Dynamic function allocation in edge serverless computing networks

Shuo Li, Ejder Başţuğ, Catello Di Martino, Marco Di Renzo

Dynamic function allocation in edge serverless computing networks

Shuo Li*†, Ejder Başтuğ*, Catello Di Martino*, Marco Di Renzo†
*Nokia Bell Labs, 7 Route de Villejust, 91620 Nozay, France
†Université Paris-Saclay, 3 Rue Joliot Curie, 91190 Gif-sur-Yvette, France
shuo.2.li@nokia.com, ejder.bastug@nokia-bell-labs.com,
lelio.di_martino@nokia-bell-labs.com, marco.di-renzo@universite-paris-saclay.fr

***Abstract*—Edge serverless computing has been widely used in mobile and web applications thanks to its simplicity and cost efficiency of network resource management. Serverless functions, event-driven and often stateless, are deployed on edge servers to reduce the latency caused by data transfer and the workload on cloud servers. However, in many edge serverless application scenarios, such as in IoT networks, the network is usually hierarchical, with limited edge storage and computational capacity, and requests received by different edge nodes might be heterogeneous and time-varying. The function deployment policy becomes a key issue in achieving guaranteed network performances. In this paper, we focus on function allocation policy in edge serverless computing networks to help edge nodes dynamically choose the functions to deploy based on incoming requests, aiming to reduce latency and minimize request outage rate. A Deep Deterministic Policy Gradient (DDPG)-based function allocation algorithm is proposed for a multi-tier edge-serverless computing scenario where a set of functions are deployed to the edge nodes strategically. Simulation results show that our proposed method can reduce the latency of requests while also minimizing the outage rate.**

***Index Terms*—function allocation, edge serverless computing, deep deterministic policy gradient, deep reinforcement learning.**

I. Introduction

Since Amazon launched AWS lambda in 2014, serverless computing has attracted a lot of attention from industry and academia. While serverless computing is originally intended for centralized cloud computing, it is quickly discovered that its ease of function development, device management, and operation was well suited for deployment in edge networks [1]. In recent years, edge serverless computing platforms, such as Microsoft Azure Serverless, AWS Lambda@Edge, Apache Openwhisk, and others, have gained widespread adoption and achieved great success, for instance, in the Internet of Things (IoT) and content delivery applications. Edge serverless computing aims to bring cloud computing capabilities closer to where the data is generated by bringing serverless functions partially or completely close to the network edge. It leverages the advantages of edge and serverless computing to provide high-quality and low-cost network services [2].

In the edge serverless computing paradigm, serverless functions are triggered by events and often processed purely by given input. These functions are typically stored in the cloud server's database and partially deployed across the edge servers' databases according to the function allocation policy. Each node can only execute requests whose functions are stored in its database [3]. Considering the heterogeneity of edge servers and their limited database size and computing capacity, the function allocation policy became one of the main concerns in realizing more reliable and efficient network services.

We have seen a lot of work on the optimization of edge serverless computing networks. For example, in [4], the authors formulated the task scheduling in edge serverless computing networks as a partially observable stochastic game problem and proposed a dueling double deep recurrent Q-network (D3RQN) based task scheduling algorithm. In [5], a novel platform is proposed, called NEMO, for realizing edge serverless computing without dedicated edge servers. In [6], the authors propose an adaptive function allocation algorithm to minimize function execution cost while maximizing the utility of servers, however without real-time tuning of function allocation nor latency impact of warm/cold starts. Our focus in this paper is to optimize function allocation using novel methods from deep reinforcement learning (DRL) while taking into account warm/cold start and other key aspects.

Modern communication and computing networks, such as IoT networks, are indeed becoming increasingly large-scale, decentralized, and autonomous, making it necessary for network devices to make decisions in uncertain and stochastic situations [7]. However, the computational complexity of modeling and solving modern networks is often extremely high. As an emerging optimization method, reinforcement learning has achieved prominent success in solving decision-making problems in both wired and wireless communication and computing networks. For example, in [8], the author proposed a multi-agent q-learning algorithm to solve the spreading factor and transmission problem in LoRa networks, and in [9], a multi-agent DRL algorithm with both Double Deep-Q Network and Advantage Actor-Critic agents is proposed to allocate transmit power to users.

