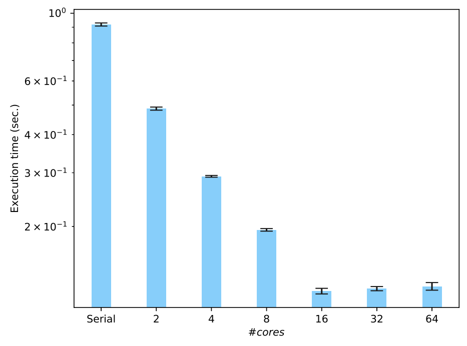
Figure 8c shows the execution times of C-SAA and PG-SAA algorithms for the small instances. We observe that even for such small instances, the PG-SAA is very competitive in terms of the execution time. While the PG-SAA takes less than one second to solve problems with less than 100 users, the execution time of the C-SAA for instances with 100 users is greater than 100 seconds.

5.3.2 Results for large-scale instances

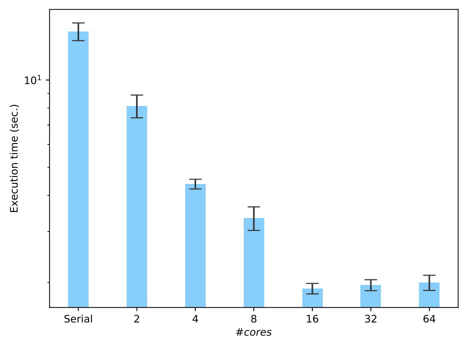
In the next set of experiments, we investigate the performance of PG-SAA on large-scale instances of the application placement problem. We run the PG-SAA algorithm on instances with different number of users ranging from 1000 to 4000.

The execution times for these large size instances are presented in Figure 9 We use the label "Serial" to denote the execution of the PG-SAA on only one core. We observe that the running time of the PG-SAA on only one core (serial execution) is still within a reasonable range for problem instances with up to 1000 users, but for larger instances the serial version of PG-SAA takes a considerable amount of time to solve the problem. The PG-SAA performs very well in terms of the running time when executed on multiple cores. The small execution time of PG-SAA for large scale instances makes PG-SAA very suitable for deployment in mobile edge computing systems.

In Section 4, we showed that the number of time slots (τ) is a determinant factor in the runtime complexity of PG-SAA. Here, we perform an analysis to experimentally investigate how the running time of our algorithm is affected by this parameter. We perform the analysis by running the parallel algorithm on 16 cores as well as its serial version. The results of this analysis are shown in Figure 10. We observe that for both instances with 1000 users (Figure 10a) and 4000 users (Figure 10b), the increase in the running time of the serial algorithm is linear but more significant than the

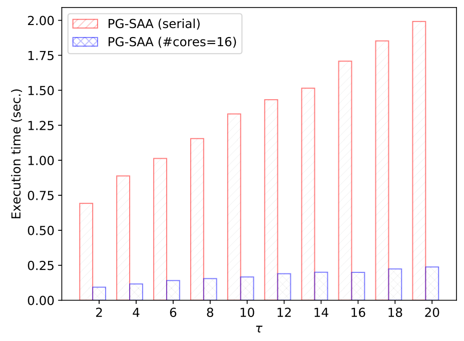


(a) $N = 1000$

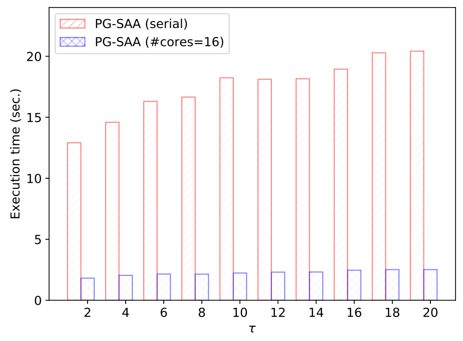


(b) $N = 4000$

Fig. 9: PG-SAA: Execution time vs. number of cores for systems with 1000 and 4000 users.



