RAP-G: Reliability-aware service placement using genetic algorithm for deep edge computing

Abdellah Kaci, Soraya Aït-Chellouche, Yassine Hadjadj-Aoul, Miloud Bagaa

▶ **To cite this version:**

Abdellah Kaci, Soraya Aït-Chellouche, Yassine Hadjadj-Aoul, Miloud Bagaa. RAP-G: Reliability-aware service placement using genetic algorithm for deep edge computing. CCNC 2023 - IEEE 20th Consumer Communications & Networking Conference, Jan 2023, Las Vegas, United States. pp.255-260, 10.1109/CCNC51644.2023.10060108 . hal-04368569

HAL Id: hal-04368569
https://inria.hal.science/hal-04368569

Submitted on 1 Jan 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

RAP-G: Reliability-aware service placement using genetic algorithm for deep edge computing

Abdellah KACI*, Soraya Ait-Chellouche*, Yassine Hadjadj-Aoul*, Miloud Bagaa†
* University of Rennes 1, IRISA Lab., INRIA, France
{abdellah.kaci,soraya.ait-chellouche,yassine.hadjadj-aoul}@irisa.fr
‡ Department of Electrical and Computer Engineering, Université du Québec
à Trois-Rivières, Trois-Rivières, QC, Canada. miloud.bagaa@uqtr.ca

***Abstract*—To ensure low latency, service providers are increasingly turning to edge computing, pushing services and resources from the Cloud to the Edge of the network, as close as possible to users. However, since video and image processing applications are particularly computationally intensive, their deployment is typically based on distributed provisioning between the Edge and the Cloud, which can increase the risk of failure when relying on unreliable networks. In this work, we proposed the algorithm RAP-G (Reliability-Aware service Placement with Genetics), which considers the reliability of network links and distributes services between the Cloud and the Edge using a genetic algorithm (GA). We have also developed a new variant of the first-fit algorithm called RF2 (Reliability-Aware First-Fit) that considers reliability within a reasonable time. The performance of the RAP-G algorithm was evaluated and compared with the RF2 algorithm. The experimental results show the importance of considering reliability in service delivery and the superiority of RAP-G.**

***Index Terms*—Edge Computing, Artificial Intelligence, Ultra-Reliable Low Latency Communications, Service Orchestration**

I. Introduction

The wide adoption of Cloud Computing and the Internet of Things (IoT) by industries leads to the emergence of the edge computing paradigm due to its advantages. Placing the resources at the user's edge reduces the end-to-end latency and boosts the local bandwidth utilization allowing better network bandwidth. Processing the video content in the cloud negatively impacts the end-to-end delay and bandwidth due to the tremendous data that should be transferred between the data producer and consumer [1]. To save the bandwidth and reduce the latency further, a new concept named Deep Edge Computing [2] is advised for processing the generated data at the source, allowing fast decision-making. In addition, new lightweight deep learning models have been employed at the deep edge for processing the video content [3]. However, these solutions focus more on reducing the complexity of deep learning techniques.

Deep Edge Computing (DEC) has limited and scarce resource capacity that should be optimally used by placing services according to their requirements and preventing resource overflow. For this reason, smart service orchestration has been widely investigated for placing and relocating services between DEC and the central cloud. Smart service orchestration relies on 5G Ultra-Reliable Low-Latency Communication (URLLC) and a reliable service placement distributed on different levels: Cloud, Edge, and Deep Edge. In order to enhance the latency and ensure continuity of service in offline mode, we have suggested two efficient solutions for placing the services at DEC to minimize resource consumption and end-to-end latency, and enhance bandwidth utilization and network reliability. In the present work, we aim to propose an efficient and smart service placement solution. To achieve that, we propose a first-fit solution that ensures an optimal service placement regarding nodes requirements (processor and memory) and links requirement (bandwidth and reliability). In addition, we advise a novel genetic service placement technique that considers the reliability in addition to nodes and links requirements. The proposed solutions are designed to enhance the latency and ensure continuity of service in offline mode in DEC. The main contributions of the present work are:

- Design of solutions that minimize the resources consumption and maximize link reliability;
- Enhance the latency and ensure continuity of service in offline mode in deep edge computing;
- Suggest a generic architecture for deep edge computing;
- Propose a novel solution for reliability-aware service placement named RF2 (Reliability-Aware First Fit);
- Develop a new efficient genetic service placement technique, named RAP-G (Reliability-Aware service Placement with Genetic algorithm);
- Concept and Develop RASim (Reliability-Aware Simulator) to evaluate the performances of proposed solutions.

