A microservice migration approach to controlling latency in 5G/6G networks

Kiranpreet Kaur, Fabrice Guillemin, Francoise Sailhan

A microservice migration approach to controlling latency in 5G/6G networks

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

***Abstract*—The microservice paradigm is widely adopted in the design of Virtualized Network Functions (VNFs). Still, 5G/6G networks require to carefully orchestrate the allocation and re-arrangement of (micro)services to avoid a largely segmented solution space while services arrive and leave the network. To support latency-effective service provisioning, we introduce a novel placement and migration strategy that chooses the microservice(s) to migrate and selects the optimal destination (data center) while considering the impact of the migration on other microservices. For this purpose, we devise fast and effective heuristics and we implement a prototype that maps the service in the deployment field. Extensive simulations show our proposed approach significantly reduces the service latency.**

***Keywords*-Microservices migration; Containerized network function**

I. Introduction

Network Function Virtualization (NFV) has been introduced to facilitate the management and provision of network capabilities using virtualized software applications hosted on Commercial off-the-shelf (COTS) servers [1]. Initially, VNFs were based on Virtual Machines (VMs) with the aim of replacing hardware-based physical networking functions. Then, the current trend in the evolution of NFV design involves the usage of containers to support the so-called cloud-native network functions (CNFs). In practice, the cloud-native approach adopted by many telecom network operators involves small and loosely-coupled microservices that are deployed in containers and scaled (up and down) as needed [1], [2].

The growing use of microservices for 5G/6G based services is driving the telecom industry to find efficient ways of exploiting new communication and computing technology. In particular, various placement strategies [3], [4] have been proposed to initially allocate VNFs and allow efficient usage of computing resources while inducing reduced load or latency. However, the initial placement of services is sometimes not able to continuously meet the Service Level Agreement (SLA), resulting in degradation of communication network performance across time. Microservices also tend to naturally scatter throughout the softwarized network while performing repetitive allocation operations, e.g., addition or removal of a service. Meanwhile, real-time applications in the 5G/6G network require to be placed close to the end-user and migrated to follow the mobile user that moves from one position to another. Such situations tend to degrade the performances due to the fact that there is an increase in the number of communication among microservices that are split into multiple servers and/or data centers. In this case, online reallocation/migration of services can significantly improve the efficiency of a softwarized network.

978-1-7281-7705-2/21/$31.00 ©2021 IEEE

Aiming to support an efficient migration of NFVs at a scale while meeting the latency requirements is the topic addressed by this article that introduces a novel migration strategy that (re-)allocates microservices on the optimal data center(s) considering the server load and service latency. The design rationale puts the more forth on user-centric and highly-active microservice while selecting and migrating them vertically (from the edge up to the cloud) or horizontally (between the data centers at the same layer). This approach tends to select few microservices to migrate instead of all which positively impacts the global latency factor of the whole network.

After positioning the proposed solution with respect to the state of the art in the next section, the paper details the following contributions:

- We formalize the problem of migrating microservices across several data centers. In particular, we are trying to solve this ever-demanding migration problem by ensuring the lesser number of microservices are moved while keeping the placement optimal.
- We introduce an approximate problem-solving solution with three heuristics that considerably reduce run time of the migration algorithm.
- We provide a simulation-based evaluation that shows that the migration of services is efficient using our proposed heuristic algorithm and minimizes the latency.

This paper is organized as follows: in Section II, we review existing work on microservice migration. The model considered in this paper (in particular the cloud infrastructure) is presented in Section III. The dynamic system as well as the metrics considered as quality indicators are presented in Section IV. The placement and migration algorithms are described in Section V. Simulation results are reported in Section VI. Concluding remarks are presented in Section VII.

II. Related work

The problem of allocating VNFs or Cloud-Native Network Functions (CNFs) is getting the attention of researchers and practitioners. As detailed below, various heuristics and machine learning-based approaches [4] have been proposed to support the dynamic migration of VNFs/CNFs instances.

In [5], authors address the VNF migration problem and propose a Mixed Integer Linear Programming (MILP) model that attempts to minimize the Service Function Chain (SFC) delays, considering constraints related to resources (CPU & memory) requirements, network delay, based on affinity & anti-affinity factors, migration delay (time required to discover service and to propose new placement). Based on their MILP model they analyse the impact of migration on VNF and choose accordingly a greedy approach that reduces the downtime while considering the established constraints.

