MEC Resource Orchestration for Heterogeneous Networks and Services Using Reinforcement Learning

Shreya K. Chari*, Luis A. Garrido*, John S. Vardakas*†, Kostas Ramantas*, Christos Verikoukis‡§
* Iquadrat Informatica S.L., Barcelona, Spain
† Department of Informatics, University of Western Macedonia, Greece
‡ University of Patras, Patras, Greece
§ Industrial Systems Institute (ISI) | Athena Research Center, Patras, Greece
*{s.chari, l.garrido, kramantas}@iquadrat.com, †ivardakas@uowm.gr, ‡cveri@ceid.upatras.gr

***Abstract*—Rapidly changing wireless network paradigms and technologies that can serve diverse applications and services have become the focal point. Multi-access Edge Computing (MEC) offers promising solutions to mitigate the complications created by such dense networks within the Management and Orchestration (MANO) frameworks. The MEC is expected to handle demands from different services that have heterogeneous performance constraints. Reinforcement Learning (RL) technique is used to solve the resource orchestration and allocation tasks at the MEC in this paper. Soft Actor-Critic (SAC) is used as the RL agent to learn the local network resource orchestration model and is further used for resource allocation also. Network resources such as CPU and memory are considered within the model for meeting the required latency of heterogeneous network services. The simulation results prove that the SAC based resource orchestrator gives a better performance in comparison to the random service orchestrator. The prospects of using such technique for multitude of use cases in Future Wireless Networks (FWN) appears to be indisputable.**

***Index Terms*—Beyond 5G, MEC, Resource orchestration, Heterogeneous networks, Reinforcement Learning (RL), Soft Actor-Critic (SAC)**

I. Introduction

Future Wireless Networks (FWN) would be serving heterogeneous services with different resource requirements. FWNs should support multiple operational standards for exploitation of expected network heterogeneity which is evident due to different types of Base Stations (BSs), User's Equipment (UE) and associated traffic variability. In such a scenario, MEC at the edge of the cellular network infrastructure offers promising solutions. The sophisticated MEC architecture is also responsible for orchestration of other essential services such as computation offloading and caching.

Dynamic nature of the heterogeneous services with different performance constraints at the network edge has made the problem of task offloading with the given resources non-trivial [1]. Several attempts have been made to solve the problem of this nature using closed-form solutions and AI. Lyapunov optimization techniques were utilized in [2] to maximize average revenue of the operator while considering different types of slices. A distributed learning framework is proposed for making the offloading decisions in beyond 5G networks in [3]. Resource coupling in resource orchestration has been considered as a key challenge by the authors in [4] and is solved using joint-network slice instantiation. Few hybrid models as proposed in [5] and as in [6] has shown significant improvement in slicing involving the Random Access Network (RAN) level. Further, federated learning-based training of RL agents is leveraged in [7] and [8] to optimally allocate communication resources. The authors of [9] use a deep Q-learning-based offloading approach for vehicular networks for resource orchestration. The work in [10] demonstrates deployment of on demand emergency network slices after redistributing its resources using deep learning and long short-term memory networks. A combination of discrete and continuous actor-critic model has also been attempted to optimize radio resources in [11] and proved to outperform existing benchmarks. On the other hand, the research work in [12] and [13] tackles the problem of computation offloading by breaking them into smaller subtasks involving RL algorithms.

However, several key issues have not yet been addressed in the state-of-the-art literature. Computational load generated from heterogeneous services associated with the end users, MEC or any other additional services is not considered together. A problem formulation that could allow a more flexible solution by traversing through different network services for 5G/6G use cases is required. With the aforementioned observations, in this paper we propose a RL solution that solves these issues by establishing relative utility among the network services thus allowing the orchestration strategy to flexibly change its preferences depending on the state of the MEC. The strategy also takes into account both elastic and inelastic network requirements by not relying on a rigid inflexible system model. The following contributions were made addressing the aforementioned limitations within the state-of-the-art literature:

- This paper explores an intelligent resource orchestration strategy at the MEC using RL. The strategy thus accommodates resource constraints from different network

services without degrading overall network performance. The performance of the resource orchestration strategy is measured by observing the resource distribution amongst different services.

