Latency and network aware placement for cloud-native 5G/6G services

Kiranpreet Kaur, Fabrice Guillemin, Veronica Quintuna Rodriguez, Francoise Sailhan

Latency and network aware placement for cloud-native 5G/6G services

Kiranpreet Kaur[1,2], Fabrice Guillemin[1], Veronica Quintuna Rodriguez[1], Francoise Sailhan[2]
[1] Orange Labs, 2 Avenue Pierre Marzin, Lannion, France.
[1]{kiranpreet.kaur, fabrice.guillemin,veronica.quintunarodriguez}@orange.com
[2] Cedric Laboratory, CNAM Paris, 292 rue St Martin, Paris, France.
[2]{firstName.lastName}@cnam.fr

***Abstract*—To meet ever more stringent requirements in terms of latency, 5G/6G networks are evolving from centralized to distributed architectures, for which the cloud-native paradigm with services decomposed into microservices is utmost relevant. This in turn raises the issue related to the distribution of network functions. In this paper, we introduce a novel microservice placement strategy considering the internal service composition, notably the communication between microservices. We formulate the placement as an optimization problem with the aim of minimizing end-to-end service latency. We solve the optimization problem with a combination of greedy and genetic algorithms.**

***Keywords*-Microservice placement; Virtualized network function**

I. Introduction

The advent of Network Function Virtualization (NFV) [1] and the recent adoption of cloud-native principles have deeply modified network operations by dissociating the hosting hardware from network functions and by decomposing network functions into a large number of smaller and ready-to-run microservices. Services are then implemented as bundles of microservices interconnected to each other and distributed on container-based infrastructures (e.g., Kubernetes).

In parallel, the ever growing performance requirements in terms of latency and throughput of new services (involved e.g. in virtual reality) pave the way towards architectures, where the network edge plays a primary role. As a matter of fact, latency sensitive network functions need to be deployed as close as possible to end users to meet real time requirements. Closeness of microservices is however not always feasible due to capacity constraints of the hosting data centers, especially when considering the relative resource scarcity of those data centers located at network edge. It is therefore necessary to design an efficient placement strategy of microservices while considering both resources availability and service demand.

The placement of Virtualized Network Functions (VNFs) has been widely studied in the literature. Most research works focused on load balancing, examine the resource (CPU, RAM, disk) needs of network functions with respect to the resource availability in data centers [2], [3]. Only a few works on microservice placement address latency requirements by considering either the processing delay or the link capacity. To address the deployment of network functions across distributed data-centers, microservice placement should be reexamined from different viewpoints, notably by considering the microservice inter-dependency and the amount of traffic among microservices, which increase the service latency due to the delay associated with messages transiting through the transport network connecting data-centers.

In order to minimize the latency of new 5G/6G services, we propose two approaches that consider the communication affinity, i.e., the number of internal communication messages within a distributed service. The first one minimizes the delay associated with messages transiting between data centers. This first approach is network aware as it explicitly depends on latency along transmission links. The second approach is network agnostic and introduces a metric that accounts for (i) communication affinity among microservices and (ii) the location of the user. This metric relying on a weighting strategy, privileges a placement near to the user and the collocation of microservices that involve heavy message exchanges.

To the best of our knowledge, the present work is one of first works considering these aspects. We formalise the problem of placing microservices via Integer Linear Programming (ILP), and we propose an approximate problem-solving solution combining two heuristics (a greedy and a genetic algorithm), which considerably reduces run time of the placement algorithm. In particular, the fast greedy algorithm provides an initial allocation of services, which is subsequently improved by an advanced genetic algorithm; this latter one rearranges through mutation and crossover the microservices to reduce the global latency, while preserving the placement of microservices achieving a high latency gain.

The paper is organized as follows. Based on a realistic model of a ISP infrastructure (§ II), we formalize the problem of allocating microservices (§ III) and we introduce an approximate problem-solving solution (§IV). We then evaluate the algorithms (§ V) and conclude with a related work (§ VI) and a summary of our contributions.

II. Model description

The model under consideration (Fig. 1) is made of two key elements: the Substrate Network (III-A) and the Services (II-B). The substrate network reasonably represents the network of an European Operator [4], which is represented as a

tree composed of a central cloud datacenter, fog nodes located at regional Points of Presence and edge nodes near to users.