In [6], a migration strategy for edge and containerized applications is introduced. In particular, an Integer Linear Programming (ILP) model minimizes the service downtime and latency while handling the migration across a cloud-edge environment. For instance, a re-instantiating is considered in case of node failure. In such a case, a heuristic that minimizes downtime has been implemented to address the computational complexity and lack of scalability faced by the proposed mathematical model.

In [7], the migration strategy is framed as multiple dimensional Markov Decision Process (MDP) for fog computing, which is solved using combined algorithms of Deep Q-learning and Deep Neural Networks (DNN). The system states are delay, power consumption and migration cost. Further, the actions are following a greedy approach which contains the selection policy that chooses the source container to be migrated depending on under-/over-utilization of nodes. At under-utilized node, all the containers are migrated to minimize the power consumption whereas at over-utilized node, the container with minimum migration cost is migrated; allocation policy selects the target node for each migrated container. To support faster learning, the reinforcement learning (Double DQN) and Prioritized Experience Replay (PER) is used during the training process. The outcome of the resulting solution shows better results in comparison with existing baseline Q-learning-based strategies, considering VNF and CNF.

The work [8] focuses on reassigning the containers of the same service close to each other. The placement maps the new containers with the aim of balancing the load and reducing the communication cost, using a customized version of Worst Fit Decreasing (WFD). Re-assignment approach relies on Sweep&Search algorithm that re-optimizes the initial placement by minimizing the total cost associated with communication, occupied resource and residual resource balance.

In [9], an online container placement strategy considers the inter-container traffic. First, a one-shot offline integer programming optimization problem has been formulated with quadratic constraints to fetch the traffic flow. Further, an online scheme following a primal-dual method has been proposed: the placement is taking place at each service arrival.

Load balancing strategy [10] is applied only on long-lived containers that occupy the resources for a long duration. Long-lived containers are arranged according to the CPU resource they require; then highly occupied hosts are selected to swap containers with low occupied hosts. The scheduling algorithm is based on a random-first-fit algorithm. These processes are run continuously to uniform the load.

In [11], a migration problem for shared VNFs in a multi-domain federated network is addressed. In case of failure, the coordination algorithm migrates the shared and chained VNFs using the information provided by each orchestrator.

The dynamic migration approach [12] introduces a novel heuristic to reduce workload, deal with user-mobility and keep to a minimum migration time. The algorithm shortlists the containers characterised by high latency. For each selected container, the selected neighbor is the one that is less or moderately utilized and geographically near the user.

The ongoing research works on VNFs placement and re-arrangement ignore the joint problem of chaining the microservices: in practice, network functions are placed focusing on the resource availability/need and/or migration time while omitting to consider the latency associated with the communication between the chained microservices and the end-user that would allow the end-to-end latency to be optimized.

Our goal in this paper is not only to find an optimal target node while satisfying load, delay or latency constraint but also to acquire a way that must co-join the chain of microservices. Therefore, compared to previous works, the proposed work jointly tackles the optimal placement and migration strategy for real-time scenarios, aiming at minimizing the network delay and end-to-end latency between users and services.

III. Model description

A. Cloud infrastructure

We consider in Fig. 1 a network that includes several clouds organized in three levels. Upper level is composed of a centralized cloud corresponding to a national cloud with huge capacity. The second level consists of series of regional clouds with intermediate capacities; those regional clouds are connected to the central cloud and to other cloud nearest to end users. This last level of cloud is composed of edge clouds with limited capacity but close to end users; edge clouds are connected to their respective regional cloud. This three level hierarchy reasonably represents the cloud infrastructure of a network operator hosting VNFs.

B. Placement of services

We consider the problem of placing a set of services on a cloud infrastructure composed of the set of data centers $D_1, \ldots, D_N$; where N is the number of data centers. Each service S is composed of J_S microservices $\sigma_1, \cdots \sigma_{J_S}$; each microservice σ_j (for $j = 1, \ldots, J_s$) requires a certain amount of CPU, disk and RAM. In practice, RAM and CPU are both the most scarce in cloud infrastructures. We denote by $c(\sigma_j)$

