Optimized Placement of Scalable IoT Services in Edge Computing

Adyson M. Maia*†, Yacine Ghamri-Doudane†, Dario Vieira‡, Miguel F. de Castro*
*GREat Lab - Federal University of Ceará (UFC), Fortaleza, Brazil
†University of La Rochelle, La Rochelle, France
‡Engineering School of Information and Digital Technologies (EFREI), Villejuif, France
Email: adysonmaia@great.ufc.br, yacine.ghamri@univ-lr.fr, dario.vieira@efrei.fr, miguel@great.ufc.br

***Abstract*—Edge computing is a promising concept to enable the Internet of Things (IoT) vision, especially for supporting time-sensitive applications. A challenge in this topic is the service placement problem, which concerns the decision of where to place multiple applications according to their Quality of Service (QoS) requirements on the one hand and the computing resource availability on the other hand. In this paper, we jointly investigate the load distribution and placement of scalable IoT services, both vertically and horizontally, to minimize the potential violation of their QoS requirements due to the limitations of edge computing resources. We formulate the problem as an integer nonlinear programming. In order to handle the computational complexity, we propose two approaches, one through linearization techniques and another based on genetic algorithm. Experiment results show that our linearization proposal has low levels of violation in a small-scale scenario, and both proposals outperform other methods in a large network.**

***Index Terms*—service placement, load distribution, edge computing, Internet of Things, quality of service, service scaling.**

I. Introduction

Over the past decade, the cloud computing model was broadly adopted in Information Technology (IT) domain. Despite its success, the cloud computing adoption has to overcome several challenges facing the emergence of the Internet of Things (IoT) [1], [2]. First, the rapid growth in the number of IoT devices (e.g., sensors, actuators, mobile phones, and other access devices) generates very large volumes of data that may lead to traffic congestion on the network core, data center overload, and high financial cost. Second, the large physical distance between IoT devices and cloud data centers results in high communication delays, which may be unacceptable for some time-sensitive applications (e.g., high-quality video streaming, interactive mobile gaming, augmented reality, and mission-critical applications) requiring low end-to-end latency (e.g., 10 ms or even 1 ms). Third, it is difficult for applications deployed in the cloud to quickly adapt to changes in the local context (e.g., precise user location and local network conditions) of distributed mobile devices.

Aiming to address these cloud challenges, recent research efforts introduced similar concepts extending the cloud computing capabilities closer to end-users (i.e., at the edge of networks), such as (i) Cloudlet [3], Fog Computing [4], and Mobile Edge Computing or Multi-access Edge Computing (MEC) [5]. We use the term Edge Computing (EC) to encompass these different, but partially overlapping and complementary concepts. EC adds a new layer of distributed computing nodes between end-user devices and cloud data centers. Therefore, applications running on EC can perform actions close to its users before connecting to the cloud, thus (i) reducing the network overhead, (ii) providing faster responses, and (iii) getting local contextual information in a most efficient way [6], [7].

As promising as EC is, it has some limitations. In particular, the EC nodes are more heterogeneous and have fewer capability resources (e.g., processing, memory and storage resources) compared to cloud data centers [2]. Thus, it is usually infeasible in a practical scenario to run all applications on EC. Faced with this limitation, a relevant problem to be addressed is deciding where to place multiple applications (i.e., whether on a node in the edge or within the cloud) according to infrastructure's resource constraints, applications' Quality of Service (QoS) requirements and other desired goals. This decision issue is known as the application or service placement problem, which is a non-trivial problem considering the vast, distributed, dynamic and heterogeneous edge computing environment [8].

Some studies on the service placement problem in cloud computing can be found in the literature [9], [10]. However, these proposed solutions cannot be directly applied to EC because they do not take into account the aforementioned characteristics of the edge computing environment (e.g., heterogeneous and distributed networks) and the time requirement of latency-sensitive applications. Moreover, the existing works for service placement in EC [8], [11], [12] have some limitations. First, some of them assume there are enough resources for all applications in the edge layer. In a practical scenario, some applications are deployed in a remote cloud data center due to resource constraints in the edge layer, which may lead in some cases to a violation of the application's maximum tolerable delay requirement. Second, some solutions do not take into account this time requirement or do not deal with both processing and communication delays. Third, they do not consider the vertical and horizontal scaling of applications. Horizontal scaling refers to add or remove replicas of an application in the system. On the other hand, vertical scaling

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

means to add resources to (or remove resources from) a single application's replica. At last, most of the works do not examine the impact of nodes' workload in meeting applications' QoS requirements.

Motivated by the above facts, in this paper, we intend to address the service placement problem in edge computing. Hence, we formulate the problem taking into consideration the infrastructure capacity constraints and applications' characteristics (QoS requirements, resources demand, scalability, and workload) to minimize application's QoS violation (i.e., time exceeded after a tolerable delay). The main contributions of this paper are as follows:

- We present a system model where multiple replicas of an application can be placed in different nodes and requests for this application are distributed among the replicas.
- We formulate the service placement optimization problem in the form of a Mixed-Integer Nonlinear Programming (MINLP) problem considering both node assignment (i.e., where to deploy an application) and load distribution to minimize the potential occurrence of QoS violations.
- In order to deal with the high computational complexity of solving MINLP, we propose a linearization with relaxation approach to transform MINLP into a Mixed-Integer Linear Programming (MILP) problem. Then, the MILP problem can be solved using a well-known solver tool.
- We further propose a meta-heuristic solution based on biased random-key genetic algorithm to solve the placement problem as well.
- We conduct performance evaluation over a cellular network with edge computing capabilities and compare the results of our proposed solutions with those of a greedy algorithm.

The rest of the paper is organized as follows. Section II reviews related works. Section III presents our system model. We formulate the MINLP problem and the linearization version of it in Sections IV and V, respectively. Section VI describes our genetic algorithm solution for the service placement. Then, the evaluation results are shown in Section VII. Finally, Section VIII concludes the paper.

