Chinese Journal of Electronics
vol. 33, no. 4, pp. 899–909, July 2024
https://doi.org/10.23919/cje.2023.00.105

Special Focus on Multi-Dimensional QoS Provision of Intelligent Edge Computing for IoT

RESEARCH ARTICLE

A Deep Deterministic Policy Gradient-Based Method for Enforcing Service Fault-Tolerance in MEC

Tingyan LONG[1], Peng CHEN[2], Yunni XIA[1], Yong MA[3], Xiaoning SUN[4], Jiale ZHAO[1], and Yifei LYU[1]

1. *College of Computer Science, Chongqing University, Chongqing 400044, China*
2. *School of Computer and Software Engineering, Xihua University, Chengdu 610039, China*
3. *School of Computer and Information Engineering, Jiangxi Normal University, Nanchang 330022, China*
4. *School of Computer and Information Science, Chongqing Normal University, Chongqing 401331, China*

Corresponding author: Yunni XIA, Email: xiayunni@hotmail.com
Manuscript Received March 30, 2023; Accepted December 12, 2023

Abstract — Mobile edge computing (MEC) provides edge services to users in a distributed and on-demand way. Due to the heterogeneity of edge applications, deploying latency and resource-intensive applications on resource-constrained devices is a key challenge for service providers. This is especially true when underlying edge infrastructures are fault and error-prone. In this paper, we propose a fault tolerance approach named DFGP, for enforcing mobile service fault-tolerance in MEC. It synthesizes a generative optimization network (GON) model for predicting resource failure and a deep deterministic policy gradient (DDPG) model for yielding preemptive migration decisions. We show through extensive simulation experiments that DFGP is more effective in fault detection and guaranteeing quality of service, in terms of fault detection accuracy, migration efficiency, task migration time, task scheduling time, and energy consumption than other existing methods.

Keywords — Mobile edge computing, Service, Generative optimization network, Fault tolerance, Deep deterministic policy gradient.

Citation — Tingyan LONG, Peng CHEN, Yunni XIA, *et al.*, "A Deep Deterministic Policy Gradient-Based Method for Enforcing Service Fault-Tolerance in MEC," *Chinese Journal of Electronics*, vol. 33, no. 4, pp. 899–909, 2024. doi: 10.23919/cje.2023.00.105.

I. Introduction

Mobile edge computing (MEC) is a proximity-based paradigm with many benefits, such as low latency, efficient communication, and high system responsiveness [1], [2]. Recently, large-scale federated edge architecture deployments have been achieved by leveraging multiple independent edge computing providers, enabling seamless interconnection of multiple edge devices in a federated setting [3]. In such an environment, edge nodes are with limited power sources and can be deployed in a complicated and volatile environment that may cause failures or faults due to network faults, unexpected device breakdowns, or process faults. Faulty nodes may cause inaccurate sensing outcomes, erroneous data processing, and incorrect data communications. It is thus a challenging task to enforce fault tolerance because of the essential restrictions of MEC, such as unreliable connections, random mobility, small-bandwidth for communication, restricted power, and fixed storage. Luckily, techniques of proactive fault-tolerant can be highly suited for those cases, where it prevents edge node failures from running applications by preemptively migrating tasks out before occurrences of faults [4].

However, it remains a great challenge to enforce high-quality fault-tolerance of MEC applications. In an MEC environment, MEC servers manage data transmis-

Associate Editor: Prof. Jiwei HUANG, China University of Petroleum, Beijing.

sion and broadcast to neighboring wireless networks. For enforcing fault tolerance, tasks keep being moved out and migrated among MEC terminals connected via an edge network. The instability of wireless network can thus strongly affect the efficacy of fault-tolerant activities. Moreover, fault tolerance itself could bring in extra system overhead, which usually refers to resource required for fault compensation and cost of building/running task replicas. Such overhead should be taken into account and properly optimized to avoid a high loss of user-perceived service quality and system responsiveness. Fall into categories. The formers reallocate tasks in advance. In contrast, the latter aims to compensate.

In this paper, we propose a deep deterministic policy gradient (DDPG)-based method for enforcing mobile service fault-tolerance in MEC. The proposed framework is comprised of a predictor built upon a generation optimization network model for predicting failures of edge nodes and a preemptive migration decision maker built with a DDPG for yielding high-quality schedules of fault tolerance. The rest of the paper is organized as follows: Related work is overviewed in Section II. Section III describes the system model and formulates the problem. Section IV presents the DFGP fault-tolerant method proposed. Section V shows and discusses simulation results. Conclusions and future research suggestions are presented in Section VI.

II. Related Work

In an MEC environment, computing offloading is essential as it aims to reduce the latency, save bandwidth, and improve resource utilization. There is a lot of existing work on computing offloading [5]–[7]. Liu *et al.* studied the joint task offloading and resource allocation problem in device-edge-cloud collaborative framework for minimizing the task handling latency [5]. They proposed partitioning tasks into subtasks and allocating them proportionally to device, edge, and cloud, obtaining the optimal tasks offloading and resource allocation policy by Lagrangian dual method. Huang *et al.* proposed a computation offloading and resource allocation (CORA) algorithm based on a deep reinforcement learning method for obtaining the optimal offloading scheme with the objective of minimizing the cost of processing tasks in a dynamic network environment [6]. Chen *et al.* presented a game model among terminal devices named quality of service (QoS)-aware computation offloading (QCO) game for obtaining the Nash equilibrium offloading strategy with minimizing the total QoS cost for multiple IoT devices [7]. However, most of the computation offloading works reckon without the failure of edge equipment during task offloading transmission in harsh environments. Currently, proactive fault-tolerant techniques are widely used in distributed systems and clouds. For example, Liu *et al.* proposed a proactive co-ordinated fault tolerance method that is capable of predicting physical machine failure and conducting a particle swarm-based optimization for deciding fault compensation times [8]. Rawat *et al.* proposed a threshold-based adaptive fault tolerance approach. It consists of a stochastic failure predictor for predicting faulty virtual machines (VMs) and an adaptive manager for deciding recovery schemes [9]. Ray *et al.* proposed a preference-based fault management algorithm for predicting faulty VMs and employed an integer linear programming model for deciding VM reallocation schemes for fault tolerance and maximizing system profit [10].

Recently, machine learning and deep learning methods and models have shown high potency in dealing with fault-tolerant optimization problems. For instance, Zhang *et al.* proposed an online failure detection approach by using a systematic parameter-search model built upon a supporting vector machine. In addition, it leverages a prediction algorithm that can be updated round-by-round with dynamic feedback [11]. Hu *et al.* provided an unsupervised fault recognition model by using a deep adaptive fuzzy clustering framework [12]. It integrates stacked sparse autoencoder into adaptive weighted Gath-Geva clustering for detecting faults. He *et al.* presented a topology-ware multivariate time series anomaly detector (TopoMAD) for detecting anomalies in clouds. The detector leverages a long short-term memory (LSTM) model for judging system status [13]. Tuli *et al.* proposed a preemptive migration prediction model. It utilizes a generative adversarial network (GAN) for detecting node failures in MEC caused by overload [14].

