Latency and availability driven VNF placement in a MEC-NFV environment

Louiza Yala*, Pantelis A. Frangoudis‡, and Adlen Ksentini‡
*IRISA/University of Rennes 1, France
‡EURECOM, Sophia Antipolis, France
Email: *louiza.yala@irisa.fr, ‡name.surname@eurecom.fr

***Abstract*—Multi-access Edge Computing (MEC) is gaining momentum as it is considered as one of the enablers of 5G ultra-Reliable Low-Latency Communications (uRLLC) services. MEC deploys computation resources close to the end user, enabling to reduce drastically the end-to-end latency. ETSI has recently leveraged the MEC architecture to run all MEC entities, including MEC applications, as Virtual Network Functions (VNF) in a Network Functions Virtualization (NFV) environment. This evolution allows taking advantage of the mature architecture and the enabling tools of NFV, including the potential to apply a variety of service-tailored function placement algorithms. However, the latter need to be carefully designed in case of MEC applications such as uRLLC, where service access latency is critical. In this paper, we propose a novel placement scheme applicable to a MEC in NFV environment. In particular, we propose a formulation of the problem of VNF placement tailored to uRLLC as an optimization problem of two conflicting objectives, namely minimizing access latency and maximizing service availability. To deal with the complexity of the problem, we propose a Genetic Algorithm to solve it, which we compare with a CPLEX implementation of our model. Our numerical results show that our heuristic algorithm runs efficiently and produces solutions that approximate well the optimal, reducing latency and providing a highly-available service.**

I. Introduction

Ultra-Reliable Low-Latency Communications (uRLLC) services, such as connected car, industry 4.0 and critical IoT, are expected to be deployed using the upcoming 5G systems. uRLLC services require very low latency access between the remote server and the clients, while running in a highly reliable environment to guarantee service continuity. While 5G introduces novel mechanisms to ensure low latency, via MAC layer scheduling and network slicing [1], it requires running the servers belonging to uRLLC at the cloud edge to maintain a low end-to-end latency. Indeed, Edge Computing allows the deployment of applications close to end users to reduce access latency, as compared to the central cloud access. Two technologies emerged to enable the edge cloud, FoG and Mobile (or Multi-access) Edge Computing (MEC). The former is expected to run at the edge router, while the latter, supported by the European Telecommunications Standards Institute (ETSI) as well as network operators, is expected to run close to 4G/5G base stations (i.e., eNodeBs). Several ETSI documents have been issued [2], [3] to define a reference architecture and functional blocks to enable running MEC applications at the edge. These documents (i) define interfaces to interconnect MEC applications with the mobile network data plane to interact with the user data, (ii) specify interfaces to run MEC applications within a virtualization platform, and (iii) provide MEC services to leverage the mobile operator, by exposing low-level Radio Access Network (RAN) information to third-party application developers, allowing the development of context-aware services. MEC is more mature and ready to be deployed compared to FoG, where the first release of the architecture has been recently issued,[1] and mainly targets IoT services. Moreover, ETSI has recently started working on updating the MEC reference architecture to run MEC entities in a Network Functions Virtualization (NFV) environment [4], to take advantage of the latters mature deployment environment in terms of implementation and enabling tools, such as the Open Source Mano (OSM)[2] and OpenBaton[3] orchestration suites. The NFV orchestration process, known as the NFV Orchestrator (NFVO), is in charge of the automation of Network Service (NS) deployment over the virtualized infrastructure, which mainly consists in the instantiation, placement, and life-cycle management of VNFs composing a NS. By integrating MEC in NFV, ETSI considers MEC applications as classical VNFs, hence using the same orchestration and management process at a centralized NFVO. It is important to mention that MEC applications need to be hosted at the edge aiming at guaranteeing low latency access to the service. However, most of the used placement algorithms in NFV [5] aim to reduce deployment costs, increase the users Quality of Service (QoS), ensure service availability, etc., ignoring MEC applications constraints in terms of low latency. This may MEC application instances to be run at central clouds, as the NFVO and its related placement algorithms do not differentiate between MEC applications and other VNFs.