The main contribution of this paper, based on the observations and explanations above, is to propose a novel dynamic function allocation algorithm for edge serverless computing networks using Deep Deterministic Policy Gradient (DDPG),

which has not been investigated earlier. It allows the edge nodes to dynamically adjust their deployed serverless functions according to request arrival patterns, taking into account the latency introduced by cold start, thus improving the load balancing and operational efficiency of the network. The rest of this paper is organized as follows. Our edge serverless computing model is defined in Section II. The proposed DDPG-based function allocation algorithm is presented in Section III. In Section IV, we show the performance of the algorithm with respect to baselines. Finally, the paper is concluded in Section V.

II. System model

A. Network model

In this paper, we consider an edge serverless computing network consisting of three types of servers in a geographical area, including one cloud server and two types of edge servers, which we refer to as small edge servers (SEs) and intermediate edge servers (IEs), respectively. We assume that IEs have a larger database and higher processing speed than SEs, but at the same time, the number of IEs is smaller. Denote by $\mathcal{I} = \{I_1, I_2, ..., I_{N_\mathrm{I}}\}$ the set of IEs and by $\mathcal{E} = \{E_1, E_2, ..., E_{N_\mathrm{E}}\}$ the set of SEs. Each SE connects to the nearest IE in addition to the cloud server, and IEs play the role of regional coordinators in our network. They do not receive data directly from users but are primarily responsible for sharing workloads as regional low-cost data centers. Let d define the distance between two network entities. Considering that the transmission time of the network increases with distance due to the number of hops and network traffic, we model the transmission delay between two directly connected network entities as $2\alpha d^{\beta}$, where α and β are constants [2].

Let $\mathcal{F} = \{f_1, f_2, ..., f_{N_F}\}$ be the set of all serverless functions in our edge serverless computing network. These functions are stored in the database of cloud server with sufficiently large storage. Meanwhile, serverless functions are deployed to SEs and IEs according to the deployment policy to accelerate data processing. To reflect the impact of the limited storage space of the edge server on the network, we define the maximum number of functions that the SE and IEs can store as D_E and D_I, respectively. The number of functions actually deployed on the edge server must not exceed this limit. We define $\mathcal{F}_{\mathrm{E},i} \subset \mathcal{F}$ and $\mathcal{F}_{\mathrm{I},j} \subset \mathcal{F}$ as the set of cached functions in SEs and IEs, where $i \in [1, N_\mathrm{I}]$ and $j \in [1, N_\mathrm{E}]$, respectively.

B. Service model

We define $\mathcal{M} = \{m_1, ...m_{N_m}\}$ as the set of function requests and assume that the arrival of function requests for each SE follows a Poisson process with an average arrival rate of ϵ. Moreover, we model the function popularity distribution with a Power law, which represents the ordered probability that a function is requested by a client [10]:

$$p(f_i) = (\eta - 1) i^{-\eta}, \tag{1}$$

where i denotes the index of functions, and η is the shape factor of the distribution with lower values resulting in a uniform behavior, whereas higher values indicate a steeper distribution. Note that a user request may contain multiple function requests in a real network environment. After arrival to the network, the user request is separated into multiple function requests and sent to its target server for execution. Therefore, in the following of this paper, all requests are treated as function requests unless otherwise specified.

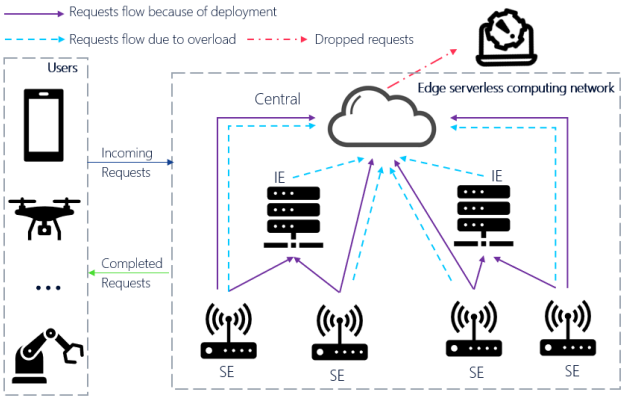

Fig. 1: An example of the service model.

