Hindawi
Wireless Communications and Mobile Computing
Volume 2023, Article ID 2753085, 13 pages
https://doi.org/10.1155/2023/2753085

Research Article

A Network Fault Prediction-Based Service Migration Approach for Unstable Mobile Edge Environment

Haiyan Wang, Weihao Tang, and Jian Luo

Jiangsu Key Laboratory of Big Data Security and Intelligent Processing, Nanjing University of Posts and Telecommunications, Nanjing, China

Correspondence should be addressed to Haiyan Wang; wanghy@njupt.edu.cn

Received 26 September 2022; Revised 11 November 2022; Accepted 3 June 2023; Published 8 August 2023

Academic Editor: Peiying Zhang

How to perform efficient service migration in a mobile edge environment has become one of the research hotspots in the field of service computing. Most service migration approaches assume that the mobile edge network on which the migration depends is stable. However, in practice, these networks often fluctuate greatly due to the fault of edge devices, resulting in unexpected service interruptions during the migration process. Besides, most of the existing solutions do not consider the migration cost and path selection in the event of edge network fault. Aiming at the above problems, we propose a service migration approach based on network fault prediction (SMNFP) for mobile edge environment. The SMNFP method first introduces the software-defined network as a global controller, which is used to monitor and collect the changing of the edge network and schedule the migration tasks. Second, a network fault prediction model based on Wide&Deep model is proposed to predict the upcoming faults in the network according to the alarm information of network equipment. Finally, the service migration problem is constructed as a Markov decision process, and a fault penalty function is introduced to avoid faulty nodes, together with the deep Q-learning method to solve the migration strategy. Simulation experiments are conducted on the public metro network fault dataset, and results show the proposed method can effectively predict network faults and complete service migration.

1. Introduction

In recent years, the development of mobile Internet has enhanced the performance of the network, such as bandwidth, transmission rate, and throughput rate. Mobile edge computing (MEC) allows us to deploy servers geographically closer to users, provide computing power closer to smartphones or various types of mobile terminals, and sink these computing power into base stations. However, in MEC, the limitations of edge server coverage, the mobility of edge end users, and the differences in mobile requests in different regions often cause load imbalance between servers, which in turn leads to service quality degradation and even service interruptions [1, 2].

To ensure the continuity of services when users move, service migration technology has begun to receive extensive attention. In the mobile edge network scenario, service migration refers to migrating the application services used by the user from the connected edge server according to a certain algorithm or decision-making mechanism under the premise of ensuring the minimum cost and delay during the rapid movement of the user to the best server at different times.

Existing service migration methods usually assume that the user's moving path is known [3, 4], and some research work has predicted the user's mobility, using mobility prediction and perception methods to carry out their work [5, 6], in which services are premigrated to relevant areas, effectively reducing the migration workload. In addition, some researchers use Markov decision process to model service migration [7, 8], reducing the overall computational complexity of the model.

However, we emphasize that there are two main problems with the previous methods. One is that these methods assume the edge network in which they migrate is in a stable condition, and these methods are valid with a limitation to the fault-free edge network. In reality, the edge network is unstable. As shown in Figure 1, edge devices may cause the

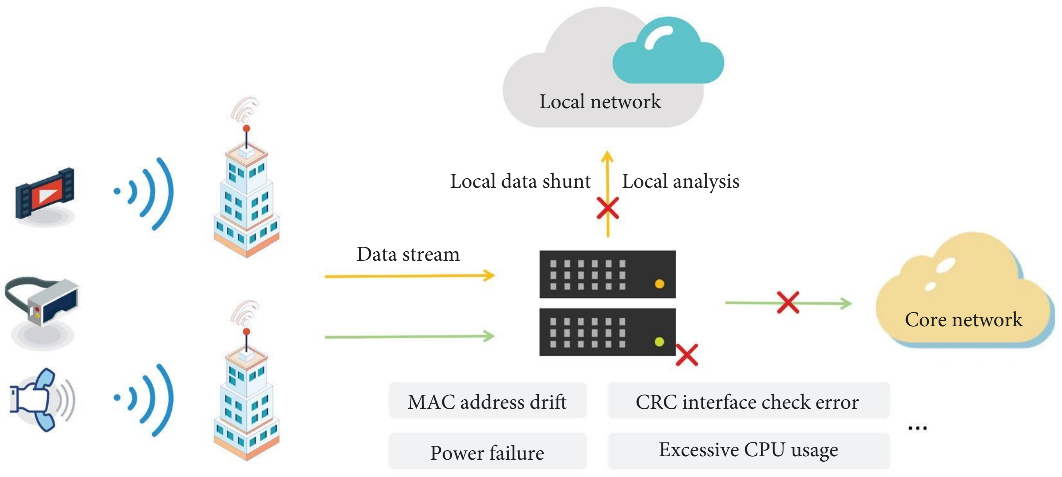


FIGURE 1: A network fault scenario caused by equipment faults.

temporary failure of edge nodes due to various reasons. The other is that most of the previous studies only consider the migration path selection problem in intact edge networks, lacking the collection and aggregation of all migration tasks and network topology information in edge networks and thus lacking a unified real-time scheduling means for service migration for different network situations.

To address the above issues, we employ the software-defined network (SDN) mechanism to predict the failures of the edge network. We then put forward the service migration approach based on network fault prediction (SMNFP) method to circumvent the faulty nodes and make reasonable migration path selection. The main contributions of this paper are as follows:

(i) First, we proposed an edge network fault prediction module network fault prediction model based on Wide&Deep model (NFP-WD), which is used to predict the fault of the entire edge network within a fixed time window and mark all edge servers that may fail before the next time window.
(ii) Second, we present the SMNFP method, in which the service migration problem is constructed as a Markov decision process, and a fault penalty function is designed to avoid faulty nodes in the migration path selection. Finally, the deep *Q*-learning method is used to solve the service migration strategy.
(iii) Finally, we introduced the SDN framework as the migration controller for the model as a whole. Then we conduct simulation experiments on the MAN fault data set, and the experimental results show that our SMNFP method has a better migration effect than the baseline method.

2. Related Work

2.1. Service Migration. In terms of service migration, many researches and methods are aimed at cost balancing and optimization in the migration process. Liang et al. [9] used a combinatorial optimization algorithm and integer relaxation iterative algorithm to optimize the offload rate, mobility, and MEC throughput of services in cellular networks, which indirectly reduced the migration cost. Wang et al. [10] investigate a user-centric service migration and exit point selection problem which introduces a neural network-based smart migration judgment to navigate the performance and computation overhead tradeoff. Park et al. [4] formalized the migration cost, communication cost, and energy consumption associated with the migration process as a complex optimization problem, employing deep reinforcement learning to approximate the optimal policy. Wang et al. [8] designed a service migration framework Mig-RL by using the reinforcement learning method; when encountering similar migration patterns, the migration strategy can be directly retrieved, which significantly reduces the decision-making cost.

Another part of the research work is based on user mobility perception and prediction. Yin et al. [5] proposed a mobility-aware service migration mechanism, which selects the target node for migration according to the migration cost and the moving direction. Labriji et al. [6] used the mobility prediction method to solve the problem of vehicle service migration. The method combines neural network and Markov chain for vehicle mobility prediction, which can still maintain a good performance in the scene of large-scale traffic flow. Xu et al. [11] proposed a service migration method based on the Bernoulli test and made a quantitative analysis of delay and user mobility prediction through theoretical analysis and simple probability statistics, which effectively reduced service communication delay and migration cost. Miao et al. [12] proposed a mobility-enabled service migration scheme, called MSM, for real-time decision-making on service migration.