II. Related Work

Virtual Machine (VM) placement is a well-studied topic in cloud computing. In [10], the authors review methods for VM placement and migration in a cloud environment. Pires and Barán [9] propose a taxonomy to classify these solutions. However, these approaches to conventional centralized cloud computing do not consider that an edge computing environment is more distributed, heterogeneous, latency-sensitive and it has limited resources.

Computation offloading refers to the transfer of tasks from a device to an external platform, such as the edge and cloud computing. Hence, it enables running intensive computational applications at a device with constrained resources while reducing its energy consumption. Moreover, a crucial part of offloading is deciding whether to offload or not. In [13], the authors' survey concerns computation offloading in the context of mobile edge computing scenarios. In this work, we are not interested in this offloading decision process, but service placement and computation offloading can be seen as complementary problems.

Some works address the service placement problem in the context of edge computing. In [8], the authors intend to optimize the problem of placing and moving applications in an EC architecture with multiple hierarchical tiers to minimize the overall running cost. A limitation of this work is the assumption of having sufficient resources for all applications. In [11], the authors jointly investigate the base station association, VM placement and tasks distribution problems for medical applications in MEC to minimize the overall cost while satisfying the maximum tolerable delay of the applications. However, the authors only examine the application deployment in base stations. Zhao and Liu [12] also address the problems of VM placement and load balancing in MEC. Although the objective of the work is to minimize the average response time of a request, it does not take into account the deadline requirement of this response time, especially for latency-sensitive applications.

Regarding the QoS violation, in [14], the authors propose a strategy for VM placement and migration to minimize non-green energy consumption of EC nodes and applications' end-to-end delay requirement violation. However, the work assumes EC nodes (cloudlets) have the same resource capabilities, and VMs have the same resource demands. Katsalis *et al.* [15] investigate VM scheduling and placement decision in MEC with the goal of (i) maximizing infrastructure provider revenue, (ii) minimizing Service-Level Agreement (SLA) violations, and (iii) ensuring fairness in resource allocation among service providers. Even though the work investigates SLA violation in terms of response time, it considers the processing time responsible only for the response delay, neglecting the network delay.

Few works propose a solution based on a genetic algorithm as we do. Skarlat *et al.* [16] examine the application placement in a hierarchical and distributed fog architecture to maximize the number of applications placed on fog nodes rather than the cloud while satisfying the execution deadline of the applications. However, the chromosome encoding proposed can generate infeasible solutions, then a violation penalty for these cases is added in the fitness calculation. To solve the problem of generating invalid solutions, [17] proposes a biased-random-key chromosome encoding to provide a resilient placement of mission-critical applications on geo-distributed clouds. A shortcoming of this work is that it does not consider the applications' delay requirement.

In order to overcome some limitations of these existing works, in this paper, we investigate the offline service placement in a distributed, heterogeneous and resources limited edge computing environment with scalable applications aiming to minimize QoS violations. In the offline case, we do not consider application migration and user mobility. In addition, the sets of applications and computing nodes are known in advance and do not change during placement.

III. System Model

This section introduces our system model for an edge computing environment.

A. Edge Computing System

Our EC system is a generalization of those proposed in [11], [12] for MEC. Unlike these works, in our system, computational nodes can be placed in different parts of a network and not just in base stations of a cellular network. In our proposal, the EC network consists of computational nodes (edge and cloud nodes), end-user devices, and links connecting the nodes. Furthermore, devices (fixed or mobile) are connected directly with some nodes over a wired or wireless link. Over time, devices require services from an application. Several applications, mainly IoT services, can be deployed and executed in different computing nodes, where the infrastructure and service providers define the deployment locations of these services. Then, a request sent by a device is routed among the nodes up to a node hosting the required application. Lastly, the request is processed, and its response is sent back to the device.

To illustrate our proposal, we describe the use case of a cellular network with edge computing capabilities (e.g., 5G network), as shown in Fig. 1. In this case, applications can be hosted on computing nodes located on the Radio Access Network (RAN), core network, and cloud computing regions. If an application is running on a Base Station (BS), then a request can be routed only among neighboring Base Stations (BSs) to reduce traffic at the core and to decrease transmission delays. However, not all applications can be deployed to BSs because of the limited computing resources in this region. Hence, some applications are hosted in the core or the cloud while carrying about not violating some placement criteria defined by the providers.

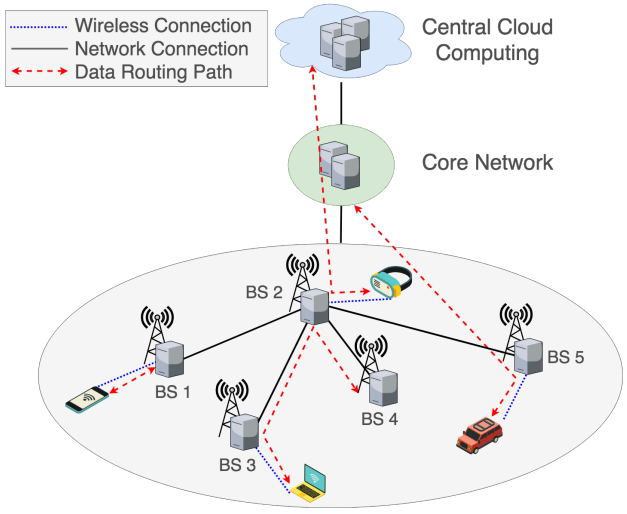


Fig. 1. Proposed edge computing system for the 5G network case.

B. Network Model

We model the network as a unidirectional graph $G = (\mathcal{V}, \mathcal{E})$, where the vertices $\mathcal{V}$ are network nodes and the edges $\mathcal{E}$ are network links between the nodes. We assume all vertices are accessible by any other vertex in the graph through multiple hops. In addition, end-user devices and their connections are not represented in G.

Each network link $e \in \mathcal{E}$ has the following property:

- **Transmission Delay** $D_{net}^{a,e}$ is the amount of time it takes for a request for application a to be transmitted in the network link e.