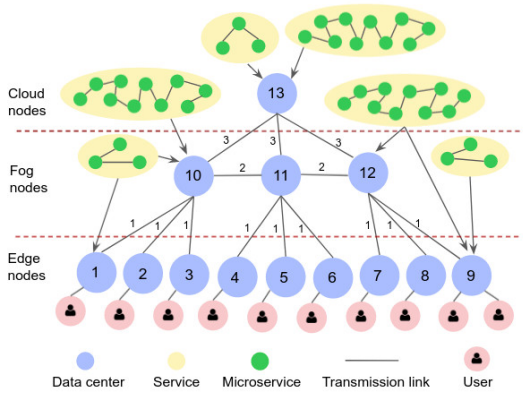


Fig. 1: Mapping a set of services onto a substrate network

A. Substrate network

The network of data centers is represented by means of a graph $G = (V, E, W)$, where the set V of vertices is composed of data centers, the set of edges E corresponds to the bidirectional links interconnecting the data centers, and the set of weights W reflects the links characteristics. While an edge weight may represent diverse factors (e.g. bandwidth, the length, the transmission delay along the link), we herein focus on the latency in the execution of a service and hence we assume that the weight $w(e(v_i, v_j))$ equals to the transmission delay $\delta(v_i, v_j)$ between the two data centers v_i and v_j.

A vertex v offers the IT resources to execute the hosted services and is characterized by a capacity $C(v)$ that may reflect various types of resources (e.g., CPU, RAM, disk). As the most limited resource is either CPU or RAM, the allocation will be based on a single resource type, even if the algorithms presented in § III could handle various types of resources.

B. Services

The substrate network hosts services that are decomposed into microservices, each of them having specific requirements in terms of IT resources. A service Σ_j corresponds to a graph composed of K_j vertices, denoted $\sigma_1^j, \ldots, \sigma_{K_j}^j$, representing the microservices, and edges representing the communication between microservices. A microservice σ_i requires an amount of IT resources equal to $c(\sigma_i)$ and may send messages to another microservice σ_j. The number of messages exchanged between the two microservices, on both way, is denoted by $\nu(\sigma_i, \sigma_j)$, with $\nu(\sigma_i, \sigma_j) = \nu(\sigma_j, \sigma_i)$.

Each service is associated with a user lying at the edge of the operator network (see the leaves in Fig. 1) and exchanging traffic with the microservices. It is important to consider the latency associated with the user traffic as the access network usually contributes a significant amount to the global latency budget. For that purpose, we introduce a dummy service σ_0 located, where the user stands and which has no resource requirements, i.e., $c(\sigma_0^j) = 0$. The service Σ_j is thus composed of microservices $\sigma_0^j, \ldots, \sigma_{K_j}^j$.

III. Problem formulation

We consider a substrate network G composed of N nodes (data centers), in which each node v_n (with $n = 1, \ldots, N$) has a capacity $C(v_n)$. The network G accommodates J services and each service Σ_j is decomposed into $K_j + 1$ microservices (including the end user). The J services can be allocated to the substrate network if and only if the total resource demand is not greater than total resource capacity of the substrate network, that is,

$$\sum_{j=1}^{J} \sum_{i=1}^{K_j} c(\sigma_i^j) \leq \sum_{n=1}^{N} C(v_n).$$

If this condition is violated, then some services are rejected. In the present paper, we do not address this issue and focus on how perform placement so as to globally reduce latency. We precisely consider a static configuration to better understand issues related to latency when placing microservices. In practice, we should consider the case when services randomly arrive and leave the system. We would then obtain a stochastic knapsack, which will be addressed in a further study. In order to optimize the microservices placement, we propose to either:

- minimize the overall latency accounting for the transmission delays, thereby considering the structure-of – and delay induced by - the network substrate (§ III-A),
- maximize the amount of microservices that are collocated and placed near to the user, ignoring the substrate network (network-agnostic approach § III-B)

