applied sciences

Article

Virtual Machine Migration Strategy Based on Multi-Agent Deep Reinforcement Learning

Yu Dai [1,*], Qiuhong Zhang [1] and Lei Yang [2]

[1] Software College, Northeastern University, Shenyang 110169, China; 1971188@stu.neu.edu.cn
[2] Computer Science and Engineering, Northeastern University, Shenyang 110169, China; yanglei@mail.neu.edu.cn
* Correspondence: daiy@swc.neu.edu.cn

Abstract: Mobile edge computing is a new computing model, which pushes cloud computing power from centralized cloud to network edge. However, with the sinking of computing power, user mobility brings new challenges: since it is usually unstable, services should be dynamically migrated between multiple edge servers to maintain service performance, that is, user-perceived latency. Considering that Mobile Edge Computing is a highly distributed computing environment and it is difficult to synchronize information between servers, in order to ensure the real-time performance of the migration strategy, a virtual machine migration strategy based on Multi-Agent Deep Reinforcement Learning is proposed in this paper. The method of centralized training and distributed execution is adopted, that is, the transfer action is guided by the global information during training, and only the local observation information is needed to obtain the transfer action. Compared with the centralized control method, the proposed method alleviates communication bottleneck. Compared with other distributed control methods, this method only needs local information, does not need communication between servers, and speeds up the perception of the current environment. Migration strategies can be generated faster. Simulation results show that the proposed strategy is better than the contrast strategy in terms of convergence and energy consumption.

Keywords: mobile edge computing; deep reinforcement learning; virtual machine migration

check for updates

Citation: Dai, Y.; Zhang, Q.; Yang, L. Virtual Machine Migration Strategy Based on Multi-Agent Deep Reinforcement Learning. *Appl. Sci.* **2021**, *11*, 7993. https://doi.org/10.3390/app11177993

Academic Editor: Sungho Kim

Received: 13 July 2021
Accepted: 25 August 2021
Published: 29 August 2021

1. Introduction

With the development of science and technology, there are many applications that require high computing power and are extremely sensitive to delay, which are far beyond the service capacity that mobile devices can provide. In addition, due to the limitations of the development of battery technology, the power consumption of running these applications is unbearable for mobile devices. These problems have become an opportunity for the development of Mobile Cloud Computing (MCC) technology, through the integration of cloud computing capabilities into the mobile network, so that mobile users can access and use cloud computing services [1]. The Mobile Capabilities Augmentation using Cloud Computing (MCACC) framework proposed in reference [2] divides mobile applications into a group of services. The framework is deployed in the application layer, and each service is executed locally or remotely on the cloud based on dynamic unloading decisions, so that mobile applications are not limited by mobile device resources. However, because mobile cloud computing uses a centralized model, there is often a very long distance between the server and mobile users, which will lead to high network delay. Additionally, in the longer path, the probability of data being attacked on the way will be greatly increased. The European Telecommunications Standards Institute (ETSI) introduced the concept of Mobile Edge Computing (MEC) in 2014. According to the white paper released by ETSI for the first time, mobile edge computing is defined as: "Mobile edge computing can provide IT service environment and cloud computing capabilities within the range of Ratio Access Network (RAN) near mobile devices [3]". Mobile edge computing releases the mobile device from the heavy computing work. By transferring the computing task on the mobile device to the MEC server, which is closer from the space [4], it can reduce the user-perceived delay, reduce the network congestion probability and prolong the battery life of the mobile device.

Although MEC greatly improves the computing power of mobile users, the unpredictable user mobility in wireless networks also brings new challenges. Reducing latency and smoothing the user experience as users move cannot be solved by simply pushing cloud capabilities to the edge of the network. In order to ensure the business continuity of users between different edges, it is necessary to adopt an efficient mobility management scheme at the network edge.

