Reinforcement Learning based Cloud and Edge Resource Allocation for Real-Time Telemedicine

Ivana Kovacevic*†, Rana Inzimam Ul Haq*‡, Jude Okwuibe*, Tanesh Kumar*, Savo Glisic§, Mika Ylianttila*, Erkki Harjula*
*Centre for Wireless Communication, University of Oulu, Oulu, Finland
†Nokia, Espoo, Finland.
‡Nokia, Oulu, Finland.
§Department of Physics, Worcester Polytechnic Institute, Worcester, Massachusetts, USA
{firstname.lastname}@oulu.fi*, sglisic@wpi.edu§

***Abstract*—Future healthcare services will extensively exploit wireless telehealth solutions in various healthcare use cases from preventive home monitoring to highly demanding real-time scenarios, such as monitoring an emergency patient's vital functions in an ambulance or ICU unit. Reliable real-time communications and computing are needed to enable these highly critical health services. However, the majority of current telehealth use cases are cloud-based, which poses a challenge to provide sufficient Quality of Service (QoS). The traditional centralized cloud infrastructure cannot meet the latency and reliability requirements due to long and unreliable communication routes. Therefore, the most advanced cloud solutions integrate edge computing as an integral part of the computational architecture to bring a part of the computational infrastructure to the proximity of the data sources and end-nodes, thus constituting an edge-cloud continuum. This continuum is capable of serving applications with real-time requirements. However, since edge computing capacity is a limited resource, solutions are needed for deciding which tasks should be run on edge and which at the data center. In this paper, we propose a machine learning-based solution to prioritize ultra-low-latency tasks for running on the edge to meet their strict delay requirements while leaving other tasks to be executed at remote servers. Our proposed solution in comparison to the baseline has a significantly lower dropping rate and outperforms fixed-interval scheduling solutions in terms of resource efficiency.**

***Index Terms*—edge-cloud continuum, internet of medical things, real-time computing, latency-limited computing, resource allocation, reinforcement learning, Q-learning.**

I. Introduction

The integration of novel communication technologies, such as 5G and Wireless Body Area Network (WBAN), along with advanced Internet of Medical Things (IoMT) sensing capabilities, help providing novel types of ubiquitous healthcare services. These services enable greater mobility without restricting normal human activities, as the medical personnel can observe the patient's health conditions based on the data received through the wireless network [1]. Wireless medical sensors are typically small-sized to facilitate comfortable in-body and wearable sensing, thus meaning very limited battery capacity and, furthermore, limited computational capabilities. Therefore, in medical sensing, cloud-based analytics is a standard solution for extracting meaningful information, data analytics, and decision-making [2]. Mission-critical and real-time medical sensing applications in, e.g., emergency response use cases, however, set strict requirements for the reliability and end-to-end latency of the task execution, which current centralized cloud systems cannot fulfill.

Due to these challenges, cloud computing architectures are extending closer to the end-nodes with the introduction of edge computing [3]. Edge computing enables parts of the cloud computing capabilities to be brought closer to end-devices and data sources, enabling pre-processing and filtering of raw data in their proximity to, e.g., reduce latency, network burden and avoid unnecessary transmission of private health raw data. Also, analysis of medical data and decision-making can be done in proximity for improved latency [4], [5]. A standardized edge computing solution, Multi-access Edge Computing (MEC) has been defined by European Telecommunications Standards Institute (ETSI) [6]. However, for MEC to be located in relative proximity to the end-nodes, a dense deployment of servers would be necessary, and therefore in comparison to cloud servers, they are limited in size and capacity [7]. In addition to reducing service latency, for cases such as MIoT and other delay critical services, MEC servers have other advantages over cloud. Namely, user data is not propagated throughout the network to geographically distant cloud centers locations, often outside the mobile operators' network. This not only reduces the overall network traffic and in turn, resource and power consumption, but it also improves privacy and security, since data stays within the providers network [8].

