Extended Abstract

PhD DISSERTATION

Stochastic Optimization Methods for Resource Management in Edge Computing Systems

by

HOSSEIN BADRI
Department of Industrial & Systems Engineering, Wayne State University, Detroit, MI 48202, USA
Email: *hossein.badri@wayne.edu*

1 Introduction

Mobile Edge Cloud (MEC) has been introduced with the aim of reducing the response time of mobile applications by allowing them to perform their computation at the edge of the network. The concept of edge computing dates back to the time when Akamai [8] introduced Content Delivery Networks (CDNs) to reduce the average time of loading pages. In this system, CDN nodes are placed close to the users to prefetch and cache web content. These nodes also customize some content of web pages. For example, they customize the advertisement on the web pages according to the location of users. CDNs especially improve the performance of video content, because the bandwidth savings from caching is significant [11]. Edge computing generalizes the CDN concept in which cloud computing infrastructure is leveraged by deploying local edge nodes. The edge nodes' proximity to users helps improve the user's application performance by: (i) reducing the response time of services, (ii) increasing the scalability of the network, and (iii) enhancing the fault tolerance of the cloud services [11].

One of the *major challenges in edge computing is how to efficiently allocate the edge servers' resources to user applications efficiently.* Due to the mobility of users, a poor allocation might impose high execution costs. Computation offloading in edge computing has to consider several issues that were not present in the data-center or cloud computing settings. After the initial placement, mobile users may move to different locations. Therefore, 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 practice, there are lots of uncertainties in the network that make optimal decision making demanding.

Markov Decision Processes (MDP) have been used by several researchers to model application placement problems in MEC. 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 modeled as an MDP, and the possibility of having fast MDP-based algorithms for solving the placement problem. In this dissertation, *we address this gap in knowledge by developing novel efficient parallel algorithms for solving the placement problem* that are able to exploit the huge computing power offered by the current and future multi-core computer systems.

1.1 Contributions of this research

In this dissertation, we develop novel stochastic programming methods for solving the application placement problem in edge computing that take into account several important nondeterministic parameters of

the edge computing system. We model the application placement problem in edge computing systems as a multi-stage stochastic program. 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. We *design a parallel Sample Average Approximation (SAA)-based algorithm* to estimate the expected value of the recourse function in the multi-stage model, and 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 placement problem in a reasonable amount of time by exploiting the huge computing power of te current multi-core computer systems.

We also model the problem of energy-aware application placement in edge computing systems as a multi-stage stochastic program, where 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 parallel SAA algorithm, and conduct an extensive experimental analysis to evaluate the performance of the proposed algorithm using real-world trace data for the mobility of users.

We also *propose a risk-based optimization method based on chance-constrained programming for application placement in MEC systems*, where the objective is to maximize the total QoS of the system. In this model the resource requirements of applications is assumed to be a nondeterministic parameter. We use SAA to solve the chance-constrained program, and *develop a learning-based algorithm to solve the SAA model*.

The research presented in this dissertation has been published in several top venues in edge computing including IEEE Transactions on Parallel and Distributed Systems [7], Proceedings of the 6th IEEE International Conference on Cloud Engineering (IC2E 2018) [5], Proceedings of the 2017 ACM/IEEE Symposium on Edge Computing (SEC 2017) [4], and Proceedings of the 2018 ACM/IEEE Symposium on Edge Computing (SEC 2018) [6].

2 Parallel Sample Average Approximation for Service Placement in Edge Computing Systems

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. 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. 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.

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) Execution cost:* the cost of executing a unit of request on each cloud container, *(ii) Communication cost:* the cost of communication between 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* δ_{ij}^{st} 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. 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.

2.1 Our proposed method

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 [2]. 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 research, we employ the Sample Average Approximation (SAA) method [2, 12] which is a Monte Carlo simulation-based approach to obtain a reliable estimation of the expected value of the recourse cost function.

We *design a parallel greedy SAA-based algorithm* (called PG-SAA) for solving the application placement problem. The algorithm evaluates the candidate solutions in parallel and uses a greedy procedure to solve the deterministic equivalent models corresponding to each sample.