In this paper, we fill this gap by proposing a placement algorithm tailored to uRLLC services in the context of MEC in NFV. First, the proposed scheme classifies all cloud resources (central and edge cloud) according to their access latency

This work has been partially supported by the European Union's H2020 5G-Transformer Project (grant no. 761536)

[1]https://www.openfogconsortium.org/ra/
[2]https://osm.etsi.org
[3]https://openbaton.github.io/

to the user data plane, which avoids the need for additional information to differentiate between edge and central cloud resources, thus being in compliance with the MEC in NFV concept. Then, the placement algorithm is formulated as an optimization problem, where the objectives are to minimize latency and to maximize service availability (reliability); these correspond to the main deployment criteria for uRLLC services.

This paper is organized as follows. In Section II we present related work on VNF placement in different environments. We propose a formulation of the VNF placement problem for uRLLC in Section III. We provide solutions to this problem by proposing a suitable Genetic Algorithm (Section IV), but also by using the CPLEX solver as a benchmark. Our results are shown in Section V, before we conclude in Section VI.

II. Related work

The VNF placement problem in federated clouds has received significant attention. It is similar to the Virtual Machine (VM) placement problem, as VNFs are composed of virtual instances (VMs or containers) that execute network functions. A suitable VNF placement is mostly motivated by maximizing resource utilization or overall performance, while minimizing cost in terms of energy consumption, network traffic or penalties associated with SLA violations under various constraints [6], [7]. Although these metrics are well suited for NFV, putting MEC in the picture may require other metrics to be considered, such as latency. Indeed, one of the advantages of placing VNFs at the edge is the proximity to end-users, which leads to a low-latency service deployment. Therefore, placement algorithms should be updated to integrate latency as a criterion, in addition to the aforementioned metrics.

Only few works have addressed the problem of VNF placement in a mixed environment that includes edge and central clouds [8], [9]. Generally, VNFs are placed either in physical machines within central cloud infrastructures only [10], or in MEC servers [11], [12]. Other works [13], [14] investigate the placement in the context of a hybrid federated cloud, using a combination of different cloud infrastructures, without however involving the edge. In this context, the authors in [9] also propose a VNF placement solution that captures the trade-off between cost efficiency and Quality of Experience (QoE) in a multi-cloud environment.

Similarly to our work, some recent approaches [8], [15] outstrip the common cost and resource optimization by introducing more specific VNF placement and provisioning optimization strategies over edge and central cloud infrastructures. Jemaa et al. [8] take into account QoS requirements such as minimum bandwidth and maximum end-to-end latency. In particular, their objective is to minimize the maximum utilization of the edge cloud by deploying more VNFs in the traditional cloud.

In our prior work [5], we studied the problem of jointly allocating compute resources to virtual instances and their placement on a traditional cloud infrastructure for the provision of CDN-as-a-Service. Our focus was on simultaneously addressing the conflicting requirements for improving service availability and reducing management cost. This paper is similar in spirit but is put in a different context. In particular, as in our prior work, we provide a multi-objective optimization problem formulation and apply the service availability model we introduced in [5]. However, from a system model perspective, this paper has the following distinct differences: (i) physical machines are not considered identical regarding their reliability and the cost of deploying VMs on top of them, thus differentiating between edge and central cloud hosts, (ii) latency minimization is one of the objectives, (iii) the CPU resource requirements per VNF instance to deploy are part of the problem input.

III. Problem formulation

We assume that an operator manages a set of data centers (DCs), which can either be installed at the network edge, and therefore close to end-users, or in the traditional central cloud. The uRLLC service is composed by a number of VNFs, which are in turn composed of a set of VMs (a VNF could be composed of one or more VMs to guarantee scalability). The operator aims to derive a suitable placement of those virtual instances (VMs) in DCs, where each DC has a set of physical machines (PMs) where VMs can be launched according to the demand. The objective is to find an appropriate placement of the n VM composing a service instance on a subset of the m available PMs.[4] This is translated in our system as a binary assignment matrix $X = (x_{ij})$ with $i \in [1, n]$ and $j \in [1, m]$, where x_{ij} is equal to 1 if VM_i is hosted at PM_j, and 0 otherwise. The number of VMs to instantiate, as well as the CPU requirements of each VM, are part of the problem input. The calculation of the optimal assignment should minimize the average access latency of the service deployment, consider the CPU capacities of the underlying physical hosts, not exceed a budget, and respect a service availability constraint. We recall that we target a uRLLC service, which requires both low-latency access and reliability.