While it is beneficial to offload as many computational tasks as possible to nearby resource-constrained edge servers, MIoT systems should ensure sufficient resources for delay-critical applications that cannot be offloaded to the distant cloud due to low-latency requirements. To achieve the trade-off between these conflicting demands, sophisticated decision-making algorithms are required. With the diversification of the network use cases and the recent proliferation of IoT devices, it has been noted that low latency traffic requirements can significantly differ in required data throughput, end-to-end latency, and reliability [9]. Mission-critical real-time telemedicine use cases have particularly demanding requirements, where the computational tasks have to be completed within a strictly defined time period. We refer to these tasks as latency-limited traffic, e.g., to maintain the perceptual illusion in virtual reality

used in remote/robotic surgery, images must be processed in a few milliseconds before human eyes can detect the lag.

In our previous work [10], we propose joint computational resources allocation and server destination selection algorithm for MEC and Cloud server offloading with strict task latency requirements. In that work, we assumed that a computational task could be executed both at the MEC or cloud server, and the proposed algorithm emulates the optimal solution, i.e., allocation is calculated in the fixed time intervals for a batch of requests. However, for the aforementioned novel use cases, due to extremely lower latency demands, delay-critical tasks cannot be queued for execution at the server. Also, computational requests should not be waiting for batch processing in fixed scheduling periods but handled immediately upon arrival. Dedicated computational resources should be assigned to the task from the vacant capacities of the servers.

This paper proposes a reinforcement learning-based algorithm that assigns the task with minimal processing delay to the appropriate server and allocates the necessary computational capacity to meet strict delay requirements. The goal of the algorithm is to maximize the number of requests offloaded to the MEC server without blocking it for future arriving requests with extremely low delay requirements that cannot be offloaded to the cloud due to the longer network delays.

The main contributions of this paper are as follows:

- Q-learning-based resource allocation algorithm for latency-limited tasks in the two-tier computational offloading infrastructure that maximizes the number of MEC task offloading, i.e., minimizes the resource usage while at the same time maximizes the number of allocated tasks.
- Simulation results analysis, which demonstrates the performance of our proposed algorithm.
- An analysis showing that fixed interval scheduling is not resource efficient for latency-limited tasks.

The rest of the paper is organized as follows. Related works on computational offloading to edge-cloud infrastructure and the use of edge computing in MIoT is presented in Section II. Section III details the system model and formulates the problem of destination server selection and resource allocation for latency-limited traffic. Our proposed reinforcement learning-based offloading algorithm is introduced in Section IV. Section V presents the simulation results showing the efficiency of the proposed solution as well as other validated benefits, while we conclude the paper in Section VI.

II. Related Work

Effective utilization of distributed service architecture and edge intelligence is one of the key research challenges for real-time mission-critical healthcare scenarios in order to optimize end-to-end performance, efficiency and improve fault tolerance [11]. Some recent works have studied edge-cloud architecture framework in the context of healthcare scenarios. The edge-based hybrid network system architecture is proposed in [12]. They evaluated 3 use cases: wearable safety monitoring sensor network, a healthcare monitoring application, and a smart hospital application, and demonstrate promising capabilities of the edge concept in support of IoT applications. MIoT systems are also characterized by a high diversity of the devices, and data [13]. Authors in [13] propose edge–cloud framework to optimize the efficiency of heterogeneous MIoT data processing. In [14] resource scheduling algorithm is proposed to offload healthcare services to 5G edge computing infrastructure. They model a system as a Nash bargaining game and propose Lyapunov-based proportional-fairness resource scheduling algorithm.

The number of works on general computational offloading assumes that the end-device has sufficient computational capabilities, which is applicable only to some categories of user equipment (UE) and use cases. Since the UE is an established term for end-nodes, we use the UE from now on to refer to end-nodes. When considering most wireless health sensing applications, the minimal computational capacity of sensor nodes typically means that almost all sensor data processing is processed outside the sensing node, as discussed in the previous section. The decision criteria for whether to offload the computational task or perform it locally at the UE is widely studied topic [15] [16].

Most of the state-of-the-art research related to computational offloading mainly focuses on reducing the latency without taking into account use cases with strict latency requirements. However, in some recent studies, this aspect is also considered, e.g., the work in [17] considers both latency and power constraints and proposed an energy-efficient mechanism for end nodes by jointly optimizing both the computational and radio resources of the edge devices. A closed-form solution for the delay-constrained cloud offloading is given in [18]. While this work formulated the scheme using single-tier offloading (UE to server), we have taken two-tier offloading problem in this work (UE to edge or cloud server).