In our edge serverless computing network, every server is deployed with a serverless computing platform. When a function is invoked by request, the server initializes a container and allocates computational resources to the container according to the function's requirements (cold-start) or reuses previously executed containers (warm-start) if the function has just been invoked [11]. In this paper, we model the service resources as the maximum number of containers it can support simultaneously in SEs, IEs, and cloud server, written as $\mathcal{C}_\mathrm{E}$, $\mathcal{C}_\mathrm{I}$ and $\mathcal{C}_\mathrm{C}$, respectively. In addition, the cache capacity of pending requests is limited in edge serverless computing nodes. We model the pending requests in each server as a first-in-first-out queue and define $\mathcal{K}_\mathrm{E}$, $\mathcal{K}_\mathrm{I}$ and $\mathcal{K}_\mathrm{C}$ as the size of the queue in SEs, IEs, and cloud server. For an incoming request, when no container is available on the target server, the request will be stored in its server queue. When this queue is overloaded, the IEs and SEs pass the request to the cloud server. The cloud server is in charge of handling forwarded requests from SEs and IEs, due to function not being deployed locally or queue overload. Newly arrived requests to the cloud server might be dropped in case of queue fullness.

Due to hardware heterogeneity of servers, such as different CPUs between tiers, the processing speed of requests might vary depending on the scenario. We define $\mu_\mathrm{E}, \mu_\mathrm{I}$, and μ_C as the service rate of SEs, IEs, and the cloud server, with $\mu_\mathrm{E} < \mu_\mathrm{I} < \mu_\mathrm{C}$. Hence, the service time for a request $m \in \mathcal{M}$ can be written as:

$$t_{s,m} = \frac{1}{\mu_m}, \tag{2}$$

where $\mu_m \in \{\mu_\mathrm{E}, \mu_\mathrm{I}, \mu_\mathrm{C}\}$ is the service rate of target server of request m.

The cold-start problem has been a longstanding issue in serverless computing frameworks, which makes the total processing time of a function potentially much longer than the intended time of execution [12]. Warm-start means that after serving a request, the container is held for a period of time. During this period, if another request with the same function arrives, the processing time of the new request will be reduced because the container initialization process is skipped. In our scenario, we define that the processing time is reduced by multiplying a reduction rate w in the warm-start case. An example of our service model is illustrated in Figure 1.

C. Optimal Function Allocation

Based on the above system model, we choose the average delay of requests and the outage rate of the system as our target performance parameters. We aim to reduce the average latency of requests while ensuring zero outage rate by properly allocating the functions deployed in each server. According to the service model, we define outage rate ξ as the ratio of dropped requests by the cloud server due to overload and the number of incoming requests, and the latency of a request includes the transmission time t_t, the waiting time t_w as well as the service time t_s, written as $l = t_t + t_w + t_s$. Therefore the optimization problem can be stated as follows:

$$\begin{aligned} \underset{\mathcal{P}}{\text{minimize}} \quad & \frac{1}{N_m} \sum_{m \in \mathcal{M}} (t_{t,m} + t_{w,m} + t_{s,m}) \\ \text{subject to} \quad & 0 < |\mathcal{F}_{S,i}| \leq D_{\mathrm{E}}, 0 < |\mathcal{F}_{I,j}| \leq D_{\mathrm{I}}, \\ & \xi = 0, \end{aligned} \tag{3}$$

where $i \in [1, N_{\mathrm{I}}], j \in [1, N_{\mathrm{E}}]$ and $\mathcal{P}$ is the function allocation policy. In this problem, the request's latency and the network's outage rate are only related to the current function allocation policy, and the problem can be described as a Markov process. At the same time, the complexity of optimizing the network using traditional methods is high due to the growing number of functions in the network and the need for perfect knowledge, which motivates us to solve this problem using DRL.