- The considered MEC infrastructure takes into account two of the most crucial resources, that is, *CPU* and *memory*. Within the infrastructure each network service is assumed to consume each of these resources.
- To drive resource orchestration decisions at the MEC, Soft Actor-Critic (SAC) agent has been utilized. A comparison with the baseline random service orchestrator is made. The SAC based resource orchestrator provides a better performance by attaining higher *usage efficiency* ratio which is close to 1 while adhering to the constraints in the formulation.
- The work accomplished in this paper facilitates further exploration within the field of resource orchestration at the MEC for heterogeneous service requirements.

The remainder of the paper is organized as follows. Section II describes the system model implemented for the purpose of simulating a network edge scenario and for conducting the experiments. Section III formulates the MEC resource orchestration as a RL problem. Section IV presents all the results and observations from the work. A comprehensive conclusion is arrived at in section V. Finally, in section VI of the paper future work is presented.

II. System Model

A network infrastructure consisting of a MEC server represented as S closely coordinating with the service demands put forth by the users at the RAN domain is considered while conducting other crucial functions as shown in Fig.1. The resources within S are categorized as a set of resource capacities consisting of $[CPU, memory]$. Within the infrastructure, N network slices are deployed that provide different network services to the end-users connected to the network. These network slices can have heterogeneous functional and performance constraints, the latter of which is expressed in the Service Level Agreements (SLA) as maximum permissible delay requirements. Network slices with distinct performance requirements in order to meet the SLA have been assumed from the state-of-the-art standards as mentioned in Table I [14, 15]. The individual resource capacities at S are known beforehand and all the network slices are presumed to be admitted into S.

Network delay is used as the Key Performance Indicator (KPI) for observing the SLA violations. The overall delay model is a combination of individual delay models based on *CPU* and *memory* resources. For the computational delay a simple model based on the CPU demand-allocation ratio as in (1) is used. In (1), β is the latency gain used for the purpose of regulation. The computational delay model can be perceived by analyzing the demand-allocation ratio. Assuming that the CPU demand is lesser than the maximum capacity at the server S and is higher than the allocated CPU, then the delay thus incurred will be higher and vice versa.

TABLE I
SLA Requirements

Type	Services	Delay (s)
I	*Critical medical application I*	0.02
II	*Cyber-physical systems*	0.01
III	*Critical medical application II*	0.1
IV	*Video, imaging and audio application*	0.00075

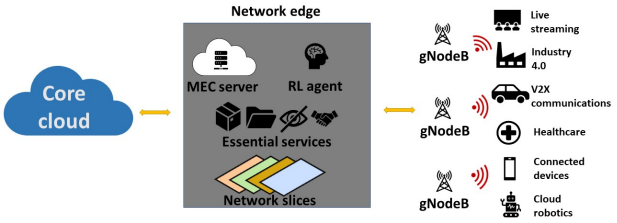

Fig. 1. System model

$$l_c = (\beta CPU_{demand})/CPU_{alloc} \tag{1}$$

Delay experienced due to memory utilization is modelled as a piece-wise equation as shown in (2) after referencing standard memory performance models [16]. There are two scenarios under consideration, (i) when the memory allocated is higher than or equal to the memory demanded (ii) when the memory allocated is lesser than the memory demanded. In the first scenario, the server has the memory capacity to meet the demands leading to a lower delay value and in the second scenario, an analogy inverse to the first scenario is observed. In (2), $l_{surplus}$ and l_{scarce} are assumed to be the minimum delays that are incurred in the two cases. ϕ is the assumed slope based on reasonable delay values suitable for this work.