Learning algorithms have recently been gaining more and more attention of the research community as a promising solution to various resource management problems. Several research works propose different learning algorithms for resource allocation in edge-cloud offloading problem. Computational task offloading without strict latency constraints is considered in [19] [20] [21] [22] and [23]. In [19] and [20] predefined policies are used to determine which users can be allocated to the edge and which to the cloud. In contrast, our work proposes a dynamic allocation scheme. Work in [21] tackles the problem of resource allocation for the interdependent tasks in distributed edge-cloud architecture. They propose a modified deep deterministic policy gradient algorithm for task allocation. In [22] deep Q-learning algorithm is used to minimize the weighted sum of energy and delay. Scheme with edge server collaboration is studied in [23], under the assumption that computational tasks can be split into portions and distributed over different servers. The cloud offloading in this scheme is possible only when edge capacities are occupied. In general, our solution is different in two ways compared to these works. First, offloading decision, whether to allocate a task to edge or cloud server, is dynamic and does not follow any predefined policies. This allows for better resource utilization. Second, over-provisioning of the resources is not possible, since latency is not minimized but strict delay

requirements are met for each task.

Latency constrained computational task offloading problem in edge-cloud infrastructure is studied in [24] and [25]. In contrast to our work, these papers do not assume that some tasks can not be allocated to the cloud because of the low latency requirements. Additionally, they operate in fixed scheduling intervals which decreases the delay budget for computation and therefore increases the number of computational resources needed to execute tasks with already very low latency needs.

III. System Model

In this section, we present the system model, which is based on our previous work in [10]. In Section IV, we elaborate on the novel Q-learning resource allocation algorithm. In our system, we consider a two-tier computation offloading scenario as shown in Fig.1, where UEs, generating a need for limited-latency computational tasks, are connected to the base station (BS), which has access to external computation servers. The MEC server comprises multiple interconnected physical machines and can be connected directly to a BS using high-speed fibers or to an edge switch to enable multiple BSs to share computing resources. The placement of MEC servers in the access network is flexible, and while the communication latency between the BSs and the MEC is low, the computational capacity of the MEC server is often more limited compared to a cloud server. Computational requests from a BS can also be forwarded through a core network to a distant cloud server (C). The computational capacity of a cloud server is typically very high and easily scalable, whereas the communication latency between a BS and a cloud server is usually high. For the purpose of this analysis, the capacity of a cloud server is considered unlimited.

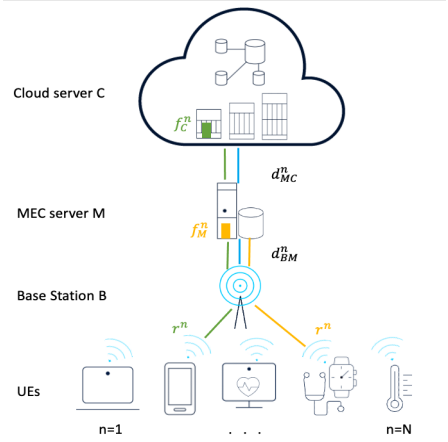

Fig. 1: System model [10]