2.2. Network Fault Prediction. At present, the related research on network fault prediction is mainly based on the methods of machine learning and deep learning. In terms of machine learning, Lin et al. [13] predicted faults in smart distribution networks by introducing multiple support vector machines (SVMs) and an improved voting random forest algorithm, which improved the accuracy rate and recall rate. Yadwad and Vatsavayi [14] combine hidden Markov models with Bayesian networks for outage prediction of network devices. In terms of deep learning, Google's Wide&Deep model [15] is widely used in the field of recommender systems; some researchers [16] used the Wide&Deep model to carry out network fault prediction work. In addition, Klein et al. [17] used two-dimensional convolutional neural networks to extract temporal feature data streams and then used graph convolutional neural networks to extract spatial features, combined with domain expert knowledge to jointly predict network faults. Tefera et al. [18] used long short-term memory network (LSTM) and gated recurrent unit for early prediction of base transceiver stationfaults caused by power system and environmental anomalies.

2.3. Deep Reinforcement Learning. Reinforcement learning is an important machine learning method. It takes the next action based on the feedback of the environment, through constant interaction and trial and error with the environment, and achieves the final goal in the case of obtaining the maximum benefit as a whole. Q-learning is a typical reinforcement learning method based on Q value. Wang et al. [8] leveraged the Q-learning approach to design a service migration framework for reducing the total service cost in mobile edge environments. However, in the actual service migration environment, the edge network environment is relatively complex, which easily leads to large state space. Therefore, it is unrealistic for reinforcement learning to store action values through a Q table. To solve this problem, Mnih et al. [19] first combined the convolutional neural network and Q-learning method and proposed a deep Q-network (DQN) model for processing visual perception-based processing. The control task is a pioneering work in the field of deep reinforcement learning. It not only has the perception ability of deep learning but also has the decision-making mechanism in reinforcement learning. van Hasselt et al. [20] innovatively proposed the deep DQN algorithm to improve the problem of overestimating Q value in deep reinforcement learning, which is more accurate in Q value estimation. Subsequent researchers [21] added a recurrent neural network structure to the DQN model, which enabled the model to have time memory capabilities and better process time-series data. At present, the models of deep reinforcement learning are developed in the direction of structural diversification and complex modules. There are many kinds of deep learning methods that can be integrated into reinforcement learning [22, 23]. By adding an attention mechanism to the model, the intelligent physical ability makes more reasonable judgments according to the importance of the system environment and state space, that is, automatic decision-making and tuning.

3. Proposed NFP-WD Model

In the wide model part, we introduce the field-aware factorization machine (FFM) to process the characteristics of conventional network alarm logs. In the deep model part, we use LSTM to process features with time series in device alarm data. The proposed model structure of the NFP-WD is shown in Figure 2.

3.1. Improved Wide Model Based on FFM. The Wide side of the model uses the FFM to deal with a large number of sparse features in the network alarm log data. The linear model of combined features simply considers each feature independently and does not consider the relationship between features. Therefore, we consider using the field-aware factorization machine model FFM to characterize the correlation between features.

There are some sparse features belonging to the same field in the actual data. For example, in the network alarm log data, "alarm level" belongs to a general field feature, which consists of fields such as "prompt," "important," "minor," "urgent," etc. When combining features, we should generalize these sparse features into the same feature field. The field-aware factorization machine can divide the same features into the same field, and the output of the FFM can be expressed as follows:

$$\phi_{\mathrm{FFM}}(w, x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} < \mathrm{v}_{i,f_j}, \mathrm{v}_{j,f_i} > x_i x_j. \tag{1}$$

In which, ω_0 is the initial weight, for each one-dimensional feature component x_i, the model automatically learns an implicit vector $\mathbf{V}_{i,f_j}$ for the field f_j where the other feature is located. Using the FFM model as the structure on the wide side can make the model generates multiple independent latent vectors better and learn new warning features. After that, the output of the FFM will be connected to the fully connected layer, and the fully connected layer will extract the cross features generated by the FFM.

3.2. Improved Deep Model Based on LSTM. The deep side of the model adopts the LSTM to train the time series features of the network fault alarm information in the edge environment. LSTM saves the past state information by introducing the unit state c_t, where the forgotten gate f_t determines the content that needs to be forgotten in the unit state, and the input gate i_t determines the content that needs to be newly added to the unit state.The output gate o_t is used to decide whether the cell state c_t will be propagated to the final state h_t. The relevant recursive equations are as follows:

$$\begin{aligned} i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) \\ f_t &= \sigma\left(W_f[h_{t-1}, x_t] + b_f\right) \\ o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\ c_t &= f_t \times c_{t-1} + i_t \times \tanh(W_c[h_{t-1}, x_t] + b_c) \\ h_t &= o_t \times \tanh(c_t), \end{aligned} \tag{2}$$

where i_t, f_t, o_t, c_t, h_t represent the input gate, forget gate, output gate, unit state, and hidden state, respectively, b_i, b_f, b_o, b_c are their corresponding bias terms.σ, tanh represent the sigmoid activation function and the tanh activation function, respectively.

The fully connected layer of the model combines the output of static features after passing through the wide