The remaining of the paper is organized as follows. First, in section II, we discuss related works and present the generic deep edge computing architecture in section III. Then, in section V-B, we define the proposed reliability-aware service placement solution RF2 (Reliability-Aware First Fit), followed by the novel genetic service placement technique, RAP-G (Reliability-Aware service Placement with Genetic algorithm) in section IV. In section V the performances of the two solutions are studied. Finally, in section VI, we conclude our work and present some future works.

II. Related works

Virtual Network Embedding (VNE) is a key research issue in the field of network virtualization. To sum up, a virtual network consists on a set of virtual nodes and links that, depending on the provided services, requires a certain amount of computational and networking resources. VNE consists then on a node-to-node and link-to-path mapping from the virtual network to the substrate physical network considering the latter capacities.

VNE has been extensively studied for more than a decade. In a survey [4] Fischer et al. list near a hundred of VNE algorithms. Most of works commonly modeled the VNE problem as an optimization problem considering mainly the substrate network resources' utilization (CPU, storage, bandwidth, etc.). VNE problem is NP-hard, [5] [6] and the proposed algorithms in literature could then be classified in three categories: exact [7], heuristic [8] and meta-heuristic-based algorithms. The exact solutions aim, first of all, to propose optimal VNE for small network instances but also to serve as reference to evaluate heuristic and meta-heuristic solutions.

In [9], Jang et al. formulated the Service Function placement as an MILP problem and proposed a polynomial time algorithm based on linear relaxation and rounding to approximate the optimal solution of the MILP to increase the service capacity and the acceptable flow rate. In [10], Zhong et al. investigated the orchestration of SFCs across multiple data centers with the goal to minimize the overall cost. An ILP model was formulated and a meta-heuristic algorithm. Authors in [11] study the joint problem of service function chain deploying and path selection for bandwidth saving and VNF reuse. They described the problem as a multi-objective and multi-restriction problem, and then a heuristic service function chain deployment algorithm based on longest function. Bagaa et al. in [12] proposed an optimized orchestration framework that considers both QoS and network security levels to ensure trusted deployment. In [13] the authors encode physical and virtual networks as images, which are then perceivable by a convolutional deep neural network. Li et al., in [14], proposed a service chain mapping algorithm based on reinforcement learning to reduce the average link delay and improve the load balancing. The proposed algorithm place each VNF by considering the network status and the feedback function value. In [15], Fu et al. proposed a deep reinforcement learning-based SFC embedding scheme in NFV-enabled IoT in which complex VNFs are decomposed into smaller virtual network function components (VNFCs) to make more effective decisions. These works differ from each other in term of the considered resources by the optimization algorithm and the targeted objectives (cost, network usage, load balancing, etc.).

Leveraging the edge resources can efficiently elevate meeting the increasingly constrained requirements of IoT services, especially in terms of delay, traffic volume or continuity of service [16]. However, shifting computing to the network edge induces new challenges. Indeed, at the Edge, resources are usually limited and heterogeneous especially when dealing with mobile resources. In such a context, reliability becomes a major issue to consider when placing network services, especially when literature has already investigated the reliability estimation as in [17]. Motivated by it, this paper will design a reliability-aware VNE that take in consideration the heterogeneity and scarceness of resources at the edge.