III. System Models

1. System model

The MEC environment is illustrated in Figure 1. It comprises local edge infrastructure (LEI) with a set of heterogeneous edge nodes and a set of mobile users. An environment consists of m edge nodes, i.e., $E = \{e_1, e_2, \ldots\}$, $B = \{b_1, b_2, \ldots\}$ denotes the collection of edge brokers, and $U = \{u_1, u_2, \ldots\}$ denotes the set of user. e_h denotes the hth edge node and $e_h \in E$. Each edge node has its own coverage R_h, and each LEI has its own edge broker responsible for sensing the status of contacting edge nodes. We consider that all edge nodes can commu-

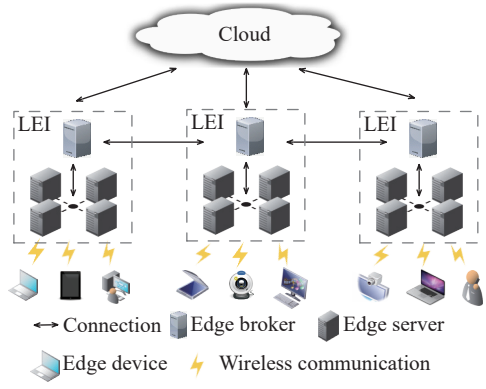


Figure 1 Mobile edge computing environment.

nicate with each other by using an edge broker for interconnection. An edge broker can obtain resource utilization indicators for all edge nodes and is capable of sensing the status, i.e., CPU usage, RAM usage, disk resource, network bandwidth, and failure records, of all edge nodes in the LEI group [15]. The parameters used in this paper are listed in Table 1.

Table 1 The notaion of term

Notation	Description
k	The length of sliding window
u	The features of edge nodes
n	The number of active tasks
c_h	Real-time bandwidth of the edge node
e_h	The hth edge node in an enumeration of E
b_g	The gth edge broker in an enumeration of B
y_t	Fault label by prediction
f_t	Fault score by prediction
a_i^t	The ith task in A_t
b_i^t	The amount of data of task a_i^t
d_i^t	The number of calculation instructions of task a_i^t
u_j^x	The longitude of the jth mobile user
u_j^y	The latitude of the jth mobile user
e_h^x	The longitude of the hth edge node
e_h^y	The latitude of the hth edge node
E	The set of edge nodes, $E = \{e_1, e_2, \ldots\}$
B	The set of edge brokers, $B = \{b_1, b2, \ldots\}$
U	The set of edge devices, $U = \{u_1, u_2, \ldots\}$
I_t	The tth interval time in simulation time T, and $t \in (0, T)$
A_t	The set of active tasks at the interval I_t
C_t	The set of completed tasks at the interval l_{t-1}
N_t	The set of new tasks
O_t	The remaining tasks at interval I_{t-1}
M_t	The set of migratable tasks at interval time t
S_t	Scheduling decision at interval time t
G_t	Undirected topology graph of environment
W_t	Time-series window
R_h	The coverage of edge node e_h
M_t^*	Migration decision at interval time t
E_j^t	The set of edge nodes for user u_j at I_t
W_{t+1}	The time-series window at interval I_{t+1}
$\check{W}_{t+1}$	The reconstruction window at interval I_{t+1}

2. Workload model

The computing and bandwidth requirements for tasks vary over time due to changing user demands and mobility IoT devices. We thus assume that task execution timeline can be divided into fixed-sized scheduling intervals [16], where I_t denotes the tth scheduling interval (t ranging from 0 to T), $s(I_0) = 0$ denotes the starting time of I_0, and $s(I_t) = s(I_{t-1}) + \Delta$, where Δ indicates scheduling intervals of equal durations. $A_t = \{a_0^t, a_1^t, \ldots, a_n^t\}$ represents active tasks being performed on the edge nodes in each scheduling interval I_t and $n = |A_t|$. Mobile device u_j will select an edge node from E_j^t for offloading and $\text{dist}(u_j, e_h) < R_h$, where $\text{dist}(u_j, e_h)$ denotes the distance between u_j and e_h. The delay is decided by the uplink delay $\lambda_i^{t,\text{up}}$, the download delay $\lambda_i^{t,\text{do}}$, the executed delay $\lambda_i^{t,\text{ex}}$, and the migration delay $\lambda_{hv}^{t,\text{mi}}$ from edge node e_h to edge node e_v [17].

$$\begin{aligned}\text{delay}_i^t(e_h) &= \lambda_i^{t,\text{up}} + \lambda_i^{t,\text{do}} + \lambda_i^{t,\text{ex}} + \lambda_{h,v}^{t,\text{mi}} \\ &= \frac{b_i^t}{c_h \eta_i^h} + \xi + \frac{d_i^t}{f_h} + \phi \text{dist}(e_h, e_v) \end{aligned} \tag{1}$$

where c_h is the average bandwidth, η_i^h is a parameter that depends on distance, ξ is a constant value, and the value is 0.1 in [18], f_h is average computing power of e_h, d_i^t is the number of calculation instructions of task a_i^t, and ϕ is the distance coefficient and the value is 0.01. $u_j(t) = (u_j^x, u_j^y)$ denotes the user movement trajectory and the distance between the edge node e_h and the user u_j is

$$\text{dist}(u_j, e_h) = \sqrt{(u_j^x - e_h^x)^2 + (u_j^y - e_h^y)^2} \tag{2}$$

where u_j^x and u_j^y represent the longtitude and latitude of user u_j, respectively, e_h^x and e_h^y represent the longtitude and latitude of edge node e_h, respectively. C_t is the set of completed tasks before I_t. Hence, the set of tasks for the next interval can be expressed as $O_t = A_{t-1} \setminus C_t$. N_t represents the set of newly arrived tasks in I_t. Let $M_t \subseteq O_t$ be the set of migratable tasks due to the edge node failure. Therefore, the set of active tasks is the union of O_t and N_t at scheduling interval I_t, i.e., $A_t = O_t \cup N_t$. G_t denotes undirected topology graph of MEC environment. Here, we consider only individual tasks and each task has an associated service level objectives (SLO) deadline. Apart from this, S_t denotes the scheduling decisions for new tasks N_t and active tasks O_t at the start of the interval I_t.

3. Fault model

There are multiple fault types in an MEC environment, i.e., hardware failure, software failure, and resource overflow, all of which lead to task failures and bad user-perceived service quality [19]. In our work, we consider four fault types in an MEC environment, including CPU overload, RAM contention, Disk attack, and DDoS attack. The fault data for each edge server can be collected by using fault injection model [20]. Then the failure of edge server is simulated through discrete events with random fault injection.

4. Problem formulation

In this work, our goal is to prevent task failure caused by the failure of edge node through a predictive method. Due to the fact that edge brokers manage both the resource and schedule tasks in LEI, the objective is thus to optimize the performance of the scheduler with proactive fault-tolerance implemented. The capability of scheduler in each scheduling interval is L and L_t is the loss of the interval I_t. The state is State_t, residual active tasks, O_t, and new tasks are N_t. The action is Action_t, which refers to selecting the appropriate edge node $\{E_i\}$ for task a_i^t in A_t.

