A Sample Average Approximation-Based Parallel Algorithm for Application Placement in Edge Computing Systems

Hossein Badri
Dept. of Ind. & Systems Eng.
Wayne State University
Detroit, USA
hossein.badri@wayne.edu

Tayebeh Bahreini
Dept. of Computer Science
Wayne State University
Detroit, USA
tayebeh.bahreini@wayne.edu

Daniel Grosu
Dept. of Computer Science
Wayne State University
Detroit, USA
dgrosu@wayne.edu

Kai Yang
Dept. of Ind. & Systems Eng.
Wayne State University
Detroit, USA
kai.yang@wayne.edu

***Abstract*—Mobile Edge Computing (MEC) is a new paradigm that aims at decreasing the response time of running mobile applications by offloading the component of the applications on the servers located at the edge of the network instead of on the cloud servers. In this paper, we address a very important problem in the management of MEC systems, that is, the problem of finding an efficient application placement on the edge servers such that the cost of execution is minimized. We develop a multi-stage stochastic programming model for the application placement problem in edge computing systems and design a novel parallel greedy algorithm based on the Sample Average Approximation method to solve it. We evaluate the performance of the proposed algorithm by conducting extensive experimental analysis using data extracted from a real-world dataset. The experimental results show that the proposed algorithm can solve the problem efficiently.**

***Index Terms*—Mobile edge computing, multi-stage stochastic programming, sample average approximation, parallel algorithm**

I. Introduction

Mobile Edge Computing (MEC) [2], [5] 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. Efficient placement of mobile applications on edge servers is one of the main challenges in MEC. Due to the mobility of users, a poor application placement might impose high execution costs. 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 minimizes the cost of execution for all users and takes into account the mobility of users and the availability of server resources.

There exist several approaches for solving the application placement problem in cloud computing [4], [6], 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.

Markov Decision Processes (MDP) have been used by several researchers to model application placement problems in MEC. Wang et al. [14] proposed 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. Adopting a similar approach, Urgaonkar et al. [13] modeled the application placement problem as an MDP and designed an online algorithm for it that is provably cost-optimal. 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 [9]. 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 placement problem. In this paper, we develop a new algorithm for solving the placement problem that is not based on MDPs and is fast and suitable for parallelization.

In this paper, we model the application placement problem in edge computing systems as a multi-stage stochastic program [1]. The stochastic programming approach allows us to take the dynamics of the location of users into account when making placement decisions. This is a significant issue in the design of efficient application placement methods in edge computing systems. Exact evaluation of the recourse function (i.e., the cost of execution in the future time slots) in stochastic optimization problems is very complex due to the large number of scenarios that have to be considered. We design a parallel implementation of the Sample Average Approximation (SAA) method to estimate the expected value of the recourse function in the multi-stage model. We also propose 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 on problem instances of different sizes considering different settings for the parameters of the algorithm.

II. The Application Placement Problem in MEC: Stochastic Programming Formulation

We consider an edge computing system consisting of a set of M servers $\{S_1, S_2, \ldots, S_M\}$ which provides services to a set of N users $\{U_1, U_2, \ldots, U_N\}$. Server S_j can create up to Q_j, $j = 1, \ldots, M$, cloud containers, each having the same processing power. User U_i, $j = 1, \ldots, N$, requests to offload and execute an application on the edge servers which requires R_i time units to be completed on a container. We assume that each container serves the request from only one user. We also assume that the placement decisions are taken at periodic intervals. We call these intervals time slots. The locations of users do not change during one time slot. The location of a user is specified by its coordinates in a two-dimensional grid of cells. 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.

The objective of the application placement problem is to minimize the total cost of the execution of users' applications. The total cost of executing the applications in a given time slot consists of three main components: (i) *computation cost:* the cost of executing a unit of request on each cloud container; (ii) *communication cost:* the cost of communication between users and servers (proportional to the distance between the location of users and servers); and, (iii) *relocation cost:* the cost associated with changing the assignment of user's request to servers during the execution. Therefore, if a user's request is completely fulfilled by a single server, no relocation cost is incurred. The relocation cost is also proportional to the distance between the location of the two servers.