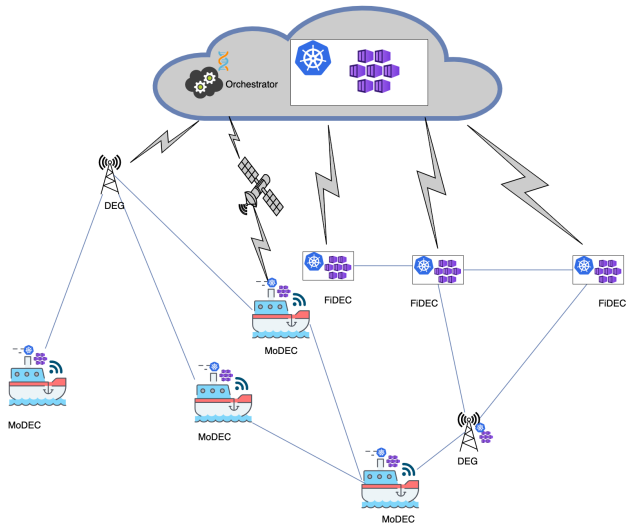

Fig. 1. The deep edge computing framework.

III. Generic architecture for Deep Edge Computing

In the present section, we present the proposed deep edge computing architecture and expose the reliability-aware service placement problem. As shown in Fig. 1, the deep edge architecture is composed of sensors (e.g., cameras and position sensors), a collection of geographically dispersed DEC clusters, a set of Deep Edge Gateways (DEGs) to ensure connectivity and Cloud service.

Undoubtedly, the edge is characterized by limited available resources, as well as connections and links among them are mostly wireless and/or unreliable. Therefore, since some services in deep edge computing should be placed at the edge level to ensure the required service level agreement (SLA), the service placement should consider resource limitations and link reliability when placing those services. Furthermore, special attention should be given to services that reqiure high reliability when placing and orchestrating them.

The DEC clusters are the backbone of the deep edge system (DES). They host virtual network functions (VNFs) that form service function chains (SFCs) corresponding to network requests. DES consists of two types of DEC clusters. First, Mobile DEC clusters (MoDEC clusters) could be vehicles, smartphones, boats, or any entity that can move. Meanwhile, the second type of FiDEC clusters (Fixed DEC clusters) is fixed and represents On-Site DEC clusters hosted in fixed geographical locations. For example, they could be located in the 5G/B5G core network of an MNO (Mobile Network Operators) or in the data centers of the service providers and their partners.

The DEGs ensure connectivity between MoDECs, FiDECs, and the central Cloud service. To ensure the quality of service (QoS), the DEGs could augment the network with extra network functions. Moreover, either a mesh network or satellite communication could be employed to ensure communication among MoDECs. While we aim for the former, the latter communication type could be used as a last resort. We have suggested a closed loop orchestration system of DeepEdge clusters (FiDECs and MoDECs) to ensure the desired quality of experience (QoE). Furthermore, the cloud orchestration system pushs the execution of greedy resource services toward central clouds to alleviate the overhead on DEGs. For instance, it is more relevant to train the Deep Learning models offline at the central cloud level while propelling the prediction toward the DEGs level.

We consider the set of fixed DEC nodes (Φ), the set of mobile DEC nodes (Ψ), and the set of nodes in DEG Getaways (Υ) that provide resources for Deep Edge services. The set Δ represents the set of all nodes in the Deep Edge clusters. In other terms, Δ represents the substrate network where virtual network functions could be hosted:

$$\Delta = \{n/n \in \Phi \cup \Psi \cup \Upsilon\} \tag{1}$$

In the reliability-aware service placement problem, we focus on the connectivity between the nodes of the substrate network (the elements of Δ). The reliability of the substrate network's physical links varies due to the different connection technologies and components used to connect nodes among the different clusters. In addition, other link properties (such as the bandwidth) are considered to ensure the required QoS. The substrate network is represented a graph $G_s = (\Delta, \xi, \Omega)$. The set ξ represents the set of physical links that connect the nodes of the set Δ. Ω represents the set of links properties (in this work, we consider two properties: bandwidth (w) and reliability (l)). The variable $U^l(n_i, n_j)$ represents the reliability of the physical link connecting the node n_i to the node n_j. Likewise, the variable $U^w(n_i, n_j)$ represents the available bandwidth in the physical link connecting node n_i to node n_j. In the case where n_i and n_j are not directly connected in the substrate network, then $U^l(n_i, n_j) = U^w(n_i, n_j) = 0$.