A. Latency model

We consider two types of hosts (PMs): ones which are located at centralized clouds/NFV infrastructures, and ones at MEC DCs. Hosts are grouped by DC and all hosts belonging to the same DC are assumed to share the same access latency value. Note that by characterizing DCs/hosts by latency, we do not need to explicitly differentiate between MEC and central cloud PMs; for a delay-minimizing algorithm, only the latency property is relevant.

We define access latency as the latency for users to access the uRLLC service. $D = [d_1\ d_2\ \dots\ d_m]^T$ is a vector where d_j is the latency characterizing the $j-$th PM and m is the total number of available PMs across all DCs. A specific assignment X is characterized by a n-dimension latency vector $D' = [d'_1\ d'_2\ \dots\ d'_n]^T$, whose value depends on the PMs that are

[4] In the rest of the paper, unless otherwise noted, the terms *host*, *PM*, and (physical) *server* are used interchangeably.

utilized to host the n VMs composing the service, and which we formulate as

$$D' = X \times D.$$

The element $d'_i = \sum_{j=1}^{m} x_{ij} d_j$ corresponds to the latency of the PM which is hosting the $i-$th VM (since a VM is placed at exactly one PM, only a single term in the above sum will be non-zero).

We define the latency of an assignment X as the average of the elements of D', i.e., the average latency over all deployed VMs:

$$L(X) = \frac{\sum_{i=1}^{n} d'_i}{n} \tag{1}$$

B. Availability model

In our model, availability is the ability of the system to offer at least a minimal functional and accessible service. In particular, a service instance is considered available if at least one of its constituent VMs remains accessible, which implies that the PM hosting it is accessible as well.

We further make the following assumptions:

- We consider two types of PMs, central and edge cloud ones.
- The probability of failure of a VM i is $q_i^{(V)}$, independently of the other VMs and PMs, and irrespectively of the load imposed on the VM.
- The probability of failure of a PM j is $q_j^{(P)}$, independently of the other PMs or the load imposed on it. It is reasonable to assume that at the edge, hosts have a greater failure probability and are thus less reliable. Indeed, VM replication or migration inside the edge is harder to ensure due to the scarcity of resources compared to the central cloud.
- If a PM fails, all VMs deployed on it fail because of that.
- A VM may become inaccessible either because it fails or because the PM that hosts it fails.
- The system operator knows the value of the PM and VM failure probabilities, as a result of measurement studies, prior experience, or other historical information.

Since VM failures can be correlated due to their dependence on the underlying PMs, we define a *correlated group* of VMs as the set of VMs that are executed on the same PM. A correlated group is available under the following conditions:

- The PM is up.
- At least one of the VMs deployed on the PM does not fail.

We define the probability that a correlated group deployed on PM j is available as follows:

$$a_j = (1 - q_j^{(P)})(1 - \prod_{i \in [1,n] | x_{ij}=1} q_i^{(V)}) \tag{2}$$

At least one correlated group should be available in order to have a service/application deployment available. Since correlated groups fail independently, the probability that a service deployment is available is given by

$$\begin{aligned} A(X) &= 1 - Pr\{\text{All correlated groups fail}\} \\ &= 1 - \prod_{j \in [1,m] | \Sigma_{i=1}^{n} x_{ij} > 0} (1 - a_j) \\ &= 1 - \prod_{j \in [1,m] | \Sigma_{i=1}^{n} x_{ij} > 0} \left[q_j^{(P)} + (1 - q_j^{(P)}) \prod_{i \in [1,n] | x_{ij}=1} q_i^{(V)} \right] \end{aligned} \tag{3}$$

Note that, by construction, any feasible solution includes at least one PM with at least one VM assigned to it; therefore, both product terms in (3) are over non-empty sets.

C. Cost model

In our prior work [5], we considered that the deployment of a VM comes with a fixed management overhead. In this work, however, not all VMs come with the same cost. In particular, we treat cost as a function of the associated VM workload (in terms of the number of vCPUs allocated to the VM and under the assumption that more powerful VMs handle more application workload). This cost may also account for the energy consumed for booting the VM or for operating other system- or service-level components. Therefore, we define a VM capacity requirement vector $P = [p_1\ p_2\ \dots\ p_n]$, which represents the CPU capacity needed for the allocation of each VM; p_i is the service/application requirement in terms of the number of virtual CPUs for VM_i.

We also assume a PM-level cost, which is a fixed overhead and is not a function of the PM workload nor the number of VMs hosted by it (e.g., energy cost for keeping the physical machine in an operating state).