C. Resource Model

The proposed system model allows specifying different types of resources, where $\mathcal{R}$ is the set of these considered resources. For instance, the set $\mathcal{R} = \{CPU, RAM, Storage\}$ is made up of processing (CPU), Random-Access Memory (RAM) and disk storage resources. RAM and disk storage are measured in bytes, while CPU can be measured in cycles per second (clock rate in Hz) or instructions per second (IPS).

D. Node Model

A node, or vertex, in the graph G, represents a server with specific resource capabilities to run applications. Although multiple servers may be located on a single network node, we view these multiple servers as a single unit. Therefore, there is precisely one server for each network node. We use the terms node and server interchangeably throughout this paper. We also use the terms application and service without distinction.

Each node $v \in \mathcal{V}$ has the following feature:

- **Resource Capacity** $C_{v,r}$ is a number describing the total capacity of resource $r \in \mathcal{R}$ on node v.

We assume the cloud has unlimited resources, $C_{cloud,r} = \infty$, due to the capacity difference between cloud and edge nodes.

E. Application Model

Let $\mathcal{A}$ be the set of all different applications to be placed over the network. We consider that one or more instances, or replicas, of these applications can be deployed within the system, but these instances are independent of each other. Furthermore, a node can host only one instance of each application.

An application $a \in \mathcal{A}$ has the following parameters:

- **Maximum Tolerable Delay** D_a is a number specifying the maximum time (i.e., deadline) allowed for responding a request for application a. The response time comprises the network delay plus the processing delay.
- **Maximum Number of Instances** N_a is a value describing how application a scales horizontally.
- **Resource Demand** $f_a^r(\lambda)$ is a non-decreasing function specifying the (average) amount of resources $r \in \mathcal{R}$ required by a replica of a with a workload $\lambda \geq 0$. We define this workload λ as the (average) arrival rate of requests in an instance of a. This function describes how an application scales vertically.

- **Processing Work Size** W_a is a value indicating the (average) amount of processing required to get a response to a request for a. It is measured by the number of instructions or clock cycles required to process a request.
- **Request Rate** λ_a is the average request generation rate for a of each end-user device requesting this application. It is determined by a Poisson distribution.

F. User Model

End-user devices, or users, are not aware of where applications are deployed and which application's instance will handle their requests. Therefore, we can distribute these requests among multiple replicas placed on the system. For the safe of simplicity, we assume each user requests for only one application. Then, let U_a^v be the number of users connected to node v requesting application a, and U_a is the total number of users requesting application a in the system.

IV. Problem Statement and Formulation

A. Problem Statement

In a practical scenario, it is not possible to place all applications on the network edge given the resource limitations of EC nodes. Consequently, some applications are deployed further (i.e., in the core network or the cloud) from their end-users. This considerable distance between node and user may result in the response time of a request to exceed the deadline specified by some applications. Moreover, an overloaded server also increases response time, thus distributing the load among application's replicas may mitigate this issue. Therefore, we investigate the joint problem of service placement and load distribution to minimize the violation of QoS (maximum tolerable delay).

The next subsection presents an estimation to determine the response time of a request.

B. Response Time Estimation

We define a request flow $F_a^{u,v}$ as the requests for a replica of application $a \in \mathcal{A}$ hosted on node $v \in \mathcal{V}$ (target node) and generated by users connected to node $u \in \mathcal{V}$ (source node). Thus, (1) specifies the average response time of a request flow $F_a^{u,v}$, where $\overline{d}_{net}$ is the average time to send requests to a from users in u to node v and $\overline{d}_{proc}$ is the average processing time of requests on v. We estimate both network and processing delay as follow.

$$\overline{d}(F_a^{u,v}) = \overline{d}_{net}(F_a^{u,v}) + \overline{d}_{proc}(a,v) \tag{1}$$

*1) **Network Delay**:* The network delay of a request includes: (i) the communication delay between the requesting end-user device and the node to which it is attached, and (ii) the transmission delay from this latter node to a server hosting the application following a multi-hop routing path. It is important to note that a user's attachment node can host the application and process its requests, and therefore the transmission delay of the second part is zero. Since we are examining the offline problem case, the communication delay between a device and its attachment node does not affect the placement decision [12]. Therefore, we do not consider this communication delay in the network delay estimation.

We estimate the average network delay of a request flow $F_a^{u,v}$ as:

$$\overline{d}_{net}(F_a^{u,v}) = D_{net}^{a,u,v} = \begin{cases} 0 & \text{if } u = v \\ \sum_{e \in \mathcal{P}_{u \to v}} D_{net}^{a,e} & \text{otherwise} \end{cases} \tag{2}$$

where $\mathcal{P}_{u \to v}$ is the set of links in a routing path from u to v. This set can be determined by some shortest routing path algorithm, such as the Floyd–Warshall algorithm [18], [19].

*2) **Processing Delay**:* We model the request processing time as an M/M/1 queueing model. In this model, users continuously generate requests for an application a according to a homogeneous Poisson process with ratio λ_a. Furthermore, request arrival rate λ_a^h for application a running on node v is defined as the sum of all requests arriving at this node. Eq. (3) expresses this request arrival rate, where $\delta_a^{u,v} \in [0, Q_a^u]$ is an integer variable indicating the size of request flow $F_a^{u,v}$ (i.e., number of requests in the flow), and $Q_a^u = \lceil U_a^u \lambda_a \rceil$ is the number of requests for a generated by users connected in u.

$$\lambda_a^v = \sum_{u \in \mathcal{V}} \delta_a^{u,v} \tag{3}$$

Service times have an exponential distribution with rate parameter μ, where $1/\mu$ is the average service time in an M/M/1 queue. Thus, we express $1/\mu_a^h$ as the time to perform the request's CPU work W_a with the resources allocated for a replica of application a in node v as:

$$\frac{1}{\mu_a^v} = \frac{W_a}{f_a^{CPU}(\lambda_a^v)} \tag{4}$$

Finally, (5) gives the average processing time of requests for application a running on node v according to Little's law.

$$\overline{d}_{proc}(a,v) = \frac{1}{\mu_a^v - \lambda_a^v} \tag{5}$$

C. Problem Constraints

A solution to the offline service placement problem is feasible only if all the following constraints are met.