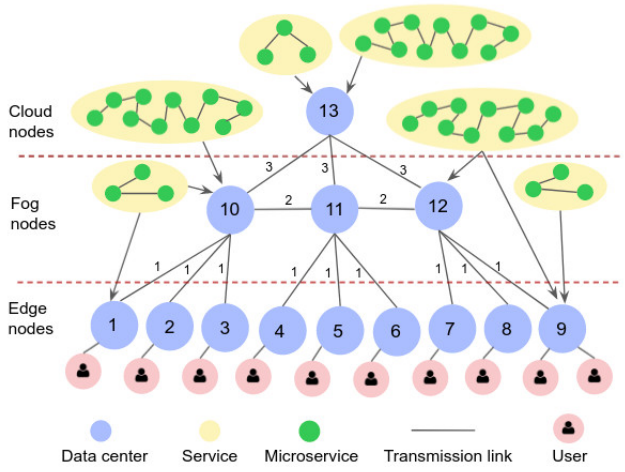


Fig. 1. Cloud infrastructure of a network

and $r(\sigma_j)$ the resource requirements of microservice σ_j in terms of CPU and RAM, respectively.

The placement problem consists of finding a mapping function h from the set $\mathcal{S}$ of services to the set $\mathcal{D}$ of data centers. More precisely, we consider the mapping $h : S \in \mathcal{S} \to (h(\sigma_1), \ldots, h(\sigma_{J_S})) \in \mathcal{D}^{J_S}$, where $h(\sigma_j) = D_n$ if microservice σ_j is placed on data center D_n. If microservice σ_j cannot be placed because of resource exhaustion, then we set $h(\sigma_j) = 0$. In that case, no microservices of S are placed and $h(S) = \mathbf{0} \stackrel{def}{=} (0, \ldots, 0)$.

Let $\mathcal{M}_n^{(h)}$ denote the set of microservices placed on data center D_n under placement h. Given C_n and R_n denoting the CPU and RAM capacities of data center D_n, respectively, the following constraints apply:

$$\sum_{\sigma \in \mathcal{M}_n^{(h)}} c(\sigma) \leq C(D_n) \text{ and } \sum_{\sigma \in \mathcal{M}_n^{(h)}} r(\sigma) \leq R(D_n). \quad (1)$$

The set of services having a microservice hosted by data center D_n is $\mathcal{S}_n^{(h)} = \{S \in \mathcal{S} \mid \exists \sigma \in S \text{ and } h(\sigma) = D_n\}$. The set of services (resp. microservices) that can be placed is $\mathcal{S}^{(h)} = \bigcup_{n=1}^{N} \mathcal{S}_n^{(h)}$ (resp. $\mathcal{M}^{(h)} = \bigcup_{n=1}^{N} \mathcal{M}_n^{(h)}$). The mapping h has to satisfy constraint (1) while additional criteria can be introduced, e.g., load balancing between data centers, maximization of the number of placed services, etc. In the former case, the following optimization problem in which only CPU is considered, needs to be solved:

$$\min_h \max_{n \in \{1,\ldots,N\}} \frac{1}{C_n} \sum_{\sigma \in \mathcal{M}_n^{(h)}} c(\sigma) \quad (2)$$

while h has to satisfy global placement objectives, for instance the maximum utilisation of the global capacity of the cloud infrastructure, whose objective reads

$$\max_h \frac{\sum_{S \in \mathcal{S}^{(h)}} \sum_{\sigma \in S} c(\sigma)}{\sum_{n=1}^{N} C_n}, \quad (3)$$

or the maximum fraction of services which can accepted in the system, that is, $\max_h \frac{|\mathcal{S}^{(h)}|}{|\mathcal{S}|}$. Finally, anti-affinity rules can be introduced to prevent two microservices being placed on the same data center (for instance for security or resilience reasons).

C. Latency of services

In the following, we are interested in the global latency experienced by a service S composed of microservices $\sigma_1 \cdots \sigma_{J_S}$. A dummy microservice σ_0 is added to represent the location of the user, which is attached to an edge node of the cloud infrastructure (Fig. 1). Any microservice σ_j (with $j = 0, \ldots, J_S$) may exchange messages. We define the message exchange matrix $\nu_S = (\nu(\sigma_i, \sigma_j))$ for service S, where $\nu(\sigma_i, \sigma_j)$ for $i, j = 0, \ldots, J_S$, is the global number of messages exchanged between microservices σ_i and σ_j. We clearly have $\nu(\sigma_i, \sigma_j) = \nu(\sigma_j, \sigma_i)$ and $\nu(\sigma_i, \sigma_i) = 0$. Moreover, ν_S is a $(j_S + 1) \times (j_S + 1)$ symmetric matrix.