Therefore, the cost of an assignment X at the VM level is given by

$$C_V(X) = e_V \sum_{i=1}^{n} p_i, \tag{4}$$

where e_V is the fixed cost per vCPU assigned to a VM.

The cost at the PM level is determined by the number and the type of the PMs involved. Note that we consider it more costly to operate on edge hosts. This is due to the value and scarcity of edge resources. We assume that when involving a PM in an assignment, this comes with a fixed PM-level cost. We encode these fixed costs in vector $K = [k_1\ k_2\ \dots\ k_m]$, each element of which corresponds to the cost associated with a specific PM. The PM-level cost of an assignment is defined as the sum of the fixed costs of all the PMs that are used, i.e., the PMs that host at least one of the service's VMs:

$$C_P(X) = \sum_{j=1}^{m} k_j \mathbb{1}(\sum_{i=1}^{n} x_{ij} > 0). \tag{5}$$

The overall cost of a placement is then given by

$$C(X) = C_V(X) + C_P(X). \tag{6}$$

D. Objective

The objective of the system is to derive an optimal assignment that minimizes latency while maximizing availability. Each criterion drives VM placement towards a different type of solutions. Focusing on a low latency service deployment will lead the system to place the service components at edge servers, which provide shorter response times but are more expensive. On the contrary, traditional central DCs offer higher availability but with longer delays. Since it is not possible to optimize for both criteria at the same time, we apply a scalarization approach to transform the problem to a single-objective one. The relative importance of the two criteria in deriving an optimal placement is dictated by a specific *policy*, which is encoded in a pair of weights, w_l for latency and w_a for availability, such that $w_l + w_a = 1$. Given a specific policy that depends on the particular uRLLC service, the system operator derives the optimal solution to the following problem:

$$\underset{X}{\text{minimize}} \quad w_l L(X) - w_a A(X) \tag{7}$$

$$\text{subject to} \quad C(X) < E \tag{8}$$

$$A(X) > A \tag{9}$$

$$\sum_{j=1}^{m} \mathbb{1}(x_{ij} > 0) \leq 1, \forall i \in [1, n] \tag{10}$$

$$\sum_{i=1}^{n} p_i \times x_{ij} \leq u_j, \forall j \in [1, m], \tag{11}$$

with $U = \{u_1, u_2 ... u_m\}$ representing the available CPU capacity for each of the m PMs.

In the proposed formulation we consider a budget constraint (8) which limits the cost of the service deployment to below E and ensures that an initial budget agreement with the customer is not violated. Constraint (9) ensures a service with a minimum availability. Constraint (10) ensures that each VM is assigned to a single host, since a VM cannot be split, and constraint (11) guarantees that the capacity of each host is not exceeded.

IV. A GENETIC ALGORITHM FOR VM PLACEMENT

Many VM or network function placement problems are known to be NP-hard [16] and various heuristics are being proposed for this reason. Therefore, we present in this section a Genetic Algorithm (GA) *meta-heuristic* tailored to our model, that we selected to tackle our VM placement problem. A GA solves an optimization problem applying iteratively two main operations: *crossover* and *mutation*. It encodes each potential solution as a chromosome (a sequence of characters) of properties called genes and characterizes each solution with a fitness value, which is an expression of the solution's quality. The GA starts with an initial pool of candidate solutions and iteratively tunes them to produce new generations of better quality by using *crossover* to generate new chromosomes from selected parents and *mutation* to randomly adjust a chromosome.

Our genetic algorithm is inspired by the work of Xu and Fortes [17], who proposed a Grouping Genetic Algorithm (GGA) to solve the problem of VM placement on physical resources, aiming to optimize, as in our case, a multi-objective function. Their objectives are minimizing resource wastage, power consumption and maximum thermal dissipation. These three objectives are conflicting because placing VMs on a small number of PMs increases thermal dissipation in the used PMs, while the two other objectives decrease.

In the same spirit, our GGA integrates our availability and latency objectives. Unlike Xu and Fortes, though, who use a fuzzy logic system as a fitness function to jointly consider the three objectives, our fitness function is dictated by the weights (policy) and is a direct application of our objective function equation (7).

Our GGA takes as input the number of VMs and creates an initial solution by generating S random assignments of VMs to PMs. For each solution in S, each group of VMs on a PM is a gene. Once the initial solution pool is generated, the GGA runs for G generations.