In this system, a server S_j, $j = 1, \ldots, M$ is characterized by its *computation cost* γ_j. The servers in the system are heterogeneous in terms of their computation costs. The *communication cost* $\delta_{ij}^{s_t}$ between user U_i and server S_j in time slot t is dependent on the location of the user which is part of a scenario s_t. A scenario s_t consists of the location of all users at the beginning of time slot t. Here, we assume that the distance between users and servers is the Manhattan distance (i.e., if user U_i is located in cell (x_u, y_u) and server S_j is located in cell (x_s, y_s) in time slot t, then the distance between the user and the sever is given by $|x_u - x_s| + |y_u - y_s|$). 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 are known at the beginning of slot t, while the locations of users in the future slots are uncertain. We denote by $s_{[t']} := (s_t, \ldots, s_{t'})$ the history of the scenarios for the location of users from time slot t up to time slot t'.

We develop a multi-stage stochastic programming model of the 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 the first stage costs and the expected objective function value of the recourse costs (i.e., the costs of future stages) is minimized [1]. The objective function of the application placement problem for time slot t is given by:

$$c_t(X^t, Y^t, s_t) := \sum_{i=1}^{N} \sum_{j=1}^{M} \left((\delta_{ij}^{s_t} + \gamma_j) \cdot x_{ij}^t + \sum_{j'=1}^{M} \rho_{j'j} \cdot y_{ij'j}^t \right) \quad (1)$$

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:

$$\begin{aligned} \min_{X,Y} \quad & c_t(X^t, Y^t) + \mathbb{E}[c_{t+1}(X^{t+1}(s_{[t+1]}), Y^{t+1}(s_{[t+1]}), s_{t+1}) + \\ & \cdots + c_{t+T}(X^{t+T}(s_{[t+T]}), Y^{t+T}(s_{[t+T]}), s_{t+T})] \quad (2) \\ & \text{subject to:} \\ & \sum_{j=1}^{M} \sum_{t'=t}^{t+T} x_{ij}^{t'} \geq R_i \quad \forall i \quad (3) \\ & \sum_{i=1}^{N} x_{ij}^{t'} \leq Q_j \quad \forall j, t' \quad (4) \\ & \sum_{j=1}^{M} x_{ij}^{t'} \leq 1 \quad \forall i, t' \quad (5) \\ & 1 - \sum_{j=1}^{M} \sum_{l=1}^{t'-1} \frac{x_{ij}^l}{R_i} \leq \sum_{j=1}^{M} x_{ij}^{t'} + B(1 - \sum_{j=1}^{M} x_{ij}^{t'-1}) \quad \forall i, t' \quad (6) \\ & x_{ij}^{t'} - x_{ij'}^{t'-1} \leq y_{ij'j}^{t'} \quad \forall i, j, j', t' \geq 2;\ j \neq j' \quad (7) \\ & x_{ij}^{t'}, y_{ij'j}^{t'} \in \{0, 1\} \quad \forall i, j, t' \quad (8) \end{aligned}$$

The objective function (2) is to minimize the total execution cost in the first stage, and the recourse cost which is the expected value of the total cost in the future stages. Here, T is the time horizon for which the objective function is defined. Constraint (3) ensures that the requested service by a user is completely satisfied. Constraint (4) prevents loading a server beyond its capacity. Constraint (5) ensures that the request of a user is assigned to at most one server during each time slot. Constraint (6) guarantees that processing of a user's request is not interrupted in the system. In this constraint, B is a very large integer. Constraint (7) ensures that when a relocation happens the decision variables are set accordingly. Constraint (8) 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 [1] which is a Monte Carlo simulation-based approach to obtain a reliable estimation of the expected value of the recourse cost function.

III. Parallel SAA-based Placement Algorithm

The SAA method has received a considerable attention as an efficient method for solving multi-stage stochastic programs [1], [8], [12]. 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 applied in each stage (time slot) t. All the parameters 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. These samples will be used to estimate the recourse function. The next step 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. In the next step, candidate solutions are evaluated using the generated sample. Finally, the selected solution is the solution with the smallest objective value among the evaluated candidates.

In multi-stage stochastic programs, the total number of scenarios grows exponentially with the number of stages. 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 increases. 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 develop a fast parallel SAA-based greedy algorithm to solve the multi-stage stochastic program corresponding to the application placement problem.

The basic idea in the proposed parallel SAA-based algorithm for solving the application placement problem is to evaluate the candidate solutions in parallel and use a greedy procedure to solve the deterministic equivalent models corresponding to each sample. The parallel greedy SAA-based application placement algorithm, called PG-SAA, is given in Algorithm 1.