A. Network-aware approach

The latency $\mathcal{L}_j$ of a service Σ_j is

$$\mathcal{L}_j = \sum_{i=0}^{K_j} \sum_{i'=i}^{K_j} \nu(\sigma_i^j, \sigma_{i'}^j) L(\sigma_i^j, \sigma_{i'}^j), \tag{1}$$

where the latency between microservices σ_i^j and $\sigma_{i'}^j$ deployed across distinct data centers verifies:

$$L(\sigma_i^j, \sigma_{i'}^j) = \sum_{n=1}^{N} \sum_{m=n}^{N} \delta(v_n, v_m) \mathbb{1}(v_n, \sigma_i^j) \mathbb{1}(v_m, \sigma_{i'}^j). \tag{2}$$

The objective of placement is then to minimize the global latency, which leads to the following optimization problem:

$$\min_{\substack{\mathbb{1}(v_n, \sigma_i^j) \\ n \in [\![1, N]\!],\ j \in [\![1, J]\!],\ i \in [\![1, K_j]\!]}} \mathcal{L} \tag{3}$$

$$i \in [\![1, K_j]\!],\ j \in [\![1, J]\!],\ \sum_{n=1}^{N} \mathbb{1}(v_n, \sigma_i^j) \leq 1, \tag{4}$$

$$n \in [\![1, N]\!],\ \sum_{j=1}^{J} \sum_{i=0}^{K_j} c(\sigma_i^j) \mathbb{1}(v_n, \sigma_i^j) \leq C(v_n), \tag{5}$$

where

$$\mathcal{L} = \sum_{j=1}^{J} \mathcal{L}_j \tag{6}$$

As defined in Equation (4), a microservice should be allocated at most once. If a microservice σ_i^j cannot be placed (the sum in equation (4) equals to 0), then the entire service σ_j is removed.

B. Network agnostic approach

If several microservices that exchange some messages are collocated (resp. hosted on distant datacenters), then the latency of the service tends to decrease (resp. increase). We intuitively introduce a metric quantifying the amount of messages exchanged by the microservices collocated on the same data center since those ones do not increase the global latency of the service. For service Σ_j, the metric is defined by

$$\tilde{L}_j = \sum_{n=1}^{N} \sum_{i=0}^{K_j} \sum_{i'=i}^{K_j} \tilde{\nu}(\sigma_i^j, \sigma_{i'}^j) \mathbb{1}(v_n, \sigma_i^j) \mathbb{1}(v_n, \sigma_{i'}^j).$$

where $\tilde{\nu}(\sigma_i^j, \sigma_{i'}^j) = \nu(\sigma_i^j, \sigma_{i'}^j)$ for $1 \leq i \leq i'$ (i.e., the actual number of messages exchanged between microservices). The product $\mathbb{1}(v_n, \sigma_i^j)\mathbb{1}(v_n, \sigma_{i'}^j)$ is equal to 1 only if microservices σ_i^j and $\sigma_{i'}^j$ are on the same data center. Moreover, to force the placement of microservices close to users, we give more weight to the communications between users and microservices. Thus, $\tilde{\nu}(\sigma_0^j, \sigma_i^j) = \kappa_i(\sigma_0^j, \sigma_i^j)$, where κ_i is a parameter giving more weight to the messages exchanged between the user and any microservice i (with $i > 0$). Overall, the metric $\tilde{L}$ quantifying the "network agnostic latency" gives more weight to collocated microservices:

$$\tilde{L} = \sum_{j=1}^{J} \alpha^{\tilde{L}_j} \tag{7}$$

for some $\alpha > 1$. There is thus an exponential discrimination between services with many collocated microservices and those with more distributed microservices. With this latency metric, the optimization problem then reads:

$$\begin{array}{c} \max \quad \tilde{L} \\ \mathbb{1}(v_n, \sigma_i^j) \\ n \in [\![1, N]\!],\ j \in [\![1, J]\!],\ i \in [\![1, K_j]\!] \end{array} \tag{8}$$

subject to the constraints (4), (5).

C. Scalability of the Latency-aware placement

The optimization problem previously introduced could be solved by using a classical ILP solver (e.g. CPLEX[1]) or by performing an exhausting search (i.e., enumerating all the solutions and selecting the optimal one). Nonetheless, these options are not practically viable for large problem instances. Computing the optimal placement of microservices that minimizes latency, is actually a variation of the bin-packing problem, which is known as to be combinatorial NP-Hard problem in which items (i.e., microservices) of varying sizes (e.g., resource capacity) should be packed into a finite number of bins (i.e., computing nodes) with finite capacities so that the number of bins is minimized.