The virtual network requests (VNRs) are represented through a directed graph $G_v = (\Pi, \Theta, \Lambda)$. The set Π symbolizes the set of all virtual network functions (VNFs) belonging to all the VNRs should be placed. Θ conveys the set of all virtual links in the VNRs to be placed. The placement requirements for the links are specified in Λ. We consider the bandwidth requirement (w) and the reliability (r) of the virtual link. Thus, the variable $R^l(v_i, v_j)$ represents the reliability required by the physical path to place the virtual link connecting the virtual network function v_i to the virtual network function n_j. Correspondingly, the variable $R^w(v_i, v_j)$ denotes the required available bandwidth in the physical path embedding the virtual link connecting the virtual network function v_i to the virtual network function v_j.

VLS_ARRAY

	src	dst	cpu_{src}	cpu_{dst}	bw	rel_sla	vnr	ram_{src}	ram_{dst}	
vl_1	f_1	f_2	1	2	20	99.99	1	1024	512	vnr1
vl_2	f_2	f_3	2	1	20	99.99	1	512	1024	
vl_3	f_3	f_4	1	4	20	99,90	1	1024	2048	
vl_4	f_2	f_4	2	4	20	99.10	1	512	2048	
vl_5	f_5	f_6	1	2	10	98	2	1024	1024	vnr2
vl_6	f_6	f_7	2	1	30	96	2	1024	512	
vl_7	f_8	f_9	1	8	20	98.79	3	1024	1024	vnr3
vl_8	f_9	f_{10}	8	2	20	93	3	1024	1024	
vl_9	f_9	f_{11}	8	4	20	99.99	3	1024	1024	
vl_{10}	f_{11}	f_{10}	4	2	20	99.99	3	1024	1024	
vl_{11}	f_{10}	f_{12}	2	1	40	95.79	3	1024	2048	

Fig. 2. The structure of the vls_array for virtual links requirements

IV. GA-BASED RELIABILITY-AWARE PLACEMENT

In this section, we present the proposed Genetic Algorithm-based Reliability-Aware Placement solution. In the latter, we consider both the substrate network graph (SNG) properties and the virtual network requests (VNRs) requirements in term of Service Level Agreement.

The proposed RAP-G technique relies on the VLS_ARRAY shown in Fig. 2, where the requirements and information about different virtual network requests are stored. The VLS_ARRAY stores for each virtual link, the identifiers of its extremities (source and destination VNFs), the resources (CPU, memory, etc.) required of the VNFs, and the required link SLA (reliability, bandwidth, etc.). Furthermore, for each virtual link, the VLS_ARRAY mentions the identifier of its VNR. The VLS_ARRAY is extracted from the graph corresponding to the virtual network requests (Fig. 3). Let consider the three virtual network requests: vnr1 (blue), vnr2 (red), and vnr3 (green). Each virtual network request consists of a set of VNFs and virtual links. In the VLS_ARRAY, each row corresponds to a virtual link and contains its SLA requirements (required bandwidth and required reliability), the identifiers of its extremities (source and destination), the requirements of each extremity (in terms of the required number of processors and memory amount). In addition, for each virtual link, VLS_ARRAY specifies to which virtual network request it belongs.

The RAP-G technique maximizes the number of satisfied requests using the genetic algorithm. The genome of RAP-G defines the physical node in the substrate network for each virtual network function, where it will be placed (Fig. 4). The RAP-G technique uses an algorithm generated through several generations to efficiently generate a service placement that maximizes the number of placed services while considering link reliability, bandwidth, and VNFs requirements. In the latter, we mainly consider the number of processors and the amount of memory. According to the deployment context, the RAP-G could be extended to other properties (with tiny modifications) to account for other properties, such as disk storage, link delay, etc.

Let be n the number of VNFs in the VNRs graph. The genetic algorithm RAP-G runs in multiple steps. First, RAP-G generates an initial population Pop_0 of size m. Then, for each