Xu and Fortes [17] introduced a *ranking-crossover* operator, which is an improved version of the classical crossover. The latter choses genes blindly from selected parents. On the other hand, *ranking-crossover* selects the genes which have a higher rank according to the average value of specific *efficiency* functions defined per objective, hence expected to produce higher quality offspring.

In our case, for each *ranking-crossover* operation, our GGA calculates the efficiency value for each involved gene from two random chromosomes. Contrary to their work, we chose as an efficiency function the distance between a gene's derived values for latency and availability given by (1) and (3) respectively, and ideal ones. These ideal values are probably unattainable at the same time by a single gene and correspond to a *utopian* case. For example, we have set the ideal values to $u_l \approx 0$ (latency) and $u_a = 100\%$ (availability). Our efficiency function is defined as the *Euclidean distance* between the utopian point (u_l, u_a) and the point representing the gene's latency and availability values (l, a), formulated as:

$$\mathcal{E} = \sqrt{(u_l - l)^2 + (u_a - a)^2}.$$

After deriving $\mathcal{E}$ for each gene, the GGA ranks them from the smallest distance, which corresponds to the highest rank, to the largest one, and creates a new chromosome by combining the highest-ranking genes.

Note that for each generation G, this procedure is repeated in order for a number of new chromosomes to be generated, which is specified by the crossover rate r_c parameter. Therefore, the GGA produces $r_c S$ offspring on average on each generation. The solution pool population at the end of a generation is constructed by evaluating the fitness function for each chromosome and keeping the top-S of them. After G generations, our algorithm terminates by returning solution with the highest fitness function value. During the whole process our GGA eliminates solutions that violate any of the constraints.

The overall complexity of our GGA is $\mathcal{O}(Sn + GSmlogm)$. The first term refers to the generation of S initial solutions,

each corresponding to a random placement of n VMs over the m PMs. Then, for each of the G generations, $\mathcal{O}(m)$ time is required for the calculation of the efficiency function for m parents, and $\mathcal{O}(mlogm)$ time is needed for sorting these m efficiency values. The latter operation is repeated r_cS times, thus yielding a $\mathcal{O}(Smlogm)$ complexity per generation.

V. Numerical results

For our experiments, we consider different configurations, where we use different types of hosts, central cloud and MEC ones. We show the positive impact of using edge hosts in deployment delays and how we capture the trade-off between latency and availability. We also compare our scheme with a CPLEX[5] implementation in terms of the quality of the solution and execution time.

We assume a uRLLC service composed of 90 VMs that need to be deployed over a federated cloud (including edge DCs). We consider a system with 150 PMs, 100 PMs within the traditional cloud each with a capacity between 5 and 15 vCPUs, and 50 MEC servers with a capacity between 1 and 6 vCPUs each. Host capacities are selected uniformly at random. We fix the budget to $E = 320$ and assume that all PMs of a specific type incur the same fixed cost. For each edge server, the PM-level cost is $k_e = 2$, while for each central cloud host it is $k_c = 1$. The cost of one vCPU assigned to a VM is $e_v = 1$. All the VMs have the same failure probability $q_V = 0.001$. The PM failure probabilities are set to $q_{P1} = 0.004$ for MEC servers and $q_{P2} = 0.002$ for central cloud servers, since we assume that the reliabilities are different for the two types of hosts. We select uniformly at random the delays, which we set to between 1 and 5 ms for the 50 edge servers, and between 4 and 10 ms for the 100 central cloud servers.

We run both algorithms (CPLEX and GGA) for: (i) the same number of vCPUs assigned to each VMs, (ii) the same number of VMs (unless otherwise specified), and (iii) the same PM delays (D). Regarding the GGA we configure the solution pool size to $S = 80$ and the number of generations to $G = 30$. We chose those values experimentally, since we found them to represent a good compromise between execution time and quality of the solutions.

Fig. 1 outlines the availability and latency function values for our algorithm for different policies. The values are normalized by mapping the lowest and highest value per objective to 0 and 1 respectively. As it is expected, the results indicate that the more the policy is latency-oriented ($w_l > 0.6$), the more the algorithm sacrifices on availability. In particular, since our GGA is guided by the applied policy, the objective function value of equation (7) decreases because the chosen servers for the deployment have small delays (small values of d_j); more edge servers are used to deploy the service components than central ones.