Algorithm 1 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:         Create graph $G_i(V, E)$ for user $U_i$
 5:         Use Dijkstra's algorithm to find the shortest path
            between node $D$ and node $S_j^t$ in graph $G_i(V, E)$.
 6:         $w_{ij}^k \leftarrow$ cost of the shortest path
 7:     end for
 8:     Call RM-GAP ($W^k$) to obtain the placement
 9:     Record the optimal solution as ($\bar{X}^{k,t}, \bar{Y}^{k,t}$).
10: end for
11: Generate a sufficiently large sample of size $L'$, $L' \gg L$.
12: for $k = 1$ to $H$ do in parallel
13:     for $i = 1$ to $N$ do
14:         Create graph $G_i'(V, E)$ for user $U_i$
15:         Use Dijkstra's algorithm to find the shortest
            path between the dummy vertex $D$ and vertex $S_{j_i^*}$
            in graph $G_i'(V, E)$
16:     end for
17: end for
18: Out of $H$ candidate solutions, choose the one that has
        the smallest estimated objective value
19: Allocate users' requests to servers according to $\bar{X}^*$.
```

First, PG-SAA generates H samples, each including L independent scenarios for the location of users in the upcoming T stages (line 1). The next step (lines 2-10) 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 model corresponding to each sample is formulated as follows:

$$z_L^{k,t} = \min_{X^{k,t}, Y^{k,t}} \{c_t(X^{k,t}, Y^{k,t}, s_t) + \frac{1}{L} \sum_{l=1}^{L} \sum_{t'=t+1}^{T} c_{t'}(X^{k,t'}, Y^{k,t'}, s_{[t']}^l)\} \quad (9)$$

where, $X^{k,t}$ and $Y^{k,t}$ are the vectors of decision variables in the current stage. To solve these IP problems approximately, we design a greedy heuristic which is based on the idea of evaluating the expected recourse cost of each user, independently. The first step of this procedure (line 4) 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 R_i, and E is the set of edges. We denote the vertex in G_i corresponding to server S_j in stage l, by S_j^l. The edge between server S_j^l and $S_{j'}^{l+1}$ represents the expected cost when the request of user U_i is executed on server S_j in stage l and then it is executed on server $S_{j'}$ in stage $l+1$. This expected cost includes the computation, communication, and relocation costs.

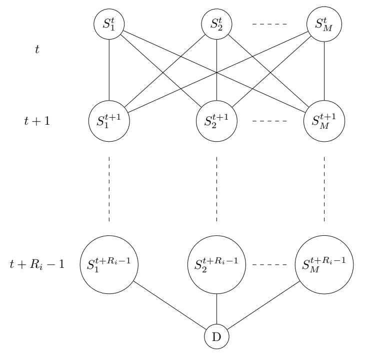


Fig. 1: Graph $G_i(V, E)$ used for computing the assignment of user U_i.

Figure 1 shows the graph $G_i(V, E)$ for the assignment of user U_i's request. Obviously, the optimal application placement of user U_i is obtained by finding the M shortest paths between the dummy vertex D and vertices S_j^t. The assignment of user U_i to the server which has the minimum value of the shortest path among all the servers is the chosen placement of user U_i.

Therefore, the expected recourse costs are obtained by finding the minimum shortest path from vertex D to vertex S_j in the current stage t. This is based on the assumption that enough capacity is available to completely satisfy the request of user U_i in R_i time slots. The cost of the shortest path, denoted by w_{ij}^k, is the expected recourse cost which includes, computation, relocation, and communication costs. Due to the limited capacity of servers, it might not be feasible to assign each user to a server as determined by the shortest paths in graphs G_i. For this reason, we need to find a refined assignment that takes into account the capacity of each server. To do that, we consider solving an associated assignment problem defined using the expected recourse costs w_{ij}^k for sample k as follows:

$$\min_X \quad \sum_{i=1}^{N} \sum_{j=1}^{M} w_{ij}^k x_{ij}^{k,t} \tag{10}$$

subject to:

$$\sum_{i=1}^{N} x_{ij}^{k,t} \le Q_j \quad \forall j \tag{11}$$

$$\sum_{j=1}^{M} x_{ij}^{k,t} = 1 \quad \forall i \tag{12}$$

$$x_{ij}^{k,t} \in \{0, 1\} \quad \forall i, j \tag{13}$$

The objective function (10) of this model is to minimize the total assignment cost. Constraint (11) ensures that the capacity of each server is not violated, constraint (12) guarantees that each user is assigned to exactly one server, and constraint (13) defines the binary variables. Obviously, this model represents the Generalized Assignment Problem (GAP) which is an NP-hard problem [7]. To solve the GAP problem, we employ the greedy algorithm proposed by Romeijn and Morales [10]. We call this algorithm RM-GAP, and for completeness, we present it in Algorithm 2. We will briefly describe the details of this algorithm later in this section. PG-SAA, calls RM-GAP to obtain a feasible assignment for the current stage (line 8), where W^k is the vector of w_{ij}^k, that is, the total costs of assignments for sample k. Next, PG-SAA records the candidate solutions obtained for each sample k as $(\bar{X}^{k,t}, \bar{Y}^{k,t})$.

Algorithm 2 RM-GAP algorithm