$$l_m = \begin{cases} l_{surplus} + \phi l_{demand} & \text{if } mem_{dem} \leq mem_{alloc} \\ l_{scarce} + \phi l_{demand} & \text{if } mem_{dem} > mem_{alloc} \end{cases} \tag{2}$$

III. Reinforcement Learning For MEC Resource Orchestration

A. Problem formulation

In this section, the formulation for the research problem in this paper is presented. A MEC server S associated with n network slices belonging to the slice set $n \epsilon N$, with fixed computational C and memory M capacities is considered. Each slice $n \epsilon N$ puts forward resource requirements A_n for its operation in the form of *CPU* and *memory* represented as a resource tuple (c_n, m_n). Server S simultaneously considers the resource demands D_n consumed by $n \epsilon N$, which is also represented as a resource tuple consisting of *CPU* and *memory* demands from all the network slices in N and orchestrates the available resources established by the RL agent´s learning.

Thus, the MEC resource orchestration problem is posed

as a multidimensional problem of a known capacity R, in which each dimension is *CPU* and *memory* resources, respectively. Each of the network slices have a utility value represented by G_n affixed to it. Keeping in mind the specific needs of services from Table I, values for G_n are selected rigorously through experimentation. While doing so, it was also considered that some services relied more on *CPU* resources and some more on *memory*. The objective is to maximize the total utility by distributing the resources C and M amongst $n\epsilon N$ at different time windows as shown in (3).

$$U = \max \sum_{n=1}^{N} G_n^{[CPU,mem]} A_n^{[CPU,mem]}(t), \forall n\epsilon N \quad (3)$$

The variable G_n represents the gain attached with each resource type for each service, with values deduced through experimentation as listed in Table II. While achieving the objective, several constraints to ensure strict surveillance of the orchestration policy are added. Equation (4) limits the resource allocation based on the available resource capacity at the MEC. SLA violation is prevented with the indication given by the delay KPI as in (5). Equation (5) is able to monitor the SLA by keeping the probability of allocated resources to be smaller than the demanded resources D_n below a maximum acceptable value ϵ_n. Finally, (6) prevents over-provisioning by not allocating all the available resources to a particular service, m_n is the maximum acceptable value for resource allocations ratio.

$$\sum_{n=1}^{N} A_n^{[CPU,mem]} \leq R^{[CPU,mem]}, \forall n\epsilon N \quad (4)$$

$$Pr(A_n^{[CPU,mem]}(t) < D_n^{[CPU,mem]}(t)) \leq \varepsilon_n, \forall n\epsilon N \quad (5)$$

$$\frac{A_n^{[CPU,mem]}(t)}{D_n^{[CPU,mem]}(t)} \leq m_n, \forall n\epsilon N \quad (6)$$

B. RL formulation

In this section, resource orchestration problem is mapped into a Markovian Decision Process (MDP). The three required fundamental elements are the action space, the state space and the reward function as shown in Fig.2 and defined as follows:

- *State space:* The state space comprises of the maximum resource capacities of the server C and M, the allocated resources A_n to the slices $n\epsilon N$, the resource demand at every network slice D_n to the slices $n\epsilon N$ and delay l. It is reminded that each A_n and D_n are represented in terms of *CPU* and *memory* resources.

$$s_t = [C, M, D_1, .., D_N, A_1, .., A_N, l_1, .., l_N]$$

- *Action space:* The action space comprises of the set of the resources arbitrarily assigned to the network slices as learnt by the RL policy. Therefore, the action taken for each slice consists of predicted *CPU* and *memory* resources.

$$a_t = [c_1, m_1, ..., c_N, m_N]$$

- *Reward:* The reward guides the learning process of the agent by rigorously evaluating the actions taken at each time step. For this work, (7) is used to award the agent with rewards as explained below.