Thus, $\text{Action}_t = \{e_h \in E \text{ for task } a_i^t | a_i^t \in M_t \cup N_t\}$, which indicates the preemptive migration decision for tasks in M_t and scheduling decision for tasks in N_t. The problem can be formulated as

$$\begin{aligned} &\min_{\mathrm{S}} \sum_{t}^{T} L_t \\ L_t = \ &\alpha \times \text{AEC}_t + \varphi \times \text{ART}_t + \delta \times \text{AMT}_t + \eta \times \text{SLAV}_t \\ \text{s.t.} \ \ &\text{a) } \forall t,\ \text{Action}_t = S(\text{State}_t) \\ &\text{b) } \forall t\ \forall a_i^t \in M_t \cup N_t,\ \{E_i\} \leftarrow \text{Action}_t(a_i^t) \end{aligned} \tag{3}$$

where S is scheduler model, α, φ, δ, and $\eta \geq 0$, and $\alpha + \varphi + \delta + \eta = 1$. In order to optimize L_t, we consider L_t as a convex combination of AEC_t, ART_t, AMT_t, and $\text{SLAV}(a_i^t)$ for interval I_t. AEC_t is the average energy consumption for interval I_t in (4). ART_t is the average response time for tasks in C_t in (5). AMT_t is the average migration time for active tasks in A_t in (6). And $\text{SLAV}(a_i^t)$ is the average count of optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers (SLA) violations for completed tasks [21] in (7).

$$\text{AEC}_t = \frac{\sum_{e_h \in E} \int_{z=s(I_t)}^{s(I_{t+1})} P_{e_h}(z)\mathrm{d}z}{n \times \sum_{e_h \in E} P_{e_h}^{\max}(z_{h+1} - z_h)} \tag{4}$$

$$\text{ART}_t = \frac{\sum_{a_i^t \in C_{t+1}} R(a_i^t)}{|C_{t+1}| \max_t \max_{a_i^t \in C_t} R(a_i^t)} \tag{5}$$

$$\text{AMT}_t = \frac{\sum_{a_i^t \in A_t} M(a_i^t)}{n \times \max_t \max_{a_i^t \in C_t} R(a_i^t)} \tag{6}$$

$$\text{SLAV}_t = \frac{\sum_{a_i^t \in C_{t+1}} \text{SLA}(a_i^t)}{|C_{t+1}|} \tag{7}$$

where $P_{e_h}(t)$ is the power function of edge node e_h at t and $P_{e_h}^{\max}$ the maximum possible power of e_h. $R(a_i^t)$ is the response time function of task a_i^t, $M(a_i^t)$ is the migration time function of task a_i^t, and $\text{SLA}(a_i^t)$ is the SLA for tasks a_i^t [21].

IV. Proposed Fault-Tolerant Method

In this section, we present the generative optimal network (GON)-based fault prediction model and the DDPG one, and the proposed fault-tolerant (DFGP) one shown in Figure 2. Here, GON takes the graph topology G_t, time series window W_t, and task scheduling S_t as the inputs for being aware of MEC status and predicting future faults, and DDPG yields migration schedules.

1. Generative optimization networks

GON is a kind of unsupervised model [22]. It takes multivariate time-series of states $\{x_0, x_1, \ldots, x_t\}$ as inputs and aims to predict the future state of x_{t+1}. For the replication padding mechanism [23], we consider a sliding window of length k for capturing:

$$W_t = \{x_{t-k+1}, x_{t-k+2}, \ldots, x_t\} \tag{8}$$

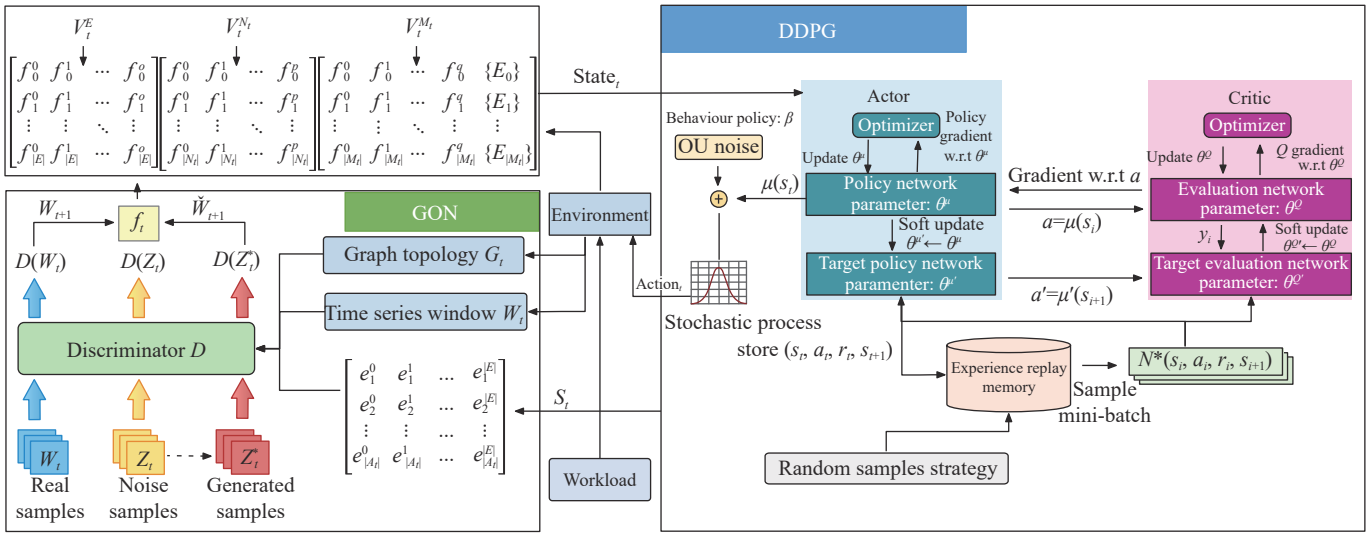


Figure 2 The DFGP model.

Figure 3 shows an example of a window with size $k = 3$. GON takes graph topology G_t and the scheduling decision S_t in Figure 2 as inputs as well. y_t represents the predicting fault label, f_t is the fault score, W_{t+1} is the next window, and $\breve{W}_{t+1}$ is the predicted reconstruction of the next window. The fault score f_t is obtained through computing reconstruction error between the true window W_{t+1} and its reconstruction $\breve{W}_{t+1}$:

$$f_t = ||W_{t+1} - \breve{W}_{t+1}|| \tag{9}$$

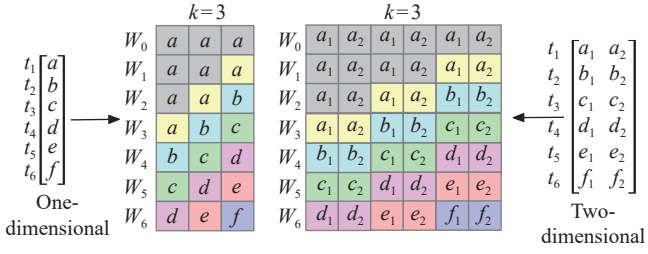


Figure 3 The example of sliding window when $k = 3$.

Figure 2 shows the GON framework. Unlike GAN [24], it incorporates only one neural network as discriminator model $D(\cdot;\theta)$, which is a differentiable multilayer perceptron with parameters θ. It includes three steps, namely training with real samples $D(G_t, W_t, \boldsymbol{S}_t; \theta)$, generating fake samples Z_t^* with through a random noise sample Z_t, and training model $D(G_t, Z_t^*, \boldsymbol{S}_t; \theta)$ with fake samples. The discriminator aims at both generating fake samples, as close to the real datas as possible, and identifying fake samples as much as possible. Equations (10) and (12) train real samples and fake ones with the cross-entropy loss by descending the stochastic gradient, and (11) generates fake samples by using random noise samples. Here, γ denotes a step parameter.

$$-\nabla_\theta \log(D(G_t, W_t, \boldsymbol{S}_t; \theta)) \tag{10}$$

$$Z_t \leftarrow Z_t + \gamma \nabla_{Z_t} \log(D(G_t, Z_t, \boldsymbol{S}_t; \theta)) \tag{11}$$

$$-\nabla_\theta \log(1 - D(G_t, Z_t^*, \boldsymbol{S}_t; \theta)) \tag{12}$$

Equation (12) implements an AdaHessian optimizer [25] for improving convergence speed. Discriminator D captures the temporal trends in the time-series data and efficiently discriminates $\breve{W}_{t+1}$ of the next window. In this work, the scheduling decision $\boldsymbol{S}_t$ is encoded as one-hot matrix of size $n \times m$. The neural network operates on $\boldsymbol{S}_t$ as a batch of n vectors, each of which is with a dimension of m, and W_t is operated as a batch with the size $n + m$ and tensors with the size of $u \times k$. In addition, G_t is obtained by conducting a graph convolution network [26] for capturing the inter-edge nodes dependencies.