*1) **Number of Instances**:* A node can only host a single replica of a given application. Then, let $\rho_a^v \in \{0,1\}$ be a binary variable to indicate whether node v hosts an instance of application a or not. Moreover, the number of instances deployed in the system must respect the limits defined by the applications, and all of them need to be placed.

$$1 \leq \sum_{v \in \mathcal{V}} \rho_a^v \leq N_a \quad \forall a \in \mathcal{A} \tag{6}$$

*2) **Request Flow Existence**:* A request flow $F_a^{u,v}$ only exists if a replica of application a is placed on v and there are users connected in u requesting a. Let $\gamma_a^{u,v} \in \{0,1\}$ be a binary variable to express whether flow $F_a^{u,v}$ exists or not.

$$\gamma_a^{u,v} \leq \rho_a^v Q_a^u \quad \forall a \in \mathcal{A}, \forall u,v \in \mathcal{V} \tag{7}$$

*3) **Request Flow Size**:* If a flow $F_a^{u,v}$ exists, its size must be at least one and at most equal to the number of requests generated by users connected in u.

$$\gamma_a^{u,v} \leq \delta_a^{u,v} \leq \gamma_a^{u,v} Q_a^u \quad \forall a \in \mathcal{A}, \forall u, v \in \mathcal{V} \tag{8}$$

*4) **Load Conservation**:* The aggregate size of all request flows for application a from the same source node u is equal to the total number of requests for a generated by users connected to this node.

$$\sum_{v \in \mathcal{V}} \delta_a^{u,v} = Q_a^u \quad \forall a \in \mathcal{A}, \forall u \in \mathcal{V} \tag{9}$$

*5) **Node's Capacity**:* The total amount of resources demanded by applications placed on a server should not exceed its capacity.

$$\sum_{a \in \mathcal{A}} \rho_a^v f_a^r(\lambda_a^v) \leq C_{v,r} \quad \forall r \in \mathcal{R}, \forall v \in \mathcal{V} \tag{10}$$

*6) **Queue Stability**:* An M/M/1 queue is stable only if the average service rate is larger than its average arrival rate. This stability needs to be guaranteed for each application placed on a node.

$$\lambda_a^v < \mu_a^v \quad \forall a, v \left(\rho_a^v = 1\right), a \in \mathcal{A}, v \in \mathcal{V} \tag{11}$$

*7) **QoS Violation**:* The response time of an existing flow $F_a^{u,v}$ should not exceed its application's deadline plus the system's QoS violation level $\varepsilon \geq 0$.

$$\gamma_a^{u,v} \overline{d}(F_a^{u,v}) \leq D_a + \varepsilon \quad \forall a \in \mathcal{A}, \forall u, v \in \mathcal{V} \tag{12}$$

D. Objective Function

We define the QoS violation level of a request flow $F_a^{u,v}$ as the difference between its average response time and the application's deadline, i.e., $\left(\overline{d}(F_a^{u,v}) - D_a\right)$. The QoS violation level of the system ε is the highest violation level among all flows in the system. Therefore, our goal is to minimize the QoS violation level ε. Then, the offline service placement problem is formulated as follows:

$$\begin{aligned} &\min \varepsilon \\ &\text{subject to} \quad \text{eqs. (6) to (12)} \end{aligned} \tag{13}$$

Table I lists the major notations used in this paper.

TABLE I
NOTATIONS USED IN THE PROPOSED SYSTEM MODEL

Symbol	Description
Input Parameters	
$\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{A}$	set of network nodes, network links, resource types, and applications respectively
$C_{v,r}$	total capacity of resource r on node v
D_a	maximum tolerable response time of application (app) a
N_a	maximum number of replicas for app a
$f_a^r(\lambda)$	demand of resource r for a replica of app a with workload λ
$K_1^{a,r}, K_2^{a,r}$	constants of a linear resource demand for app a and resource r, $f_a^r(\lambda) = K_1^{a,r}\lambda + K_2^{a,r}$
W_a	CPU work size of a request for app a
λ_a	request generation rate for app a
U_a^v	number of users connected to node v requesting app a
$D_{net}^{a,u,v}$	network delay for app a between nodes u and v
Variables	
ρ_a^v	whatever node v deploys an instance of app a or not
$\gamma_a^{u,v}$	whatever request flow $F_a^{u,v}$ exists or not
$\delta_a^{u,v}$	number of requests in the flow $F_a^{u,v}$
ε	system's QoS violation level
Others	
$F_a^{u,v}$	flow of requests from users connected to node u to an instance of app a deployed on node v
λ_a^v	request arrival rate of app a on node v
μ_a^v	service rate of app a on node v
Q_a^u	number of requests for app a generated from users connected to node u
Q_a	total number of requests for app a in the system

V. LINEARIZATION AND RELAXATION PROPOSAL

The optimization problem (13) is a Mixed-Integer Nonlinear Programming (MINLP) problem because constraints (10) to (12) are nonlinear. MINLP is usually difficult to solve due to its high computational complexity [20]. One way to reduce this complexity is to apply linearization and relaxation techniques. Therefore, we transform (13) into a Mixed-Integer Linear Programming (MILP) problem by employing these techniques to the following nonlinear constraints:

Node's Capacity. For some applications, the resource demand function $f_a^r(\cdot)$ may be nonlinear. In this case, it can be replaced by an over linear estimator ${f^*}_a^r(\cdot)$ in the domain interval $[0, Q_a]$, as shown in (14), where $K_1^{a,r}$, $K_2^{a,r}$ are constants, and Q_a is equal to $\sum_{v \in \mathcal{V}} Q_a^v$.

$${f^*}_a^r(\lambda) = K_1^{a,r}\lambda + K_2^{a,r} \tag{14}$$

Given that requests only arrive at servers running the requested application according to (3), (7) and (8), we have:

$$\rho_a^v \lambda_a^v = \lambda_a^v \tag{15}$$

By applying (14) and (15) to (10), the node's capacity constraint can be rewritten as:

$$\sum_{a \in \mathcal{A}} \left(\lambda_a^v K_1^{a,r} + \rho_a^v K_2^{a,r}\right) \leq C_{v,r} \quad \forall r \in \mathcal{R}, \forall v \in \mathcal{V} \tag{16}$$