III. DDPG-BASED FUNCTION ALLOCATION ALGORITHM

In recent years, we have seen significant success with DRL in decision-making systems. A typical reinforcement learning structure includes an environment, an agent, and an experience reply buffer. During the algorithm's running, an agent makes decisions depending on the interaction with the environment. Assume that S is the set of possible environment states. At the time t, the RL agent observes $s_t \in S$ from the environment. According to the policy π, the agent takes action $a_t \in A$, where A is the action space. As a_t is performed, the agent earns a reward r, and the environment returns a new state s_{t+1}. The action a is decided according to policy π. At state s, we describe the choice of action as $\pi(s) = a$. A value function $q(s, a)$ is introduced in such a system to evaluate the performance of policy π at current state s and time t, shown as follows:

$$q(s, a) = \mathbb{E}[G_t | s_t = s, a_t = a], \tag{4}$$

where G_t is the discounted future cumulative reward,

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^{n-t} r_n, \tag{5}$$

where γ is defined as the discount factor and is a constant and $\gamma \in [0, 1]$.

In our problem, our goal is to find the best function allocation policy $\mathcal{P}$ to minimize the average latency l_{avg} while ensuring zero outage rate. The action a should include the number as well as the type of functions deployed in each server, and the reward r must consider both average latency and outage rate. The following is our definition of the environment and the agent.

A. Environment

Our environment consists mainly of an edge serverless network and incoming function requests $\mathcal{M}$. Each request has its access point, i.e., one of the SEs in the network, and an arrival time. We divide the time into uniform time slots, denoted by $\{0, 1, ..., T-1\}$. In each time slot $t \in \{0, 1, ..., T-1\}$, the network receives a_t chosen by the agent and observes the average latency and outage rate in the current network environment.

State: The state at each time slot consists of the average latency of requests and the outage rate, shown as:

$$s_t = \{l_{\mathrm{avg}}, \xi\}, \tag{6}$$

where $\xi \in [0, 1]$ and $l_{\mathrm{avg}} > 0$.

Action: In our system, each edge server, whether IE or SE, can determine the number and type of functions it stores upon request. Therefore, the action of an edge server can be defined as:

$$a_k = \{d_k, \mathcal{F}_k\}, \tag{7}$$

where

$$\begin{cases} 0 < d_k \leq D_{\mathrm{I}} & k \in \mathcal{I}, \\ 0 < d_k \leq D_{\mathrm{E}} & k \in \mathcal{E}, \end{cases} \tag{8}$$

and $\mathcal{F}_k = \binom{N_F}{d_k}$. Hence the action of time slot t can be defined as the combination of the action of all edge servers, shown as:

$$a_t = [a_1, ..., a_{N_{\mathrm{I}}+N_{\mathrm{E}}}]. \tag{9}$$

In our system, the action space is dynamically evolving since $\mathcal{F}_k$ is a function of d_k. Meanwhile, considering that the number of serverless functions in a network can be very large, constructing such a discrete action space is extremely time-consuming. To simplify the construction of action space while keeping our action space static during the solving process using reinforcement learning, we transform the action space into a continuous action space where each action is represented by a continuous value between 0 and 1.

Reward: At each time slot, the RL agent takes an action to update the serverless function deployed on each server. The environment will update the function allocation of each server and provide feedback based on the average latency and outage rate. Meanwhile, taking into account that in edge serverless

computing, requests are usually time-limited, our reward r is defined as:

$$r = \begin{cases} \max(\cos(0.75 l_{\text{avg}}), 0) & \xi = 0, \\ 0 & \xi > 0, \end{cases} \tag{10}$$

meaning that agents are rewarded only when the outage rate is zero.