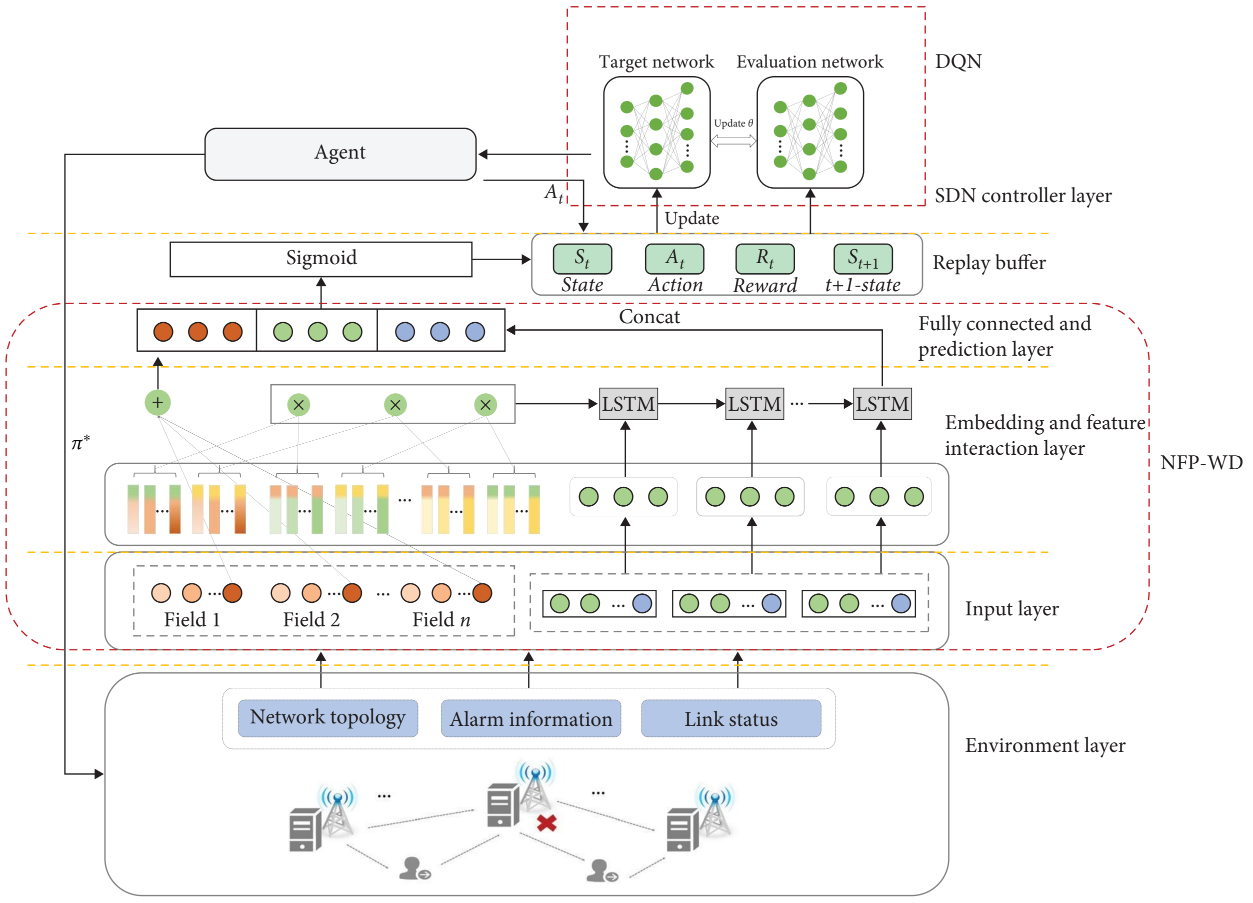


Figure 2: The architecture of SMNFP.

module and the output of dynamic time series features after passing through the deep module and uses the sigmoid activation function to output the probability value of edge network fault prediction in each time window. The output can be formally expressed as follows:

$$y_{\text{con}} = \text{sigmoid}\left(w_{\text{con}} \cdot \text{concat}\left(y_{\text{wide}}, y_{\text{deep}}\right) + b_{\text{con}}\right), \quad (3)$$

where concat is a combination function, which is used to perform vector splicing of the processing output of static nontemporal features and the output of time-series features in each time window. y_{wide} and y_{deep} are the outputs of the FFM and LSTM neural networks, and w_{con} and b_{con} are the weight and bias parameters to be trained.

3.3. Loss Function and Optimization. Our objective function consists of Wide model part and Deep model part. In the Wide part, we use the field-aware factorization machine to cross-feature combinations and generate new alert features. The model uses logistic loss as the loss function and uses the L2 penalty term. To avoid overfitting, L2 penalty term is introduced to penalize the weights of the model, encouraging the model to prefer smaller weight values, thereby reducing model complexity. At the same time, it prompts the model to assign smaller weights to irrelevant or redundant features, improving the model's generalization ability.

The optimized loss function is as follows:

$$L_W = \sum_{p=1}^{N} \log\left(1 + \exp\left(-y_p \phi\left(w, x_p\right)\right)\right) + \frac{\lambda}{2}\|w\|_2^2, \quad (4)$$

where $y_p \in \{0, 1\}$ is the label of the pth sample. λ is the regularization coefficient.

Using the LSTM neural network to predict whether the edge network will fail in the next time period is essentially a binary classification problem, and the loss function can be expressed as follows:

$$L_D = -\sum_{i=1}^{N} y \log \hat{y} + (1 - y) \log\left(1 - \hat{y}\right), \quad (5)$$

where N is the total number of samples, y is the real label of the sample fault, $\hat{y}$ is the probability value that the model predicts the sample to be a positive class value. The NFP-WD model outputs the probability value of network fault in each time window, which leads to corresponding local errors at each step. For the problem of fault prediction, the focus should be the output probability of the model in the last step. Therefore, by adjusting the proportion of the prediction probability of the last step in the global, the object that the loss function should focus on is controlled. The optimized loss function is as follows:

$$L_{D^*} = \frac{1}{N}(1-\alpha)L_D + \frac{1}{T}\alpha L_D. \quad (6)$$

Among them, T is the length value of the input sequence, and the hyperparameter $\alpha \in \{0,1\}$is used to control the importance of the output in the prediction process to the final prediction result. The overall loss function of the final model is as follows:

$$L = L_w + L_{D^*}. \quad (7)$$

The goal of NFP-WD model training is to minimize the loss function L. Based on the above design, the Wide model and the Deep model are combined through a fully connected layer, and the final network fault prediction value is obtained after joint training.

4. Proposed Service Migration Method

We first introduce an SDN controller into the mobile edge network and use the controller to monitor the operation of all edge servers, collect all observable computing tasks and network device alarm information, and predict the faults of network equipment in each time window according to the alarm information. When a user moves from one location to another, service migration will be triggered. In order to avoid passing through faulty servers during the migration process, we introduce the NFP-WD module to avoid servers that are about to fail by setting a reasonable reward function; finally, we use deep reinforcement learning to solve the service migration strategy. The overall architecture of the model is shown in Figure 2.

4.1. Service Migration Model Based on Markov Decision Process. We adopt a deep reinforcement learning method to solve the problem of service migration. Our method is based on time windows, that is, each time window t is regarded as a sampling interval, and in each sampling interval, network faults are predicted according to the edge network conditions, and the corresponding service migration decisions are made. Reinforcement learning problems can be formally represented by quintuples of Markov decision processes: $M = (S, A, P, R, \gamma)$, where S is the state space, representing a set of state states, and A is a set of actions, $P(s'|s, a)$ represents the transition probability of taking action a in state s and transitioning to state s′. R represents the reward function. γ represents the discounting factor.

4.1.1. State Space S and Action Set A. Suppose the edge network consists of edge servers with N nodes, denoted as $N = (1, 2, \ldots, n)$, the service runs on K servers, the collection of these services is represented as $SE = (se_1, se_2, \ldots, se_k)$. Define a set of nodes $N_f = (f_1, f_2, \ldots, f_n)$ indicates the nodes that will fail after being predicted by the NFP-WD module within a specific time window, during the actual migration process, these nodes will be avoided according to a certain migration strategy. Assume that at a certain moment the user enjoys the service S_{et} provided by the edge node N_t, we define $s(t)$ as the distance between user u at time slot t and the edge server N_t serving it: $s(t) = \|loc_{u_t} - loc_{N_t}\|$, where loc_{u_t} represents the location of user, loc_{N_t} represents the location of edge server. State space $S = \{s(t), t = 1, 2, \ldots, n\}$. In each time window t, the state changes from $s(t)$ to $s'(t)$ after taking action $a(s(t))$. Action set A is the set of these actions $a(s)$, where $a(s(t)) = \begin{cases} 0, & \text{No Service Migration.} \\ 1, & \text{Perform Service Migration.} \end{cases}$

4.1.2. Cost Constraints. We consider the migration cost and communication cost in the process of migrating services from a source server to a target server. Suppose the address of the origin server is l_{ori}, the target server address is l_{dest}, the user address is l_{user}. We measure the distance between two servers by the number of hops between two cellular networks: $\delta = \|l_{\text{ori}} - l_{\text{dest}}\|$, the distance between the user and the target server after the service migration is performed as $\tau = \|l_{\text{user}} - l_{\text{dest}}\|$. We define the migration cost function as $m(\delta) = \begin{cases} \omega_o + \omega_d \theta^\delta\ \delta > 0 \\ 0,\ \delta = 0 \end{cases}$. The communication cost function as $\text{n}(\tau) = \begin{cases} \mu_0 + \mu_d \lambda^\tau,\ \tau > 0 \\ 0,\ \tau = 0 \end{cases}$, where $\omega_{\text{o}}, \omega_{\text{d}}, \mu_{\text{o}}, \mu_{\text{d}}, 0 \le \lambda \le 1, \theta \ge 1$ are real values. So the total cost function is $C(s, a) = m(\delta) + n(\tau)$.