[1]https://www.ibm.com/analytics/cplex-optimizer

Our placement of microservices across several data centers is actually a multi-dimensional bin-packing problem in which (i) the resource usage and microservices latency are considered and (ii) evaluating the latency constraints adds to the complexity of the solution. Overall, the resolution of our NP-hard problem with a solver requires a prohibitive processing before reaching the optimal solution.

IV. Placement algorithms

In order to address the scalability issue of the latency-aware placement problem previously formulated, we introduce an heuristic approach which combines a greedy algorithm and a genetic algorithm. The greedy algorithm (see § IV-A) is used to generate an initial (and possibly not optimal) distribution of microservices on the substrate network. Then, the genetic algorithm (see § IV-B) further refines the initial placement and attempts to find the best solution [5] by iteratively improving the quality of the result and helping the search process to escape from local optima. We could have opted for either approach, but by combining these two algorithms we get a faster solution without sacrificing the quality.

A. Finding an initial placement using a greedy heuristic

The design rational of our greedy algorithm is to consider the substrate network characteristics (occupancy and underlying structure) to smartly place the microservices in a fair manner, i.e., without discriminating some services over others based on their internal characteristics (e.g., number of microservices or their demand in terms of resources). In particular, the greedy algorithm favours the placement of the chain of microservices composing one service within the same data center, ideally at the edge. This placement permits to minimize the service completion time and the latency perceived by the end user while favouring short-distance communications between microservices and hence avoiding as much as possible long-distance communications between data centers.

B. Enhancing the initial placement using genetic algorithm

The Genetic Algorithm (GA) mimics evolutionary processes, i.e., at each iteration of the GA, the population of individuals evolves using three genetic operators: selection, crossover, and mutation.

1) Population Encoding: The population (Figure 2) consists of a set of datacenters $C_1 \cdots C_N$. Within the population, each individual (i.e. datacenter) is represented by a *chromosome* that consists of series of genes. A *gene* is a binary variable representing the presence or absence of a microservice on a data center. We thus define the nth chromosome C_n (with $n = 1, \ldots, N$) as a binary series $\mathbb{1}(v_n, \sigma_i^j)$ for $i = 1, \ldots, K_j$. Each chromosome (data center) is characterised by its own resource capacity – in this model we only consider CPU capacity – and each gene by its required resource capacity.

Once the initial population is generated using the greedy approach (IV-A), the population evolves iteratively: at each iteration of the genetic algorithm, the population evolves by applying the selection, crossover, and mutation operators.

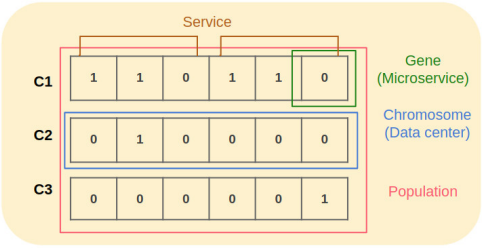


Fig. 2: Population encoding

2) Selection process: Among the population, the selection operator selects the best individuals to let them reproduce and have an *offspring* (through crossover and mutation as detailed below), i.e., a set of new individuals, which composes the subsequent generation. The chromosomes with the best fitness value envisaged through two ways of fitness value calculation defined in equations (7) and (6) - network agnostic and network-aware metrics, respectively. The total fitness of the population is expected to rise (for network agnostic metric) or to decrease (for the network aware metric) with the algorithm.

3) Crossover: After selecting certain individuals for reproduction, the crossover operator swaps genes (bit strings) between two selected parent chromosomes ($P1$, $P2$) to create two new off springs (Figure 3). Based on our analysis and literature study, the crossover probability value ranges between 0.1 and 0.8 resulted in a better solution.

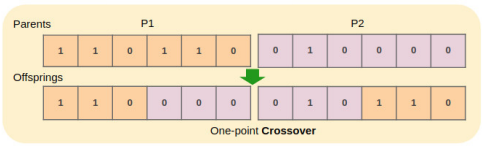


Fig. 3: One-point Crossover

4) Mutation: The mutation operator performs a mutation of certain individuals from the new offspring to diversify the generations. The mutation consists of changing a random number of genes in the chromosome of an individual. The value of mutation probability is suggested to be lesser than crossover probability, i.e., within a range of 0.01 to 0.2 (as per our sensitivity analysis carried out by multiple runs of algorithms with different probability).