We assume the MEC server M is directly connected to a BS B, as shown in Fig. 1 [10]. A set of UEs, connected to the BS B, generate $\mathcal{N} = \{1, ..., n, ...N\}$ limited-latency tasks over the period of time. The arrival of the tasks in the network is a Poisson distributed process with the average arrival rate λ. Task n is associated with a computational request characterized by the $K^n\{D^n, J^n, p_1^n, p_2^n\}$profile. The value of D^n represents the maximum permissible delay, taking into account the total communication and computational delay during task execution. In order to complete task n, a specific number of instructions or CPU cycles, denoted as J^n , must be executed. The number of instructions is typically expressed in Million Instruction (MI). We use p_1^n and p_2^n to represent the number of packets required for transmitting the task to the server and returning the computation result to the UE, respectively. The optical links have a constant transmission rate per session at the time of task allocation, and we assume that this rate is known. In order to support limited-latency communication, the network slices handling such traffic must maintain a constant rate on each link of the route, as queuing delays are not tolerable. For the limited-latency slice that relays the task between the BS and MEC server, we assume a transmission rate of r_s packets per second. The delay between BS B and MEC M for task n is denoted as $d_{B,M}^n$, while the delay between MEC M and cloud server C is represented as $d_{M,C}^n$, where $d_{B,M}^n << d_{M,C}^n$. Each UE has available transmission rate r^n over the wireless link at the moment of arrival. This rate depends on the available conditions in the network. The allocation algorithm is aware of the propagation conditions in the network at the time of allocation, i.e. it has information on available rates r^n and r_s. At the time of allocation, the total available computational capacity of a MEC server is denoted as F and is measured in terms of CPU cycles per second. The computational rate assigned to a given task n is expressed as f_M^n when executing on a MEC server and f_C^n when executed on a cloud server, where the rate is usually specified in Million Instructions per Second (MIPS).

A. Limited Latency Computation Offloading

As per our system model [10], we assume that the rate available to a UE with task n at BS B is equal to r^n packets per second. To offload a task to the BS, it takes p_1^n/r^n time, while relaying the computation result back to the UE takes p_2^n/r^n time. The delay incurred in transmitting a total of p_1^n packets over L_M links between the BS and MEC is $d_{n,1}^{B,M} = L_M/r_s + (p_1^n - 1)/r_s$ [26]. Similarly, relaying the computation result back to the UE requires $d_{n,2}^{B,M} = L_M/r_s + (p_2^n - 1)/r_s$, resulting in a total delay of $d_n^{B,M} = (2L_M + p_1^n + p_2^n - 2)/r_s$ between the BS and MEC to relay task n. The execution time for task n is given by J^n/f_M^n, and the total delay for offloading task n to the MEC must be less than the delay threshold:

$$D_M^n = \frac{p_1^n + p_2^n}{r^n} + \frac{J^n}{f_M^n} + d_n^{B,M} \leq D^n \tag{1}$$

Similar to the delay calculation for a task offloaded to the MEC server, the delay for a task offloaded to the cloud server can be determined by considering the additional delay between the MEC and cloud server, which is denoted as $d_n^{M,C}$ and calculated as $(2L_C + p_1^n + p_2^n - 2)/r_s$, where L_C represents the number of links or hops on the route between the MEC and cloud server. The condition for a delay guarantee in the cloud can be expressed as:

$$D_C^n = \frac{p_1^n + p_2^n}{r^n} + d_n^{M,B} + d_{M,C}^n + \frac{J^n}{f_C^n} \leq D^n \tag{2}$$

Computational rate sufficient for the execution of task n at the MEC server is:

$$f_M^n = J^n/(D^n - \frac{p_1^n + p_2^n}{r^n} - d_{B,M}^n) \quad (3)$$

Similarly, computational rate sufficient for the execution of task n at the cloud server is:

$$f_C^n = J^n/(D^n - \frac{p_1^n + p_2^n}{r^n} - d_{B,M}^n - d_{M,C}^n) \quad (4)$$

B. Task type

Depending on the use cases system is serving different task types from a finite set $K^n \in K$. We assume that delay thresholds D^n and available bandwidth r^n are always sufficient to allow task offloading to the MEC server:

$$D^n - \frac{p_1^n + p_2^n}{r^n} - d_{B,M}^n > 0 r^n > \frac{p_1^n + p_2^n}{D^n - d_{B,M}^n} \quad (5)$$

However, for some task types with low delay requirements, latency of the network can be larger than the requested delay:

$$D^n < \frac{p_1^n + p_2^n}{r^n} + d_{B,M}^n + d_{M,C}^n \quad (6)$$

This type of tasks can only be allocated to the MEC server. On the other hand, tasks types with the high delay requirement:

$$D^n > \frac{p_1^n + p_2^n}{r^n} + d_{B,M}^n + d_{M,C}^n \quad (7)$$

can be allocated both to the MEC and the Cloud server. From (3) and (4) follows that $f_M^n < f_C^n$, which means that MEC allocation is always more resource efficient, even without taking into account additional communication resources necessary to offload task n to the cloud. Therefore, the goal of our algorithm is to allocate as many high-delay requests to the MEC server. However, some of the high-delay arrivals should be allocated to the cloud in order not to block MEC capacity for the low-delay arrivals that can not be offloaded to the cloud. In the next section we propose the Q-learning resource allocation algorithm that learns decision policy for a given system in order to achieve this trade-off.