Queue Stability. To obtain a standard form of a MILP problem, it must remove the strictness of inequality in (11). For this, it is added a small constant $\Theta \approx 0$. Moreover, both sides of the inequality are multiplied by ρ_a^v to ensure the queue existence constraint. Applying (4), (14) and (15) to this result, we further have:

$$\begin{aligned} \lambda_a^v \left(K_1^{a,CPU} - W_a\right) + \rho_a^v K_2^{a,CPU} \geq \rho_a^v \Theta \\ \forall a \in \mathcal{A}, \forall v \in \mathcal{V} \end{aligned} \tag{17}$$

QoS Violation. Given the equations (1), (2), (4), (5) and (14), we rewrite constraint (12) as:

$$\left(\gamma_a^{u,v}\lambda_a^v D_{net}^{a,u,v} - \varepsilon\lambda_a^v - \lambda_a^v D_a\right)\left(K_1^{a,CPU} - W_a\right) + \gamma_a^{u,v}\left(K_2^{a,CPU} D_{net}^{a,u,v} + W_a\right) - K_2^{a,CPU}\left(D_a + \varepsilon\right) \leq 0 \quad \forall a \in \mathcal{A}, \forall u, v \in \mathcal{V} \quad (18)$$

However, in (18), both $\gamma_a^{u,v}\lambda_a^v$ and $\varepsilon\lambda_a^v$ are bilinear terms. We can relax these terms to obtain linear terms using McCormick's envelopes [21]. That is, we replace these bilinear terms with new variables ($\varphi_a^{u,v} = \gamma_a^{u,v}\lambda_a^v$ and $\psi_a^v = \varepsilon\lambda_a^v$) and add the following new constraints in the problem:

$$0 \leq \gamma_a^{u,v} \leq 1 \text{ and } 0 \leq \lambda_a^v \leq Q_a \text{ and } 0 \leq \varepsilon \leq E \quad (19a)$$

$$0 \leq \varphi_a^{u,v} \leq \lambda_a^v \text{ and } Q_a\left(\gamma_a^{u,v} - 1\right) + \lambda_a^v \leq \varphi_a^{u,v} \leq \gamma_a^{u,v} \quad (19b)$$

$$0 \leq \psi_a^v \leq \lambda_a^v E \text{ and } \varepsilon Q_a + \lambda_a^v E - E Q_a \leq \psi_a^v \leq \varepsilon Q_a \quad (19c)$$

where E is a constant specifying the maximum level of QoS violation allowed. Then, we can rewrite (18) with the two new variables to have a linear constraint:

$$\left(\varphi_a^{u,v} D_{net}^{a,u,v} - \psi_a^v - \lambda_a^v D_a\right)\left(K_1^{a,CPU} - W_a\right) + \gamma_a^{u,v}\left(K_2^{a,CPU} D_{net}^{a,u,v} + W_a\right) - K_2^{a,CPU}\left(D_a + \varepsilon\right) \leq 0 \quad \forall a \in \mathcal{A}, \forall u, v \in \mathcal{V} \quad (20)$$

At last, we formulate the MILP problem as follows:

$$\begin{aligned} &\min \varepsilon \\ &\text{subject to} \quad (6) \text{ to } (9), (16), (17), (19) \text{ and } (20) \end{aligned} \quad (21)$$

It is important to note that a solution to (21) is also feasible for (13), but it may present a higher objective value ε when applied to the original problem due to the bilinear relaxation.

VI. A Genetic-Based Proposal

Although well-known solvers, such as CPLEX [22], can solve MILP problems, these problems are generally NP-Hard [12]. Moreover, (21) is highly time-consuming due to a large number of integer variables. Hence, we propose a metaheuristic solution based on genetic algorithms. An advantage of a genetic approach is that it is not limited to convex or linear problems [23].

The proposed genetic algorithm uses biased random-key chromosomes [24], which is an array of randomly generated real numbers in the interval $[0, 1]$. This chromosome representation is used not to create infeasible solutions that may degrade the genetic algorithm performance [25]. The proposal uses an elitist strategy, keeping elite individuals, i.e., those with the best fitness values, to the next generation. It also adds new random generated individuals, as mutants, in the next generation. In addition, to complete the population, offsprings are generated by a parameterized uniform crossover [26] between elite and non-elite individuals.

A. Chromosome Representation

In biased random-key algorithms, a deterministic decoder algorithm takes an individual's chromosome and computes his fitness value. Thus, the representation of a chromosome and the decoder algorithm plays important rules in our proposal.

Our proposal for chromosome encoding and the description of its parts are given below:

$$C = \Big[O_1^1, O_1^2, \cdots, O_1^{|\mathcal{V}|}, \cdots, O_{|\mathcal{A}|}^1, O_{|\mathcal{A}|}^2, \cdots, O_{|\mathcal{A}|}^{|\mathcal{V}|}, M_1, M_2, \cdots, M_{|\mathcal{A}|}, V_1^1, V_1^2, \cdots, V_1^{|\mathcal{V}|}, \cdots, V_{|\mathcal{A}|}^1, V_{|\mathcal{A}|}^2, \cdots, V_{|\mathcal{A}|}^{|\mathcal{V}|}\Big]$$

1) $O_1^1, \cdots, O_{|\mathcal{A}|}^{|\mathcal{V}|}$. It is the creation order of request flows.
2) $M_1, \cdots, M_{|\mathcal{A}|}$. It describes a weight used in choosing a server to host an application.
3) $V_1^1, \cdots, V_{|\mathcal{A}|}^{|\mathcal{V}|}$. Along with the previous part, it is a parameter to compute node's priority to be chosen as a place to deploy an application. This priority of a node v for application a is given as:

$$M_a^v V_a^v + (1 - M_a^v)\frac{D_{net}^{a,u,cloud} - D_{net}^{a,u,v}}{D_{net}^{a,u,cloud}} \quad (22)$$

B. Decoder Algorithm

A simple greedy solution to the service placement problem deploys an application on the closest servers to users of this application, i.e., servers with less network delay to these users. However, the capacity limitation of nodes prevents this deployment scheme from optimally working for a large number of applications or users. To improve this solution, we propose the inclusion of another parameter (part V on the chromosome) in addition to the network delay in the node selection procedure. Furthermore, a weight factor (part M on the chromosome) balances these two parameters in the decision process.