V. Experimental results

A. Experimental setting

In order to evaluate our approach to the placement of microservices, we consider the substrate network displayed in Figure 1, which reasonably represents a telco infrastructure including, cloud, fog and edge nodes. Considering the closeness of edge and fog nodes, we assume a smaller latency (one unit) between the edge and fog nodes than that between the fog nodes and the centralized cloud, which is equal to 3 units. In general, the central cloud is the largest and consists of nodes with a large capacity (1000 units) comparing to the fog nodes (capacity of 100 units) and edge nodes (capacity of 20 units). Following, we consider two types of services that differ in the number of microservices:

- The first service type (say, a lightweight application running at the edge such a firewall) involves a relatively small number of microservices: 3 microservices exchange messages with $\nu(\sigma_1^1, \sigma_2^1) = 2$ and $\nu(\sigma_2^1, \sigma_3^1) = 4$.
- The second service type (a heavyweight application) is composed of 10 microservices exchanging messages with $\nu(\sigma_1^2, \sigma_2^2) = \nu(\sigma_4^2, \sigma_5^2) = \nu(\sigma_6^2, \sigma_7^2) = \nu(\sigma_7^2, \sigma_8^2) = 3$, $\nu(\sigma_2^2, \sigma_3^2) = \nu(\sigma_3^2, \sigma_4^2) = \nu(\sigma_5^2, \sigma_6^2) = 2$, $\nu(\sigma_8^2, \sigma_9^2) = 4$ and $\nu(\sigma_9^2, \sigma_{10}^2) = 1$.

We assume that for both service types, user exchanges only two messages with the first microservice and none with others ($\nu(\sigma_0^j, \sigma_1^j) = 2$ and $\nu(\sigma_0^j, \sigma_i^j) = 0$ for all j and $i > 1$); this reflects a single input and output message between the user and the service. In addition, in the computation of the quantities $\tilde{L}_j$, we have taken $\tilde{\nu}(\sigma_0^j, \sigma_1^j) = 100$ and $\tilde{\nu}(\sigma_0^j, \sigma_i^j) = 0$ for all j and $i > 1$. This is to force the algorithm to collocate the user and the first microservice.

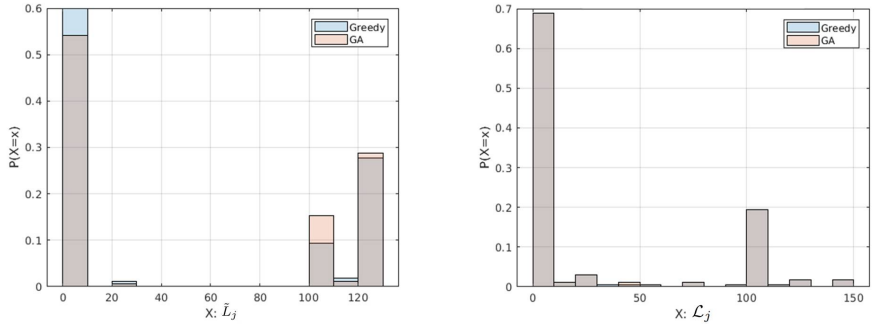
The capacity requirement for each microservice is set equal to 1. During our experiments, services of type 1 constitute $2/3$ of the services. Thus, we have a maximal population of $J_0 \approx 278$ services with $J_1 = 185$ services of type 1 and $J_2 = 93$ of type 2, which can be hosted by the system. The maximum order of magnitude of chromosome length is then $3J_1 + 10J_2 \approx 1480$ genes. In the following, we consider two representative loads: $\rho = .7$ (light load) and $\rho = .9$ (heavy load).

B. Numerical results

Our evaluation investigates to which extent maximizing the metric $\tilde{L}$ (defined in a network agnostic way) is relevant for controlling the global latency $\mathcal{L}$. For this purpose, we have first evaluated the network agnostic method (see Fig. 4) using the probability density function (pdf) of $(\tilde{L}_j)$ and that of $(\mathcal{L}_j)$ for load ρ equal to 0.7 and 0.9. We observe that the GA algorithm slightly improves $\tilde{L}$. As observed in Table I, the mean value $\mathbb{E}(\tilde{L})$ of the series $(\tilde{L}_j)$ is better with the GA algorithm. However, the improvement is very marginal when looking at global latency, given by the series $(\mathcal{L}_j)$ with mean value $\mathbb{E}(\mathcal{L})$. This indicates that the network agnostic optimization is not sufficient to significantly improve the latency. Then, we focus on the network aware metric in Figure

TABLE I: End to end latency of services.