IV. Q-Learning-based Resource Allocation

Q-learning is a simple and well-known model-free reinforcement learning method [27]. This means that the decision-making agent learns straight from the interaction with the environment. After observing the state of the system, the agent selects one of the possible actions and observes the reward. In our case, the agent makes the decision of where to allocate the computational task, while the underlying computational offloading system is considered as the environment.

The state of the system is of the form *(available MEC capacity, new request profile, available rate)*, i.e. $s = (F, K^n, r^n)$. The state space is finite because there is a finite set of task types K. The agent has a set of available actions $A = \{a_1, a_2, a_3\}$ where a_1 is: allocate the task to the MEC with the computational rate f_M; action a_2 is: allocate the task to the cloud with the computational rate f_C; and a_3 is: drop the request. Depending on the state, different subset of actions is available:

- If D^n is high and $F > f_M^n$: $\{a_1, a_2\}$
- If D^n is high and $F < f_M^n$: $\{a_2\}$
- If D^n is low and $F > f_M^n$: $\{a_1\}$
- If D^n is low and $F < f_M^n$: $\{a_3\}$.

After the action is taken and there is a new arrival, the system moves to the next state $s' = (F', K^{n\prime}, r^{n\prime})$, where F' is the available capacity of MEC at the time of the request n' arrival, and depends on the previous action, i.e. previously allocated and completed tasks. The agent receives award that is equal $R = +1$, if the task is assigned to the MEC server. If the task is assigned to the cloud server, the reward is $R = 0$. If the task is dropped, the agent receives negative reward of $R = -10$.

The steps of the algorithm are summarized in Algorithm 1. Q-learning algorithm updates the Q-table iteratively until the policy converges. At the beginning of the learning process, the probability of exploration is high, but, after the optimal policy is learned, the algorithm continues working in the exploitation phase with occasional exploration. The exploration parameter is denoted with ϵ. It is decreased after every δ iteration. The learning rate is denoted with α and the discount factor with γ.

Algorithm 1 Q-learning Resource Allocation

```
Initialize Q(s,a) = 0,ϵ,α and γ
Observe environment state s_i
Repeat
For rand < ϵ
    select random action from A_s
    Else select action a = argmax Q_s
Observe reward R and new state s'
Update Q: Q(s,a) = Q(s,a) + α(R + γ max_a Q(s',a) − Q(s,a))
s=s'
If ϵ > 0.1
    decrease ϵ = ϵ − 0.1 after δ iterations
END
```

Complexity of Q-learning algorithm is linear with number of actions $O(|a|)$, which is 3 in this case.

V. Numerical Results

Here, we analyze the performance of our Q-learning resource allocation algorithm. The results are obtained by simulation of the system and Q-learning algorithm implementation using Python programming language. We compare the performance of our algorithm to a baseline solution with predetermined offloading policy. The baseline algorithm offloads tasks to the MEC server, if sufficient space is available, and to the Cloud server, when MEC capacities are not sufficient, and the delay requirement of the task is loose enough.

In this scenario we make an assumption that the maximum total capacity of the MEC server is $70,000$ MIPS. For all tasks n, the delay between the base station and the MEC server is 0.5 ms, while the delay between the MEC server and the cloud server is 3.5 ms. For simplicity, we assume $r^n = 15$packets/second for all users. The

tasks are randomly generated with different profiles, where $D^n \in \{4ms, 5ms, 7ms\}$, $J^n \in \{100MI, 150MI, 200MI\}$, $p_1 + p_2 \in \{10, 15, 20\} packets$.