2. Deep deterministic policy learning

DDPG is a strategy gradient algorithm for continuous action space, which combines strategy gradients and a deep Q-learning network. It yields a deterministic action rather than an action probability distribution [27] and the algorithm is described as **Algorithm 1**.

Algorithm 1 Deep deterministic policy gradient algorithm

Input: V_t^E, $V_t^{N_t}$, $V_t^{M_t}$, M, γ, τ, β.
Output: Action$_t$.

1: Randomly initialize the critic network $Q(s, a|\theta^Q)$ and actor network $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ;
2: Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$;
3: Initialize replay memory buffer R;
4: **for** episode = 1, M **do**
5: Initialize a random process $\mathcal{N}$ for action exploration;
6: Receive initial observation state s_1;
7: **for** t=1, T **do**
8: Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise;
9: Execute action a_t and observe reward r_t and observe new state s_{t+1};
10: Store transition (s_t, a_t, r_t, s_{t+1}) in R;
11: Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R;
12: Calculate Q reference value, called, $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$;
13: Update critic network parameter θ^Q by minimizing the loss: $L = \frac{1}{N}\sum_i^N (y_i - Q(s_i, a_i|\theta^Q))^2$;
14: Update actor network using the sampled policy gradient: $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i^N \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$;
15: Update the target networks: $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$, $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$;
16: **end**
17: **end**
18: Action$_t \leftarrow \mu(s_t|\theta^\mu)$.

DDPG is essentially an actor-critic framework being able to select an action value for a given state. It includes four networks, i.e., a policy network, a target policy network, an evaluation network, and a target evaluation network. The policy network iteratively updates the policy network with parameters θ^μ, selects the current action $a_t = \mu(s_t|\theta^\mu)$ according to state s_t for interacting with the environment, and generates next state s_t', as well as the current reward r. The target policy network selects the next action a_t' and updates $\theta^{\mu'}$. The evaluation network iteratively updates parameter θ^Q. The target evaluation network calculates the target Q value. The objective function of DDPG is in (13). The Q function is expressed as the expectation of the reward value of the selected action with the deterministic scheme μ in (14).

$$J(\theta^\mu) = \mathbb{E}_{\theta^\mu}[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots] \tag{13}$$

$$Q^{\mu}(s_t, a_t) = \mathbb{E}[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}))] \quad (14)$$

where γ is the discount factor. Equation (15) is used for evaluating the quality of the strategy and the optimal behavior strategy μ is defined as the strategy with maximal $J_\beta(\mu)$ in (16).

$$J_\beta(\mu) = \int_{\mathcal{S}} \rho^{\beta} Q^{\mu}(s, \mu(s)) \mathrm{d}s = \mathbb{E}_{s\sim\rho^{\beta}}[Q^{\mu}(s, \mu(s))] \quad (15)$$

$$\mu = \mathrm{argmax}_{\mu} J(\mu) \quad (16)$$

where $Q^{\mu}(s, \mu(s))$ represents the expected return obtained by selecting actions in s state with strategy μ. β is an Uhlenbeck-Ornstein random process (UO process). $J_\beta(\mu)$ is the expectation of $Q^{\mu}(s, \mu(s))$ when s is distributed hinge on ρ^{β}. The network update process is as follows:

$$\text{policy network}: \begin{cases} \text{online}: \mu(s|\theta^{\mu}) & \text{gradient update } \theta^{\mu} \\ \text{target}: \mu'(s|\theta^{\mu'}) & \text{soft update } \theta^{\mu'} \end{cases} \quad (17)$$

$$Q \text{ network}: \begin{cases} \text{online}: Q(\mathrm{s}, \mathrm{a}|\theta^{Q}) & \text{gradient update } \theta^{Q} \\ \text{target}: Q'(\mathrm{s}, \mathrm{a}|\theta^{Q}) & \text{soft update } \theta^{Q'} \end{cases} \quad (18)$$

$$\text{soft update}: \begin{cases} \theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \\ \theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \end{cases} \quad (19)$$

where the soft update parameter $\tau = 0.001$.

In this paper, we model the fault-tolerance decision process of the scheduler $\boldsymbol{S}_t$ as a Markov decision process with a state space $\mathcal{S}$, an action space $\mathcal{A}$, a transition dynamics $p(s_{t+1}|s_t, a_t)$, and a reward function of $r(s_t, a_t)$. The return from state refers to the sum of discounted future reward $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$ with a discounting factor $\gamma \in [0, 1]$. $\mathcal{S}$ corresponds to the tasks in each LEI. The edge broker in each LEI determines the scheduling policy $\boldsymbol{S}_t$ based on the observed information. Specifically, at the beginning of the interval I_t, the observation is described by a vectors $\boldsymbol{V}_t^E$ of each edge node which refers to the usage of its CPU, RAM, bandwidth, disk, etc. The size of $\boldsymbol{V}_t^E$ is $m \times (u+1)$ and the last feature is fault label of edge server, $u = 4$. The size of the feature vector $\boldsymbol{V}_t^{N_t}$ is $|N_t| \times u$. The size of $\boldsymbol{V}_t^{M_t}$ is $|M_t| \times (u+1)$ and the last feature in $\boldsymbol{V}_t^{M_t}$ indicates the edge node index number selected by the previous interval I_{t-1} task. A single state is thus described by a tuple $s_t = (\boldsymbol{V}_t^E, \boldsymbol{V}_t^{N_t}, \boldsymbol{V}_t^{M_t})$, where $s_t \in \mathcal{S}$. For the action space $\mathcal{A}$, the current state of the system is observed by each LEI edge broker s_t and an action a_t, i.e., an appropriate edge node is selected for each task for each time interval t. For the reward function R, the behavior of each LEI edge broker is reward-driven, and thus the reward function r_t received by each agent after the time t step is (3).

3. Proposed method

The pseudo codes of the proposed DFGP algorithm is illustrated in Algorithm 2. Each edge broker executes DFGP algorithm and the neural network is fine-tuned periodically to adapt to changing circumstances and workload traits. We firstly training discriminator D with dataset to obtain performance and initialize the network parameters (line 1 and line 2). Then obtain the scheduling decision Action_t through Algorithm 1 (lines 3–7). The new task is assigned based on Action_t, and migrating tasks on potentially faulty edge nodes (lines 8–11). The input of GON for unsupervised fault detection method is multivariate time-series data, such as $\boldsymbol{S}_t$, W_t, G_t, and $\check{W}_{t+1}$. The simulation setting is in line 12, and the AdaHessian optimizer is used to accelerate the rate of convergence (line 13). The ReLU activation function is used to avoid zero fault scoring (line 14). The fault score was compared with the data generated by the peak value over threshold (POT) method [28] to obtain the fault label (line 15). Compile the time series windows of all brokers to W_t and label the failed edge nodes with failure labels. Finally, add the fault label to $\boldsymbol{V}_t^E$ (lines 16–18). Then, use optimization goal to update the network parameter (line 19) and fine-tune the discriminator model with a new data point by loss Loss (line 20).