Placement	Load	J_1	J_2	$\mathbb{E}(\tilde{L})$	$\mathbb{E}(\mathcal{L})$
Greedy	0.71	118	52	49.78	31.39
GA network agnostic	0.71	118	52	56.20	31.28
Greedy	0.70	109	55	54.76	33.04
GA network aware	0.70	109	55	38.95	22.01
Greedy	0.90	125	76	52.06	40.21
GA network agnostic	0.90	125	76	56.51	40.06
Greedy	0.91	148	68	46.36	33.55
GA network aware	0.9	148	68	32.87	24.79



(a) Pdf of $(\tilde{L}_j)$ for $\rho = 0.7$

(b) Pdf of $(\mathcal{L}_j)$ for $\rho = 0.7$

(c) Pdf of $(\tilde{L}_j)$ for $\rho = 0.9$

(d) Pdf of $(\mathcal{L}_j)$ for $\rho = 0.9$

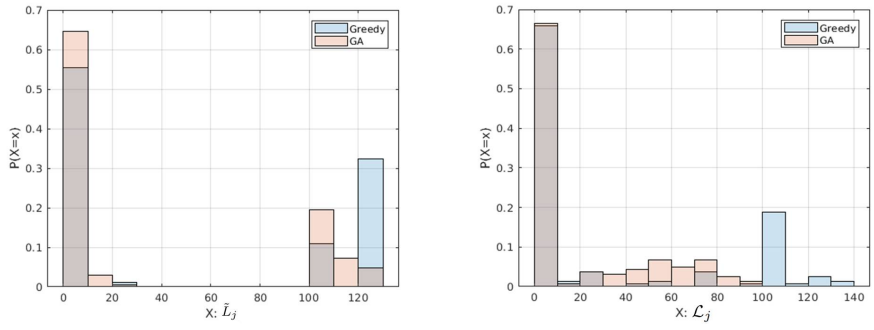
Fig. 4: Network agnostic optimization.

5 that provides the pdfs of the series $(\tilde{L}_j)$ and $(\mathcal{L}_j)$, for a load of $\rho = 0.7$ and $\rho = 0.9$. We see that the GA algorithm significantly improves the global latency; this can also be seen for mean values. It is worth noting that improving the latency does not increase the values of $(\tilde{L}_j)$. Thus, maximizing $\tilde{L}$ and minimizing $\mathcal{L}$ could go in opposite directions. Results show that GA improves the placement performed by Greedy, however the main conclusion of this work is that placement algorithms either in the infrastructure layer (e.g., in Cloud OS environments as Kubernetes and Openstack) or upper in the Orchestration layer (e.g., ONAP) need to be network aware.

VI. Related work

The VNF placement problem is a long-standing research topic [6] that is usually framed as an optimization problem considering mostly the resource utilization (computing, storage or network capacity) [2], [7], [3] and to a lesser extent the latency, which is the focus of our work. In order to reduce the latency, Alleg *et al.* [8] rely on Mixed-Integer Quadratically Constrained program to find the optimal placement of a chain of network Functions, which is the one that minimizes the processing delay. The work presented in [9] departs from the previous research work by considering several network-related criteria, e.g., the infrastructure size, request size, network connectivity and load of a system. In order to place the microservices, authors introduce an eigen-decomposition approach and follow a two steps procedure, where VNFs are first optimally placed and then the traffic is fairly distributed among VNFs following the shortest path. In [10], authors propose a multi-objective optimization strategy that places Service Function Chains (SFCs) across edge-cloud environment with the aim of reducing the deployment cost and end-to-end latency jointly. The proposed Mixed Integer Programming (MIP) model is further solved using two heuristic algorithms - GA-based algorithm and Bee Colony-based algorithm for large sized cloud/edge environments. Substantial variety of criteria are considered, including resource capacity (CPU and storage), acceptable latency requirements (end-to-end latency is calculated by combining the propagation delay, processing and virtualization delay, and queuing delay), affinity and anti-affinity (both consider as a binary variable). Comparatively, a smaller set of criteria (queuing & processing delays as well as transmission delays) is used in [11] to optimize the placement of microservices chain over an edge-cloud infrastructure. The problem formalised using MIP is further solved by a meta-heuristic based Tabu Search algorithm.