This is more clear in Fig. 2 that represents the usage rate of MEC servers as a function of the applied policy. It can be seen that when w_l increases, more hosts within the edge are used, because the delays on the edges are smaller and our algorithm honors hosts which offer minimal latencies.

Figure 3 presents the evolution of cost during the placement, where the y-axis represents the values of the cost as a function of the given policy depicted in the x-axis. The straight line represents the maximum budget set to $E = 320$ (which can be fixed with the customer). As can be seen, the cost increases substantially with w_l. We argue this by the fact that the GGA uses more MEC hosts for the VM placement, which are by definition more expensive.

In our second experiment we run our GGA by fixing the policy to $(w_a, w_l) = (0.5, 0.5)$ (uRLLC service that requires both latency and availability), using the same number of PMs and their capacities, and for the same number of vCPUs (200). We vary the number of VMs in an interval of $n \in (0 - 60]$ for 100 iterations and present results with 95% confidence intervals. The results of the latter experiment are shown in Fig. 4 as box-plots illustrating the evolution of the average latency according to the number of VMs. As depicted in Fig. 4, when n increases, the average latency decreases. In fact, adding more VMs leads to their placement in smaller DCs, which in our case corresponds to edge PMs; the more VMs we have, the less vCPUs are allocated to each one of them since the number of vCPUs remains the same for all our tests.

In Fig. 5, we highlight the evolution of availability as a function of the number of VMs, for the solutions obtained by our GGA. The straight line represents the availability constraint, set to class five (99,999%) of reliability requirement for uRLLC services.

From the resulting plot we can see that the availability slightly decreases with the number of VMs, which is compatible with the growing use of MEC servers previously pointed out. Considering the small capacities within the edge servers, the VMs with the small number of vCPUs assigned to them are placed on those servers. However, even if the values of availability decrease, the availability constraint is not violated.

Regarding the cost, Fig. 6 illustrates its evolution as a function of the number of VMs for the same case. We clearly see that the cost increases with the number of VMs, which positively supports our results from the two previous figures. The cost is affected by the use of PMs from the edge, which are more expensive.

In our third experiment, we implement our model in the CPLEX environment and use its optimizer to derive exact optimal solutions. We compare and evaluate the CPLEX results with our GGA.

Fig. 7 presents a comparison of the objective function values obtained for the same configuration for our GGA (green curve, "x" points) and CPLEX (purple, cross points) as a function of the applied policy. Since we are solving a minimization problem, lower values for a specific weight combination mean better performance. Our meta-heuristic is shown to approximate well the optimal solutions provided by CPLEX.

Fig. 8 presents the execution time results for both CPLEX

[5]http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

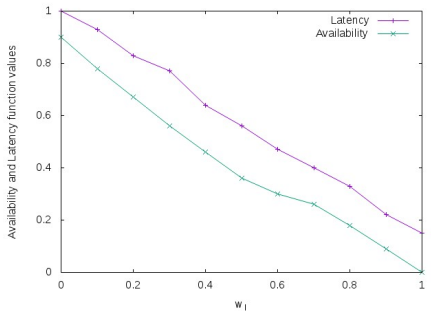

Fig. 1. Availability and Latency as a function of the selected policy.

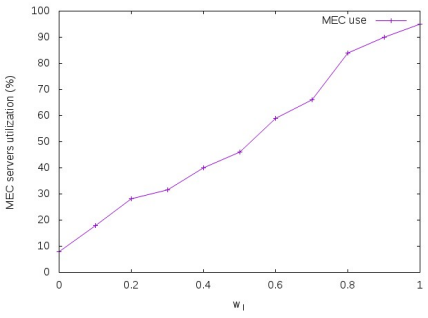

Fig. 2. MEC servers utilization as a function of the policy.

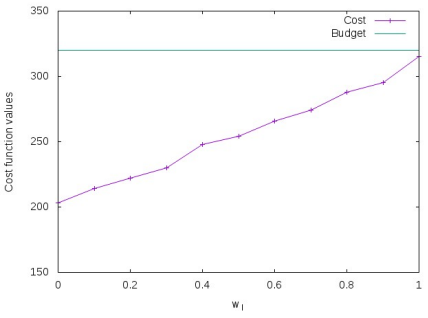

Fig. 3. Evolution of cost with latency-oriented policy. The straight line represents cost constraint (budget).