4.1.3. Reward Function. Suppose that in a certain state s, for a service to be migrated, there is an edge node sequence $N_s = \{N_1, N_2, \ldots, N_{\text{dest}}\}$ representing the migration path of the current service, $N_f = (f_1, f_2, \ldots, f_n)$ represents the set of nodes that may fail predicted by the NFP-WD module in the current state of the system. Define N_K to represent the set of nodes where the service has been deployed. In order to encourage the reinforcement learning mechanism to try to avoid faulty nodes in the migration decision, we define the fault penalty function Penalty(s). The value of the penalty function is determined by whether the faulty node is included in the current migration decision and the origin of the service request. For each state s in the state space S:

$$\text{Penalty}(s) = \sum_{f_i \in N_f} g(f_i, N_{\text{dest}}) + \sum_{n \in (N - N_K)} x_n \operatorname*{dis}_{\mu \in N_K} \{n, \mu\}, \quad (8)$$

where $g(f_i, N_{\text{dest}})$ indicates the number of paths affected by a single failed node f_i, in the network topology, it is expressed as the total number of paths without loops that reach the target node N_{dest} with the faulty node f_i as the starting point. x_n represents the total number of requests at node n. dis $\{n, \mu\}$ represents the shortest distance from node n to the first node of the deployed service. It can be seen that the penalty function is divided into two items. The first item indicates that if there is a faulty node in the migration path, the penalty will be obtained. If a service request is initiated at the edge node where the service is deployed, a penalty will be obtained, and the amount of penalty will also increase with the increase of the number of requests. So the final reward function is as follows:

$$R(s, a) = \text{Penalty } (s) - \text{Penalty } (s') - w_p C(s, a), \tag{9}$$

where $C(s, a)$ represents the cost function. If the state of the system is improved after the migration action is performed, it will receive a positive immediate reward, otherwise it will be punished. The migration strategy needs to strike a balance between the penalty function and the cost function, so we introduce a compromise weight factor w_p to achieve this purpose.

4.2. Service Migration Method Based on Deep Reinforcement Learning. Reinforcement learning is generally used in scenarios that need to interact with the environment. For a given state in the state space, the program selects a corresponding action according to a certain strategy. After the action is executed, the environment changes and the state changes to a new state. After each action is performed, the program will get a reward value, and then the program adjusts its strategy according to the size of the reward value. After all steps are executed, when the program reaches the terminal state, the sum of the rewards obtained is the largest, and the strategy obtained at this time the optimal strategy.

The Q-learning algorithm is a representative algorithm among value-based algorithms in reinforcement learning. Q (s, a) is a state-action value function in reinforcement learning, which represents the sum of the expected total rewards after taking action a in state s. The update process of Q (s, a) is as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right], \tag{10}$$

where $Q(s', a')$ is the expected total return after taking the next action a', $\alpha \in [0, 1]$ is the learning rate used to control the convergence of the model, r represents the reward obtained after taking action a in state s. $\gamma \in [0, 1]$ is discounting factor, which is used to control the degree of influence of the new Q value on the previous Q value.

However, Q-learning uses a Q table to store action values. In the service migration environment we constructed, in order to verify the impact of equipment fault on the migration effect, a large number of edge devices are required, which easily leads to an excessively large state space. Therefore, it is not practical to store the Q value of each time step by constructing a Q table. To solve this problem, we use the Deep Q Network (DQN) algorithm in deep reinforcement learning to calculate the Q value that can be obtained by selecting an action a for given state s. To prevent overfitting, DQN includes an evaluation neural network and a target neural network, which has the same structure but different weight vectors and corresponding biases of the depth neurons. Equation (12) has a similar structure to Equation (11), with the difference that the neuron weight vector in the evaluation network is θ, and the output is $Q(s, a; \theta)$, θ varies with each time step t. The parameters in the target network are the parameters $\hat{\theta}$ in the evaluation network some time ago, and the output is $\hat{Q}(s, a; \hat{\theta})$. After a period of time, the parameters of the evaluation network $\hat{Q}(s, a; \hat{\theta})$ are assigned to the target network. $\gamma \in [0, 1]$ is still discounting factor. The service migration algorithm is described as Algorithm 1. The update process of the action value function can be expressed as follows:

$$\begin{aligned} &\mathrm{Q}(s_t, a_t; \theta) \leftarrow \\ &\mathrm{Q}(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a_{t+1}} \hat{\mathrm{Q}}\left(s_{t+1}, a_{t+1}; \hat{\theta} \right) - \mathrm{Q}(s_t, a_t; \theta) \right]. \end{aligned} \tag{11}$$

In order to solve the problem of unstable training effect caused by the nonindependence of training samples, we use the experience replay buffer as the training method of neural network. Experience replay buffer refers to storing the quadruplets obtained during the training process in the experience pool and then randomly selecting a batch of quadruplets (s_t, a_t, r_t, s_{t+1}) from the experience pool as a batch for training. This random sampling can reduce the correlation between data samples and improve the training efficiency of the neural network. The loss function can be expressed as follows:

$$L(\theta) = E\left[\left(r_t + \gamma \max_{a_{t+1}} \hat{\mathrm{Q}}\left(s_{t+1}, a_{t+1}; \hat{\theta} \right) - \mathrm{Q}(s_t, a_t; \theta) \right)^2 \right]. \tag{12}$$

The execution time of Algorithm 1 is as follows:

$$\begin{aligned} &T(\text{episode } t) = t_0 + t_1 + \\ &(t_2 + t_{2.1} + (t_{2.2} + t_{2.2.1} + t_{2.2.2} + \ldots + t_{2.2.9}) \times T) \times E \\ &= t_{c1} + (t_{c2} + t_{c3} \times T) \times E \\ &= t_{c1} + (t_{c2} \times E + t_{c3} \times T \times E) \\ &= t_{c1} + t_{c2} \times E + t_{c3} \times T \times E \\ &= t_{c3} \times T \times E \\ &= T \times E. \end{aligned} \tag{13}$$

where t_0 represents the time required to initialize the experience pool. t_1 represents the time to initialize the evaluation network and the target network. For each episode, the execution time consumed is t_2, repeatedly running E times. Then, each individual step in Algorithm 1 corresponds to an independent step-by-step time. When the values of episodes and t of the algorithm are very large, the constant terms in T (episode, t) and the coefficients of T and E are negligible. The main influence of the T (episode, t) is not

```
Input:
    State set $S$, Action set $A$, discounting factor $\gamma$, explore probability $\epsilon$
Output:
    Migration strategy $\pi = (0, 1, 2, \ldots T)$.
1: Initialize the Experience Pool with a capacity of $M$
2: Initialize the evaluation network neuron weight vector $\theta$
3: Initialize the target network neuron weight vector $\hat{\theta} = \theta$, the rest of the parameters are the same as the evaluation network
4: for episode = 1, 2…$E$ do
5:     Initialize user location $loc_u$ and the location of edge server $loc_N$,initialize the first state $s_1$
6:     for $t$ = 1, 2…$T$ do
7:         Predict the faulty node $f$ and add it to the set of faulty nodes $N_f$
8:         Randomly choose action $a_t$ with probability $\epsilon$
9:         Or choose the action $a_t = \arg\max(s_t, a_t; \theta)$
10:        perform action $a_t$,calculate the penalty value p, reward value $r_t$ and the next moment state $s_{t+1}$
11:        Put the sample $|s_t, a_t, r_t, s_{t+1}|$ into the experience pool
12:        Randomly select a small batch of samples $|s_j, a_j, r_j, s_{j+1}|$ from EP
13:        if if episode terminates at step $t + 1$ then
14:            set $y_t = r_t$
15:        else
16:            set $y_t = r_t + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}; a_{t+1}; \hat{\theta})$
17:        end if
18:        Train the network according to the loss function $(y_t - Q(s_t, a_t; \theta))^2$
19:        Set $\hat{\theta} = \theta$ every $x$ steps
20:    end for
21: end for
```

Algorithm 1: Service migration based on network fault prediction and DQN.