The above off-line placement strategies may be complemented by online allocation strategies. In [12], authors handle the off-line and online allocation of microservices based on the traffic demand and network state. Two problems are addressed jointly: NF placement and flow routing problem (reduction of stretch path encountered by flows) with the constraints of number of cores per nodes, memory, link capacity, delay (sum of link delays and total nodal delay). The work presented in [13] arranges the microservices based on their affinities (defined by number and size of messages) and history of



(a) Pdf of $(\tilde{L}_j)$ for $\rho = 0.7$

(b) Pdf of $(\mathcal{L}_j)$ for $\rho = 0.7$

(c) Pdf of $(\tilde{L}_j)$ for $\rho = 0.9$

(d) Pdf of $(\mathcal{L}_j)$ for $\rho = 0.9$

Fig. 5: Network aware optimization.

resource usage on the same physical server. The prototype also allows runtime allocation of containerized microservices.

VII. Conclusion

Provisioning a reliable and timely service delivery over a ISP network is a critical issue that we addressed through the introduction of a latency-effective microservice placement strategy that allocates the microservices on computing nodes, spanning from the edge to the cloud. We proposed two mathematical formulations of the notion of latency: the former favours the allocation of microservices in the same compute node and near to end users by promoting the presence of collocated services near to the end user, thereby lowering the cost of communications. The latter minimizes the communication delay between all the data centers and avoids as much as possible long distance communications. We further propose an ILP formulation of the placement problems, which are solved by a hybrid algorithm combining greedy and genetic methods.

As part of our future work, we plan to study the online placement and migration of containerized applications or network functions while keeping the service active.

References

[1] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[2] S. Tavakoli-Someh and M. H. Rezvani, "Utilization-aware virtual network function placement using nsga-ii evolutionary computing," in *IEEE Conference on Knowledge Based Engineering and Innovation*, 2019.

[3] N. Djennane, R. Aoudjit, and S. Bouzefrane, "Energy-efficient algorithm for load balancing and vms reassignment in data centers," in *6th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*. IEEE, 2018, pp. 225–230.

[4] F. Slim, F. Guillemin, A. Gravey, and Y. Hadjadj-Aoul, "Towards a dynamic adaptive placement of virtual network functions under onap," in *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2017, pp. 210–215.

[5] R. K. Ahuja, J. B. Orlin, and A. Tiwari, "A greedy genetic algorithm for the quadratic assignment problem," *Computers & Operations Research*, vol. 27, no. 10, pp. 917–934, 2000.

[6] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1409–1434, 2018.

[7] M. Hadji, N. Djenane, R. Aoudjit, and S. Bouzefrane, "A new scalable and energy efficient algorithm for vms reassignment in cloud data centers," in *IEEE International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, 2016, pp. 310–314.

[8] A. Alleg, T. Ahmed, M. Mosbah, R. Riggio, and al., "Delay-aware vnf placement and chaining based on a flexible resource allocation approach," in *IEEE conf. on network and service management*, 2017.

[9] M. Mechtri, C. Ghribi, and D. Zeghlache, "A scalable algorithm for the placement of service function chains," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 533–546, 2016.

[10] M. A. Khoshkholghi, M. G. Khan, K. A. Noghani, and al., "Service function chain placement for joint cost and latency optimization," *Mobile Networks and Applications*, pp. 1–15, 2020.

[11] A. Leivadeas, G. Kesidis, M. Ibnkahla, and al., "Vnf placement optimization at the edge and cloud," *Future Internet*, vol. 11, no. 3, 2019.

[12] Y. T. Woldeyohannes, A. Mohammadkhan, K. Ramakrishnan, and al., "Cluspr: balancing multiple objectives at scale for NFV resource allocation," *IEEE Trans. on Net. and Serv. Manag.*, vol. 15, no. 4, 2018.

[13] A. R. Sampaio, J. Rubin, I. Beschastnikh, and al., "Improving microservice-based applications with runtime placement adaptation," *Journal of Internet Services and Applications*, vol. 10, no. 1, 2019.