Algorithm 1 The DDPG-based function allocation algorithm

Parameters: Actor learning rate lr_a, critic learning rate lr_c, soft update coefficient τ, discount factor γ, reply memory size B, mini-batch size b;
Initialize the critic network $Q_\omega(s,a)$ and actor network $\mu_\theta(s)$ with random network parameters ω and θ, respectively;
Pass the same parameters $\omega \to \omega'$ and $\theta \to \theta'$ to initialize the target critic network $Q_{\omega'}(s,a)$ and target actor network $\mu_{\theta'}(s)$, respectively;
Initialize experience replay memory $\mathcal{D}$;
for episode $= 1 \to N_{\text{episode}}$ **do**
    Initialize random noise $\mathcal{N}$ for action exploration;
    Observe the initial state s_t from the environment;
    for time step $N_{steps} = 1 \to N_{maxstep}$ **do**
        In actor network, choose continuous action a_t based on current strategy $\mu_\theta(s_t)$ and the random noise perturbation $\mathcal{N}$, $a_t = \mu_\theta(s_t) + \mathcal{N}$;
        In environment, transform continuous action a_t to deployed functions in each edge server;
        In environment, simulate network performance with requests $\mathcal{RQ}$, calculate the reward r_t, and the environment state changes to s_{t+1};
        Store (s_t, a_t, r_t, s_{t+1}) as a tuple in the experience replay buffer $\mathcal{D}$;
        Sampling random mini-batches (N_b tuples of transitions (s_t, a_t, r_t, s_{t+1}) from the replay memory $\mathcal{D}$;
        For each tuple, use the target networks to calculate $y_t = r_t + \gamma Q_{\omega'}\left(s_{t+1}, \mu_{\theta'}\left(s_{t+1}\right)\right)$;
        Update critic network by minimizing the target loss $L = \frac{1}{N_b}\sum_{t=1}^{N_b}\left(y_t - Q_\omega\left(s_t, a_t\right)\right)^2$;
        Update the actor network using the sampled policy gradient,
        $\nabla_\theta J \approx \frac{1}{N_b}\sum_{t=1}^{N_b} \nabla_a Q_\omega\left(s_t, a\right)\Big|_{a=\mu_\theta(s_t)} \nabla_\theta \mu_\theta\left(s_t\right)$;
        Update the target critic network: $\omega' \leftarrow \tau\omega + (1-\tau)\omega'$
        Update the target actor network: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$
    end for
end for

B. Agent

A large number of outstanding DRL algorithms have been proposed for solving decision-making problems, such as Deep Q-network [13], Dueling Double Deep Q-network [14], and Rainbow Deep Q-network [15]. In this paper, we choose the DDPG algorithm to design our agent. The reason for using DDPG algorithm stems from the fact that they are well suited to problems with continuous action and state space.

The DDPG algorithm is an off-policy DRL algorithm in the Actor-Critic (AC) framework. A DDPG algorithm consists of four neuronal networks, which are the actor network, the critic network, the target actor network, and the target critic network. The actor network takes the current state as input and outputs the probability of taking each action in the current state. The critic network is a value function network that takes the current state and action as input and outputs the Q value obtained by performing that action in the current state. Target networks are implemented to improve the stability of actor and critic networks [16]. The structure of DDPG algorithm is illustrated in Figure 2, and the procedure of training a function allocation algorithm is illustrated in Algorithm 1.

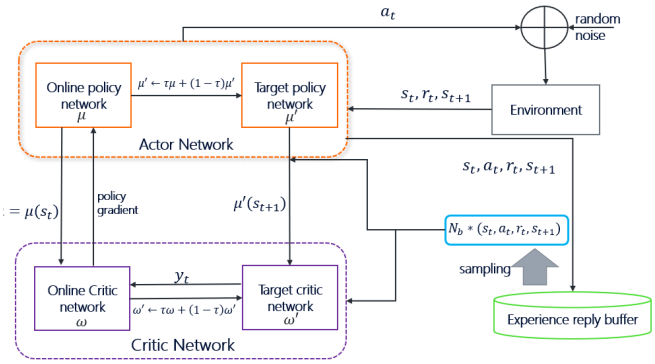

Fig. 2: The structure of DDPG algorithm.