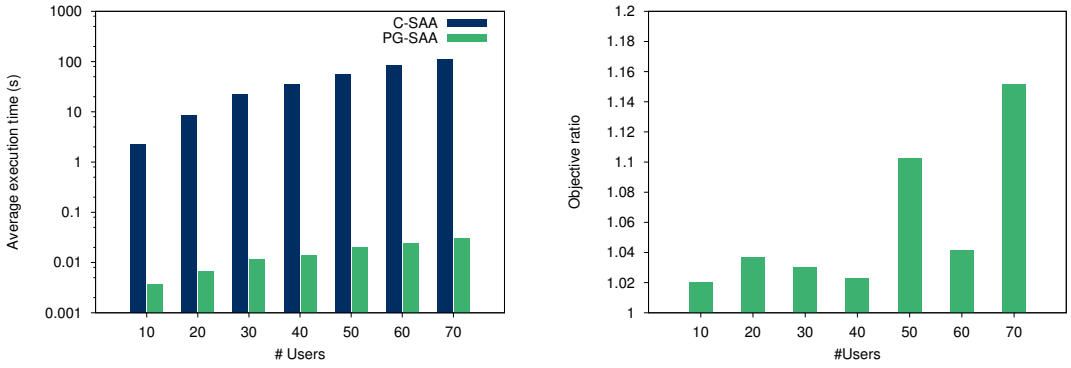
2.2 Experimental analysis

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* as follows:

$$\text{Objective ratio} = \frac{Z^{PG-SAA}}{Z^{C-SAA}} \tag{1}$$

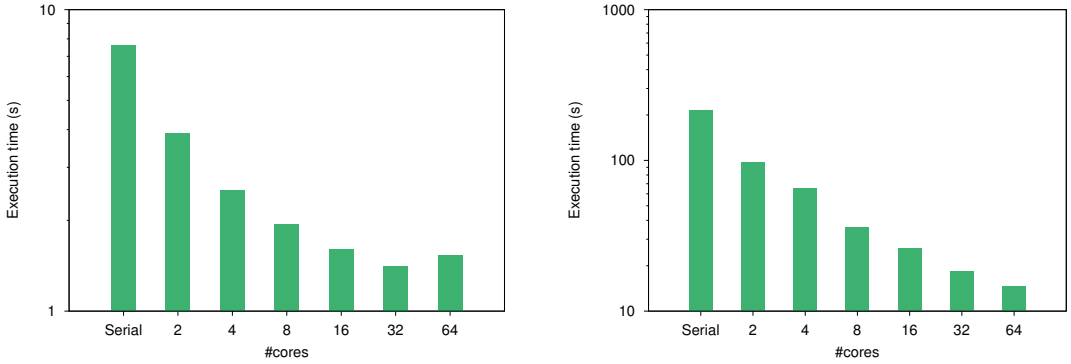
where Z^{PG-SAA} is the objective value determined by the PG-SAA and Z^{C-SAA} is the objective value determined by the CPLEX-based SAA algorithm, C-SAA.

The execution times and the objective ratios obtained for various types of small instances are presented in Figure 1. Those instances consists of $M = 5$ servers and the number of users N ranges from 10 to 70. Because of the small size of the instance 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 1a 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 1b), the solutions obtained by PG-SAA for these types of instances are within an



(a) Execution time ($M = 5$) (b) Objective ratio ($M = 5$)

Figure 1: PG-SAA vs. C-SAA: Execution time and objective ratio



(a) $N = 1000$ (b) $N = 5000$

Figure 2: PG-SAA: Execution time vs. number of cores

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.

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 different number of users ranging from 1000 to 5000, and set the other parameters as follows: $M = 20$, $T = 5$, $L = 10$, and $L' = 20$. We use two different values for the number of samples in this analysis. For the instances with $N = 1000$ we use 10 samples, while for instances with $N > 1000$ we use 20 samples. The execution times for these large size instances are presented in Figure 2. In the figures 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. An important observation on the performance of PG-SAA is that its execution time does not increase significantly with the increase in the number of users.

3 Energy-Aware Application Placement in Mobile Edge Computing

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. 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.

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, $\mathcal{U} = \{U_1, U_2, \ldots, U_N\}$. In our model, edge 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.

3.1 Our proposed method

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 [5] 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.*

Here, 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. 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* (called EPG-SAA) 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.