We designed Algorithm 1 based on the idea above. In its outermost loop (line 6), there is an iteration over all possible sources of request flow, where the first part (i.e., O_a^u part) of the chromosome defines the loop order. Then, in line 8, it checks for all possible flow targets ordered by the specification in (22). In the innermost loop (lines 10 to 17), it tries to allocate the maximum number of requests to the chosen target node, while respecting constraints (10) and (11). It is important to note that this loop is finite due to the assumption of unlimited resources of a cloud node. If the number of replicas exceeds the maximum allowed, the algorithm does a local search optimization (line 21) by replacing surplus replicas with the cloud. Finally, it computes the QoS violation level and returns this level as the fitness value for the input individual.

VII. Evaluation

In this section, we present the performance (i.e., the optimality) results of our MILP and genetic solutions by comparing it with other algorithms over a cellular network (5G) with EC capabilities.

Algorithm 1 Chromosome Decoder Algorithm.

procedure DECODER($individual$)
  initialize $\rho_a^v, \gamma_a^{u,v}, \delta_a^{u,v} \leftarrow 0$
  $O, M, V \leftarrow individual.chromosome$
  $l \leftarrow \{(a,u), \forall a \in \mathcal{A}, \forall u \in \mathcal{V}\}$
  order l according to O_a^u
  for all $(a,u) \in l$ **do**
    order $\mathcal{V}$ according to (22)
    for all $v \in \mathcal{V}$ **do**
      $q \leftarrow Q_a^u - \sum_{i \in \mathcal{V}} \delta_a^{u,i}$
      while $q > 0$ and $\sum_{i \in \mathcal{V}} \delta_a^{u,i} < Q_a^u$ **do**
        if constraints (10) and (11) are respected when receiving more q requests for application a **then**
          $\delta_a^{u,v} \leftarrow \delta_a^{u,v} + q$
          $\gamma_a^{u,v} \leftarrow 1$; $\rho_a^v \leftarrow 1$; $q \leftarrow 0$
        else
          $q \leftarrow q - 1$
        end if
      end while
    end for
    $x \leftarrow N_a - \sum_{i \in \mathcal{V}} \rho_a^i$
    if $x > 0$ **then**
      replace x replicas of a with the cloud node
    end if
  end for
  return $\varepsilon \leftarrow$ compute QoS violation with $\rho_a^v, \gamma_a^{u,v}, \delta_a^{u,v}$
end procedure

This section is structured as follows. First, the evaluated algorithms are detailed. Second, the experiment setup is presented. Then, we analyze the obtained results.

A. Evaluated Algorithms

An overview of the compared methods is given below:

- ***Cloud***. It simply places everything in the cloud.
- ***Greedy***. It follows the description of a simple greedy algorithm with the local search optimization presented in Subsection VI-B. Also, It first chooses applications with shortest deadlines to be placed.
- ***Genetic***. This is our genetic proposal. To improve performance, we include in the initial population an individual with chromosome encoding of the *Greedy* solution ($M_a = 0, V_a^v = 0$).
- ***MILP***. It presents the optimal result for problem (21) by applying the branch and cut technique in the solver.
- ***MILP-MINLP***. It returns the results for (13) based on the optimal solution found in *MILP*.
- ***MILP-MINLP-T***. It is similar to *MILP-MINLP*, but it returns the results when a timeout parameter is used to stop the solver tool. If the solver does not find a solution after the timeout, *Cloud* method is used instead.

B. Experimental Setup

We developed the experiment in Python with the CPLEX solver [22] to compare the above mentioned methods in a 5G network scenario. We followed the 5G Key Performance Indicators (KPI) [27] to specify the network delays less than 1 ms. We used three types of applications as planned for 5G: (i) enhanced Mobile Broadband (eMBB), (ii) Ultra Reliable Low Latency Communications (URLLC) and (iii) massive Machine Type Communications (mMTC). We also randomly defined application's parameters based on 5G predictions [27], [28] and so that URLLC and mMTC demand fewer resources than eMBB. Moreover, users are uniformly distributed between base stations, which are arranged in a hexagonal grid. In this grid, neighboring base stations are directly connected. Computational nodes are placed in different network parts (base stations, core network, and in the cloud), and their capacities are reduced as they descend into the network topology (i.e., from cloud to base stations). Lastly, each test case is executed 30 times to obtain results with 95% confidence interval [29].