IV. Numerical Results

We conducted extensive simulations to evaluate the performance of the proposed DDPG-based function allocation strategy. We built our deep neural network (DNN) agent with two fully connected layers made of 128 neurons, and chose $tanh$ as the activation function of each layer for both actor and critic networks. At the same time, we use Ornstein-Uhlenbeck noise as the random noise. In this section, we analyze the convergence of our algorithm and compare it with baseline algorithms to demonstrate the performance. Some of the parameters of our simulation are shown in Table I.

A. Convergence

Convergence is an important measure of the DRL algorithm, which reflects the design of neuronal network design and the selection of hyperparameters.

Figure 3, 4, and 5 shows the performance of the algorithm when the average arrival rate of function requests $\epsilon = 15$ requests per second. In the simulation, we set the shape factor of the function popularity distribution η of function requests received by the four SEs as 1.01, 1.51, 2.01, and 2.51, respectively. Meanwhile, considering that in practical applications, we are more interested in the highest reward in the whole training process and take the corresponding policy as the actual function deployment policy. Hence, besides

Parameters	Symbol	value
Actor learning rate	lr_a	5×10^{-5}
Critic learning rate	lr_c	5×10^{-5}
Soft update coefficient	τ	0.001
Discount factor	γ	0.95
Reply memory size	B	1000000
Mini-batch size	b	256
Step per episode	N_{maxstep}	200
Number of servers	N_{E}, N_{I}, N_{C}	4, 2, 1
Number of serverless functions	N_F	20
Maximum number of deployed functions	D_{E}, D_{I}	3, 5
Number of containers	$\mathcal{C}_{\mathrm{E}}$, $\mathcal{C}_{\mathrm{I}}$, $\mathcal{C}_{\mathrm{C}}$	2, 4, 8
Queue size	$\mathcal{K}_{\mathrm{E}}$, $\mathcal{K}_{\mathrm{I}}$, $\mathcal{K}_{\mathrm{C}}$	8, 10, 20
Service rate	$\mu_{\mathrm{E}}, \mu_{\mathrm{I}}, \mu_{\mathrm{C}}$	1, 2, 4 (request/sec)
Number of requests per step	$\lvert\mathcal{M}\rvert$	8000
Warm start reduction rate	w	0.5

TABLE I: List of simulation parameters.

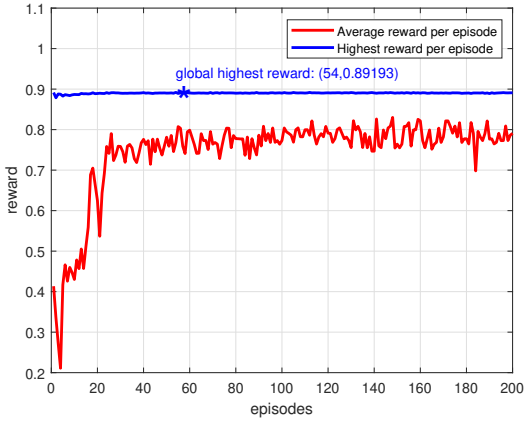

Fig. 3: Average reward and highest reward per episode.

the average value of reward and latency in each episode, the best result is also presented. Figure 3 showcases the maximum and average reward for each episode, and Figure 4 illustrates the corresponding optimal and average delay for each episode. The average outage rate of each episode is presented in Figure 5. The reason for the non-zero outage rate after algorithm convergence is that in each episode, the algorithm is still exploring new policies. We can see that our algorithm stabilizes after 40 episodes, and the optimal solution appears in the 54^{th} episode. However, the difference between the best reward after 40 episodes and the best reward across the board is less than 0.01.