If microservices σ_i and σ_j are not placed on the same data center, then the transmission across the links connecting the two data center introduce latency in the execution of the service. Let $d_{n,m}$ denote the delay between data centers n and m. In the following, we neglect the delay inside a data center (i.e., $d_{n,n} = 0$) as this delay is low compared with transmission delays between remote data centers.

For a given placement h, let us define the $(j_S + 1) \times (j_S + 1)$ delay matrix $\Delta_S^{(h)} = (d_{h(\sigma_i), h(\sigma_j)})$ for service S under placement h. Then, the latency affecting service S is

$$\ell_S^{(h)} = \sum_{0 \leq i < j \leq J_S} \nu(\sigma_i, \sigma_j) d_{h(\sigma_i), h(\sigma_j)} \quad (4)$$

Owing to the symmetry of matrices,

$$\ell_S^{(h)} = \frac{1}{2} \mathrm{Tr}(\nu_S \Delta_S^{(h)}), \quad (5)$$

Tr is the trace operator.

The global latency of the system under placement h is defined as

$$\mathcal{L}^{(h)} = \sum_{S \in \mathcal{S}^{(h)}} \ell_S^{(h)} \quad (6)$$

and the average latency as

$$\overline{\mathcal{L}}^{(h)} = \frac{1}{|\mathcal{S}^{(h)}|} \sum_{S \in \mathcal{S}^h} \ell_S^{(h)}. \quad (7)$$

With regard to placement, we can introduce the following optimization problems: Minimizing global (resp. average) latency $\min_h \mathcal{L}^{(h)}$ (resp., $\min_h \overline{\mathcal{L}}^{(h)}$) or minimizing the maximum latency of services $\min_h \max_{S \in \mathcal{S}^{(h)}} \ell_S^{(h)}$ with h satisfying in addition criterion (3) and achieving maximum acceptance rate.

IV. Dynamical system and associated metrics

A. Dynamical setting

While many studies in service placement assume a static setting, where the global set of services to be placed is known in advance and fixed, we consider a dynamic system where services join and leave the system. In that case, the placement

strategy should take account of the service dynamic in the sense that:

- Each arriving service has to be placed by taking into account the current state of the system, possibly by migrating some microservices in order to control the latency of the new service while also controlling the latency of services with migrated microservices;
- At each departure of a service, resources are released and can be used for microservices migration so as to reduce latency of services in the system, e.g., according to the optimization problems (see Section III-C).

If services are accepted on a capacity basis only, then we have a blocking system (see the seminal paper [13]). As long as the service can be placed, the service is accepted regardless of incurred latency.

In the following, we assume that there are K classes of services. Those services of class k ($k = 1, \ldots, K$) arrive according to Poisson processes with rate λ_k. A service of class k, if accepted, stays for a random amount of time with mean $1/\mu_k$. A service S_k of class k has a global resource requirement $A_k = \sum_{\sigma \in S_k} c(\sigma)$. The global capacity of the system is $C = \sum_{n=1}^{N} C(D_n)$.

If the global capacity C is finite then we have a multirate loss network [13]; this kind of model has been used to dimension multiservice circuit switched networks and the blocking probability which gives an estimation of loss probabilities in different regimes. When the capacity is infinite (as it is assumed in the following), the number of services of class K is Poisson with mean λ_k/μ_k.

In analogy with this kind of networks, the problem we consider in this paper is similar to assigning the circuits (microservices) of a multi-rate bundle (a service) to transmission capacities (data centers). Migrating microservices is similar to repacking circuits. The concept of latency of services however does not easily translate into a criterion for repacking circuits. Hence, algorithms designed for circuit repacking cannot be easily adapted to the context considered in this paper.

B. Metrics

When dealing with a QoS requirement like latency, we could impose that when a service joins the system and the QoS objectives for this service cannot be met, then the service is rejected. This may however lead to under-utilization of the system. Instead, we propose to accept all services and we use the capability of migrating microservices to keep the latency under control. An issue is to determine control metrics.