$$r(t) = \sum_{n=0}^{N} U_n^{CPU} f(\delta_{CPU}) + U_n^{mem} f(\delta_{mem}) \quad (7)$$

$$f(\delta_{resource}) = \begin{cases} 1 & \text{if } \delta_{resource} \geq 0 \\ 0 & \text{if } \delta_{resource} < 0 \end{cases} \quad (8)$$

Here, U_n^{CPU} and U_n^{mem} are the utility values associated with each resource type for each slice $n\epsilon N$. To calculate the utility values, the allocations are multiplied with gains per resource per slice i.e., $U_n^{CPU} = G_n^{CPU} c_n(t)$ and $U_n^{mem} = G_n^{mem} m_n(t)$. It is necessary to include this term in the reward to emphasize on the larger objective as introduced in (3). The reward $r(t)$ is further controlled by a function that depends on the difference δ_{CPU} and δ_{mem} between the allocations made by the agent and the actual resource demand.

Further, penalties are given if the KPI fails to perform as expected. Table I provides the requisite delay values for each network service useful for monitoring penalties. In (9), the penalty for SLA violations is considered by including the penalty loss term P^{delay}. Also, l_n is the delay at a given time step while l_{exp} is the expected delay for each slice type. Additionally, penalties for violating the constraints defined in the problem formulation are taken into account for driving the agent's policy learning. The final reward value is finally calculated as the difference between r_t and the total sum of all the aforementioned penalties.

$$p_{delay}(t) = \begin{cases} P_n^{delay}(l_n(t) - l_{exp}) & \text{if } l_n > l_{exp} \\ 0 & \text{if } l_n \leq l_{exp} \end{cases} \quad (9)$$

C. SAC driven learning

For continuous action space, such as in this paper, a mix of policy optimization and Q-learning approaches seems optimum. Hence, SAC that utilizes both approaches efficiently was implemented to model the orchestration strategy at the MEC server. It makes use of experience replay buffers and target networks to stabilize its learning. SAC uses two Q-functions but applies entropy regularization. Using entropy helps in regularizing the exploitation-exploration trade-off and policy convergence [17]. The objective function of the SAC is mathematically defined as in (10). $J(\theta)$ is maximised with θ representing the parameters of the policy. s_t and a_t are the state and action taken at a given timestep t. Immediate reward r gained at time step t is regulated by the discount factor γ while $H(\pi_\theta(.|s_t))$ is the entropy of the policy at state s_t regulated by the trade-off coefficient α.

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t (r(s_t, a_t) + \alpha H(\pi_\theta(.|s_t)] \quad (10)$$

TABLE II
SIMULATION PARAMETERS

Type	Services	G_{cpu}	G_{mem}
I	*Critical medical application I*	22950	38050
II	*Cyber-physical systems*	32000	40000
III	*Critical medical application II*	34200	42800
IV	*Video, imaging and audio application*	38450	23550

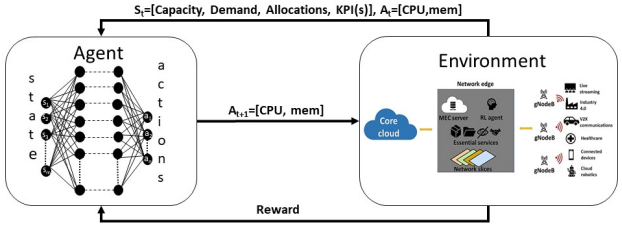

Fig. 2. RL system

D. Algorithm implementation

The system model as explained in Section II was implemented using Python 3.11 and Tensorflow 2.15 [18, 19]. The dataset required for training the RL orchestrator was acquired from the Google cluster usage-traces [20]. The SAC parameters used for the simulations are mentioned here on. There are 1024 sub-steps in each episode of the RL orchestrator algorithm implying that the RL agent interacts with the environment 1024 times before the complete environment is readjusted to the initial conditions. In addition, to avoid over-fitting, after 20 episodes the algorithm is consciously terminated early. The algorithm is trained with mini batches of size 64 while the learning rate of 0.0001 is utilized. Further, the discount factor as introduced previously is set at 0.90. All the parameters were determined through rigorous trials and tuning within the considered system model.