E alone but $T \times E$, because $T \times E$ grows much faster than E itself. Therefore, T (episode, t) = O (n_t n_e), where n_e represents the number of episodes and n_t represents the number of time steps in each episode.

5. Experiment

In this section, we will first conduct experiments on the proposed network fault prediction method and compare its prediction effect with the baseline method. Then we apply this method to the service migration process to evaluate the effect of the migration.

5.1. Datasets and Metrics. We select the network alarm log information of the public metropolitan area network from January to November 2013 to train the NFP-WD model. In order to simulate the migration situation when the edge network fails, we refer to the failure information of 10 associated devices in the data set in December 2013 to perform active fault injection on the edge servers.

We use the DeepFace face recognition application as a test program. A cloud server is set up to store the face recognition video dataset iQIYI-VID. The face recognition application is initially deployed in the edge server closest to the user. The mobile user first downloads the face recognition video from the cloud server and then uploads the video to the edge server for recognition. The SDN senses that the user moves to the coverage area of the next edge server. Service migration will be triggered during the migration process, and the face recognition application will be interrupted until after the migration is completed, the user establishes a connection with the new edge server, and the video stream continues.

The alarm log includes the alarm type, alarm severity, alarm name, alarm source, NE type, alarm time, alarm clear time, and confirmation time et, as shown in Table 1.

The alarm levels in the alarm log are divided into four different levels: prompt, minor, important, and urgent. The types of alarms are divided into common alarms and root-cause alarms. In actual situations, only the records with the alarm level of "urgent" in the alarm dataset are defined as fault conditions, and the prediction of network faults is the prediction of emergency-level alarms. Our goal is to predict whether the devices in the network will fail urgently under the conditions of a given time window based on the alarm information.

In order to evaluate the proposed network fault prediction effect, we select Recall, F-Measure, and AUC value as the evaluation indicators of the model.

$$\begin{aligned} \text{Recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}} \\ \text{AUC} &= \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \\ \text{F1} &= \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}. \end{aligned} \tag{14}$$

where TP indicates the situation that the failure is predicted to occur and the failure occurs, FP indicates the situation that the failure is predicted but does not occur, FN indicates the situation that the failure is predicted not to occur but the failure occurs, and TN indicates that the prediction does not occur and the failure does not actually occur.

In the simulation experiment part of service migration, we select the migration success rate, migration cost, and migration times as the evaluation indicators of the model. Service migration success rate is an important experimental metric in our experiments; we define it as the percentage of how many face recognition applications have completed running properly.

5.2. Baselines and Parameters Settings. First, we compare the proposed network fault prediction method NFP-WD with methods: Bayesian Network [24], Random-Forest [25], SVM [26], Wide&Deep [15], and DeepFM [27]. Afterward, we

TABLE 1: Statistics of metro network fault dataset.

ID	Alarm type	Alarm level	Alarm name	Alarm source	Positioning information	Time of occurrence
26	Source alarm	Important	Power module power failure warning	Device 1	Entity name = PWR board 8	10/16/2013 17:07:45
1253	Common alarm	Minor	Link down	Device 3	Interface index = 16	01/10/2013 08:07:43
2259	Common alarm	Urgent	Temperature exceeds threshold	Device 12	Entity name = LPU slot 2	02/07/2013 17:09:29
2259	Derived alarm	Prompt	Link down	Device 28	Interface index = 6	02/16/2014 18:53:22

compare the NFP-WD-integrated service migration method SMNFP with the following methods:

(i) ASM [28]: always perform service migration, also known as the greedy migration method. As users move, services are always migrated to the edge server that is closest to the mobile user, and this method tends to lead to large migration costs.

(ii) Mig-RL [8]: A method for service migration based on the Q-learning algorithm in reinforcement learning, which aims to minimize service cost and maximize service quality.

(iii) DSM [29]: A method for modeling service migration as a distance-aware Markov decision process, focusing on the location between mobile users and edge servers.

(iv) SRSM [30]: A service migration method based on fault state-triggered adaptation, which establishes four different fault models for network states and can constrain migration cost, delay, server resource capacity, and bandwidth for different fault conditions.

Edge servers in different geographical locations represent edge nodes. We abstracted ten edge nodes into a Docker container cluster and used the Kubernetes container management platform (K8S) to implement container resource management. Docker containers can be used to conveniently store and migrate resources and provide certain computing power, and the SDN controller can unify the container cluster management through the OpenFlow protocol.

The experimental environment is CPU Intel® Core™ i9-9980XE @3.00 GHz, 128 GB RAM, and two Titan XP graphics cards. The experimental operating system is Ubuntu20.04; we use Tensorflow to implement the algorithm. We conduct simulation experiments on the Mininet network simulation platform. The experimental environment consists of four parts: mobile terminal equipment, edge server, controller, and cloud server. Mobile devices access edge servers through wireless hotspots. In the experiment, the POX controller is selected as the SDN controller. All edge nodes install a local POX controller to collect network topology information and fault conditions, sense, and initiate service migration and schedule migration tasks. Edge nodes run Open vSwitch software to parse the OpenFlow protocol. The global controller acts as a personal PC for equipment fault monitoring and management of local SDN controllers. All parameters of the experiment are shown in Tables 2 and 3.

TABLE 2: Simulation parameters.

Parameter	Values
Number of ES	5–10
Number of mobile users	5–30
Area	2 km × 2 km
ES coverage radius	300 m
ES overlap coverage radius	45 m
Moving speed of users	5 m/s
Bandwidth for up/down	20–100 Mbps
Latency between ES and users	50 ms
Fault recovery time	20–60 s
Replay memory size	10,000
Learning rate	0.001
Number of episodes	1,800
Compromise factor w_p	0.05
Discounting factor	0.9
Exploration probability	0.1

TABLE 3: NFP-WD parameters.

Parameter	Values
FFM	Field size : 3
	Feature sizes : (64, 64, 64)
	Embedding size : 4
	Dropout shallow : (0.5, 0.5)
LSTM	Hidden size : 3, 512, 256, 128
	Num of layers : 3
	Epochs : 64
	Batchsize : 256
	learning rate : 0.003
Joint training	Activation : Sigmoid
Loss function	α : 0.9

5.3. Performance Evaluation. The experiment explores the influence of the prediction time window on the prediction effect and selects the device 4 with more faults in the network fault data set as the research object to predict whether the device will fail in a given time window. We set the time window as 10 min; the experimental results are shown in Table 4; our proposed NFP-WD model outperforms the baseline model in all three metrics.

From Table 3, we have the following observations: when the prediction time window is the same, the traditional

Table 4: Recall, F1, and AUC for predicting whether device 4 will fail when the time window is 10 min.