TABLE II
EXPERIMENT PARAMETERS

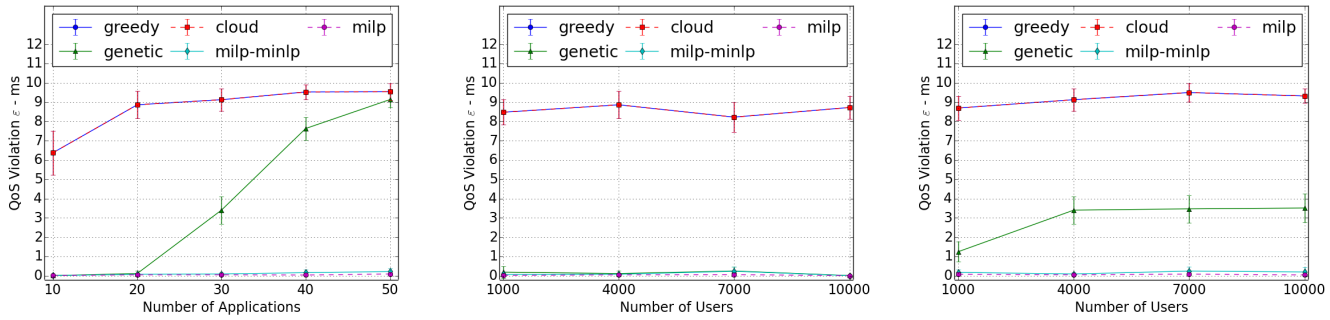
Parameter	Value
System	
Base Stations (BSs)	7, 19
Network Delay	BS-BS: 1 ms, BS-Core: 1 ms Core-Cloud: 10 ms
CPU (MIPS)	Cloud: unlimited, Core: 200000, BS: 50000
Storage (MB)	Cloud: unlimited, Core: 10000, BS: 1000
Number of Users	1000, 4000, 7000, 10000
User Proportion	70% mMTC, 20% eMBB, 10% URLLC
Number of Applications	10, 20, 30, 40, 50
Application Proportion	34% mMTC, 33% eMBB, 33% URLLC
Applications	
N_a	$[1, \lvert\mathcal{V}\rvert]$
D_a (ms)	$[50, 1000]$ mMTC, $[10, 50]$ eMBB, $[1, 10]$ URLLC
λ_a (requests/ms)	$[0.001, 0.01]$ mMTC, $[0.02, 0.01]$ eMBB, $[0.02, 0.01]$ URLLC
$K_1^{a,Storage}$, $K_2^{a,Storage}$ (MB)	$[1, 10]$ mMTC, $[1, 50]$ eMBB, $[1, 10]$ URLLC
W_a (MI)	$[1, 5]$ mMTC, $[1, 10]$ eMBB, $[1, 5]$ URLLC
$K_1^{a,CPU}$ (MIPS)	$W_a + 1$
$K_2^{a,CPU}$ (MIPS)	$[0, W_a + 1]$

Table II summarizes the major experiment parameters used, where an interval $[a, b]$ means that a value is chosen randomly within this range.

C. Results and Discussion

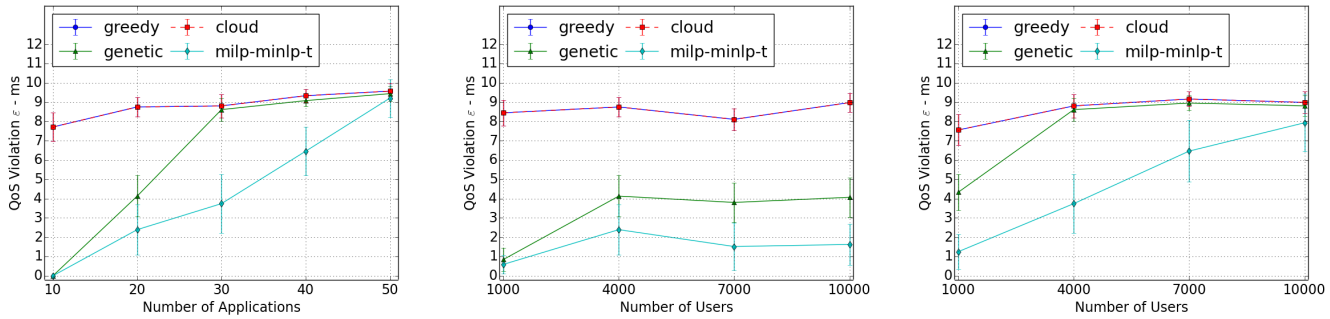
We first evaluated the performance of placement approaches in a small-scale network scenario with 7 BSs (1 BS with 6 BSs around it). We used this network size due the highly time-consuming to find the optimal solution of (21). Despite this, we evaluated how close the approaches are to the optimal solution of (21). Thus, *MILP* is a lower bound benchmark.

Fig. 2 presents the impact of increasing the number of applications and users on the level of QoS violation in a small-scale network scenario. In Fig. 2(a), we can see that *Greedy*



(a) 4000 users (b) 20 applications (c) 30 applications

Fig. 2. Performance in a 5G network with 7 base stations



(a) 4000 users (b) 20 applications (c) 30 applications

Fig. 3. Performance in a 5G network with 19 base stations.

has the same results as *Cloud*, which is partially due to the local search optimization. Considering the confidence interval, *MILP* and *MILP-MINLP* present equal results close to zero, which are the most desired results. *Genetic* shows a rise in the violation level with an increasing number of applications. An explanation for this behavior is that allocating the maximum number of requests on a server is not a satisfactory strategy in an environment with high resource demands. We can see in Figs 2(b) and 2(c) that increasing the number of users does not have much effect on performance.

In a second scenario, we analyzed the placement methods in a network with 19 BSs (1 central BS, a ring of 6 BSs around the central one and 12 BSs around the first ring). Due to processing time, *MILP* and *MILP-MINLP* are replaced by *MILP-MINLP-T*. In this scenario, *Cloud*, *Greedy*, and *Genetic* have the similar behavior as in the small-scale network when there is an increase of applications, as shown in Fig. 3(a). Contrary to the first scenario, the timeout version for *MILP-MINLP* presents a growth in the QoS violation with an increasing number of applications. This occurs due to the timeout parameter limiting the solver to find a better solution and because the relaxation in (20) may result in worse values than the optimal for the MINLP problem. Nevertheless, *MILP-MINLP-T* has slightly better values than *Genetic*.

Figs. 3(b) and 3(c) present the performance impact of increasing the number of users. We can observe an increase in violation levels from 1000 to 4000 users, and then it remains almost constant up to 10000 users, mainly for *Cloud*, *Greedy*, and *Genetic* solutions. This behavior can be explained by the placement of applications in the cloud, which increases the network delay. On the other hand, a cloud node has unlimited resources to receive the rise of users without greatly affecting processing performance. Meanwhile, we can see a steady increase in the violation level for *MILP-MINLP-T* method in Fig. 3(c), but its values are smaller than the other methods. The growth in cloud usage due to increased resource demand is an explanation for this observed result.

VIII. Conclusion

In this paper, we investigated the offline placement problem of IoT services supporting horizontal and vertical scaling in an edge computing environment. First, we formulated this problem into a MINLP problem. Then, we proposed a linearization and genetic-based methods to solve the problem and deal with its high computational complexity. Through experiments, we showed that linearization has low violation level of application's deadline requirement in a small-scale cellular network scenario. In a scenario with a network larger than the small-scale network, our two approaches (linearization and genetic-based) have close results, and both generally had better results than a simple greedy solution.