```
 1: Input: $w_{ij}$
 2: $Q'_j \leftarrow Q_j \quad \forall j \in \{1, \dots, M\}$
 3: $U = \{1, \dots, N\}$
 4: while $U \neq \emptyset$ do
 5:     Construct the feasibility set $F = \{j : Q'_j > 0\}$
 6:     for $i = 1$ to $N$ do
 7:         Find the server with the minimum cost:
            $j_i = \arg\min_{j \in F} w_{ij} \quad \forall i \in U$
 8:         Calculate the difference between the first and the second
            server with the lowest cost: $\Delta_i = \min_{k \in F, k \neq j_i} w_{ik} - w_{ij_i}$
 9:     end for
10:     Assign the user with the greatest value of $\Delta_i$ to server $j_i$:
        $\hat{i} = \arg\max_{i \in U} \Delta_i$
11:     Set $x_{\hat{i}j_i} = 1$
12:     Set $x_{\hat{i}j} = 0 \quad j = 1, \dots, M, j \neq j_i$
13:     $Q'_{j_i} \leftarrow Q'_{j_i} - 1$
14:     $U = U \backslash \{\hat{i}\}$
15: end while
```

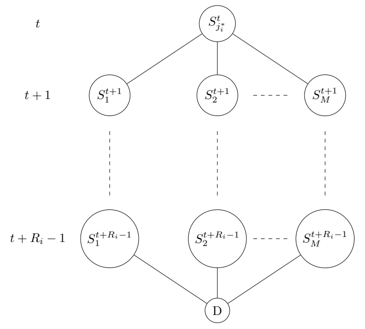


Fig. 2: Graph $G'_i(V, E)$ used when evaluating the candidate solutions.

Next, PG-SAA generates a sample with L' scenarios ($L' \gg L$) to evaluate the candidate solutions (line 11). Then, the candidate solutions are evaluated using the generated sample (lines 12-17). 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:

$$\hat{z}_{L'}^{k,t} = c_t(\bar{X}^{k,t}, \bar{Y}^{k,t}, s_t) + \frac{1}{L'} \sum_{l=1}^{L'} \sum_{t'=t+1}^{T} c_{t'}(X^{k,t'}, Y^{k,t'}, s_{[t']}^l) \tag{14}$$

where, $\bar{X}^{k,t}$ and $\bar{Y}^{k,t}$ are the vectors of solutions recorded in line 13. For the evaluation step (lines 18-20), PG-SAA uses a similar procedure as it 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 2, there is only one node in the first stage which represents the server that was assigned to user U_i 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 ($\bar{x}_{ij}^{k,t} = 1$), where $S_{j_i^*}$ is the server assigned to user U_i according to the candidate solution recorded in line 9. It should be noted that the feasibility of this assignment was already guaranteed, because this solution is obtained by the RM-GAP algorithm. Finally, the best solution is the solution with the minimum value among the evaluated candidates (lines 18-19):

$$(\bar{X}^*, \bar{Y}^*) \in \text{argmin}_{k \in \{1,\dots,H\}} \{ \hat{z}_{L'}^{k,t}(\bar{X}^{k,t}, \bar{Y}^{k,t}) \}. \quad (15)$$

We now describe briefly the RM-GAP algorithm given in Algorithm 2. We define a new variable for the capacity of servers denoted by Q'_j. First, RM-GAP sets the value of Q'_j to the total available capacity Q_j for server S_j. Here, we denote the set of scheduled users by U. RM-GAP repeats the following steps (lines 5-14) until all users are scheduled. It constructs a feasibility set which includes all the servers that have enough available capacity to create at least one container. Then, for each user it finds the server with the minimum total assignment cost w_{ij}^k. In the next step (line 8), it calculates the difference between the first and the second server with the minimum assignment costs. Then, it assigns the user with the largest difference of assignment costs (calculated in line 8) to the server which has the minimum cost (lines 10-11). In line 12, it sets all the other variables to zero, and finally in line 13 and 14 it updates the capacity of the assigned server and the set of the scheduled users.

IV. Experimental Analysis

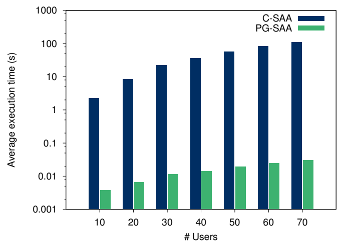