(a) $N = 1000$



(b) $N = 4000$

Fig. 10: PG-SAA: Effects of τ on the execution time for systems with 1000 and 4000 users.

that of the parallel algorithm when executed on 16 cores. This observation implies that the parallel algorithm can estimate very efficiently the expected value of the recourse function for problem instances with 4000 users and up to 20 time slots.

5.4 Discussion

In MEC systems, the mobility of users might significantly affect the QoS, and must be taken into account to have an efficient application placement. In this experimental analysis, we showed how the proposed method can be employed in a scenario driven by real-world data on the mobility of users. To apply our proposed method in real-world scenarios, one only needs know the current location of users and the historical data on the mobility of users. The mobility of users in future time slots can be modeled using the historical data traces. The mobility model is independent of our proposed method and would be obtained from the available historical data from the considered geographical area. Therefore, our method is applicable to virtually any geographical area and time period as long as there are no significant changes in the structure and the population of the area that could affect the prediction from historical data.

The experimental analysis shows that the proposed SAA-based parallel algorithm can efficiently solve large-scale instances of the energy-aware application placement problem in MEC systems. Comparing the results obtained by PG-SAA with those of C-SAA based on the ROR metric shows that the solutions obtained by PG-SAA are fairly close to the solutions obtained by the CPLEX solver. But when it comes to the execution time, PG-SAA can solve instances of size up to 4000 users within 2 seconds, while C-SAA requires 100 seconds to solve a much smaller instance with 100 users. Our experimental analysis on the performance of the parallel algorithm for large-scale instances also shows that the proposed algorithm scales reasonably well with the size of instances on systems with large number of cores.

6 Conclusion

We addressed the energy-aware application placement problem in mobile edge computing systems. To take into account the mobility of users, a challenging issue in the resource management of MEC systems, we formulated the problem as a multi-stage stochastic program. We designed a parallel greedy SAA-based algorithm for solving the application placement problem and performed an extensive experimental analysis to investigate its performance using real-world trace data. The results of the experimental analysis showed that the proposed algorithm is able to obtain very good quality solutions for the problem. The proposed algorithm requires a very small execution time when executed on large multi-core systems, making it very suitable for deployment on current and future mobile edge computing systems.

As future work, we plan to employ the proposed approach on the design of algorithms for other challenging stochastic problems in the management of mobile edge computing systems. Such problems include task offloading and resource provisioning, where the mobility of users is an important factor affecting the offloading and provisioning decisions. We also plan to employ a risk-based optimization approach to address the quality of service in MEC systems considering the movement of users.

Acknowledgment

This research was supported in part by the US National Science Foundation under grant no. IIS-1724227.

References

[1] M. Satyanarayanan, “Fundamental challenges in mobile computing,” in *Proc. 15th ACM Symp. on Principles of Distrib. Comp.*, 1996, pp. 1–7.

[2] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, “A survey of mobile cloud computing: architecture, applications, and approaches,” *Wireless Comm. and Mobile Comp.*, vol. 13, no. 18, pp. 1587–1611, 2013.

[3] F. Liu, P. Shu, H. Jin, L. Ding, J. Yu, D. Niu, and B. Li, “Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications,” *IEEE Wireless Communications*, vol. 20, no. 3, pp. 14–22, 2013.

[4] M. T. Beck, M. Werner, S. Feld, and S. Schimper, “Mobile edge computing: A taxonomy,” in *Proc. 6th International Conference on Advances in Future Internet*, 2014, pp. 48 – 54.

[5] B. Liang, “Mobile edge computing,” *Key Technologies for 5G Wireless Systems*, p. 76, 2017.

[6] S. Davy, J. Famaey, J. Serrat-Fernandez, J. L. Gorricho, A. Miron, M. Dramitinos, P. M. Neves, S. Latré, and E. Goshen, “Challenges to support edge-as-a-service,” *IEEE Communications Magazine*, vol. 52, no. 1, pp. 132–139, 2014.

[7] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, “Mobile edge computing: Survey and research outlook,” *arXiv preprint arXiv*, vol. 1701, 2017.