Algorithm 2 The DFGP fault-tolerance algorithm

Inputs: The number of scheduling intervals N, set of edge nodes E;
Output: Migration decision M_t^*;

1: Trained GON model $D(.;\theta)$ with dataset;
2: Initialize Algorithm 1 parameters;
3: **for** interval index t to T **do**
4: **for** each broker **do**
5: Send State_t to Algorithm 1;
6: $\mathrm{Action}_t \leftarrow$ output of Algorithm 1 for State_t;
7: Allocate new tasks and migration existing tasks based on Action_t;
8: Task migration is represented by $M_t^* = (e_j, e_h)$;
9: **if** $e_j \in \mathrm{LEI}_1$ and $e_h \in \mathrm{LEI}_2$ **then**
10: Migrate tasks on edge node e_j to edge node e_h in LEI_2;
11: **end if**
12: $\boldsymbol{S}_t \leftarrow \mathrm{Action}_t$;
13: $W_{t+1} \leftarrow \mathrm{Simulation}(W_t, S_t)$; //Simulate execution tasks in scheduling interval I_t
14: $\check{W}_{t+1} \leftarrow \mathrm{Adahessian}(D(G_t, W_t, S_t; \theta))$;
15: $f_t = ||\mathrm{ReLU}(W_{t+1} - \check{W}_{t+1})||$;
16: $y_t = 1(f_t \geq \mathrm{POT}(W_t))$; //Fault label
17: Compile time-series windows of all brokers as W_t;
18: Faulty nodes $\mathrm{FE} \subseteq E$ that have $y_t = 1$;
19: Adding fault label y_t in V_t^E;
20: $\mathrm{Loss} = \log D(G_t, W_{t+1}, S_t; \theta) + \log(1 - D(G_t, \check{W}_{t+1}, S_t))$;
21: Fine-tune $D(.;\theta)$ using Loss and Adam.
22: **end for**
23: **end for**

Since the time complexity of DFGP algorithm is re-

lated to GON model and DDPG algorithm model, suppose that the time complexity of GON model is $O(K_1)$, and K_1 depends on the number of nodes of input layer, the number of hidden layers, the number of output layer nodes, and the number of training iterations in the neural network model. Moreover, the number of samples and the number of neural network layers are much smaller than the number of training turns and steps, the time complexity of DDPG algorithm can be approximatively expressed as $O(\mathrm{MT})$. Since the time complexity of task simulation execution is constant and negligible, the time complexity of Adahessian algorithm is determined by the computational gradient and the complexity of the Hessian matrix, and the computational complexity can be expressed as $O(T_2 \times (N + N^2))$, where T_2 is the number of iterations and N is the number of model parameters. Therefore, the total time complexity is $O(K_1 + \mathrm{MT} + (T_2 \times (N + N^2)))$, and the simplification is approximately $O(K_1 + \mathrm{MT} + T_2 N^2)$.

V. Performance Evaluation

1. Experimental settings

In this section, we employ the Shanghai Telecom's dataset [29], [30] for testing and validating our proposed method. The dataset contains geographic locations of 7000 edge servers, resource request times to those servers, and mobile trajectories of requesters as shown in Figure 4. The selected edge nodes at different times are shown in Figure 5. Selected taxi trajectories are marked red and edge nodes are marked blue. For the LEI groups, we divided the edge servers contained in Telecom dataset into 15 LEI groups according to the 15 districts in Shanghai, as shown in Figure 6.

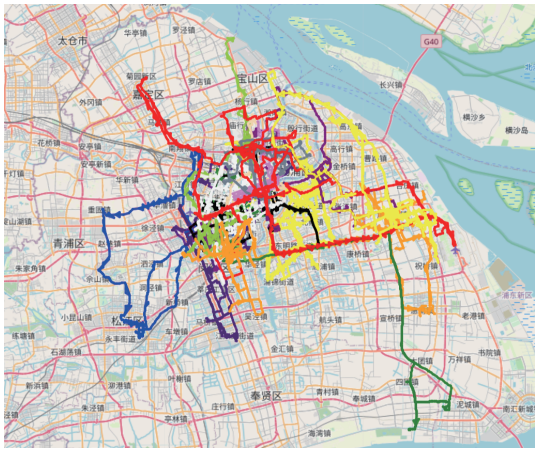


Figure 4 The dataset of Shanghai Telecom.

In addition, we use the fog time series anomaly detection (FTSAD) datasets [22], as shown in Table 2. It maintains fault records of CPU overloads, RAM contentions, Disk attacks, and DDoS attacks. The detail experimental parameters are shown in Table 3. We use the BWGD workload dataset [31] as well, which maintains

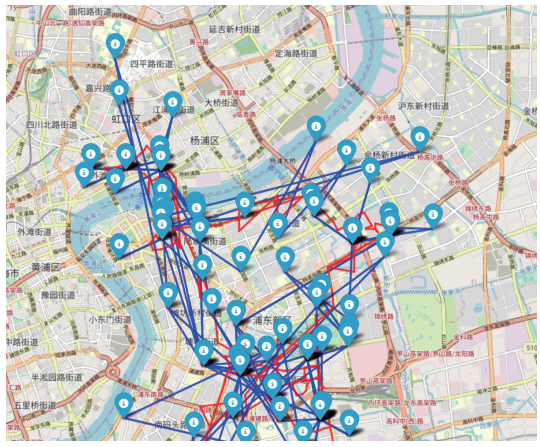


Figure 5 User trajectories and selected edge nodes.

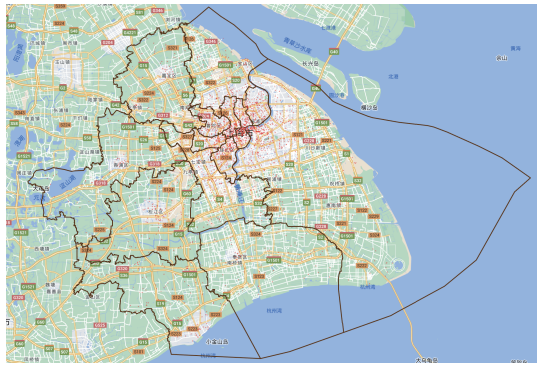


Figure 6 The fifteen districts of Shanghai.

workload records of 1750 services. We compare the proposed method against PBFM [10], IOTEF [32], TBAFT [9], TopoMAD [13], StepGAN [33], and DRAGON [34].

Table 2 Dataset statistics

Dataset	Train	Test	Dimension	Anomalies (%)
FTSAD-1	600	5000	1	12.88
FTSAD-25	574	1700	25	32.23
FTSAD-55	2158	2264	55	13.69

Table 3 Experiment parameters

Description	Value
Learning rate	0.004
Weight decay	0.0005
The number of neurons in feed-forward networks	128
Optimizer	Adam
POT coefficient	0.004
POT low quantile	0.07
Window size k	15
Minibatch size	10
Scheduling interval Δ	0.0001
ϕ	0.01
ξ	0.1
α, φ, δ, η	0.25, 0.25, 0.25, 0.25

2. Empirical results

In our experiment, we first compare our GON prediction model with other anomaly detection methods such as MAD-GAN [35], SlimGAN [36], USAD [37], CAE_M [38], and LSTM_AD [39] method, upon FS-TAD datasets.

Figure 7 shows how the accuracy and effectiveness of the classification of GON improve with rounds of training, where the y-axis indicates edge nodes and x-axis intervals, i.e., the recovery strategy anomaly scores.

Figure 8 demonstrates the results of precision and F1 scores for different dimension anomaly time series datasets. On average, the precision of the GON is 0.993 and F1 is 0.881. GON outperforms its peers in terms of precision and F1, as well as training times.