For our experiments, we consider that users are located within a two-dimensional grid of 100×100 cells. Initially, a user can be in any cell of the grid network and its location is drawn randomly from a uniform distribution over the locations of the grid. Servers are also located within the same two-dimensional grid network and the coordinates of their positions are drawn from a uniform distribution. The location of the servers is kept the same for all time slots. The capacity of servers is randomly drawn from the uniform distribution $U(1, q)$, where q is set to different values for different types of instances used in the analysis. In our setting, we assume that the mobility of the users is based on the random walk model [3]. For the sizes of the user requests R_i, we use the data from the dataset provided by the LiveLab project on real-world smart phone usage [11]. Computation costs are randomly drawn from uniform distribution $U(1, 100)$. The relocation costs are computed using $\eta d_{jj'}$, where η is a constant factor and $d_{jj'}$ is the distance between server S_j and server $S_{j'}$. Here, we set η to 10 units.

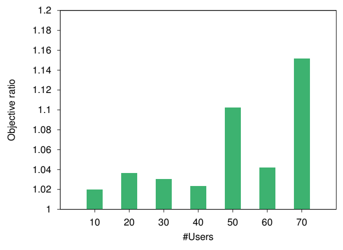
All programs implementing the algorithms were compiled using GCC version 4.8.5 and executed on a 64-core 2.4 GHz dual-processor AMD Opteron system with 512 GB of RAM and Red Hat Enterprise Linux Server as the operating system. For the experiments involving CPLEX we used the CPLEX 12 solver provided by IBM ILOG CPLEX optimization studio for academics initiative. Each experiment is performed five times and the analysis is performed based on the average value of the metrics.

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 those problems, we consider here only small sizes for the placement problem instances. We also set the maximum running time to 120 seconds for these experiments. 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 *objective ratio* $= Z^{PG-SAA}/Z^{C-SAA}$, 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.

The execution times and the objective ratios obtained for various types of small instances are presented in Figure 3. Those instances consist of $M = 5$ servers and N users ranging from 10 to 70. 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 placement problem. Figure 3a shows the execution times of C-SAA and PG-SAA algorithms for instances with five servers. We observe that for such instances, C-SAA can solve problems with up to 70 users within the time limit of 120 seconds. We also observe that the PG-SAA is very competitive in terms of execution time. While the PG-SAA takes less than one second to solve problems with less than 70 users, the execution time of the C-SAA for instances with more than 30 users is greater than 10 seconds. When it comes to the quality of solutions (Figure 3b), the solutions obtained by PG-SAA for these types of instances are within an acceptable range. For instances with 40 users or less, the objective ratio is less than 1.04, which is very close to the solution obtained by the C-SAA algorithm. We observe an increase in the objective ratio for instances with more users.

Next, we perform a set of experiments to investigate the effect of the cost parameters on the performance of PG-SAA. In order to do this, we define a metric called *cost ratio* $= \frac{b_\gamma}{\eta}$, where b_γ is the upper bound of the range of the uniform distributions used to generate the computation cost γ_i for each server in the edge system, and η is the constant factor used to calculate the relocation cost $\rho_{j'j} = \eta \cdot d_{j'j}$ between server $S_{j'}$ and server S_j. This ratio captures the magnitude of the computation costs relative to that of the communication costs. In Figure 3c, we observe that increasing the ratio negatively affects the performance of the PG-SAA. The reason for this is that by increasing the ratio, the competition among users for obtaining a server with small computation cost also increases, making the PG-SAA chose allocations leading to


(a)


(b)

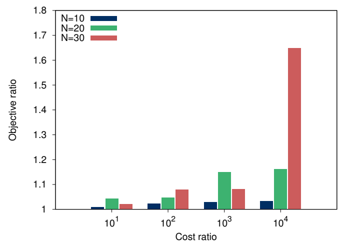
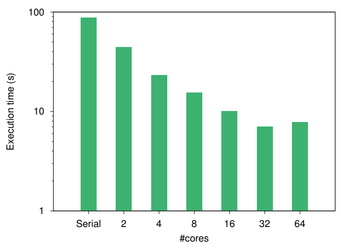