[8] Q. Chen, Z. Zheng, C. Hu, D. Wang, and F. Liu, “Data-driven task allocation for multi-task transfer learning on the edge,” in *Proc. IEEE ICDCS*, 2019.

[9] Z. Feng, S. George, J. Harkes, P. Pillai, R. Klatzky, and M. Satyanarayanan, “Eureka: Edge-based discovery of training data for machine learning,” *IEEE Internet Computing*, 2019.

[10] H. Li, K. Ota, and M. Dong, “Learning iot in edge: Deep learning for the internet of things with edge computing,” *IEEE Network*, vol. 32, no. 1, pp. 96–101, 2018.

[11] T. Bahreini, M. Brocanelli, and D. Grosu, “Energy-aware speculative execution in vehicular edge computing systems,” in *Proc. 2nd Intl. Workshop on Edge Systems, Analytics and Networking*. ACM, 2019, pp. 18–23.

[12] F. Cicirelli, A. Guerrieri, A. Mercuri, G. Spezzano, and A. Vinci, “Itema: A methodological approach for cognitive edge computing iot ecosystems,” *Future Generation Computer Systems*, vol. 92, pp. 189–197, 2019.

[13] Y. Liu, C. Yang, L. Jiang, S. Xie, and Y. Zhang, “Intelligent edge computing for iot-based energy management in smart cities,” *IEEE Network*, vol. 33, no. 2, pp. 111–117, 2019.

[14] T. Bahreini, H. Badri, and D. Grosu, “Energy-aware capacity provisioning and resource allocation in edge computing systems,” in *Proc. Intl. Conf. on Edge Computing*, 2019, pp. 31–45.

[15] F. Liu, P. Shu, and J. C. Lui, “Appatp: An energy conserving adaptive mobile-cloud transmission protocol,” *IEEE Transactions on Computers*, vol. 64, no. 11, pp. 3051–3063, 2015.

[16] S. Chen, L. Jiao, L. Wang, and F. Liu, “An online market mechanism for edge emergency demand response via cloudlet control,” in *Proc. IEEE INFOCOM*, 2019, pp. 2566–2574.

[17] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, “The cost of a cloud: research problems in data center networks,” *ACM SIGCOMM Computer Comm. Rev.*, vol. 39, no. 1, pp. 68–73, 2008.

[18] H. Badri, T. Bahreini, D. Grosu, and K. Yang, “A sample average approximation-based parallel algorithm for application placement in edge computing systems,” in *Proc. of IEEE International Conference on Cloud Engineering*, 2018, pp. 198–203.

[19] A. Shapiro, D. Dentcheva, and A. Ruszczyński, *Lectures on stochastic programming: modeling and theory*. SIAM, 2009.

[20] S. Ahmed and A. Shapiro, “The sample average approximation method for stochastic programs with integer recourse,” *ISyE Technical Report, Georgia Institute of Technology*, pp. 1–24, 2002.

[21] M. Chowdhury, M. R. Rahman, and R. Boutaba, “Vineyard: Virtual network embedding algorithms with coordinated node and link mapping,” *IEEE/ACM Transactions on Networking*, vol. 20, no. 1, pp. 206–219, 2012.

[22] D. Dutta, M. Kapralov, I. Post, and R. Shinde, “Embedding paths into trees: Vm placement to minimize congestion,” in *European Symposium on Algorithms*. Springer, 2012, pp. 431–442.

[23] T. Bahreini and D. Grosu, “Efficient placement of multi-component applications in edge computing systems,” in *Proc. of the Second ACM/IEEE Symposium on Edge Computing*, 2017, pp. 5:1–5:11.

[24] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, “Dynamic service placement for mobile micro-clouds with predicted future costs,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1002–1016, 2017.

[25] S. Wang, M. Zafer, and K. K. Leung, “Online placement of multi-component applications in edge computing environments,” *IEEE Access*, vol. 5, pp. 2514–2533, 2017.

[26] A. Al-Shuwaili and O. Simeone, “Energy-efficient resource allocation for mobile edge computing-based augmented reality applications,” *IEEE Wireless Comm. Letters*, vol. 6, no. 3, pp. 398–401, 2017.