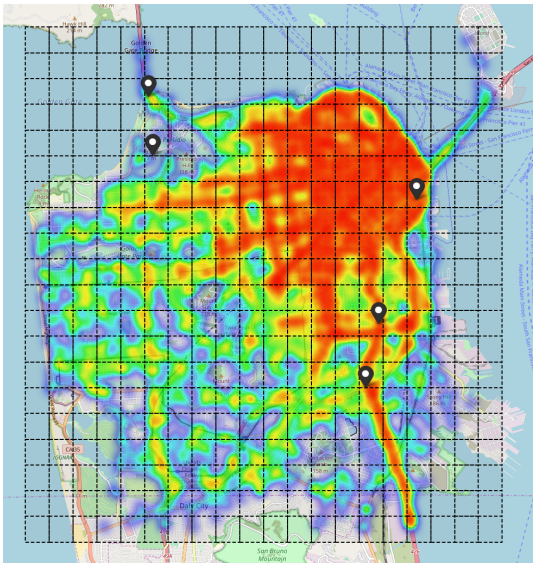
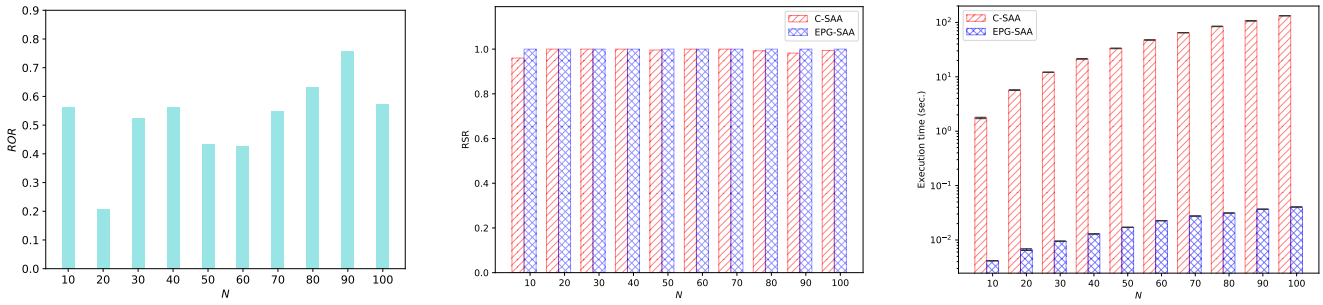


Figure 3: Distribution of users and servers in the network (servers are placed on cell towers and the locations of cell towers are marked by pointers)



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

Figure 4: EPG-SAA vs. C-SAA performance for various numbers of users, N.

3.2 Experimental analysis

We perform our analysis using a real-world data set of cab mobility traces collected in San Francisco [10] to capture distribution of demands as well as the mobility of users across the network. Figure 3 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. In our analysis, we consider that the edge servers are co-located with antenna towers. The locations of the towers are taken from [1]. We choose five base-stations registered with Verizon Wireless Co. that are shown by pointers in Figure 3. We choose Dell PowerEdge R740 Rock Server as the edge servers.

First, we compare the quality of solutions obtained by the EPG-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 EPG-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:

$$\text{ROR} = \frac{Z^{C-SAA} - Z^{EPG-SAA}}{Z^{C-SAA}} \tag{2}$$

where $Z^{EPG-SAA}$ is the objective value determined by the EPG-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

$$\text{RSR} = \frac{\sum_{i \in \mathcal{U}} \sum_{j \in \mathcal{S}} x_{i,j}}{N} \tag{3}$$

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 4. Because of the small size of the instances considered here, we execute EPG-SAA on only one core. This is because the benefits of parallelization are not observed for such small instances. We will execute EPG-SAA on multiple cores for the next set of experiments involving large-scale instances of the application placement problem. Figure 4a shows the objective ratio obtained for the small instances. The solutions obtained by EPG-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 4b, we observe that the request satisfaction ratio obtained by the EPG-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 EPG-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.

4 Risk-based Application Placement: A Learning-based Optimization Method

In this research, we propose a risked-based optimization approach to application placement in MEC systems with the aim of taking into account the risk of overloading of edge servers when making allocation decisions. Assuming that resource requirements of mobile applications are stochastic parameters, we formulate the problem as a *chance-constrained stochastic program*. In order to solve the problem in a reasonable amount of time, we employ the Sample Average Approximation (SAA) method [3] and *develop a fast machine learning based optimization method* to solve fairly large size instances. We evaluate the efficiency of the proposed approach by conducting an experimental analysis on instances with different problem settings.