E. Dataset pre-processing

Google cluster usage-traces are queried to collect information that includes job scheduling and resource utilization observations from Google's enormous data clusters [20]. The data is stored in tables with names machine events, machine attributes, collection events, instance events and instance usage. From these tables, various essential parameters like resource usage, job priority, task allocation, and the nature of workloads can be extracted. The resource requests are differentiated as jobs or alloc sets. A job can be user requested collection of tasks while an alloc set is the reserved resource. Further, the Google cluster traces classifies task scheduling into 4 distinct categories, which serves our intention of recognizing 4 different types of services. Each record in the dataset is collected over a window of 5 minutes and supplies the needed service type alongside the utilization metrics for *CPU* and *memory* resources. The data is thus conveyed to the environment of the RL orchestrator, which further prepares the state space to be utilized by the RL agent.

IV. RESULTS & DISCUSSIONS

In this section, the simulation results are presented and discussed. For performance comparison, the RL-based orchestrator is measured against the random service orchestrator.

A. Random service orchestrator

A comparative analysis is conducted between the RL resource orchestrator algorithm and a random service orchestrator. The random service orchestrator reacts to varying user demands for the services immediately, while obeying the capacity limitations imposed by the MEC server. A service is selected at random within the random service orchestrator reusing the service gains from Table II serving as weights in the selection process. Services with higher gains are favored in this method. Additionally, resources in proportion to the demand are given to each service. The random service orchestrator does not use a reward-punish mechanism or an experience replay buffer as the RL orchestrator that can be used as feedback. Therefore, it perfectly serves the purpose of differentiating the performance between an intelligent AI backed method and a non-intelligent legacy network approach.

B. Simulation settings

An early stopping was performed for the simulations after 20 episodes. Resource allocation for *CPU* and *memory* is carried out respectively for each of the 4 network services. For the comparisons, resource usage efficiency i.e., *usage efficiency*=$\frac{demand}{allocations}$ for each resource type and for each service is calculated. A *usage efficiency* ratio approximately equivalent to 1 suggests that the resource allocations are closely following the resource demand. This shows that the resources within the MEC server are distributed efficiently without wastage. Usage efficiency is used as a metric to compare the performances as it establishes a relation between the demand and allocations made by the different algorithms.

Further, a moving average with a 1024-step window is employed after considerate experimentation. These moving averages were governed until the agent reached a point of learning saturation, after which an early stopping was applied. The results for the last 5 episodes with 1024 sub-steps each is presented against the steps in terms of efficiency percentages. A significant difference in performance is observed as explained in detail below due to the sophisticated nature of the RL-based orchestrator.

C. Observed results

In Fig.3 and Fig.4, it is clearly visible that the *CPU* resource usage efficiency achieved by the SAC based RL orchestrator for *service type I* and *service type II* which were introduced beforehand in Table I and Table II is outperforming the baseline random service orchestrator. Moreover, the allocations rely on resource utilization, implying that even a small over-allocation can significantly affect the availability of resources

at the server. Therefore, it is observed that the SAC based orchestrator is able to observe the network demands of the services more meticulously and is able to strike a comparatively superior balance between the resource demand and the allocations made. Conversely, the MEC server needs to manage its resources sensibly to ensure future user demand requests are met without compromising the fulfillment of current demands. A similar trend is observed with other service types where the RL orchestrator performs better than the random service orchestrator in judiciously allocating *CPU* resources while complying to all established constraints.