So far, we have defined in Section III-C latency of services in a static situation. We can nevertheless define a random variable $\ell^{(h)}$ taking values in the set $\{\ell_S^{(h)}, S \in \mathcal{S}^{(h)}\}$. When dealing with a dynamic system, we compute the latency of those services that are in the system. Contrary to the static case, the service latency can vary in time due to migration. If a service S has a holding time τ_S, then we define the mean latency under a migration strategy m as

$$\overline{\ell}_S^{(m)} = \frac{1}{\tau_S} \int_{t_S}^{t_S+\tau_S} \ell_S^{(m)}(u) du,$$

where t_S is the arrival date of service S and $\ell_S^{(m)}(t)$ is the latency experienced by service S at time t.

When considering a population of services under service migration policy m and placement h, we define the mean latency

$$\mathbb{E}(\ell^{(m)}) = \frac{1}{|\mathcal{S}^{(h)}|} \sum_{S \in \mathcal{S}^{(h)}} \overline{\ell}_S^{(m)}).$$

This is a global metric reflecting the efficiency of a migration policy m in terms of latency.

V. Algorithms for placement and migration of services

We consider in this section several algorithms to place new services (§V-A) and migrate the services when needed (§ V-B).

A. Placement of new services

In order to place arriving services, we consider two greedy algorithms: the Greedy First Fit algorithm and the Greedy Best Fit algorithm.

1) The Greedy First Fit (GFF) algorithm: This one greedily allocates the services on the various data centers by proceeding as follows:

1) Place randomly the user of the service at a edge node n (the user does not consume any resources).
2) Place as many microservices (a service is an ordered chain of interacting microservices) as possible on this edge node. If all microservices can be placed, then stop.
3) Otherwise, place the remaining microservices on the fog node to which the edge node is attached to. If all remaining microservices can be placed on the fog node, then stop the placement.
4) Otherwise, continue to place the remaining microservices on the cloud node. (Recall that we assume that the capacity of the central cloud is infinite.)

2) Greedy Best Fit (GBF) algorithm: This other algorithm aims at reducing the service fragmentation (i.e., the number of data centers hosting the microservices of the serivce) by greedily placing all the microservices as much as possible on the same data center.

The GFF or GBF algorithms are adopted to place the microservices on each service arrival while microservices migration is suitable when a service leaves.

B. Migration Strategy

Upon departure, a service releases resources in data center(s) that may be overloaded (e.g., typically the edge or fog data center) and some microservices could be migrated to reduce the service latency. Thus, it is worth identifying the set of fragmented microservices which (i) incur high latency because they are located on distant data centers and/or (ii) exchange many messages with end-user. In such a case, it

is suitable to migrate the microservices to the data center(s) hosting the paired microservice which increases the latency.

To migrate a microservice S, resources should be available on the destination data center. If no resources are available in the saturated data center, it may be envisaged to migrate some candidate microservices, which are placed on the same data center (or a neighbouring data center) and which exchange the least number of messages with the other microservices of the candidate service. This happens only if sufficient resources are released to host the microservices of S and if there is a latency gain, i.e., if the difference between the latency reduction achieved by migrating the microservice of S and the latency increase due to the migration of the candidate microservices is positive. Overall, the migration strategy has been explained in Algorithm 1.

Algorithm 1: $Migration(\mathcal{D}, \mathcal{C})$ algorithm for the migration of a service after the departure of a service on a set of data centers $\mathcal{D}$ with capacity $\mathcal{C}$

Input : Set of Data centers $\mathcal{D}$; Total available capacity $\mathcal{C}$;
Required CPU capacity $c(\sigma)$ for microservice $\sigma \in S$;
Output: Updated set $\mathcal{D}$ of data centers;
S_f^*: set of fragmented services;
$\mathcal{D}_S$= current placement in the datacenters ;
$S_f^* = sortDescendOrder(S_f^*)$;
for *each iteration* $i = length(S_f^*)$ **do**
  $S_i = MaxFragmentedService(S_f^*)$;
  $find(\sigma_i, \sigma_j)$; // find 2 μservices in different DCs and inducing the maximum delay;
  $find(\mathcal{D}_{\sigma_i}, \mathcal{D}_{\sigma_j})$ // Get current location of μservice and end-user;
  if $c(\sigma_j) > C(D_{\sigma_i})$ **then**
    $S_j = MinFragmentedService(S_f^*, D(\sigma_i))$;
    $select(\sigma) == c(\sigma_j)$ // Select the set of μservices equals to capacity required to place the migrated μservice;
    $FirstFit(D_\sigma)$;
  end
  if $new_{L_{S_j}} + new_{L_{S_i}} < old_{L_{S_j}} + old_{L_{S_i}}$ **then**
    $Migrate(\sigma_j, D_{\sigma_i})$; $Update[D_S]$;
    remove S_i from S_f^*
  end
end