We consider a two-level (i.e., cloud and edge) MEC system, and denote the set of levels by $\mathcal{L}$, where $\ell \in \mathcal{L}$, and $\ell = 1$ represents the edge level, and $\ell = 2$, the cloud level. There is one cloud server and M^1 servers at the edge level. The set of servers at the edge level and the set of all servers are denoted by $\mathcal{M}^1$ and $\mathcal{M}$, respectively. These servers provide computing resources to N independent requests from mobile applications. In our model, edge 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. In this model, cloud servers are located in a distant area from users, and the cloud server is assumed to be uncapacitated. We assume that the size of request U_i is characterized by two parameters, (i) the amount of transferred data t_i, and (ii) the size of the requested container R_i which is expressed in terms of the equivalent number of unit-size containers needed to complete the application.

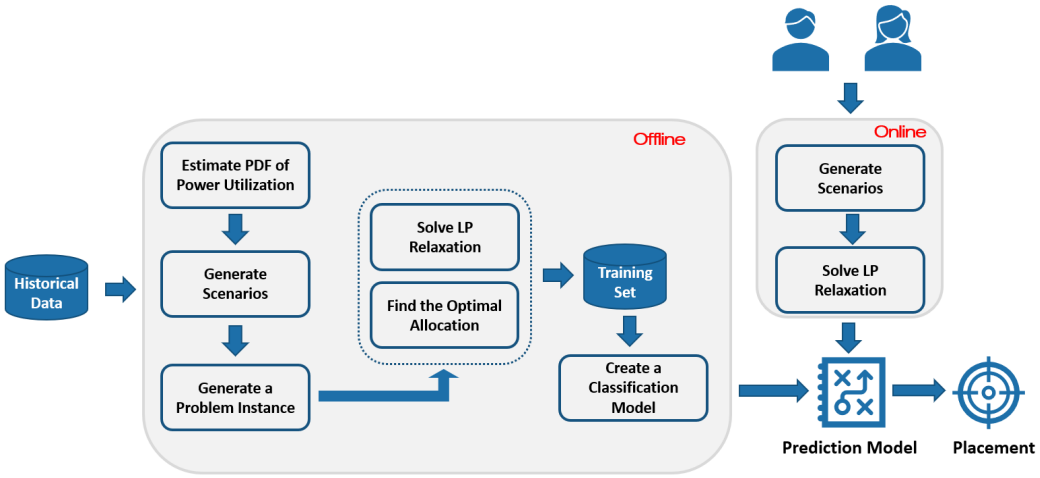


Figure 5: Learning-based Application Placement Framework.

4.1 Our proposed method

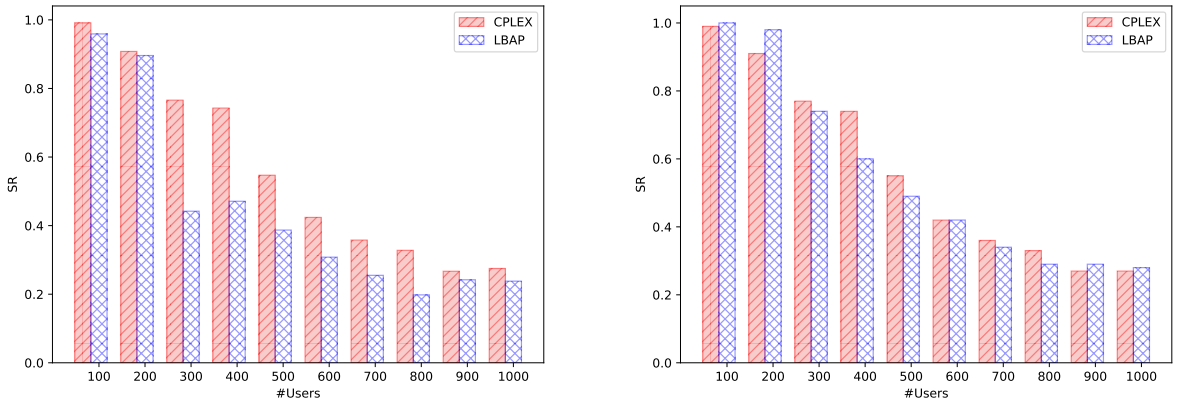
We formulate the risked-based resource provisioning in MEC systems as a chance-constrained stochastic program. In our model, the objective function is to maximize the quality of service of the system which is defined as the sum of the QoS of individual requests who receive services. Here we define the QoS, based on two important factors that determine the latency, the distance between user and server, and the size of the transfered data from the mobile device to the system. Thus, the quality of service that user U_i receives from server S_j is defined as,

$$QoS_{ij} = \frac{t_i}{d_{ij}} \tag{4}$$

where, d_{ij} is the distance between user U_i and server S_j.