A near identical result is seen in Fig.5 and Fig.6 for *memory* allocations in *service type II* and *service type IV*. Indistinguishably, the RL orchestrator outshines the random orchestrator's performance. Following in the same direction, the other network services provide higher resource usage efficiency when the SAC based orchestrator is utilized. It can be undoubtedly asserted that the RL SAC agent is capable of managing various resource types, including both *CPU* and *memory*, while continuing to maintain high levels of resource utilization. This also confirms the initial assumptions that a network infrastructure with diverse requirements creates the need for RL based intelligent algorithms.

As per the proposed formulation, it is necessary that the RL SAC orchestrator algorithm is able to meet the KPI requirements for all the network slices. Fig.7 displays the maximum permissible delay for *service type III*. It is noticed that both the RL SAC orchestrator and the random service orchestrator successfully meet this objective. Such a phenomenon can be attributed to the low resource usage efficiency values for the random service orchestrator. This means that the resource allocations were probably much greater than the actual demand which indicates the possibility of unessential over-allocation followed by improper server usage. Similar outcomes are observed for the other service types as well resulting in similar conclusions.

D. Discussion

The results presented in the graphs showcases that the RL SAC orchestrator has a greater grasp of the resource

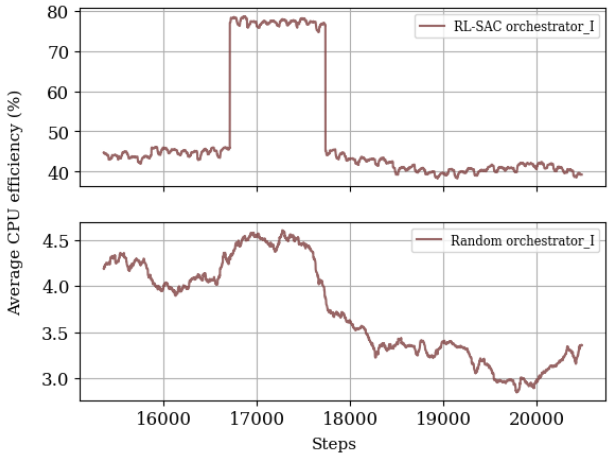

Fig. 3. Average CPU resource usage efficiency for *service type I*

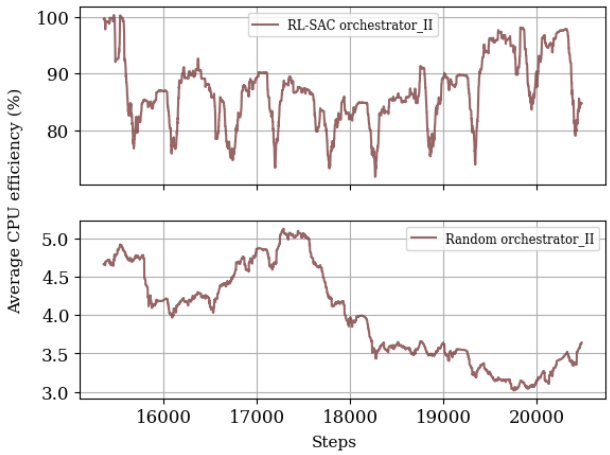

Fig. 4. Average CPU resource usage efficiency for *service type II*

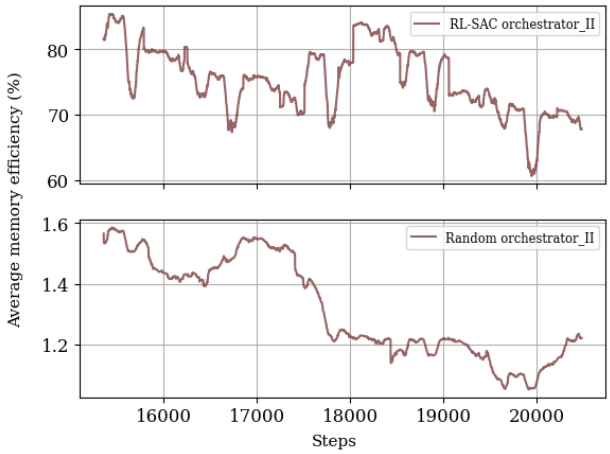

Fig. 5. Average memory resource usage efficiency for *service type II*

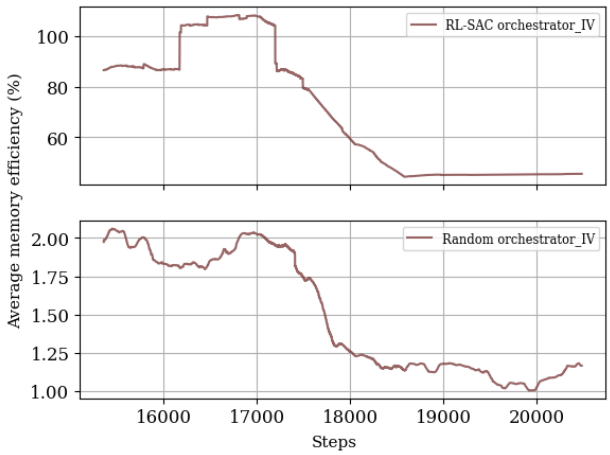

Fig. 6. Average memory resource usage efficiency for *service type IV*

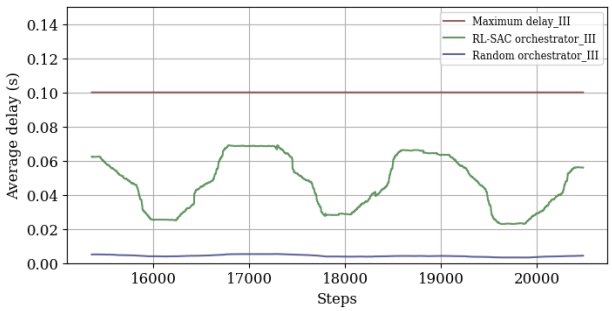

Fig. 7. Average delay for *service type III*