VI. Experimental results

A. Simulation setting

We consider a cloud infrastructure as the one illustrated in Fig. 1; this infrastructure is composed of a cloud node (with infinite capacity), 3 fog nodes (with 100 capacity each) and 9 edge nodes (with 20 capacity each). Further, we have two types of services differing in the number of microservices:

- **Small services:** They correspond to lightweight applications (e.g., a firewall) and are composed of a small number of microservices, namely 3 microservices exchange messages with $\nu(\sigma_1^1, \sigma_2^1) = 2$ and $\nu(\sigma_2^1, \sigma_3^1) = 4$; the number $\nu(\sigma_0^1, \sigma_1^1)$ of messages between the end user and the first microservice may change.
- **Large services:** They are related to large and heavy-weight applications; we specifically assume that each service comprises 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 $\nu(\sigma_9^2, \sigma_{10}^2) = 1$; as above, the number $\nu(\sigma_0^1, \sigma_1^1)$ may change.

The required capacity of each microservice equals to 1.

Services arrive according to Poisson processes with rate λ_k for class $k = 1, 2$ and stay in the system for an exponentially distributed period of time with mean $1/\mu_k$.

B. Numerical results

In the following, we compare the proposed approaches: the *Greedy First Fit (GFF)*, the *Greedy Best Fit (GBF)* and the *Migration strategy* taking place along with GBF based on number of departures, considering the latency and fragmentation associated with small and large services. The latency (see Eq. (4)) accounts for: (i) the messages exchanged between the microservices itself as well as those exchanged with the end-user and (ii) the distance between the data centers. The average fragmentation reflects to which extent microservices are fragmented on different data centers.

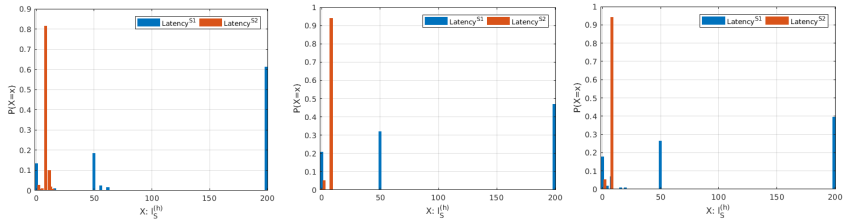
TABLE I
Mean of global latency

Service Type	Methods	Number of messages			
		2	5	10	20
Small	GFF	5.95	13.944	27.223	53.931
	GBF	4.422	11.151	22.246	44.817
	Migration	4.873	10.291	19.197	37.821
Large	GFF	8.136	8.114	8.160	8.129
	GBF	7.647	7.627	7.622	7.606
	Migration	7.637	7.640	7.606	7.648

In a first step, we let vary the number of the messages exchanged between the end user and the first microservice. As shown in Table I, our migration algorithm results in a lower average global latency for small and large services. As expected, the global latency increases with the number of messages exchanged between the microservices and the end-user since the first microservices may be placed in fog and cloud data centers.

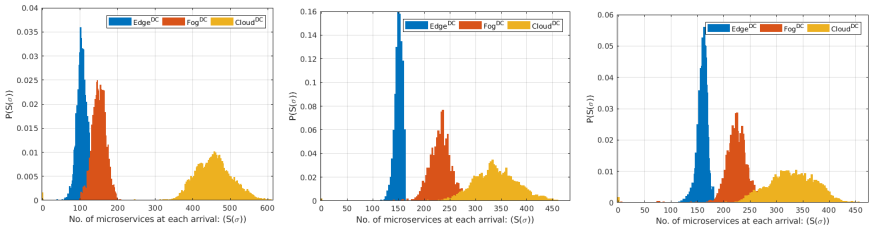
Figure 2 compares the strategies in terms of the probability mass function (pmf) of latency $\ell_S^{(h)}$ for small and large services with a number of exchanges equal to 50 and 2, respectively. We observe that the migration strategy (Fig. 2 (c)) minimizes the service latency in comparison to the GFF and GBF strategies (Fig. 2 (a) & (b)).