The parameters of the Q-learning algorithm are as follows: $\epsilon = 0.9$, the learning rate $\alpha = 0.81$ and the discount factor $\gamma = 0.96$.

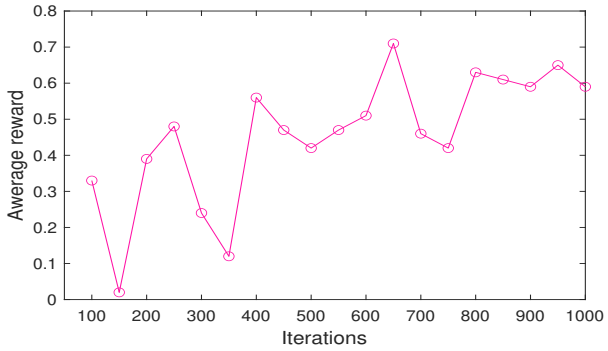

Fig. 2: Average reward

Fig. 2 presents the average reward acquired by the Q-learning agent throughout the simulation. The reward is averaged over 100 iterations of the algorithm. The Q-learning algorithm converges as the average reward increases with the simulation time. However, due to the system's randomness, such as the arrival of many low-delay requests in a short period, the average reward does not increase steadily but oscillates instead.

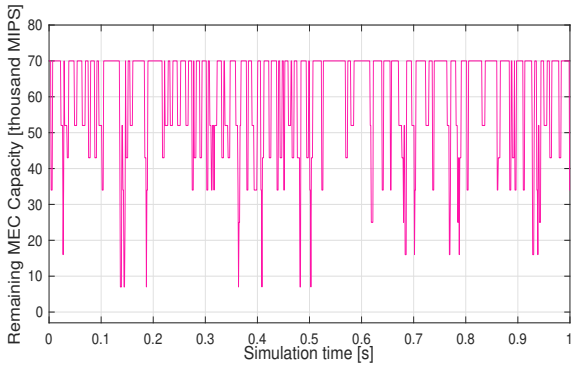

Fig. 3: Available MEC capcity

In Fig.3, the available capacity of the MEC server throughout the simulation is presented. Occasionally, the remaining MEC capacity is not sufficient for requests with high computational demand. If these requests require low delay, they get dropped.

In Fig. 4, the cumulative number of dropped tasks throughout the simulation time is presented. Here, the Q-learning resource allocation algorithm has significantly less dropped tasks compared to the baseline algorithm. This is because the congestion at MEC server is anticipated, and some of the high-delay tasks are offloaded to the cloud before the congestion takes place. The number of dropped packets does not drop to zero because sometimes the number of low-delay packets is too high for the system capacity. We simulated two scenarios:

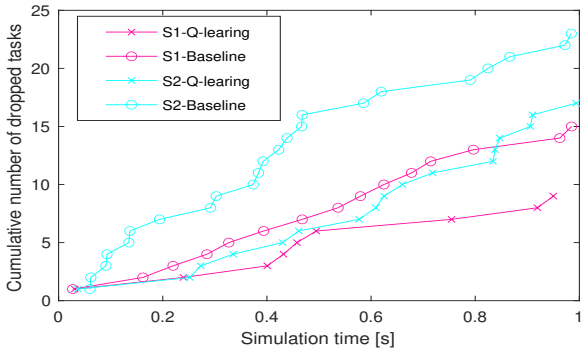

Fig. 4: Cumulative number of dropped tasks

S_1, where the distribution over all task types is constant, and S_2, where the percentage of low delay tasks is higher. The number of dropped packets increases with the increase in low-delay tasks. The percentage of dropped packets out of all generated tasks decreases from 10% to 5% in S_1 and from 19% to 9% in S_2.