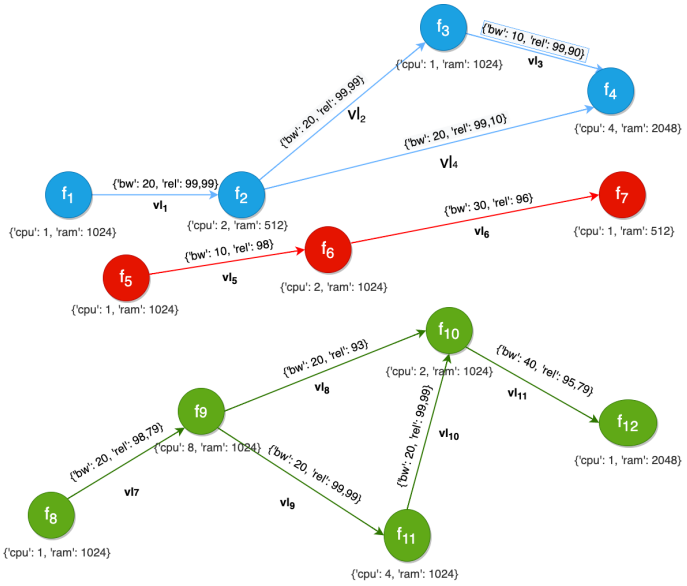

Fig. 3. The graph of virtual network requests.

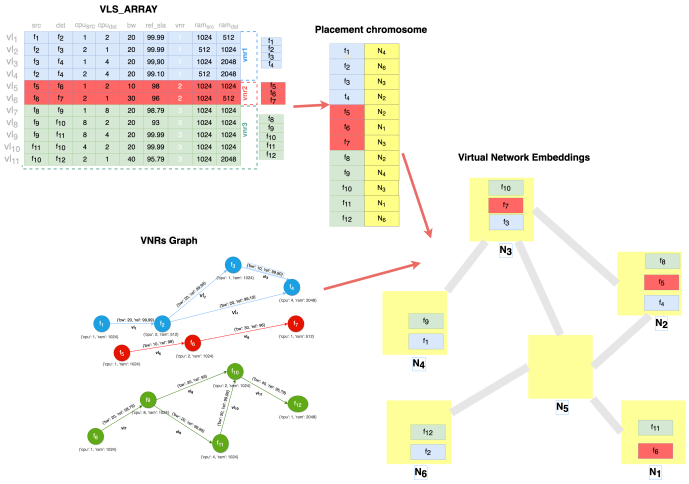

Fig. 4. The genetic based placement.

chromosome $i \in Pop_0$ in the population, RAP-G specifies for each VNF f_j the physical node from the substrate network where it should be placed ($j \in 1, 2, ..., n$). To enhance the reliability of the placement and ensure high availability, RAP-G places the VNFs of the same service (virtual network request) in different physical nodes. However, the placement can host functions from different services. Indeed, as shown in the placement chromosome of Fig. 4, RAP-G placed in each physical node N_k many VNFs belonging to different services.

From a generation t to a generation $t+1$, RAP-G first uses the fitness value of each solution in the population Pop_k to select a subset using the Top Rank method. Then, in the cross-over phase, for every two parents, A and B (in Top Rank, the fitness value of parent A is greater or equal to the fitness value of B). Then, according to a Bernoulli law $\beta(p, n)$ of a success probability p, the son chromosome will copy the placement of the functions of each service vnr_s with a probability of p. In the other case, the service will follow the placement of

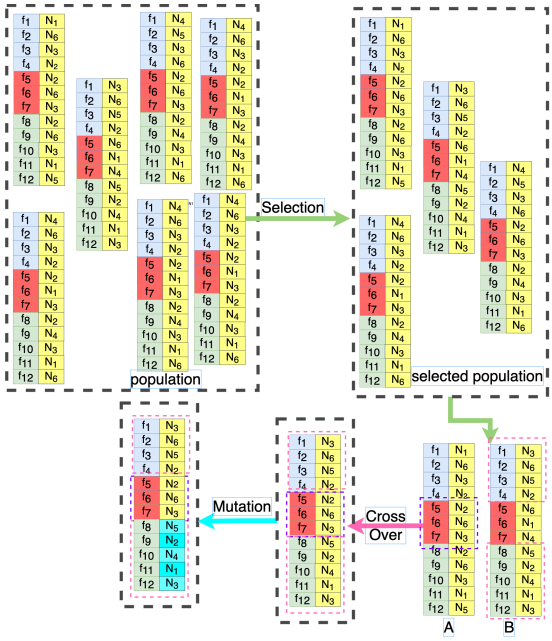

Fig. 5. The genetic algorithm RAP-G.

parent B. For instance, after the cross-over, as depicted in Fig. 5, the placement of the second service is copied from parent A. Meanwhile, the placement of the services vnr_1 and vnr_3 are extracted from parent B with success probability assumed to be p = 1/3.