In this research, we aim at leveraging machine learning techniques to develop a Learning-based Application Placement (LBAP) algorithm which is an extremely fast algorithm to solve the SAA model for the risk-based application placement in MEC systems. The framework of the LBAP algorithm is illustrated in Figure 5. Our methods consists of two major components, offline and online. The aim of the offline component is to create a prediction model based on the historical data on the size of the transferred data as well as the resource requirements of applications. The first step of this component is to estimate the Probability Distribution Function (PDF) of the power utilization of applications. The PDF is then used to generate scenarios for the resource requirements of applications. Once the scenarios are generated using the PDF, a problem instance is created and an exact solution method is employed to find the optimal solution. We also solve the LP relaxation model of the problem.

Features and the target values corresponding to the current problem instance are added to the training set. This process is repeated multiple times and for different size of instances. Once the training set is large enough, we use a multi-class classification method to create the prediction model. Since this procedure is performed in an offline fashion, the complexity of the solution algorithm and the classification method is not a challenge. In our proposed approach, it is assumed that the number servers are fixed, but every time the MEC system might receive a different number of requests. The online component of the LBAP shows how we use the prediction model for application placement in an online fashion. Every time that the optimizer in the MEC system receives a batch of requests, it generates scenarios for the resource requirements of the received requests. We should note that the size of the transferred data is deterministically known at this stage. Then the LP relaxation model is solved and the value of all the features are obtained. The trained prediction model is then used for application placement.



(a) Basic features

(b) Basic & LP relaxation features

Figure 6: Service ratio vs. number of users

4.2 Experimental Analysis

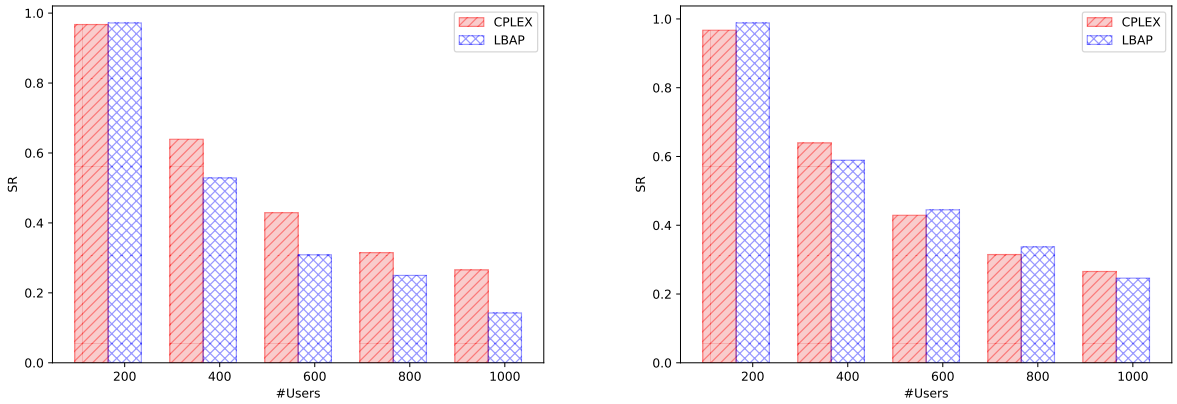
To evaluate the performance of the proposed application placement algorithm, we perform an extensive experimental analysis using real-world data. We aim at evaluating the quality of solutions and the running times of the proposed method for problem instances of different sizes. We investigate the performance of the LBAP model when only basic features are considered in the classification model, and evaluate the impact of considering features from the solution of the LP relaxation model in addition to the basic features. We evaluate the performance of the LBAP model compared to the optimal placement obtained by solving the problem by CPLEX using the Service Ratio (SR) metric which is defined as the proportion of the requests allocated to an edge server, that is,

$$SR = \frac{\sum_i \sum_j \bar{x}_{ij}^1}{N} \tag{5}$$

where, $\bar{x}_{ij}^1$ is 1 if request U_i is allocated to the edge server s_j, 0 otherwise.