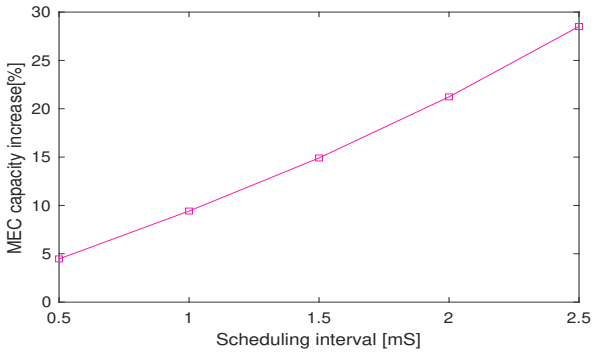

Fig. 5: Additional MEC capacity required for fixed interval scheduling compared to on arrival

Our algorithm processes requests upon their arrival. If the requests would be waiting to be processed in the fixed scheduling intervals, the delay budget D^n would be reduced for the difference between arrival time and length of the scheduling interval. The computational capacity necessary for execution of these tasks, according to (3) and (4), increases compared to our approach. Fig. 5 presents the increase in the computational capacity required to serve a single task in one scheduling interval in the case the scheduling is performed periodically instead of immediately. For simplicity, we assumed that all arrivals are assigned to the MEC. With the increase in the length of the scheduling interval, time budget for the computation decreases, and therefore more computational resources are necessary to complete the task within required latency limit. At the same time, with the decrease in the scheduling interval, the time benefits of processing requests in the batch and finding optimal or close to the optimal allocation decrease since the batch sizes decrease as well.

VI. Conclusion

In this paper, we studied the problem of two-tier limited-latency computational task offloading in the context of real-time telehealth applications. The objective was to assign an incoming task with minimal processing delay to an appropriate serving node in the edge-cloud continuum so that strict delay requirements are met while avoiding the congestion of the critical edge computing resources. We proposed a resource allocation algorithm based on reinforcement learning, which is capable of learning straight from the interaction with the environment in real-time. When we compared our proposed solution to the baseline scheduling solution, the first observation was that the drop rate was reduced significantly. Our algorithm enables decision-making upon request arrival, which is critical for very low-latency computational tasks. Maximizing the number of accepted requests helps to fulfill the strict latency requirements of delay-critical telehealth applications. The second observation is that our algorithm outperforms the fixed-interval allocation solution in terms of the required computational capacity for successful task execution, which improves resource-efficiency. Minimizing the usage of resources, especially avoiding unnecessary cloud offloading, improves the scalability and reduces the costs of the system, as the required quality of service can be provided with less hardware resources. In our context, this is an important contribution in improving healthcare cost-efficiency, one of the biggest challenges of aging societies with deteriorating dependency ratios. As the future work, we aim to develop our algorithm to work in a three-tier edge-cloud continuum, where local computing capacity found at, e.g., a hospital room or an ambulance, could be used as an additional local edge tier. We expect this approach to further improve resilience to network connectivity problems during, e.g., ambulance transportation.

Acknowledgment

This research has been partially supported by the Academy of Finland Digihealth and 6G Flagship programs (grant 346208) and Business Finland Tomohead project (grant 8095/31/2022).

References

[1] F. Khan, A. U. Rehman, J. Khan, M. T. Sadiq, R. M. Asif, and Z. Mehmood, "Real-time health monitoring using wireless body area network," *Journal of Applied and Emerging Sciences*, vol. 10, no. 1, 2020. [Online]. Available: http://journal.buitms.edu.pk/j/index.php/bj/article/view/332

[2] K. S. Awaisi, S. Hussain, M. Ahmed, A. A. Khan, and G. Ahmed, "Leveraging IoT and fog computing in healthcare systems," *IEEE Internet of Things Magazine*, vol. 3, no. 2, pp. 52–56, 2020.

[3] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1657–1681, thirdquarter 2017.

[4] K. Wei, L. Zhang, Y. Guo, and X. Jiang, "Health monitoring based on internet of medical things: Architecture, enabling technologies, and applications," *IEEE Access*, vol. 8, pp. 27 468–27 478, 2020.

[5] X. Li, X. Huang, C. Li, R. Yu, and L. Shu, "EdgeCare: Leveraging edge computing for collaborative data management in mobile healthcare systems," *IEEE Access*, vol. 7, pp. 22 011–22 025, 2019.

[6] Y. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing a key technology towards 5G," *Sophia Antipolis, France, ETSI, White Paper*, pp. 1–16, 2015.

[7] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, thirdquarter 2017.

[8] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, Feb 2018.