availability patterns at the MEC server resulting in highly efficient utilization of the resources. On the other hand, the random service orchestrator exhibits notable overutilization of the server resulting in unessential spending of resources. Further, it is anticipated that the random service orchestrator meets KPI requirements in a non-judicious way. The disparity in resource allocation can hinder the MEC orchestration abilities. Moreover, KPI(s) for the services need to be met in order to avoid degradation in the smooth functioning of the services. SLA violations are bound to occur as the number of associated network services increase. Therefore, a random orchestrator or any similar orchestration strategy is not suitable for a scalable solution.

V. Conclusions

To summarize, a SAC based RL resource orchestrator for resource management at the network edge is proposed and implemented within this work. The resource orchestration process involves network resources in the form of *CPU* and *memory* to be managed and allocated to multiple services with heterogeneous service requirements. The RL orchestrator and the random service orchestrator is then compared to highlight the outstanding performance of the RL resource orchestrator. The encouraging results prove that the MEC resource orchestration using RL is indeed possible and can be useful for beyond 5G applications.

VI. Future Work

For future work, reward functions based on other KPI(s) can be explored along with a distributed RL framework including several MEC servers. Additional resources such as storage, bandwidth or GPU may also be taken into consideration in the problem formulation. A comparison against other AI methods and a possibility of test bed integration can be attempted as well. Inclusion of more RAN components to extend the results to other problems such as admission control or scheduling.

VII. Acknowledgement

This research was funded by the AC3 (Agile and Cognitive Cloud-edge Continuum management) project (Grant No. 101093129) and by the H.F.R.I project ENABLE-6G (ID:16294).

References

[1] Ali Shakarami, Mostafa Ghobaei-Arani, Ali Shahidinejad, “A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective”, Computer Networks, vol. 182, 2020.

[2] J. Feng, Q. Pei, F. R. Yu, X. Chu, J. Du and L. Zhu, ”Dynamic network slicing and resource allocation in mobile edge computing systems,” IEEE Transactions on Vehicular Technology, vol. 69, no. 7, pp. 7863-7878, July 2020.