We perform our analysis using two different approaches. First, we investigate the performance of the proposed method excluding the features from the solution of the LP relaxation model. We aim at investigating the accuracy of the LBAP model when only features from the SAA MIP model are included in the classification model. In this approach, we do not need to solve the LP relaxation model, and the placement is obtained without having to solve any optimization model in the online fashion. In the second part of our experimental analysis, we include the features from the solution of the LP relaxation model. Based on this approach, once the system receives the requests the LP relaxation model is solved and the optimal value of the decision variables are used as features to generate the placement using the LBAP model. Figure 6 compares the service ratio obtained by the LBAP model with the optimal service ratio obtained by solving the MIP model using the CPLEX solver. We observe that the service ratio obtained by the LBAP model is fairly close to that of the CPLEX. As expected, the LBAP model performs significantly better when LP relaxation features are used as features in the classification model 6b. In few instances, we observe that the LBAP model gives a higher service ratio compared to the CPLEX results, that is, the overloading risk is higher than the accepted risk factor (α).

In real world applications, it might not be practical to train the classification model using all possible size of instances. Therefore, we might need to train a model using only a subset of sizes. Here, we perform an analysis to investigate the performance of the LBAP model when it is generalized to be used for the sizes that were not included in the training set. For this purpose, we train the model using instances with 100, 300, 500, 700, and 900 requests and test it on instances with 200, 400, 600, 800, and 1000 requests. Figure 7 shows the service ratio versus the number of users with basic features (7a) and basic & LP relaxation features (7b). We observe that LBAP performs very well when is used for the size of



(a) Basic features

(b) Basic & LP relaxation features

Figure 7: Service ratio vs. number of users: unseen size of instances

instances that were not included in the training set. These results indicate that the LBAP model can be efficiently used for real world applications.

References

[1] Fcc registered antenna towers in san francisco, ca. http://www.city-data.com/towers/cell-San-Francisco-California.html. Accessed: 08/02/2019.

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

[3] Shabbir Ahmed and Alexander Shapiro. Solving chance-constrained stochastic programs via sampling and integer programming. In *State-of-the-Art Decision-Making Tools in the Information-Intensive Age*, pages 261–269. Informs, 2008.

[4] Hossein Badri, Tayebeh Bahreini, Daniel Grosu, and Kai Yang. Multi-stage stochastic programming for service placement in edge computing systems: poster. *Second ACM/IEEE Symposium on Edge Computing (SEC-2017)*, 2017.

[5] Hossein Badri, Tayebeh Bahreini, Daniel Grosu, and Kai Yang. Parallel sample average approximation for service placement in edge computing systems. *Proc. of The IEEE International Conference on Cloud Engineering (IC2E-18)*, 2018.

[6] Hossein Badri, Tayebeh Bahreini, Daniel Grosu, and Kai Yang. Risk-based optimization of resource provisioning in mobile edge computing. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 328–330. IEEE, 2018.

[7] Hossein Badri, Tayebeh Bahreini, Daniel Grosu, and Kai Yang. Energy-aware application placement in mobile edge computing: A stochastic optimization approach. *IEEE Transactions on Parallel and Distributed Systems*, pages 1–1, 2019.

[8] John Dilley, Bruce Maggs, Jay Parikh, Harald Prokop, Ramesh Sitaraman, and Bill Weihl. Globally distributed content delivery. *IEEE Internet Computing*, 6(5):50–58, 2002.

[9] Christos H Papadimitriou and John N Tsitsiklis. The complexity of markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[10] Michal Piorkowski, Natasa Sarafijanovoc-Djukic, and Matthias Grossglauser. A Parsimonious Model of Mobile Partitioned Networks with Clustering. In *The First Intl. Conf. on Communication Systems and NetworkS*, January 2009.

[11] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[12] Alexander Shapiro, Darinka Dentcheva, and Andrzej Ruszczyński. *Lectures on stochastic programming: modeling and theory*. SIAM, 2009.