For the mutation phase, RAP-G mutates a certain number of services (corresponding to a mutation rate), such that the mutation is done at the level of the service and not at the virtual network functions levels. Indeed, for each muted service, RAP-G will generate a new random placement that respects the constraint that the VNFs of the same service are placed in different physical nodes from the substrate network. For instance, in the mutation phase of Fig. 5, the VNFs of the service vnr_3 are mutated and placed in different locations.

V. Experimental Study

In the present section, we evaluate the performance of the proposed reliability-aware service placement technique RAP-G. In the balance of this section, we will start by describing the experimental setup used in the performance evaluation. Then, in section V-B, we present the proposed solution RF2 (Reliability-Aware First Fit), which is a first fit algorithm that considers link reliability. Last but not least, we present and discuss the experimental evaluation results.

A. Experimental setup

To evaluate the performance of the proposed techniques, we developed a simulator anointed RASim (Reliability-Aware Simulator). The simulator uses a JSON configuration file that specifies the substrate network and the virtual network specifications. For instance, in the substrate network, we specify the node characteristics, such as processors and RAM amounts, the physical link properties, such as Bandwidth and failure probability, and the network topology (i.e., file name).

RASim simulator generates a graph with the specified network topology and properties based on the earlier information. Likewise, the RASim simulator will use the virtual network specifications to generate its corresponding graph, simulating several virtual network requests with specified requirements for VNFs and links.

B. Reliability-Aware First Fit

The proposed RF2 (Reliability-Aware First Fit) solution considers the substrate network graph (SNG), the set of virtual network requests (VNRs), and the reliability SLA threshold (slaThreshold), which specifies the required service reliability. Algorithm 1 describes the RF2 placement. First, RF2 extracts the properties of the nodes of the substrate network. Then, RF2 tries to satisfy all the requests present in the VNRs requests on a one-by-one basis.

For each request i, RF2 extracts the set of VNFs described in the request (VNFsToPlace). For each VNF j in the set VNFsToPlace, RF2 finds the node with enough resources to host the requirement of the VNF j. If one VNF from the set VNFsToPlace could not be placed, the full service i is not placed.

If all the VNFs of the request i could be satisfied (VNFsToPlace = 0), the RF2 technique checks the requirements of the virtual link. The request i is satisfied only all its virtual links could be placed. In this case, the resource allocation is processed for all the VNFs and virtual links within the request i.

Algorithm 1: RF2

```
Input: SNG: Graph, VNRs: Array, slaThreshold: Number
BEGIN
nodesProps = extractNodesProperties(SNG)//CPU and RAM
for i = 1 To |NumReq| do
    VNFsToPlace = VNRs[i].getVNFs()
    for each j in VNFsToPlace do
        PlacementSol[j] = (NULL, (0, 0) )
    end for
    for each j in VNFsToPlace do
        for each k in nodesProps.keys() do
            if nodesProps[k].satisfies(VNRs[i].getReq(j) then
                PlacementSol[j] = (k, (plCpu, plRam) )
                VNFsToPlace = VNFsToPlace - 1
                nodesProps.exclude(k)
                break
            end if
        end for
    end for
    if VNFsToPlace = 0 then
        placementVLs = False
        isVLsPlaced = placeVLs(VNRs[i], slaThreshold)
        if isVLsPlaced then
            nbPlacements = nbPlacements + 1
            process(PlacementSol[j])
        end if
    end if
end for
END
```

C. Experimental results

Based on the two graphs (representing the substrate network and the virtual network), the RASim simulator will execute the placement algorithms (RAP-G and RF2) according to the desired setup. We first studied and compared the acceptance rate of the two techniques RAP-G and RF2. The results are shown in Fig. 6 and Fig. 7. In these experiments, we considered 30 generations and populations of 100 chromosomes.