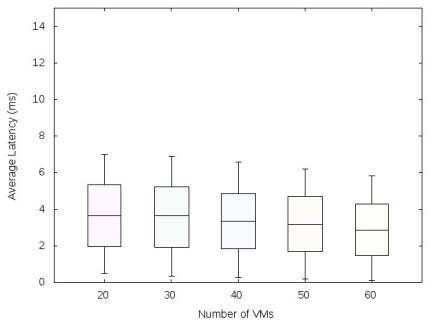

Fig. 4. Box-plot showing VMs impact on average Latency of a service deployment.

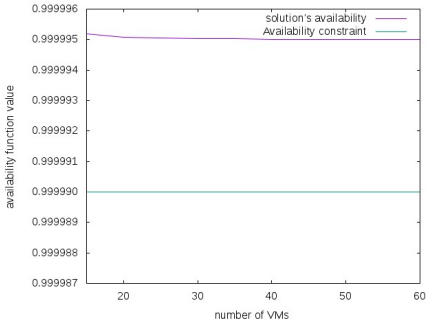

Fig. 5. Availability as a function of the number of VMs. The straight line represents availability constraint.

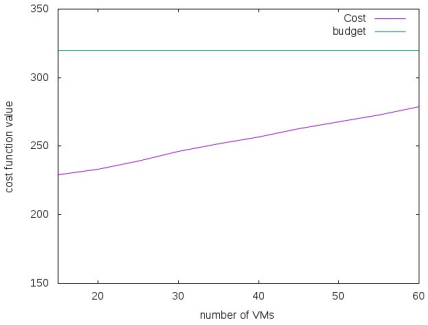

Fig. 6. Cost objective value as a function numbers of VMs. The straight line represents cost constraint.

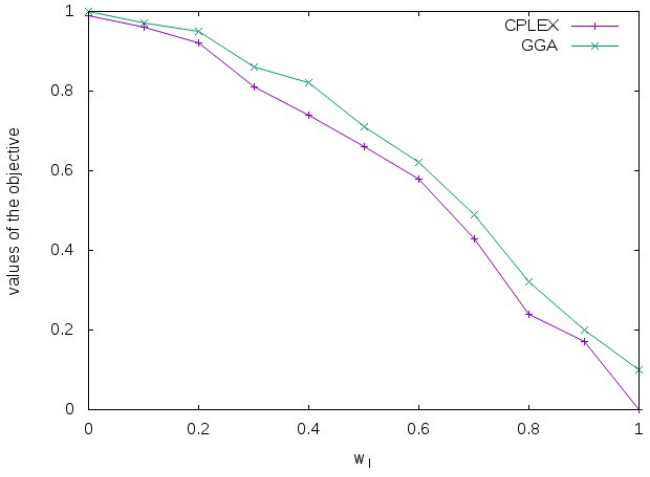

Fig. 7. Comparison between our GGA and CPLEX in terms of the objective for a specific policy.

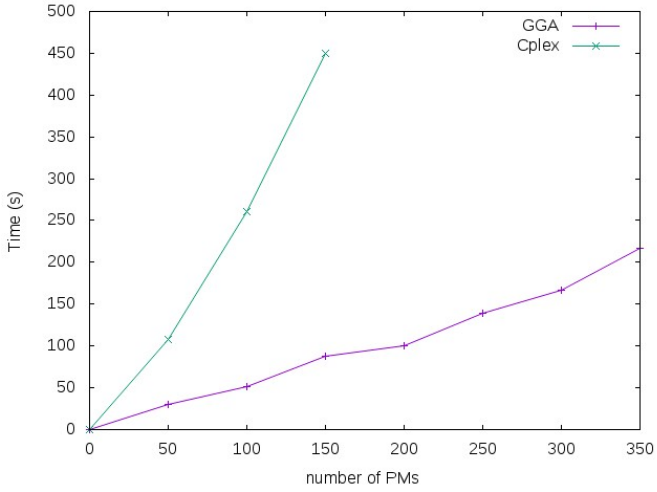

Fig. 8. Execution time as a function of the number of PMs for our GGA and CPLEX.

and GGA for increasing numbers of available servers. Our experiments were performed on an Intel i7 machine with 8 CPU cores and 8 GB of RAM, running Ubuntu 14.04. It can be seen that our GGA scales well as the number of PMs grows. Even for more than 200, it produces a solution close to the optimal in an order of a couple of minutes. On the other hand the execution time of CPLEX increases dramatically, and it stops producing solutions for problem instances with more than approximatively 150 PMs. Note that we carried out those experiments by fixing the policy to $w_l = w_a = 0.5$.