Methods	Recall	F1	AUC
Bayesian net	0.7345	0.7252	0.6893
SVM	0.8386	0.8294	0.8147
Random forest	0.8771	0.8681	0.8433
Wide&Deep	0.8882	0.8743	0.8521
DeepFM	0.8943	0.8917	0.8696
NFP-WD	0.9116	0.9032	0.8819

machine learning classification model lacks the generalization ability of the model features compared with the three subsequent models integrated with the deep neural network, so the prediction effect is poor. After the DeepFM model replaces the LR part of the Wide&Deep model with FM, it can learn the low-order and high-order feature combinations of the alarm information at the same time without manual feature engineering, which alleviates the sparsity problem in the alarm data set to a certain extent, and the prediction effect is obtained. At the same time, we noticed that there is a lot of time series information in the alarm log in the actual situation, and the DeepFM model does not have the ability to process time series features due to the lack of memory vectors or memory neural units. In order to solve this problem, our NFP-WD model introduces the LSTM neural network in the Deep layer, which enhances the memory of the model; and the introduction of FFM can distinguish the importance of different combined features compared to FM, for example, in the alarm log, the combination of alarm severity and alarm time is an essential feature. After combining these two advantages, the prediction effect of the model for equipment fault has been improved to a certain extent.

All our experiments take the method of controlling variables and study the influence of a certain factor on the experiment when other factors are the same.

We first explore the service interruption time during the service migration. In our experiments, we calculate the service interruption time every 100 episodes and then calculate the average of all interruption times. As shown in Figure 3, the dashed lines represent the average service interruption times of various methods, and our SMNFP method achieves the smallest average service interruption time of 4.2 s, with SRSM, Mig-RL, DSM, and ASM values of 7.4, 15.5, 17.3, and 34.6 s, respectively. This is because Mig-RL, DSM, and ASM work in a fault-free network environment; for the face recognition detection service, if there are servers in the migration planning path that are about to fail in the short term, the face recognition program will be temporarily hung resulting in service interruption until a specific fault recovery time is experienced and the migration process will continue, so the service interruption time for these three methods is longer than the fault-triggered adaptive method SRSM and our SMNFP method.

Figures 4 and 5 show the effect of different numbers of edge servers on the number of service migrations and the average cost. In the scenario of network fault, as the number of servers continues to increase, the state space continues to

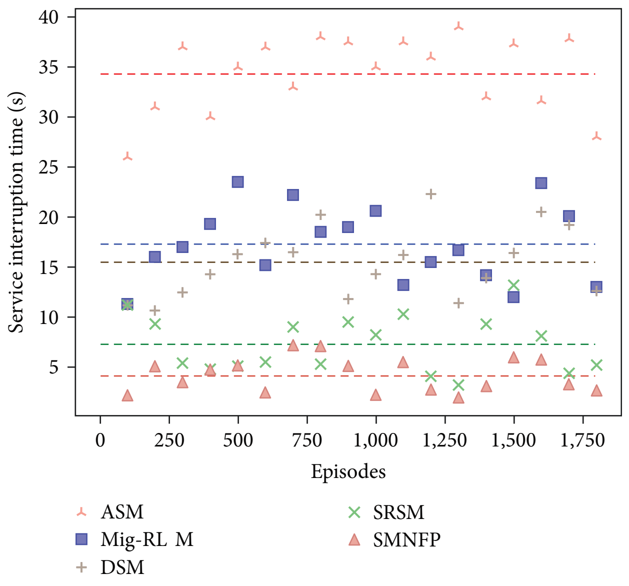


Figure 3: Service interruption time for different methods.

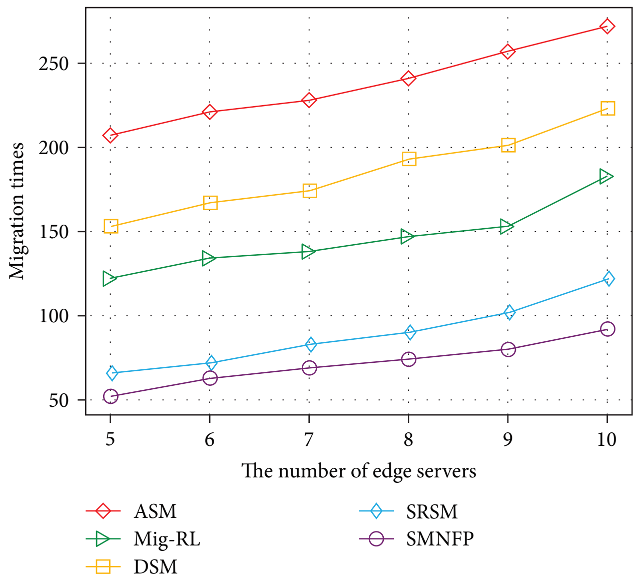


Figure 4: Service migration times under different number of edge servers.

expand, and the number of hops between the user and the original server that remains connected increases when the user moves quickly within a certain period. To complete the migration goal, the number of migrations and the total cost needs to be increased accordingly. When the number of edge servers is 10, the number of migrations and the average cost of the ASM method are the highest, reaching 272 and 641, respectively, because it is always connected to the server closest to itself and constantly initiates migration requests during the frequent movement of users. The number of migrations and the average cost of the DSM method are 217 and 473, respectively. The number of migrations and the average cost of Mig-RL are 165 and 437, respectively.

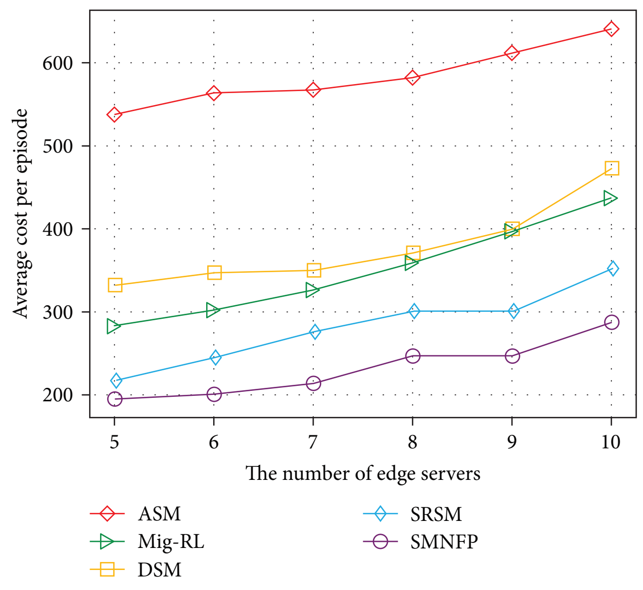


FIGURE 5: Average cost per episode under different number of edge servers.

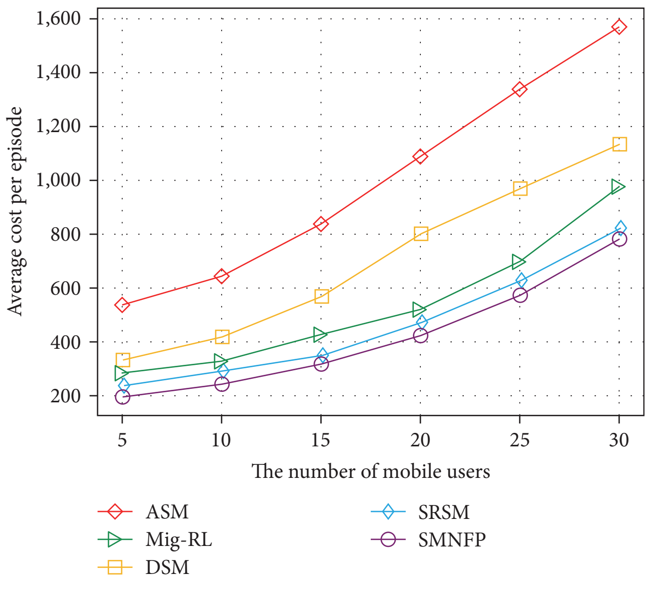


FIGURE 6: Average cost per episode under different number of mobile users.

SRSM and SMNFP use the DQN to make migration decisions, which can still maintain good performance when the state space increases.