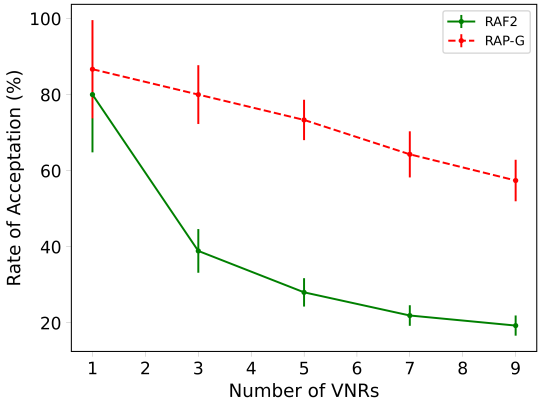

Fig. 6. Comparative study.

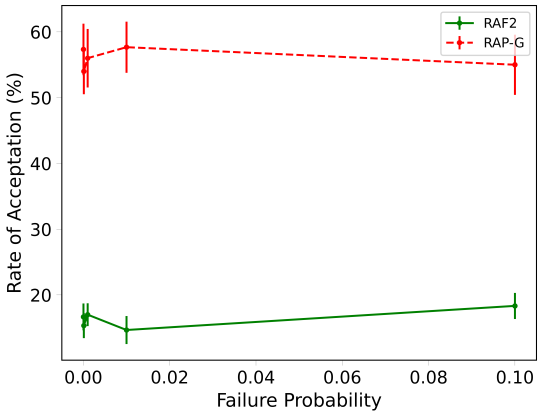

Fig. 7. Failure probability effect.

In the second step, we studied the behavior of the RAP-G technique considering the acceptance rate and the execution time. According to the number of generations, the evolution of the acceptance rate and the execution time of the RAP-G technique are shown in Fig. 8 and 9, respectively. In addition, the evolution of the execution time is according to the number of virtual network requests.

As shown in Fig. 6, the acceptation rate of the proposed RAP-G technique is better than the first fit RF2 technique. In addition, with the augmentation of the number of the requested services, the acceptance rate decreases due to network saturation. However, the proposed RAP-G technique is better than the first fit RF2 one. Considering ten service requests and different failure probabilities for the substrate network, Fig. 7 shows the ability of RAP-G to deal with the failures while having a higher acceptance rate.

The RAP-G technique enhances the quality of the placements when the number of generations and the population size increase (Fig. 8). However, the execution times increase with the augmentation of the number of generations, the population

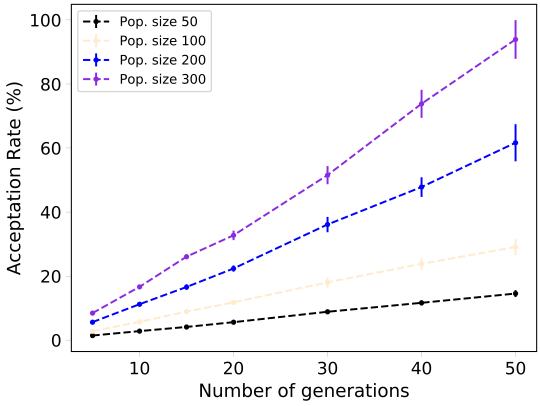

Fig. 8. Acceptation rate evolution.

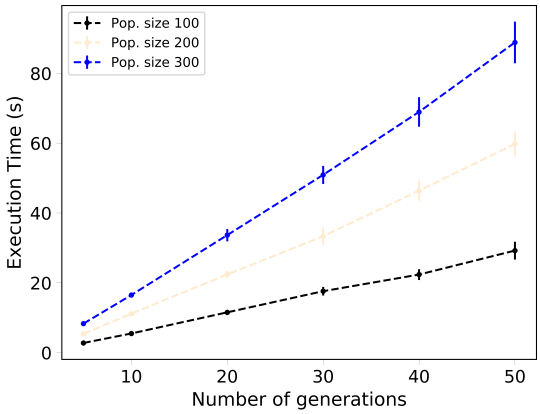

Fig. 9. Execution time evolution.

size, and the number of requests, as shown in Fig. 9 and 10, respectively.

VI. Conclusion and Future Works

In the present work, we suggested RAP-G, a novel genetic algorithm for reliable distributed service orchestration to reduce the latency in Deep Edge Computing. The proposed RAP-G algorithm was compared to a variant of the First Fit Algorithm, namely RF2 (Reliability-Aware First Fit). Experimental results demonstrate that RAP-G outperforms the RF2 algorithm. In future work, we will use Machine Learning to ensure dynamic service placement.