Figure 9 shows the comparison between DFGP algorithm and other comparison algorithms in terms of task migration number. The scheduling interval is 100. We can see that the DFGP algorithm on average shows 17.2%, 8.2%, 8.9%, and 17.2% increases than DRAGON, IoTEF, PBFM, and StepGAN, respectively, while 61.4% and 78%

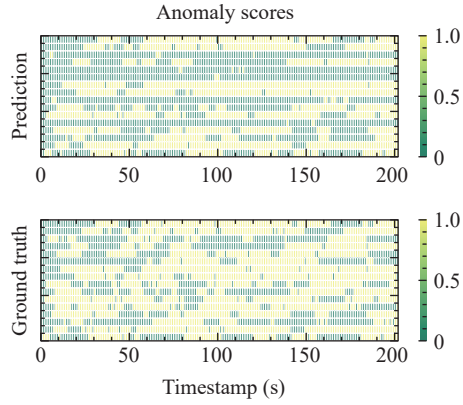


Figure 7 Fault prediction with ground truth labels.

decreased than TopoMAD and TBAFT methods, respectively. This is because different fault prediction models calculate the abnormal scoring criteria and the threshold of abnormal selection is different. DFGP algorithm uses POT method to determine the threshold. From the perspective of task migration number, DFGP shows better stability and load-balance than other as well.

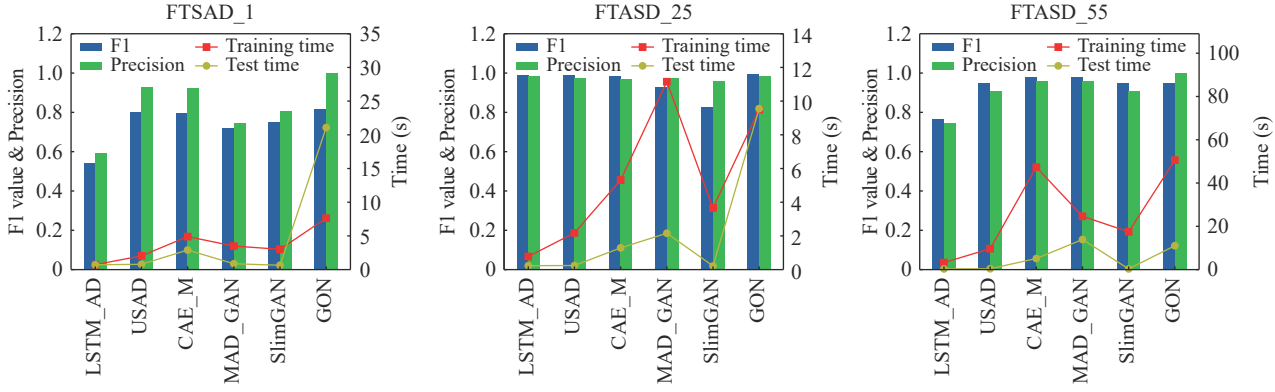


Figure 8 The F1 value and precision.

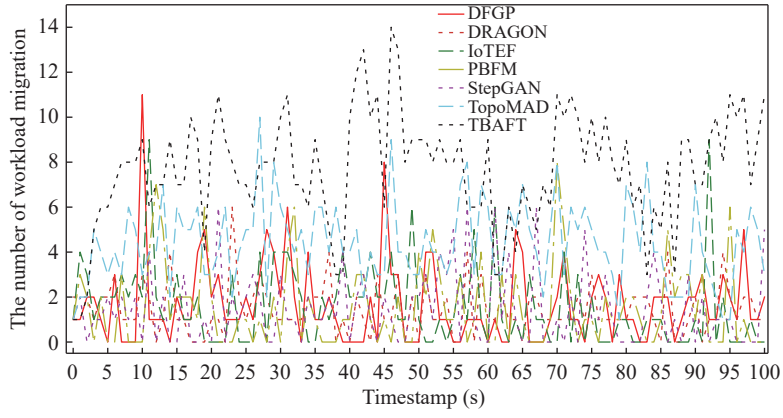


Figure 9 The number of workload migration.

Figure 10 demonstrates the averaged migration times of different approaches. It can be seen that the average migration time of DFGP is 82.1%, 76.6%, 62.4%, 73.4%, 91.7%, and 94.5% lower than its peers, respectively. This is because the DFGP method can quickly detect the failure or interruption of edge nodes and immediately trigger the task migration process, thus reducing migration time. This shows that DFGP fault-tolerant method ex-

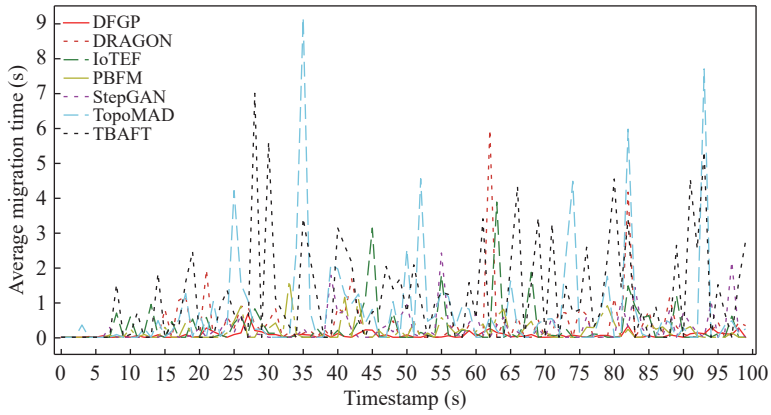


Figure 10 Average migration time.