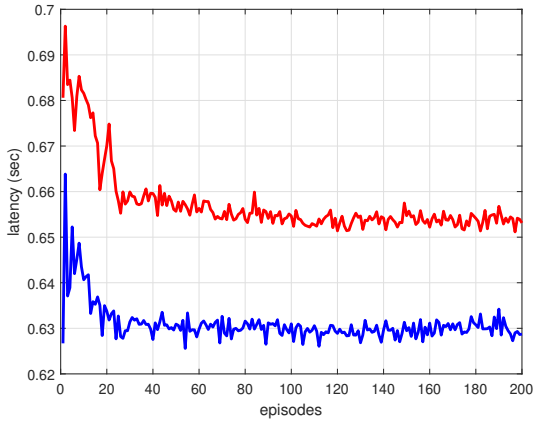

Fig. 4: Average latency and best latency per episode.

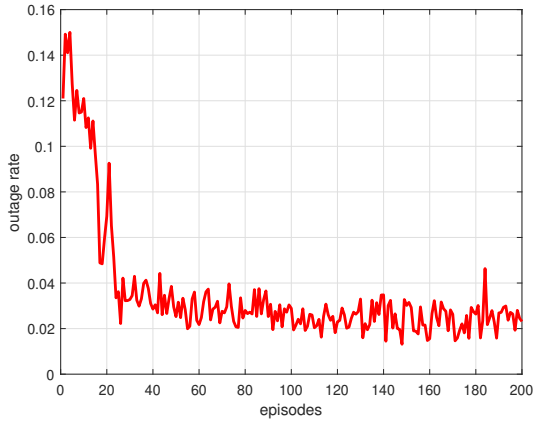

Fig. 5: Average outage rate per episode.

B. Performance Analysis

In order to further analyze the performance of the DDPG-based function allocation algorithm, we compare our proposed method with two baseline algorithms.

1) *Greedy*: The greedy algorithm is an online optimization algorithm with partial or no knowledge of the environment. In the greedy algorithm, the intelligence makes a decision with κ probability of randomly selecting an unknown action, leaving $1-\kappa$ probability of selecting the action with the largest action value among the existing actions. It is widely used in action selection for reinforcement learning to balance exploration and exploitation. In our simulation, we set the probability $\kappa = 0.6$ and the reward discount factor $\gamma = 0.9$.
2) *Global popularity*: Inspired by [2], we propose a static function allocation algorithm based on the global function popularity, where the SEs cache the most popular functions and the IEs cache the following popular functions. It is an offline algorithm with perfect knowledge of the network as well as the incoming requests. The number of functions deployed is fixed at the maximum

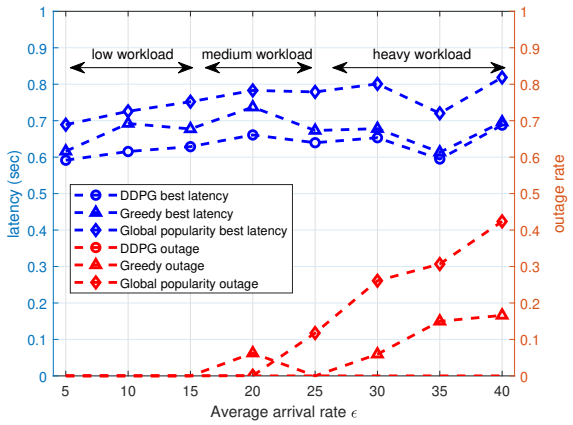

Fig. 6: Performance comparison of three algorithms.

number of functions that can be deployed.

We analyzed the average latency and outage rate of the three algorithms for different request arrival rates. Since the DDPG and greedy algorithms must be trained to obtain the desired solution, we compare the performance of these two algorithms after $30,000$ training steps and take the results achieved by the best of these strategies for comparison. Figure 6 shows the performance achieved by the three algorithms at an arrival rate of $\epsilon = [5, 10, 15, 20, 25, 30, 35, 40]$ per second. For the purpose of analysis, we divide the figure into three regions, low workload region, medium workload region, and heavy workload region, based on the outage rate of the greedy algorithm and the global popularity algorithm. In the first region, the workload is low with an average request arrival rate $\epsilon = [5, 10, 15]$ per second, and all three algorithms achieve zero outage rate. The dropped requests are observed for greedy and global popularity algorithms in the second region with average request arrival rate $\epsilon = 20$ and 25. Moreover, when the average request arrival rate ϵ is greater than 25, the function deployment policy based on both baseline algorithms leads to a large number of dropped requests due to overload, even though the global popularity algorithm has global knowledge of the network. On the contrary, our DDPG-based function allocation algorithm guarantees zero outage rate and provides lower average request latency in all three regions. Meanwhile, at $\epsilon = 35$, we observe an unexpected decrease in latency. The main reason for this is that in this simulation case, the arrival rate of requests with the same function is approximately equal to its service rate. A large number of requests were served in the warm-start.