In terms of the number of migrations, the SRSM method and SMNFP method are 127 and 92, respectively, and the average cost is 352 and 287, respectively. The service migration method based on fault adaptation is more complex when faults are frequently triggered and also leads to a high average cost when self-adaptation fails. By predicting and avoiding faulty nodes, our method improves the success rate of migration and reduces the time cost and communication cost of migration as well as the number of migrations.

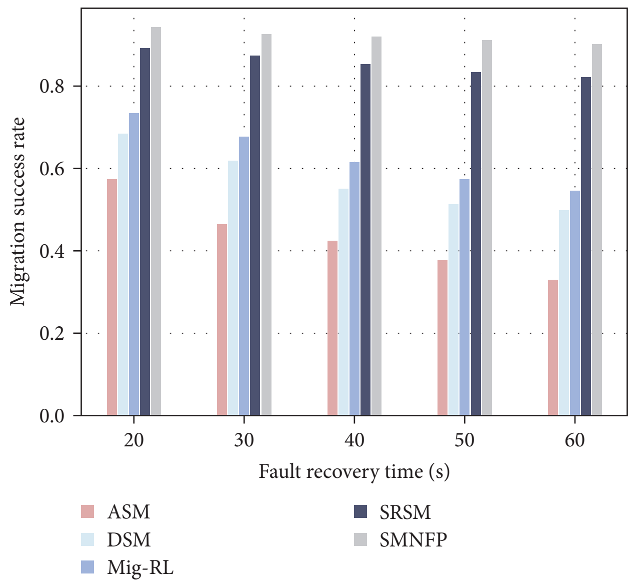
We also study the effect of different numbers of mobile users on migration costs, as shown in Figure 6. Compared with the cost impact of the number of servers, the cost of each method increased significantly when the number of mobile users increased from 5 to 30. This is because as the number of users increases, the total amount of data requested by users increases, and the amount of data that needs to be migrated also increases. Due to the limitation of storage capacity and bandwidth resources, some services cannot be migrated for a short period during the mobile process, resulting in a sharp increase in communication costs. Our proposed SMNFP method has the lowest migration cost among all methods, with a value of 792 when the number of users is 30.

To better simulate the migration situation when a network fault occurs, we choose to periodically clear the fault after a short period of active fault injection into the device. A short fault recovery time can reduce the service interruption time during the migration process, thereby improving the success of the migration rate. When the number of mobile users is five, Figure 7(a) shows the effect of fault recovery time on the migration success rate from 20 to 60 s, and SMNFP achieves the highest migration success rate at different periods. All methods show a decreasing trend as the recovery time increases. When the time is 60 s, the success rate of migration reaches the lowest, which are 0.327, 0.497, 0.544, 0.821, and 0.903, respectively, and the migration success rate of the three methods of ASM, DSM, and Mig-RL decreases greatly. The migration rate of SMNFP has a little downward trend; this is because after NFP-WD predicts the faulty node, only a very small part of the services will be migrated to the faulty node, so the fault recovery time has little effect on SMNFP.

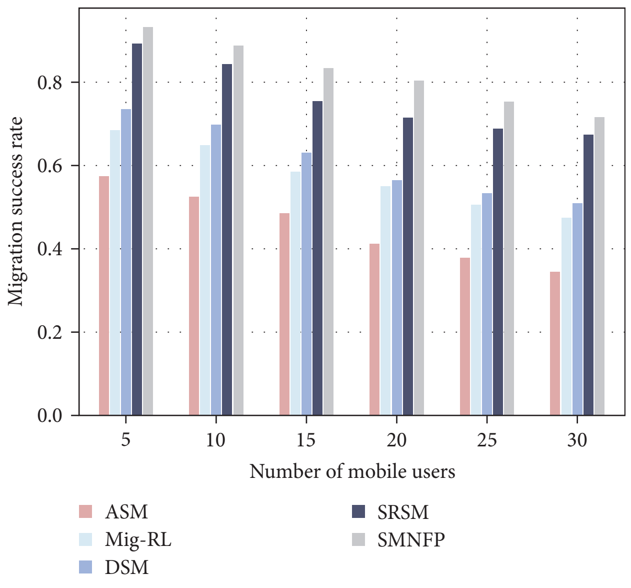
When the fault recovery time is 20 s, we study the effect of the number of concurrent requests on the success rate of migration. The experiment selects different numbers of mobile users from 5 to 25. It can be seen from Figure 7(b) that with the increase in the number of mobile users, different migration methods all showed a significant downward trend. When the number of mobile users is 25, the success rates are 0.343, 0.473, 0.508, 0.642, and 0.714, respectively. This is because, with the increase of users, the number of concurrent face recognition applications increases. Due to the limitation of its bandwidth, computing power, and frequent faults, edge servers have caused a large number of service computing faults and migration faults, and the migration success rate has dropped significantly.

Besides, we study the effect of network bandwidth and the number of edge servers on the success rate of migration. From Figure 7(c), we can see that the success rate of migration decreases with the increase of network bandwidth; when the bandwidth increases from 60 to 80 Mbps, the increase is the highest. When the bandwidth is 100 Mbps, the migration success rate of each method is the highest, which are 0.572, 0.683, 0.733, 0.891, and 0.931, respectively. Continuing to increase bandwidth has little effect on improving the success rate because the factor limiting the success rate of migration is no longer bandwidth but other computer hardware factors.

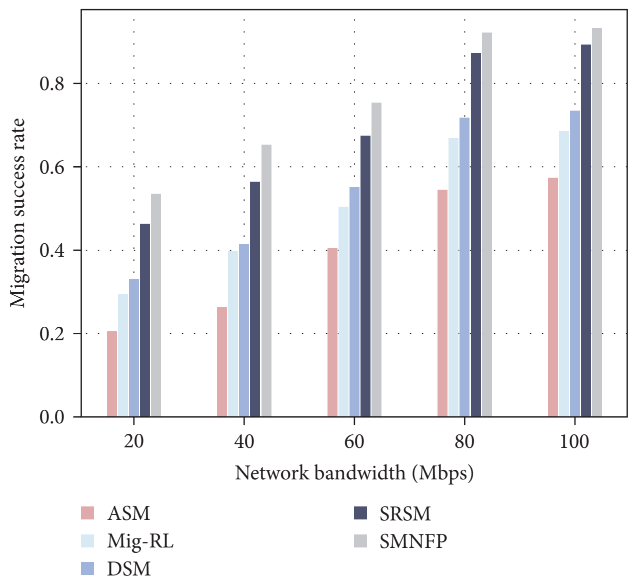
Figure 7(d) shows the effect of the number of mobile edge servers on the migration success rate. For the added servers, we also follow the previous fault injection method.



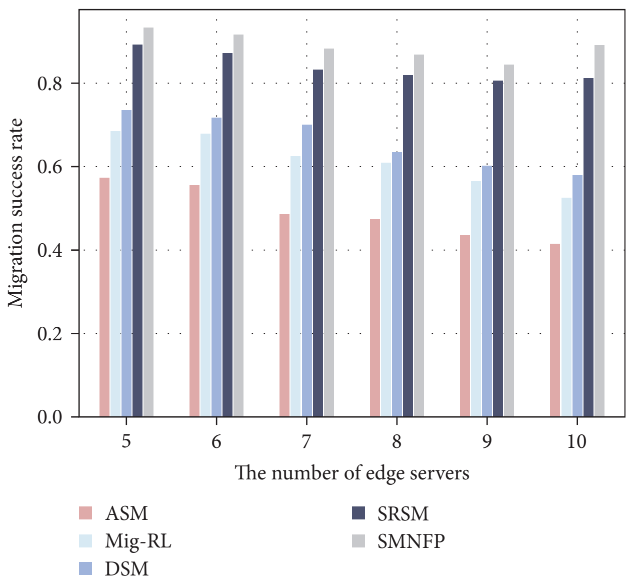
(a)



(b)



(c)



(d)

Figure 7: Comparisons of migration success rate for five methods under different parameters: (a) migration success rate under different fault recovery time; (b) migration success rate under different numbers of users; (c) migration success rate under different network bandwidth; (d) migration success rate under different number of edge servers.