(c)

Fig. 3: PG-SAA vs. C-SAA: (a) Execution time, (b) Objective ratio, and (c) Objective ratio vs. cost ratio.


(a)

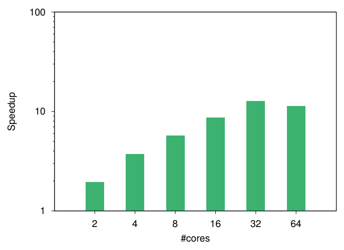

(b)

Fig. 4: PG-SAA: (a) Execution time, and (b) Speedup vs. number of cores

larger total costs.

In the next set of experiments, we investigate the performance of PG-SAA on large instances of the application placement problem. We run the PG-SAA algorithm on instances with 3000 users, and set the other parameters as follows: $M = 20$, $T = 5$, $L = 10$, and $L' = 20$. The execution times and the speedup for these large size instances are presented in Figure 4. In the figure, we use the label ”Serial” to denote the execution of the PG-SAA on only one core. The PG-SAA performs very well in terms of the running time when executed on multiple cores.

We also characterize the performance of PG-SAA in terms of Speed-up $= \frac{T_s}{T_p}$, where T_s is the execution time of the serial algorithm, and T_p is the execution time of the PG-SAA. Figure 4b shows the speed-up obtained by PG-SAA for the instances considered in the experiment. PG-SAA obtains reasonable speedup of up to 10.5 when executed on a multi-core systems with 32 cores. The small execution time of PG-SAA for large scale instances makes it very suitable for deployment in mobile edge computing systems.

V. Conclusion

We designed an efficient parallel greedy SAA-based algorithm for solving the placement problem in MEC and performed an extensive experimental analysis to investigate its performance. The experimental analysis shows that the proposed algorithm is able to obtain very good quality solution for the placement problem. The proposed algorithm requires a very small execution time and has very good speedup when executed on large multi-core systems, making it very suitable for deployment on current and future mobile edge computing systems.

References

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

[2] M. T. Beck, M. Werner, S. Feld, and S. Schimper. Mobile edge computing: A taxonomy. In *Proceedings of the Sixth International Conference on Advances in Future Internet*, pages 48 – 54, 2014.

[3] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless communications and mobile computing*, 2(5):483–502, 2002.

[4] M. Chowdhury, M. R. Rahman, and R. Boutaba. Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Transactions on Networking (TON)*, 20(1):206–219, 2012.

[5] 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 Comm. Mag.*, 52(1):132–139, 2014.

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

[7] M. L. Fisher, R. Jaikumar, and L. N. Van Wassenhove. A multiplier adjustment method for the generalized assignment problem. *Management Science*, 32(9):1095–1103, 1986.

[8] A. J. Kleywegt, A. Shapiro, and T. Homem-de Mello. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 12(2):479–502, 2002.

[9] C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of markov decision processes. *Math. of Operations Res.*, 12(3):441–450, 1987.

[10] H. E. Romeijn and D. R. Morales. A class of greedy algorithms for the generalized assignment problem. *Discrete Applied Mathematics*, 103(1):209–235, 2000.

[11] C. Shepard, A. Rahmati, C. Tossell, L. Zhong, and P. Kortum. Livelab: measuring wireless networks and smartphone users in the field. *ACM SIGMETRICS Performance Evaluation Review*, 38(3):15–20, 2011.

[12] C. Swamy and D. B. Shmoys. Sampling-based approximation algorithms for multi-stage stochastic optimization. In *proc. 46th IEEE Symp. Foundations of Comp. Science*, pages 357–366, 2005.

[13] 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*, 91:205 – 228, 2015.

[14] 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*, pages 1–9, 2015.