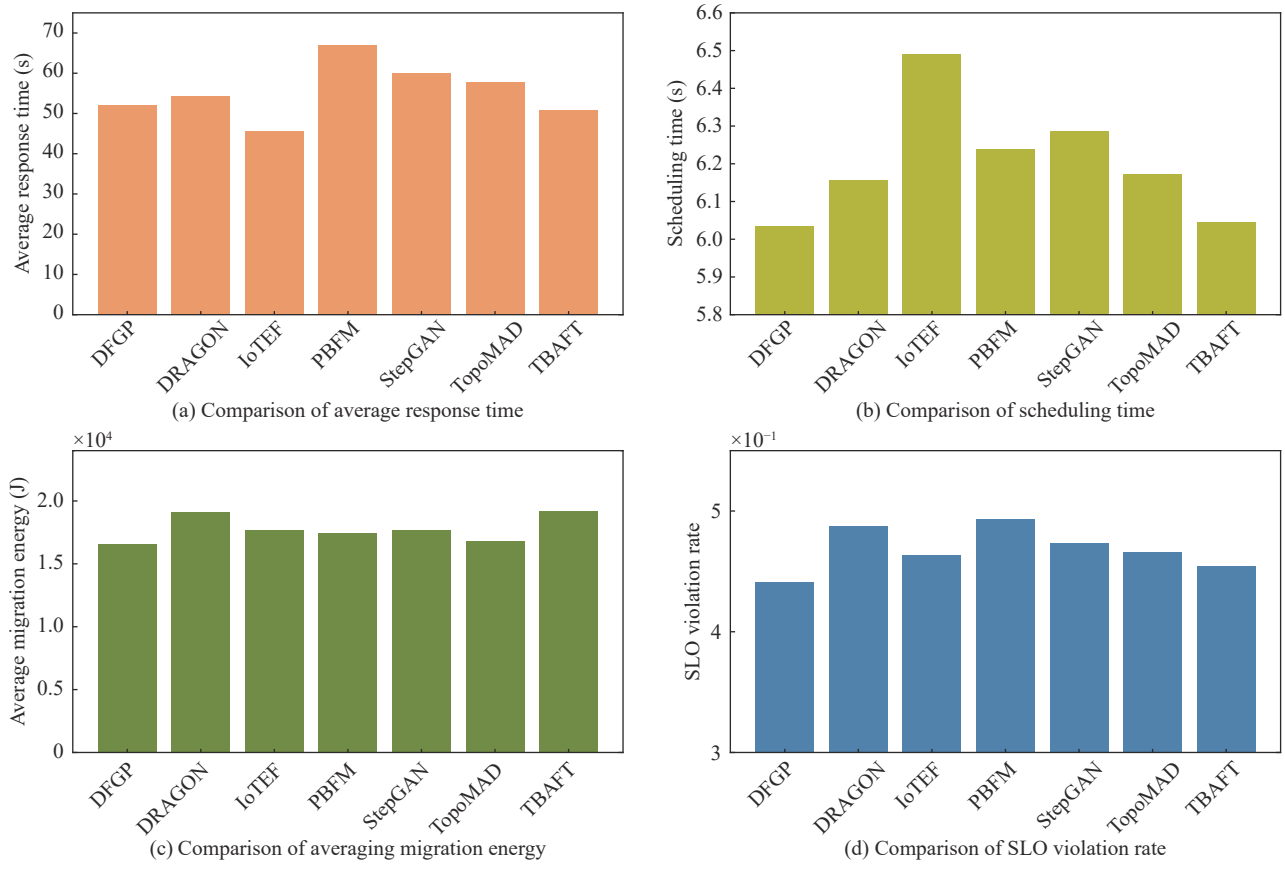


Figure 11 Comparison of QoS parameters of DFGP against baselines.

hibits better stability than its peers as well.

Figure 11(a) illustrates the averaged response time of different fault-tolerant methods. DFGP is 4.0%, 22.0%, 13.1% and 9.6% decreases than DRAGON, PBFM, StepGAN, and TopoMAD, respectively, while 14.2% and 2.5% increases than IoTEF and TBAFT, respectively. As indicated in Figure 11(b), DFGP shows 2.0%, 7.0%, 3.3%, 4.0%, 2.2%, and 0.2% decreases in the scheduling time than its peers, respectively. As indicated in Figure 11(c), DFGP shows 13.2%, 6.6%, 5.2%, 6.2%, 1.5%, and 1.4% decreases of the migration time than its peers. And Figure 11(d) indicates that DFGP shows 9.5%, 4.8%, 10.5%, 6.9%, 5.4%, and 3.0% decreases of SLA violations than its peers.

VI. Conclusion

In this paper, we develop a novel deep policy gradient-based fault tolerance approach for enforcing fault-tolerance of mobile services in mobile edge computing (MEC). It synthesizes a generative optimization network (GON) model for predicting and optimizing the task scheduling of edge devices and a deep deterministic policy gradient (DDPG) model for yielding preemptive migration decisions. We conducted evaluation experiments on the real-workd data set and empirical results show that the proposed DFGP method is more effective in fault detection

and guaranteeing quality of service (QoS) than its peers, in terms of multiple metrics. In the future, we plan to use decentralized method for task offloading problem in end-edge-cloud system [40].

Acknowledgements

This work was supported by the National Key R&D Program of China (Grant No. 2018YFB1403602), the Chongqing Technological Innovation Foundations (Grant Nos. cstc2019jscx-msxm0652 and cstc2019jscx-fxyd0385), the Chongqing Key RD Project (Grant No. cstc2018jszx-cyzdX0081), the Jiangxi Key RD Project (Grant No. 2018 1ACE50029), the Postgraduate Research and Innovation Project of Chongqing (Grant No. CYB22064), the Graduate Research and Innovation Foundation of Chongging (Grant. No. CYS22112), the Technological Program Organized by SGCC (Grant No. 52094020000U), and the Technology Innovation and Application Development Foundation of Chongqing (Grant No. cstc2020jscx-gksbX0010).

References

[1] Y. Chen, J. Zhao, X. K. Zhou, *et al.*, "A distributed game theoretical approach for credibility-guaranteed multimedia data offloading in MEC," *Information Sciences*, vol. 644, article no. 119306, 2023.

[2] Y. Chen, J. Zhao, J. T. Hu, *et al.*, "Distributed task offloading and resource purchasing in NOMA-enabled mobile edge computing: Hierarchical game theoretical approaches," *ACM Transactions on Embedded Computing Systems*, vol. 23, no. 1, article no. 2, 2024.

[3] X. F. Cao, G. M. Tang, D. K. Guo, *et al.*, "Edge federation: Towards an integrated service provisioning model," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1116–1129, 2020.

[4] C. Engelmann, G. R. Vallee, T. Naughton, *et al.*, "Proactive fault tolerance using preemptive migration," in *Proceedings of the 2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, Weimar, Germany, pp. 252–257, 2009.

[5] F. Z. Liu, J. W. Huang, and X. B. Wang, "Joint task offloading and resource allocation for device-edge-cloud collaboration with subtask dependencies," *IEEE Transactions on Cloud Computing*, vol. 11, no. 3, pp. 3027–3039, 2023.

[6] J. W. Huang, J. Y. Wan, B. F. Lv, *et al.*, "Joint computation offloading and resource allocation for edge-cloud collaboration in internet of vehicles via deep reinforcement learning," *IEEE Systems Journal*, vol. 17, no. 2, pp. 2500–2511, 2023.

[7] Y. Chen, J. T. Hu, J. Zhao, *et al.*, "QoS-aware computation offloading in LEO satellite edge computing for IoT: A game-theoretical approach," *Chinese Journal of Electronics*, in press, doi: 10.23919/cje.2022.00.412, 2023.

[8] J. L. Liu, S. G. Wang, A. Zhou, *et al*, "Using proactive fault-tolerance approach to enhance cloud service reliability," *IEEE Transactions on Cloud Computing*, vol. 6, no. 4, pp. 1191–1202, 2018.

[9] A. Rawat, R. Sushil, A. Agarwal, *et al.*, "A new adaptive fault tolerant framework in the cloud," *IETE Journal of Research*, vol. 69, no. 5, pp. 2897–2909, 2023.

[10] B. K. Ray, A. Saha, S. Khatua, *et al.*, "Proactive fault-tolerance technique to enhance reliability of cloud service in cloud federation environment," *IEEE Transactions on Cloud Computing*, vol. 10, no. 2, pp. 957–971, 2022.

[11] P. Y. Zhang, S. Shu, and M. C. Zhou, "An online fault detection model and strategies based on SVM-grid in clouds," *IEEE/CAA Journal of Automatica Sinica*, vol. 5, no. 2, pp. 445–456, 2018.

[12] X. F. Hu, Y. B. Li, L. Jia, *et al.*, "A novel two-stage unsupervised fault recognition framework combining feature extraction and fuzzy clustering for collaborative AIoT," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 2, pp. 1291–1300, 2022.

[13] Z. L. He, P. F. Chen, X. Y. Li, *et al.*, "A spatiotemporal deep learning approach for unsupervised anomaly detection in cloud systems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 4, pp. 1705–1719, 2023.

[14] S. Tuli, G. Casale, and N. R. Jennings, "PreGAN: Preemptive migration prediction network for proactive fault-tolerant edge computing, " in *Proceedings of 2022 IEEE Conference on Computer Communications*, London, United Kingdom, pp. 670–679, 2022.

[15] P. Dasgupta, R. C. Chen, S. Menon, *et al.*, "Design and implementation of the clouds distributed operating system," *Computing Systems*, vol. 3, no. 1, pp. 11–46, 1990.

[16] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, *et al.*, "IFogSim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software:Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[17] A. Al-Shuwaili, O. Simeone, A. Bagheri, *et al.*, "Joint uplink/downlink optimization for backhaul-limited mobile cloud computing with user scheduling," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 3, no. 4, pp. 787–802, 2017.

[18] X. Chen, L. Jiao, W. Z. Li, *et al.*, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[19] D. Satria, D. Park, and M. Jo, "Recovery for overloaded mobile edge computing," *Future Generation Computer Systems*, vol. 70, pp. 138–147, 2017.