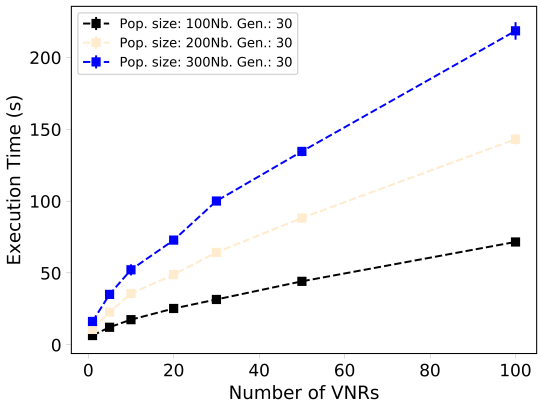

Fig. 10. Execution time and problem size.

VII. Acknowledgment

This research work is conducted as part of the NaviDEC project, funded by the Britany Region, France, under grant agreement #21004179.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] J.-H. Kim, N. Kim, and C. S. Won, "Deep edge computing for videos," *IEEE Access*, vol. 9, pp. 123348–123357, 2021.

[3] S. Y. Nikouei, Y. Chen, S. Song, R. Xu, B.-Y. Choi, and T. R. Faughnan, "Real-time human detection as an edge service enabled by a lightweight cnn," in *2018 IEEE International Conference on Edge Computing (EDGE)*, pp. 125–129, IEEE, 2018.

[4] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[5] M. Rost and S. Schmid, "Charting the complexity landscape of virtual network embeddings," in *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, pp. 1–9, 2018.

[6] E. Amaldi, S. Coniglio, A. M. Koster, and M. Tieves, "On the computational complexity of the virtual network embedding problem," *Electronic Notes in Discrete Mathematics*, vol. 52, pp. 213–220, 2016. INOC 2015 – 7th International Network Optimization Conference.

[7] H. Cao, L. Yang, Z. Liu, and M. Wu, "Exact solutions of vne: A survey," *China Communications*, vol. 13, no. 6, pp. 48–62, 2016.

[8] H. Cao, H. Hu, Z. Qu, and L. Yang, "Heuristic solutions of virtual network embedding: A survey," *China Communications*, vol. 15, no. 3, pp. 186–219, 2018.

[9] I. Jang, D. Suh, S. Pack, and G. Dán, "Joint optimization of service function placement and flow distribution for service function chaining," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2532–2541, 2017.

[10] X. Zhong, Y. Wang, and X. Qiu, "Service function chain orchestration across multiple clouds," *China Communications*, vol. 15, no. 10, pp. 99–116, 2018.

[11] D. Li, J. Lan, and P. Wang, "Joint service function chain deploying and path selection for bandwidth saving and vnf reuse," *International Journal of Communication Systems*, vol. 31, no. 6, p. e3523, 2018.

[12] M. Bagaa, T. Taleb, J. B. Bernabe, and A. Skarmeta, "Qos and resource-aware security orchestration and life cycle management," *IEEE Transactions on Mobile Computing*, 2020.

[13] M. Dolati, S. B. Hassanpour, M. Ghaderi, and A. Khonsari, "Deepvine: Virtual network embedding with deep reinforcement learning," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 879–885, IEEE, 2019.

[14] W. Li, H. Wu, C. Jiang, P. Jia, N. Li, and P. Lin, "Service chain mapping algorithm based on reinforcement learning," in *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pp. 800–805, 2020.

[15] X. Fu, F. R. Yu, J. Wang, Q. Qi, and J. Liao, "Service function chain embedding for nfv-enabled iot based on deep reinforcement learning," *IEEE Communications Magazine*, vol. 57, no. 11, pp. 102–108, 2019.

[16] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[17] P. L'Ecuyer, G. Rubino, S. Saggadi, and B. Tuffin, "Approximate zero-variance importance sampling for static network reliability estimation," *IEEE Transactions on Reliability*, vol. 60, no. 3, pp. 590–604, 2011.