[3] X. Chen, C. Wu, Z. Liu, N. Zhang and Y. Ji, “Computation offloading in beyond 5G networks: A distributed learning framework and applications,” IEEE Wireless Communications, vol. 28, no. 2, pp. 56-62, April 2021.

[4] Salvatore D’Oro, Leonardo Bonati, Francesco Restuccia, Michele Polese, Michele Zorzi, and Tommaso Melodia,“SI-edge: network slicing at the edge,” in Proc. Mobihoc, 2020.

[5] Z. Ning, X. Wang, J. J. P. C. Rodrigues and F. Xia, “Joint computation offloading, power allocation, and channel assignment for 5G-enabled traffic management systems,” IEEE Transactions on Industrial Informatics, vol. 15, no. 5, pp. 3058–3067, 2019.

[6] Y. -J. Liu, G. Feng, Y. Sun, S. Qin and Y. -C. Liang, “Device association for RAN slicing based on hybrid federated deep reinforcement learning,” IEEE Transactions on Vehicular Technology, vol. 69, no.12, pp. 15731–15745, 2020.

[7] J. Ren, H. Wang, T. Hou, S. Zheng and C. Tang, “Federated learning-based computation offloading optimization in edge computing-supported Internet of Things,” IEEE Access, vol.7, pp. 69194–69201, 2019.

[8] Nanliang Shan, Xiaolong Cui, and Zhiqiang Gao, ““DRL + FL”: An intelligent resource allocation model based on deep reinforcement learning for mobile edge computing,” Computer Communications, vol. 160, pp. 14–24, 2020.

[9] P. Dai, K. Hu, X. Wu, H. Xing and Z. Yu, “Asynchronous deep reinforcement learning for data-driven task offloading in MEC-empowered vehicular networks,” in Proc. IEEE INFOCOM, 2021.

[10] A. Okic, L. Zanzi, V. Sciancalepore, A. Redondi and X. Costa-Pérez, “π-ROAD: A learn-as-you-go framework for on-demand emergency slices in V2X scenarios,” in Proc. IEEE INFOCOM, 2021.

[11] Kai-Hsiang Liu and Wanjiun Liao, “Intelligent offloading for multi-access edge Computing: A new actor-critic approach”, in Proc. ICC, 2020.

[12] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji and M. Bennis, “Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning,” IEEE Internet of Things Journal, vol. 6, no. 3, pp. 4005–4018, 2019.

[13] Zhaolong Ning et al, “When deep reinforcement learning meets 5G-enabled vehicular networks: A distributed offloading framework for traffic big data,” IEEE Transactions on Industrial Informatics, vol. 16, no. 2, pp. 1352–1361, 2020.

[14] 3GPP, “Service requirements for video, imaging and audio for professional applications (VIAPA),” Technical specification group services and system aspects, 3GPP TS 22.263 V17.4.0, 2021.

[15] 3GPP, “Service requirements for cyber-physical control applications in vertical domains,” Technical specification group services and system aspects, 3GPP TS 22.104 V19.1.0, 2023.

[16] Kerbyson DJ, Hoisie A, Pakin S, Petrini F, Wasserman HJ, “A performance evaluation of an alpha EV7 processing node,” The International Journal of High Performance Computing Applications, Vol. 18, no. 2, pp. 199-209, 2004.

[17] OpenAI spinning up, “Soft Actor Critic,” [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/sac.html. [Accessed January 8, 2024].

[18] Python [Online]. Available: https://www.python.org/. [Accessed January 11, 2024].

[19] Tensorflow [Online]. Available: https://www.tensorflow.org/. [Accessed January 11, 2024].

[20] Google, “Google cluster-usage traces v3,” [Online]. Available: https://github.com/google/cluster-data/blob/master/ClusterData2019.md. [Accessed January 11, 2024].