As a future work, we plan to investigate the online service placement problem, which includes application migration, user mobility, and other dynamic changes in the network.

REFERENCES

[1] J. Pan and J. McElhannon, "Future edge cloud and edge computing for internet of things applications," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 439–449, Feb 2018.

[2] H. Liu, F. Eldarrat, H. Alqahtani, A. Reznik, X. de Foy, and Y. Zhang, "Mobile edge cloud system: Architectures, challenges, and approaches," *IEEE Systems Journal*, vol. 12, no. 3, pp. 2495–2508, Sept 2018.

[3] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct 2009.

[4] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 13–16. [Online]. Available: http://doi.acm.org/10.1145/2342509.2342513

[5] ETSI, "Mobile edge computing (mec); framework and reference architecture," *ETSI, DGS MEC*, vol. 3, 2016.

[6] H. El-Sayed, S. Sankar, M. Prasad, D. Puthal, A. Gupta, M. Mohanty, and C. Lin, "Edge of things: The big picture on the integration of edge, iot and the cloud in a distributed computing environment," *IEEE Access*, vol. 6, pp. 1706–1717, 2018.

[7] K. Bilal, O. Khalid, A. Erbad, and S. U. Khan, "Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers," *Computer Networks*, vol. 130, pp. 94 – 120, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128617303778

[8] W. Tärneberg, A. Mehta, E. Wadbro, J. Tordsson, J. Eker, M. Kihl, and E. Elmroth, "Dynamic application placement in the mobile cloud network," *Future Generation Computer Systems*, vol. 70, pp. 163 – 177, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X16302060

[9] F. L. Pires and B. Barán, "A virtual machine placement taxonomy," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2015, pp. 159–168.

[10] M. C. S. Filho, C. C. Monteiro, P. R. Inácio, and M. M. Freire, "Approaches for optimizing virtual machine placement and migration in cloud environments: A survey," *Journal of Parallel and Distributed Computing*, vol. 111, pp. 222 – 250, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S074373151730240X

[11] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang, "Cost efficient resource management in fog computing supported medical cyber-physical system," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 1, pp. 108–119, Jan 2017.

[12] L. Zhao and J. Liu, "Optimal placement of virtual machines for supporting multiple applications in mobile edge networks," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 7, pp. 6533–6545, July 2018.

[13] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, thirdquarter 2017.

[14] X. Sun and N. Ansari, "Green cloudlet network: A sustainable platform for mobile cloud computing," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2018.

[15] K. Katsalis, T. G. Papaioannou, N. Nikaein, and L. Tassiulas, "Sla-driven vm scheduling in mobile edge computing," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, June 2016, pp. 750–757.

[16] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, "Optimized iot service placement in the fog," *Service Oriented Computing and Applications*, vol. 11, no. 4, pp. 427–443, Dec 2017. [Online]. Available: https://doi.org/10.1007/s11761-017-0219-8

[17] B. Spinnewyn, R. Mennes, J. F. Botero, and S. Latré, "Resilient application placement for geo-distributed cloud networks," *Journal of Network and Computer Applications*, vol. 85, pp. 14 – 31, 2017, intelligent Systems for Heterogeneous Networks. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804516303149

[18] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, no. 6, pp. 345–, Jun. 1962. [Online]. Available: http://doi.acm.org/10.1145/367766.368168

[19] S. Warshall, "A theorem on boolean matrices," *J. ACM*, vol. 9, no. 1, pp. 11–12, Jan. 1962. [Online]. Available: http://doi.acm.org/10.1145/321105.321107

[20] S. Burer and A. N. Letchford, "Non-convex mixed-integer nonlinear programming: A survey," *Surveys in Operations Research and Management Science*, vol. 17, no. 2, pp. 97 – 106, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1876735412000037

[21] G. P. McCormick, "Computability of global solutions to factorable nonconvex programs: Part i — convex underestimating problems," *Mathematical Programming*, vol. 10, no. 1, pp. 147–175, Dec 1976. [Online]. Available: https://doi.org/10.1007/BF01580665

[22] IBM ILOG CPLEX optimization studio. Accessed: Sept. 1, 2018. [Online]. Available: https://www.ibm.com/products/ilog-cplex-optimization-studio

[23] Y. Cui, Z. Geng, Q. Zhu, and Y. Han, "Review: Multi-objective optimization methods and application in energy saving," *Energy*, vol. 125, pp. 681 – 704, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360544217303584

[24] J. F. Gonçalves and M. G. C. Resende, "Biased random-key genetic algorithms for combinatorial optimization," *Journal of Heuristics*, vol. 17, no. 5, pp. 487–525, Oct 2011. [Online]. Available: https://doi.org/10.1007/s10732-010-9143-1

[25] R. Mennes, B. Spinnewyn, S. Latré, and J. F. Botero, "Greco: A distributed genetic algorithm for reliable application placement in hybrid clouds," in *2016 5th IEEE International Conference on Cloud Networking (Cloudnet)*, Oct 2016, pp. 14–20.

[26] W. M. Spears and K. D. De Jong, "On the virtues of parameterized uniform crossover," NAVAL RESEARCH LAB WASHINGTON DC, Tech. Rep., 1995.

[27] N. Alliance, "5g white paper," *Next generation mobile networks, white paper*, pp. 1–125, 2015.

[28] P. Schulz, M. Matthe, H. Klessig, M. Simsek, G. Fettweis, J. Ansari, S. A. Ashraf, B. Almeroth, J. Voigt, I. Riedel, A. Puschmann, A. Mitschele-Thiel, M. Muller, T. Elste, and M. Windisch, "Latency critical iot applications in 5g: Perspective on the design of radio interface and network architecture," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 70–78, February 2017.

[29] R. Jain, *The art of computer systems performance analysis*. John Wiley & Sons Chichester, 1991, vol. 182.