[20] K. J. Ye, Y. Y. Liu, G. Y. Xu, *et al.*, "Fault injection and detection for artificial intelligence applications in container-based clouds," in *Proceedings of the 11th International Conference on Cloud Computing*, Seattle, WA, USA, pp. 112–127, 2018.

[21] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

[22] S. Tuli, S. Tuli, G. Casale, *et al.*, "Generative optimization networks for memory efficient data generation," *arXiv preprint*, arXiv: 2110.02912, 2021.

[23] G. L. Liu, K. J. Shih, T. C. Wang, *et al.*, "Partial convolution based padding," *arXiv preprint*, arXiv: 1811.11718, 2018.

[24] P. Chen, H. Y. Liu, R. Y. Xin, *et al.*, "Effectively detecting operational anomalies in large-scale IoT data infrastructures by using a GAN-based predictive model," *The Computer Journal*, vol. 65, no. 11, pp. 2909–2925, 2022.

[25] Z. W. Yao, A. Gholami, S. Shen, *et al.*, "ADAHESSIAN: An adaptive second order optimizer for machine learning," in *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, Virtual Event, pp. 10665–10673, 2021.

[26] S. Zhang, H. H. Tong, J. J. Xu, *et al.*, "Graph convolutional networks: A comprehensive review," *Computational Social Networks*, vol. 6, no. 1, article no. 11, 2019.

[27] D. Silver, G. Lever, N. Heess, *et al.*, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning*, Beijing, China, pp. I-387–I-395, 2014.

[28] A. Siffer, P. A. Fouque, A. Termier, *et al.*, "Anomaly detection in streams with extreme value theory," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Halifax, NS, Canada, pp. 1067–1075, 2017.

[29] S. G. Wang, Y. L. Zhao, J. L. Xu, *et al.*, "Edge server placement in mobile edge computing," *Journal of Parallel and Distributed Computing*, vol. 127 pp. , pp. 160–168, 2019.

[30] S. Y. Liu, Y. H. Liu, L. M. Ni, *et al.*, "Towards mobility-based clustering," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data*

Mining, Washington, DC, USA, pp. 919–928, 2010.
[31] S. Q. Shen, V. V. Beek, and A. Iosup, "Statistical characterization of business-critical workloads hosted in cloud datacenters," in *Proceedings of the 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, Shenzhen, China, pp. 465–474, 2015.
[32] A. Javed, J. Robert, K. Heljanko, *et al.*, "IoTEF: A federated edge-cloud architecture for fault-tolerant IoT applications," *Journal of Grid Computing*, vol. 18, no. 1, pp. 57–80, 2020.
[33] Y. Feng, Z. J. Liu, J. L. Chen, *et al.*, "Make the rocket intelligent at IoT edge: Stepwise GAN for anomaly detection of LRE with multisource fusion," *IEEE Internet of Things Journal*, vol. 9, no. 4, pp. 3135–3149, 2022.
[34] S. Tuli, G. Casale, and N. R. Jennings, "DRAGON: Decentralized fault tolerance in edge federations," *IEEE Transactions on Network and Service Management*, vol. 20, no. 1, pp. 276–291, 2023.
[35] D. Li, D. C. Chen, B. H. Jin, *et al.*, "MAD-GAN: Multivariate anomaly detection for time series data with generative adversarial networks," in *Proceedings of the 28th International Conference on Artificial Neural Networks*, Munich, Germany, pp. 703–716, 2019.
[36] L. Hou, Z. H. Yuan, L. Huang, *et al.*, "Slimmable generative adversarial networks," in *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, Virtual Event, pp. 7746–7753, 2021.
[37] J. Audibert, P. Michiardi, F. Guyard, *et al.*, "USAD: UnSupervised anomaly detection on multivariate time series," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, New York, NY, USA, pp. 3395–3404, 2020.
[38] Y. X. Zhang, Y. Q. Chen, J. D. Wang, *et al.*, "Unsupervised deep anomaly detection for multi-sensor time-series signals," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 2, pp. 2118–2132, 2023.
[39] Z. W. Ji, J. H. Gong, and J. R. Feng, "A novel deep learning approach for anomaly detection of time series data," *Scientific Programming*, vol. 2021, article no. 6636270, 2021.
[40] Y. Chen, J. Zhao, Y. Wu, *et al.*, "QoE-aware decentralized task offloading and resource allocation for end-edge-cloud systems: A game-theoretical approach," *IEEE Transactions on Mobile Computing*, vol. 23, no. 1, pp. 769–784, 2024.

Tingyan LONG received the M.S. degree with the School of Computer Science & Technology, Guizhou University, Guiyang, China, in 2019, and the Ph.D. degree from the College of Computer Science, Chongqing University, Chongqing, China, in 2023. Currently, she works at the School of Computer Science and Technology, Guizhou University, Guiyang, China. Her research interests include fault-tolerant, edge computing, and reinforcement learning method.
(Email: l5469369@hotmail.com)

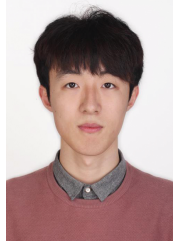
Peng CHEN received the B.E. degree in computer science and technology from University of Electronic Science and Technology of China, Chengdu, China, in 2001, the M.S. degree in computer software and theory from Peking University, Beijing, China, in 2004, and the Ph.D. degree in computer science and technology from Sichuan University, Chengdu, China, in 2017. He is currently a Professor with the School of Computer and Software Engineering, Xihua University, Chengdu, China. His research interests include machine learning, service computing, and time series analysis.
(Email: chenpeng@mail.xhu.edu.cn)

Yunni XIA received the B.S. degree in computer science from Chongqing University, Chongqing, China, in 2003, and the Ph.D. degree in computer science from Peking University, Beijing, China, in 2008. He is currently a Professor with the College of Computer Science, Chongqing University, Chongqing, China. He is the author or co-author of more than 100 research publications. His research interests are in Petri nets, software quality, performance evaluation, and edge computing and cloud computing system dependability.
(Email: xiayunni@hotmail.com)

Yong MA received the M.S. degree in computer science from Xidian University, Xi'an, China, in 2003, and Ph.D. degree in computer science from Wuhan University, Wuhan, China, in 2006. In 2018, he worked on the integrated control and dispatching of energy in microgrid with Malardalens University, Sweden. He is now a Professor with the School of Computer and Information Engineering, Jiangxi Normal University, Nanchang, China. His current research focuses on cloud computing, edge computing, and data science.
(Email: mywuda@126.com)

Xiaoning SUN received the B.S. degree in computer science and the Ph.D. degree in software engineering from in 2015 and 2022, respectively, both from Chongqing University, Chongqing, China. Since July 2022, she has been a Lecturer with the School of Computer and Information Science, Chongqing Normal University, Chongqing, China. Her research interests include service computing, performance evaluation, and edge computing.
(Email: sxiaoning@hotmail.com)

Jiale ZHAO received the B.E. degree in information security from Huaibei Normal University, Huaibei, China, in 2017, and the M.S. degree in computer technology from Jiangxi Normal University, Nanchang, China, in 2021. He is currently pursuing the Ph.D. degree in computer science and technology with Chongqing University, Chongqing, China. His research interests include fault-tolerance, edge computing, and cloud computing.
(Email: zhaojiale0415@163.com)



Yifei LYU is currently pursuing the M.S. degree with the College of Computer Science, Chongqing University, Chongqing, China. His research interests include the areas of cloud computing, edge computing, and reinforcement learning method.
(Email: yuzhe1334021@163.com)