[27] C. You, K. Huang, H. Chae, and B.-H. Kim, “Energy-efficient resource allocation for mobile-edge computation offloading,” *IEEE Trans. Wireless Comm.*, vol. 16, no. 3, pp. 1397–1411, 2017.

[28] J. Guo, Z. Song, Y. Cui, Z. Liu, and Y. Ji, “Energy-efficient resource allocation for multi-user mobile edge computing,” in *IEEE Global Communications Conference*, 2017, pp. 1–7.

[29] J. Zhang, X. Hu, Z. Ning, E. C.-H. Ngai, L. Zhou, J. Wei, J. Cheng, and B. Hu, “Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks,” *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2633–2645, 2018.

[30] S. Wang, Y. Zhao, L. Huang, J. Xu, and C.-H. Hsu, “Qos prediction for service recommendations in mobile edge computing,” *Journal of Parallel and Distributed Computing*, vol. 127, pp. 134–144, 2019.

[31] A. H. Sodhro, Z. Luo, A. K. Sangaiah, and S. W. Baik, “Mobile edge computing based qos optimization in medical healthcare applications,” *Intl. J. of Information Management*, vol. 45, no. 1, pp. 308–318, 2019.

[32] R. Buyya, A. Beloglazov, and J. Abawajy, “Energy-efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges,” *arXiv preprint arXiv:1006.0308*, 2010.

[33] A. Beloglazov, J. Abawajy, and R. Buyya, “Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing,” *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012.

[34] A. Verma, P. Ahuja, and A. Neogi, “pmapper: power and migration cost aware application placement in virtualized systems,” in *Proc. of the 9th ACM/IFIP/USENIX Int. Conf. on Middleware*. Springer-Verlag New York, Inc., 2008, pp. 243–264.

[35] S. Srikantaiah, A. Kansal, and F. Zhao, “Energy aware consolidation for cloud computing,” in *Proc. of the Conference on Power Aware Computing and Systems*, vol. 10, 2008, pp. 1–5.

[36] Y. C. Lee and A. Y. Zomaya, “Energy efficient utilization of resources in cloud computing systems,” *The Journal of Supercomputing*, vol. 60, no. 2, pp. 268–280, 2012.

[37] J. Torres, D. Carrera, K. Hogan, R. Gavaldà, V. Beltran, and N. Poggi, “Reducing wasted resources to help achieve green data centers,” in *Proc. IEEE Int. Symp. on Parallel and Distributed Processing*. IEEE, 2008, pp. 1–8.

[38] A. Beloglazov, R. Buyya, Y. C. Lee, and A. Zomaya, “A taxonomy and survey of energy-efficient data centers and cloud computing systems,” in *Advances in Computers*. Elsevier, 2011, vol. 82, pp. 47–111.

[39] A. Hameed, A. Khoshkbarforoushha, R. Ranjan, P. P. Jayaraman, J. Kolodziej, P. Balaji, S. Zeadally, Q. M. Malluhi, N. Tziritas, A. Vishnu *et al.*, “A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems,” *Computing*, vol. 98, no. 7, pp. 751–774, 2016.

[40] B. Gao, Z. Zhou, F. Liu, and F. Xu, “Winning at the starting line: Joint network selection and service placement for mobile edge computing,” in *Proc. IEEE INFOCOM*, 2019, pp. 1459–1467.