Software Defined Network (SDN) [5] is an emerging technology, which provides a simplified method to dynamically control multiple simple switches through a single controller program. In the fog computing architecture based on SDN [6], routing logic and intelligent management logic are deployed on the SDN controller, which greatly simplifies the operation and management of the network. However, the placement of dynamic service profiles is not simple. On the one hand, the user-perceived delay is determined by both the communication delay and the computing delay [7,8]. Therefore, if each user′s service profile is placed on the nearest MEC node, some MEC nodes may be overloaded, resulting in increased computing latency. On the other hand, when the MEC computing node is overloaded or unable to calculate, the computing services need to be migrated to the Virtual Machines (VMs) of other nodes for computing. In order to improve Quality of Service (QoS) such as TCP throughput, in Reference [9], the author proposed a VM migration method based on expected TCP throughput, which migrates VMs from crowded nodes to another node on the mobile edge; this method is a VM migration method considering edge node congestion without considering user mobility in MEC environment. Moreover, during the migration, you also need to ensure that the resources (CPU, network bandwidth, etc.) of the migration operating system will not be unnecessarily disrupted by contention for active services [10]. When computing services need to be frequently migrated between multiple MEC nodes, this will incur additional operational costs. A resource allocation strategy based on VM migration is proposed in Reference [11]. It is considered that the MEC server is deployed near the terminal, and the MEC controller allocates the servers accessed by the terminals in different areas to maximize the system Energy Efficiency (EE). User equipment can harvest radio frequency energy from multiple frequency bands as they move. Energy-efficient mobility management is an important issue in modern networks. In Reference [12], the author introduces handover costs into total energy consumption. Therefore, the effective dynamic service layout strategy should pay attention to: (1) coordinating communication delay and computing delay to minimize user-perceived delay; (2) weighing performance and cost in a cost-effective way.

In this paper, the VM migration method is used to keep the VM synchronized with the user all the time to ensure the quality of service. MEC places computing resources on the edge of the network closer to mobile devices in a distributed way. Considering that it is difficult to ensure the real-time exchange of information in distributed systems, this paper proposes a VM migration strategy based on Multi-Agent Deep Reinforcement Learning (MADRL). The strategy adopts the combination of centralized training and distributed execution, evaluates the migration action through the global information during training, and transmits the evaluation in reverse to train the state value network, so as to learn the migration strategy with the lowest energy consumption. During execution, only local observation information is needed to obtain the migration action. Compared with the centralized control method, the proposed method alleviates the communication bottleneck of the VM migration strategy based on MADRL. Compared with other distributed control methods, this method only needs local information and does not need communication between servers. it speeds up the perception of the current environment, and the migration strategy can be generated more quickly. Finally, the simulation results show that the proposed migration strategy has a better effect than the contrast strategy in terms of convergence, energy consumption and migration success rate.

2. Related Works

Service placement has been widely studied in the paradigm of cloud computing. In MEC, a key challenge in achieving efficient service placement is to consider device mobility. In the existing research, some work is based on the prediction of future information. For example, in Reference [13], the author solved the tradeoff between execution overhead and delay by using mobility-based prediction schemes, and proposes a data transfer throughput estimation scheme (DTT), MDC handoff time estimation scheme (HTE) and service migration management scheme (SMM). In addition, the author further studies how to place services by predicting the costs of future data transmission, processing and service migration in Reference [14]. In Reference [15], the author proposed a FMeC-based framework in the autopilot scenario, which is based on the analysis of vehicle movement patterns and makes a tradeoff between reducing service migration costs and maintaining end-to-end QoS. The proposed scheme meets the delay requirements of autopilot and minimizes the overall cost.

However, it is not realistic to accurately predict future information such as user mobility in the real environment. In order to meet the challenge of user mobility that is unpredictable in practice, another recent work applied the Markov Decision Process (MDP). In Reference [16], through Markov chain analysis of whether to migrate services, the author discusses the impact of service migration on mobile users' perceived delay. In References [17,18], the authors determine the optimal threshold of service migration based on MDP. In addition, the author in Reference [19] extended the service migration algorithm to adapt to 2D migration scenarios. In Reference [20], the optimal service migration strategy is designed by defining the service placement problem as a sequential decision problem. The above studies are based on the centralized system model, and the global information of the network needs to be predicted when solving the scheduling strategy. However, MEC is a highly distributed computing environment; obtaining the status of all servers may consume a lot of network resources, and cannot ensure the real-time performance of the migration strategy. In order to solve the above problems, this paper constructs a distributed migration model and proposes a VM migration strategy based on MADRL. The migration process is mainly through live VM migration technology, and live migration is a key function of system virtualization technology [21]. The methods of centralized training and distributed execution are adopted to alleviate the communication bottleneck, speed up the perception of the current environment, and generate the migration strategy more quickly.

3. System Model

As shown in Figure 1, after the user establishes a connection with a server, the computing task is unloaded to the server, and then the server creates a dedicated VM for the calculation of the task. During the period of establishing a connection to the end of the service, users still upload their location information regularly, and the server sends back the navigation information regularly. With the continuous movement of users, the distance from the server will become farther and farther, resulting in an increase in transmission delay, so as to reduce the quality of user experience. Therefore, the above problems can be solved by actively migrating VMs closer to mobile users.