We further study the resulting placement of the services at the different network layer (edge, fog and cloud) considering the GFF, GBF and "best fit along with migration" approaches, regarding small services (Fig. 3) and large services (Fig. 4). In particular, we consider the number of microservices placed on each layer which reflects the CPU resources that are consumed by the services, at each service's arrival. As expected, GFF (Fig. 3 (a)) first consumes the edge resources and then moves to the cloud layer. GBF consumes more of the edge and fog for small services (Fig. 3 (b)) compared to large services (Fig. 4 (b)) that are mostly placed on cloud DCs. Likewise, migration strategy along with GBF (Fig. 3 (c) and 4 (c)) shows a similar trend but performs better than the GBF strategy:



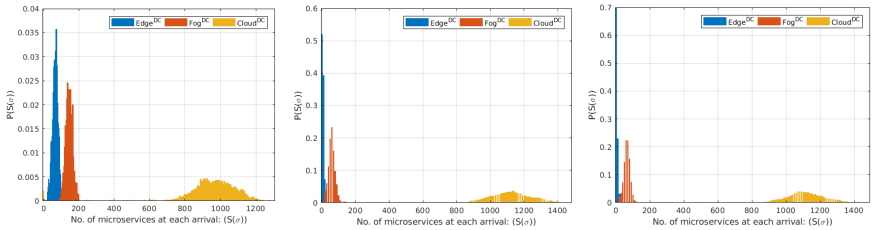
(a) Greedy First Fit (GFF) (b) Greedy Best Fit (GBF) without migration (c) GBF with Migration

Fig. 2. Latency of (Large versus Small) Services for 50 number of exchange b/w user & microservice



(a) First Fit (b) Best fit (c) Migration

Fig. 3. Placement of Small Microservices on Different Layers



(a) First Fit (b) Best fit (c) Migration

Fig. 4. Placement of Large Microservices on Different Layers

more microservices migrate near the end user while less-active microservices moves on upper layer to make the space for actively communicating microservices.

VII. Conclusion

This work aims at improving the placement of chains of microservices in terms of load and end-to-end latency using a migration approach. The proposed heuristic algorithms consider distributed data-centers along with the ephemeral nature of containerized services. The simulation based evaluation shows that migration performs better and reduces the latency in comparison with static placement (e.g., GBF and GFF considered in this paper). This work may be extended by using deep-learning approach through our future work.

References

[1] Y. Hu, M. Song, and T. Li, "Towards full containerization in containerized network function virtualization," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017.

[2] N. Alliance, "Cloud native enabling future telco platforms," 2021.

[3] K. Kaur, F. Guillemin, V. Q. Rodriguez, and F. Sailhan, "Latency and network aware placement for cloud-native 5g/6g services," in *IEEE Annual Consumer Communications & Networking Conference*, 2022.

[4] K. Kaur, F. Guillemin, and F. Sailhan, "Container placement and migration strategies for cloud, fog, and edge data centers: A survey," *International Journal of Network Management*, vol. 32, no. 6, 2022.

[5] H. Hawilo, M. Jammal, and A. Shami, "Orchestrating network function virtualization platform: Migration or re-instantiation?" in *IEEE International Conference on Cloud Networking (CloudNet)*, 2017.

[6] S. Aleyadeh, A. Moubayed, P. Heidari, and al., "Optimal container migration/re-instantiation in hybrid computing environments," *IEEE Open Journal of the Communications Society*, vol. 3, 2022.

[7] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration modeling and learning algorithms for containers in fog computing," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 712–725, 2018.

[8] L. Lv, Y. Zhang, Y. Li, and al., "Communication-aware container placement and reassignment in large-scale internet data centers," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, 2019.

[9] R. Zhou, Z. Li, and C. Wu, "An efficient online placement scheme for cloud container clusters," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1046–1058, 2019.

[10] U. Pongsakorn, Y. Watashiba, K. Ichikawa, and al., "Container rebalancing: Towards proactive linux containers placement optimization in a data center," in *IEEE Annual Computer Software and Applications Conference (COMPSAC)*, 2017.

[11] J. C. Cisneros, S. Yangui, and al., "Coordination algorithm for migration of shared vnfs in federated environments," in *IEEE NetSoft*, 2020.

[12] S. Maheshwari, S. Choudhury, I. Seskar, and al., "Traffic-aware dynamic container migration for real-time support in mobile edge clouds," in *IEEE ANTS*, 2018.

[13] F. P. Kelly, "Loss networks," *The Annals of Applied Probability*, vol. 1, no. 3, pp. 319–378, 1991.