V. Conclusion

In this paper, we proposed a dynamic serverless function allocation algorithm based on DDPG. We provided a detailed description of the algorithm and analyzed its convergence. In addition, we compared the performance of the DDPG-based function allocation algorithm in terms of average request latency and outage rate with the baseline algorithms. In the next step, on one hand, we will further optimize our function allocation algorithm, including experimenting with other reinforcement learning algorithms and optimizing hyperparameters. At the same time, we are going to use DRL algorithms to jointly optimize the parameters of the edge nodes to achieve deterministic networking in terms of network computation, i.e., to achieve very low outage, very low jitter, and bounded latency.

References

[1] A. Glikson, S. Nastic, and S. Dustdar, "Deviceless edge computing: extending serverless computing to the edge of the network," in *Proceedings of the 10th ACM International Systems and Storage Conference*, 2017, pp. 1–1.

[2] S. Li, E. Baştuğ, and M. Di Renzo, "On the modelling and analysis of edge-serverless computing," in *2022 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*, 2022, pp. 250–254.

[3] C. Cicconetti, M. Conti, and A. Passarella, "A decentralized framework for serverless edge computing in the internet of things," vol. 18, no. 2, 2021, pp. 2166–2180.

[4] Q. Tang, R. Xie, F. R. Yu, T. Chen, R. Zhang, T. Huang, and Y. Liu, "Distributed task scheduling in serverless edge computing networks for the internet of things: A learning approach," vol. 9, no. 20, 2022, pp. 19 634–19 648.

[5] L. Zhang, W. Feng, C. Li, X. Hou, P. Wang, J. Wang, and M. Guo, "Tapping into nfv environment for opportunistic serverless edge function deployment," vol. 71, no. 10, 2022, pp. 2698–2704.

[6] D. Xu and Z. Sun, "An adaptive function placement in serverless computing," vol. 25, no. 5. USA: Kluwer Academic Publishers, oct 2022, p. 3161–3174. [Online]. Available: https://doi.org/10.1007/s10586-021-03506-x

[7] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," vol. 21, no. 4, 2019, pp. 3133–3174.

[8] Y. Yu, L. Mroueh, S. Li, and M. Terré, "Multi-agent Q-learning algorithm for dynamic power and rate allocation in LoRa networks," in *2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications*, 2020, pp. 1–5.

[9] N. Naderializadeh, J. J. Sydir, M. Simsek, and H. Nikopour, "Resource management in wireless networks via multi-agent deep reinforcement learning," vol. 20, no. 6, 2021, pp. 3507–3523.

[10] E. Baştuğ, M. Kountouris, M. Bennis, and M. Debbah, "On the delay of geographical caching methods in two-tiered heterogeneous networks," in *2016 IEEE 17th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2016, pp. 1–5.

[11] L. Baresi and D. Filgueira Mendonça, "Towards a serverless platform for edge computing," in *2019 IEEE International Conference on Fog Computing (ICFC)*, 2019, pp. 1–10.

[12] I. E. Akkus, R. Chen, I. Rimac, M. Stein, K. Satzke, A. Beck, P. Aditya, and V. Hilt, "{SAND}: Towards high-performance serverless computing," in *2018 Usenix Annual Technical Conference 18)*, 2018, pp. 923–935.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[14] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1995–2003.

[15] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[16] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015.