It can be seen that with the increase in the number of servers, the migration success rate of various methods decreases slightly. Experiments show that the number of edge servers has a great impact on the number of service migrations but has little impact on the success rate. This may be because the number of servers has not yet reached a very large number in the actual edge network.

6. Conclusion

In this paper, we first propose a network fault prediction method NFP-WD, which is used to predict the fault of the mobile edge network. Then we model the service migration problem as a Markov decision process, and a penalty function is designed to avoid faulty nodes during migration. Simulation experiments on service migration show that our proposed SMNFP method outperforms several baseline methods.

In the follow-up work, we plan to analyze the user's movement trajectory and predict their movement patterns under network fault scenarios to further improve the success rate of migration.

Data Availability

The original dataset used in this work is available from the corresponding author on request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (62272243), Jiangsu Key Laboratory of Big Data Security & Intelligent Processing and Jiangsu High Technology Research Key Laboratory for Wireless Sensor Networks.

References

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: the communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[2] L. Wang, Y. Zhang, and S. Chen, "Computation offloading via Sinkhorn's matrix scaling for edge services," *IEEE Internet of Things Journal*, vol. 8, no. 10, pp. 8097–8106, 2021.

[3] M. Sun, Z. Zhou, X. Xue, and W. Gaaloul, "Migration-based service allocation optimization in dynamic IoT networks," in *International Conference on Service-Oriented Computing*, pp. 385–399, Springer, Cham, 2021.

[4] S. W. Park, A. Boukerche, and S. Guan, "A novel deep reinforcement learning based service migration model for mobile edge computing," in *2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pp. 1–8, IEEE, Prague, Czech Republic, 2020.

[5] L. Yin, P. Li, and J. Luo, "Smart contract service migration mechanism based on container in edge computing," *Journal of Parallel and Distributed Computing*, vol. 152, pp. 157–166, 2021.

[6] I. Labriji, F. Meneghello, D. Cecchinato et al., "Mobility aware and dynamic migration of mec services for the internet of vehicles," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 570–584, 2021.

[7] T. Taleb, A. Ksentini, and P. A. Frangoudis, "Follow-me cloud: when cloud services follow mobile users," *IEEE Transactions on Cloud Computing*, vol. 7, no. 2, pp. 369–382, 2019.

[8] Y. Wang, S. Cao, H. Ren et al., "Towards cost-effective service migration in mobile edge: a Q-learning approach," *Journal of Parallel and Distributed Computing*, vol. 146, pp. 175–188, 2020.

[9] Z. Liang, Y. Liu, T.-M. Lok, and K. Huang, "Multi-cell mobile edge computing: joint service migration and resource allocation," *IEEE Transactions on Wireless Communications*, vol. 20, no. 9, pp. 5898–5912, 2021.

[10] P. Wang, T. Ouyang, G. Liao, J. Gong, S. Yu, and X. Chen, "Edge intelligence in motion: mobility-aware dynamic DNN inference service migration with downtime in mobile edge computing," *Journal of Systems Architecture*, vol. 130, Article ID 102664, 2022.

[11] M. Xu, Q. Zhou, H. Wu, W. Lin, K. Ye, and C. Xu, "PDMA: probabilistic service migration approach for delay-aware and mobility-aware mobile edge computing," *Software: Practice and Experience*, vol. 52, no. 2, pp. 394–414, 2022.

[12] Y. Miao, F. Lyu, F. Wu et al., "Mobility-aware service migration for seamless provision: a reinforcement learning approach," in *ICC 2022-IEEE International Conference on Communications*, pp. 5064–5069, IEEE, Seoul, Korea, 2022.

[13] R. Lin, Z. Pei, Z. Ye, B. Wu, and G. Yang, "A voted based random forests algorithm for smart grid distribution network faults prediction," *Enterprise Information Systems*, vol. 14, no. 4, pp. 496–514, 2020.

[14] S. A. Yadwad and V. K. Vatsavayi, "Fault prediction for network devices using service outage prediction model," *Journal of Communications*, vol. 17, no. 5, pp. 339–349, 2022.

[15] H.-T. Cheng, L. Koc, J. Harmsen et al., "Wide & deep learning for recommender systems," in *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pp. 7–10, Association for Computing Machinery, New York, NY, United States, 2016.

[16] J. Jia, C. Feng, T. Zhang et al., "Deep fault prediction with flexible weighted mining based alarm correlation analysis of communication networks," in *2020 IEEE 20th International Conference on Communication Technology (ICCT)*, pp. 173–177, IEEE, Nanning, China, 2020.

[17] P. Klein, N. Weingarz, and R. Bergmann, "Using expert knowledge for masking irrelevant data streams in siamese networks for the detection and prediction of faults," in *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–10, IEEE, Shenzhen, China, 2021.

[18] Y. Y. Tefera, T. Kibatu, B. S. Shawel, and D. H. Woldegebreal, "Recurrent neural network-based base transceiver station power supply system failure prediction," in *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7, IEEE, Glasgow, UK, 2020.

[19] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[20] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[21] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," 2015 aaai fall symposium series, 2015.

[22] J.-K. Ge, Y.-F. Chai, and Y.-P. Chai, "WATuning: a workload-aware tuning system with attention-based deep reinforcement learning," *Journal of Computer Science and Technology*, vol. 36, pp. 741–761, 2021.

[23] J. Li, L. Xin, Z. Cao, A. Lim, W. Song, and J. Zhang, "Heterogeneous attentions for solving pickup and delivery problem via deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 3, pp. 2306–2315, 2022.

[24] K. Dejaeger, T. Verbraken, and B. Baesens, "Toward comprehensible software fault prediction models using Bayesian network classifiers," *IEEE Transactions on Software Engineering*, vol. 39, no. 2, pp. 237–257, 2013.

[25] J. Shen, J. Wan, S.-J. Lim, and L. Yu, "Random-forest-based failure prediction for hard disk drives," *International Journal of Distributed Sensor Networks*, vol. 14, no. 11, pp. 1–15, 2018.

[26] H. Zhang, Y. Liu, and L. Wang, "An intelligent alarm method for optical fiber network based on backtracking single alarm information," in *2020 International Workshop on Electronic Communication and Artificial Intelligence (IWECAI)*, pp. 56–60, IEEE, Shanghai, China, 2020.

[27] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "DeepFM: a factorization-machine based neural network for ctr prediction," arXiv preprint arXiv:1703.04247, 2017.

[28] L. Ma, S. Yi, and Q. Li, "Efficient service handoff across edge servers via docker container migration," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, pp. 1–13, Association for Computing Machinery, New York, NY, United States, 2017.

[29] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge computing based on Markov decision process," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 1272–1288, 2019.

[30] L. L. Rui, M. Zhang, Z. Gao, X. Qiu, Z. Wang, and A. Xiong, "Service migration in multi-access edge computing: a joint state adaptation and reinforcement learning mechanism," *Journal of Network and Computer Applications*, vol. 183-184, Article ID 103058, 2021.