[9] X. Jiang, H. Shokri-Ghadikolaei, G. Fodor, E. Modiano, Z. Pang, M. Zorzi, and C. Fischione, "Low-latency networking: Where latency lurks and how to tame it," *Proceedings of the IEEE*, vol. 107, no. 2, pp. 280–306, Feb 2019.

[10] I. Kovacevic, E. Harjula, S. Glisic, B. Lorenzo, and M. Ylianttila, "Cloud and edge computation offloading for latency limited services," *IEEE Access*, vol. 9, pp. 55 764–55 776, 2021.

[11] I. Ahmad, Z. Asghar, T. Kumar, G. Li, A. Manzoor, K. Mikhaylov, S. A. Shah, M. Höyhtyä, J. Reponen, J. Huusko, and E. Harjula, "Emerging technologies for next generation remote health care and assisted living," *IEEE Access*, vol. 10, pp. 56 094–56 132, 2022.

[12] F. Wu, C. Qiu, T. Wu, and M. R. Yuce, "Edge-based hybrid system implementation for long-range safety and healthcare iot applications," *IEEE Internet of Things Journal*, vol. 8, no. 12, pp. 9970–9980, 2021.

[13] Z. Yang, B. Liang, and W. Ji, "An intelligent end–edge–cloud architecture for visual iot-assisted healthcare systems," *IEEE Internet of Things Journal*, vol. 8, no. 23, pp. 16 779–16 786, 2021.

[14] X. Lin, J. Wu, A. K. Bashir, W. Yang, A. Singh, and A. A. AlZubi, "Fairhealth: Long-term proportional fairness-driven 5g edge healthcare in internet of medical things," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 12, pp. 8905–8915, 2022.

[15] L. Liu, Z. Chang, X. Guo, S. Mao, and T. Ristaniemi, "Multiobjective optimization for computation offloading in fog computing," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 283–294, Feb 2018.

[16] L. Tang and S. He, "Multi-user computation offloading in mobile edge computing: A behavioral perspective," *IEEE Network*, vol. 32, no. 1, pp. 48–53, Jan 2018.

[17] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 89–103, June 2015.

[18] X. Meng, W. Wang, Y. Wang, V. K. N. Lau, and Z. Zhang, "Closed-form delay-optimal computation offloading in mobile edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 18, no. 10, pp. 4653–4667, 2019.

[19] Y. Liu, M. J. Lee, and Y. Zheng, "Adaptive multi-resource allocation for cloudlet-based mobile cloud computing system," *IEEE Transactions on Mobile Computing*, vol. 15, no. 10, pp. 2398–2410, Oct 2016.

[20] Q. Li, L. Zhao, J. Gao, H. Liang, L. Zhao, and X. Tang, "SMDP-based coordinated virtual machine allocations in cloud-fog computing systems," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1977–1988, June 2018.

[21] Q. Zhang, L. Gui, S. Zhu, and X. Lang, "Task offloading and resource scheduling in hybrid edge-cloud networks," *IEEE Access*, vol. 9, pp. 85 350–85 366, 2021.

[22] Y. Wang, H. Ge, A. Feng, W. Li, L. Liu, and H. Jiang, "Computation offloading strategy based on deep reinforcement learning in cloud-assisted mobile edge computing," in *2020 IEEE 5th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)*, 2020, pp. 108–113.

[23] J. Chen, S. Chen, Q. Wang, B. Cao, G. Feng, and J. Hu, "iraf: A deep reinforcement learning approach for collaborative mobile edge computing iot networks," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 7011–7024, Aug 2019.

[24] T. Zhao, S. Zhou, L. Song, Z. Jiang, X. Guo, and Z. Niu, "Energy-optimal and delay-bounded computation offloading in mobile edge computing with heterogeneous clouds," *China Communications*, vol. 17, no. 5, pp. 191–210, 2020.

[25] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 7944–7956, 2019.

[26] I. Kovacevic, A. S. Shafigh, S. Glisic, B. Lorenzo, and E. Hossain, "Multi-domain network slicing with latency equalization," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2182–2196, 2020.

[27] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *MIT press*, 2018.