[41] Y. Sun, S. Zhou, and J. Xu, "Emm: Energy-aware mobility management for mobile edge computing in ultra dense networks," *IEEE J. on Selected Areas in Comm.*, vol. 35, no. 11, pp. 2637–2646, 2017.
[42] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *IFIP Networking Conference*, 2015, pp. 1–9.
[43] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Performance Evaluation*, vol. 91, pp. 205 – 228, 2015.
[44] C. H. Papadimitriou and J. N. Tsitsiklis, "The complexity of markov decision processes," *Mathematics of Operations Research*, vol. 12, no. 3, pp. 441–450, 1987.
[45] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang, "Adaptive user-managed service placement for mobile edge computing: An online learning approach," in *Proc. IEEE INFOCOM*, 2019, pp. 1468–1476.
[46] A. Shapiro, "On complexity of multistage stochastic programs," *Operations Research Letters*, vol. 34, no. 1, pp. 1–8, 2006.
[47] C. Swamy and D. B. Shmoys, "Sampling-based approximation algorithms for multi-stage stochastic optimization," in *Proc. 46th IEEE Symp. on Foundations of Comp. Sci.*,, 2005, pp. 357–366.
[48] A. J. Kleywegt, A. Shapiro, and T. Homem-de Mello, "The sample average approximation method for stochastic discrete optimization," *SIAM J. on Optimization*, vol. 12, no. 2, pp. 479–502, 2002.
[49] M. L. Fisher, R. Jaikumar, and L. N. Van Wassenhove, "A multiplier adjustment method for the generalized assignment problem," *Management Science*, vol. 32, no. 9, pp. 1095–1103, 1986.
[50] M. R. Garey and D. S. Johnson, *Computers and intractability: A guide to NP-completeness*. WH Freeman and Company, San Francisco, 1979.
[51] M. Piorkowski, N. Sarafijanovoc-Djukic, and M. Grossglauser, "A Parsimonious Model of Mobile Partitioned Networks with Clustering," in *The First Intl. Conf. on Communication Systems and NetworkS*, January 2009.
[52] "Fcc registered antenna towers in san francisco, ca," http://www.city-data.com/towers/cell-San-Francisco-California.html, accessed: 08/02/2019.
[53] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 732–794, 2015.
[54] Y. Mirsky, A. Shabtai, L. Rokach, B. Shapira, and Y. Elovici, "Sherlock vs moriarty: A smartphone dataset for cybersecurity research," in *Proc. ACM Workshop on Artificial Intelligence and Security*, 2016, pp. 1–12.
[55] (2009) IBM ILOG CPLEX V12.1 user's manual. [Online]. Available: ftp://ftp.software.ibm.com/software/websphere/ilog/docs/

Hossein Badri is a PhD student in Industrial & Systems Engineering and a MSc student in Computer Science at Wayne State University. Prior to joining WSU, Hossein studied in Iran where he received a MSc degree in Industrial Engineering from Shahed University, and a BSc degree in Industrial Engineering from Sharif University of Technology, Golpaygan College Of Engineering. His research interests include parallel approximation algorithms, stochastic optimization, game theory, and data driven optimization in distributed systems, logistics & transportation systems, and healthcare systems.

Tayebeh Bahreini is currently a Ph.D. student at Wayne State University, Department of Computer Science. She graduated from Shahed University in Iran with a M.Sc. degree in Computer Engineering, in 2014. She received her B.Sc. degree in Computer Science from University of Isfahan, Iran, in 2010. Her main research interests are distributed systems, approximation algorithms, parallel computing, and game theory. She is the recipient of the 2019 National Center for Women & Information Technology (NCWIT) Collegiate National Award for her research on mobile edge computing.

Daniel Grosu received the Diploma in engineering (automatic control and industrial informatics) from the Technical University of Iasi, Romania, in 1994 and the MSc and PhD degrees in computer science from the University of Texas at San Antonio in 2002 and 2003, respectively. Currently, he is an associate professor in the Department of Computer Science, Wayne State University, Detroit. His research interests include parallel and distributed computing, cloud and edge computing, parallel algorithms, approximation algorithms, and topics at the border of computer science, game theory and economics. He has published more than one hundred peer-reviewed papers in the above areas. He is a senior member of the ACM, the IEEE, and the IEEE Computer Society.

Kai Yang is a professor in the Department of Industrial and Systems Engineering, Wayne State University. He received the BSc degree in Electrical Engineering from China Petroleum University, in 1982. He received the MSc and PhD degrees in Industrial and Operations Engineering from University of Michigan in 1985 and 1990, respectively. Dr. Yang is a fellow of the Institute of Industrial and Systems Engineering. He is an expert in the field of quality and reliability engineering, data analytics and health informatics.