VI. Conclusion

In this paper, we presented our study on the problem of placing VNF instances on a mixed environment which considers two different types of hosts, namely PMs from the edge and the central cloud. We proposed a multi-objective optimization formulation of the problem that captures the trade-off between service access latency and availability, which correspond to the criteria of uRLLC services. To solve this placement problem, we provided an appropriate adaptation of the Genetic Algorithm meta-heuristic. Our experimental results show that our solutions are close to the optimal, while simultaneously it takes considerably less time to derive them than an exact algorithm provided by CPLEX. We have also shown how our method can reduce delays and still provide a highly-available service.

Our approach tackles important aspects regarding uRLLC services, but more needs to be done, and this is the subject of our ongoing work. First, we will consider different types of components, such as caches, which may influence the type of resources needed and thus the number of vCPUs as well as the number of VMs. Second our mechanism will take into account failure recovery in case of PM breakdowns, especially at the edge and for critical services, such as the connected car. Finally, we are exploring uses of our model with respect to the establishment of SLAs with the customer, in order to create a clear common understanding about different services.

References

[1] A. Ksentini, P. A. Frangoudis, P. Amogh, and N. Nikaein, "Providing low latency guarantees for slicing-ready 5g systems via two-level mac scheduling," *IEEE Network*, 2018, in press.

[2] *Mobile Edge Computing (MEC); Framework and Reference Architecture*, ETSI Group Specification MEC 003, 2016.

[3] *Mobile Edge Computing (MEC); Mobile Edge Management; Part 2: Application lifecycle, rules and requirements management*, ETSI Group Specification MEC 010, 2017.

[4] *Mobile Edge Computing (MEC); Deployment of Mobile Edge Computing in an NVF environment*, ETSI Group Report MEC 017, 2018.

[5] L. Yala, P. Frangoudis, G. Lucarelli, and A. Ksentini, "Balancing between cost and availability for cdnaas resource placement," in *IEEE Global Communications Conference (GLOBECOM 2017)*, 2017.

[6] J. Dong, X. Jin, H. Wang, Y. Li, P. Zhang, and S. Cheng, "Energy-saving virtual machine placement in cloud data centers," in *Proc. IEEE/ACM CCGrid*, 2013.

[7] N. Quang-Hung, N. T. Son, and N. Thoai, "Energy-saving virtual machine scheduling in cloud computing with fixed interval constraints," in *Transactions on Large-Scale Data-and Knowledge-Centered Systems XXXI*. Springer, 2017, pp. 124–145.

[8] F. B. Jemaa, G. Pujolle, and M. Pariente, "Qos-aware vnf placement optimization in edge-central carrier cloud architecture," in *Global Communications Conference (GLOBECOM), 2016 IEEE*. IEEE, 2016, pp. 1–7.

[9] I. Benkacem, T. Taleb, M. Bagaa, and H. Flinck, "Optimal vnfs placement in cdn slicing over multi-cloud environment," *IEEE Journal on Selected Areas in Communications*, 2018.

[10] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 725–739, 2016.

[11] 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 (IFIP Networking), 2015*. IEEE, 2015, pp. 1–9.

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

[13] Z. Wen, J. Cała, P. Watson, and A. Romanovsky, "Cost effective, reliable and secure workflow deployment over federated clouds," *IEEE Transactions on Services Computing*, vol. 10, no. 6, pp. 929–941, 2017.

[14] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. IEEE, 2010, pp. 228–235.

[15] A. Alleg, T. Ahmed, M. Mosbah, R. Riggio, and R. Boutaba, "Delay-aware vnf placement and chaining based on a flexible resource allocation approach," in *2017 13th International Conference on Network and Service Management (CNSM)*. IEEE, 2017, pp. 1–7.

[16] Z. A. Mann, "Allocation of virtual machines in cloud data centers–a survey of problem models and optimization algorithms," *ACM Comput. Surv.*, vol. 48, no. 1, p. 11, 2015.

[17] J. Xu and J. A. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*. IEEE Computer Society, 2010, pp. 179–188.