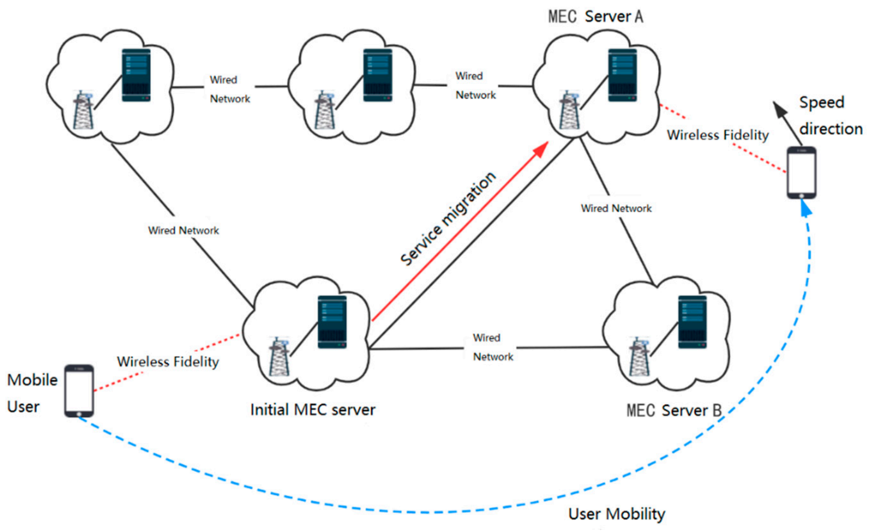

Figure 1. VM migration scenario.

In order to better represent the mobility of users and the dynamic changes of the network, time is divided into time periods according to the same interval and numbered according to time order, and the i-th time period is expressed as $slot_i$. As shown in the figure, consider a network structure composed of MEC server $\mathcal{H} = \{1, 2, \cdots, n, \cdots, H\}$. MEC server n can be represented by a triple (cap_n, r_n, l_n), where cap_n represents the remaining computing capacity of the server (in cycles per s), r_n represents the remaining disk space of the server (in byte), and l_n represents the location of the server. Each MEC node connects to a local Base Station (BS) or wireless access point via a high-speed local area network (LAN). The VMs to be migrated on the MEC server n within the time period SI_i are recorded as $\mathcal{T}_n = \{T_n^1, T_n^2, \ldots, T_n^u\}$, the VM T_n^u is represented by the quintuple $T_n^u = (\lambda_n^u, c_n^u, d_n^u, s_n^u, o_n^u)$, where λ_n^u represents the size of the VM configuration file (unit byte), c_n^u represents the calculation cycle (unit cycle) of the CPU required to perform the computing task, and d_n^u represents the maximum delay (unit seconds) that the task can tolerate. s_n^u represents the current movement speed of the device that generated the task (in meter per second), and o_n^u represents the current direction of movement of the device that generated the task, expressed as the angle between the clockwise direction and the north (in 2 degrees).

3.1. Delay Model

The delay that users can perceive can be composed of two parts: computing delay and transmission delay. The transmission delay tr_n^u is directly proportional to the size of the transmitted data and inversely proportional to the bandwidth, which can be calculated by the following formula:

$$tr_n^u = \frac{\kappa_n^u}{B_n^{tr}} \tag{1}$$

where κ_n^u represents the task size, and B_n^{tr} represents the bandwidth between the BS to which the user is connected and the destination MEC server. The computing delay com_n^u represents the time it takes for the task to obtain the result on the MEC server. The delay can be calculated by the CPU computing cycle required by the task and the computing power of the MEC server, as shown below:

$$com_n^u = \frac{c_n^u}{cap_n}. \tag{2}$$

In summary, the task T_u is executed on the MEC server n, and the total delay D_u perceived by the user can be expressed as a formula:

$$D_u = tr_n^u + com_n^u. \tag{3}$$

3.2. Energy Consumption Model

Assuming that the VM T_i^u is migrated from the MEC server H_i to H_j, the migration cost is positively related to the size of the VM profile and the distance between the two servers, as shown in Formula (4),

$$E_{i,j}^u = c_t \lambda_i^u d\left(H_i, H_j\right), \tag{4}$$

where c_t represents the resource consumed by transferring 1 byte of data per unit length.

3.3. Problem Modeling and Transformation

How to migrate is the focus of the dynamic VM migration mechanism; most migration mechanisms only rely on network conditions to determine the migration strategy, few people take into account the user's movement speed, direction and other behavior characteristics. As shown in the example shown in Section 4.1, when multiple servers meet the migration conditions of the current time period, the choice of migration strategy considering the movement direction of the user can reduce the probability of triggering VM migration in the future, thus reducing the migration energy consumption of the system.

The optimization goal of this paper is to minimize the energy consumption of migration when the constraints are met. $a_{i,j}$ represents the VM migration strategy, and when $a_{i,j} = 1$, it represents the migration of the VM T_i^u from the MEC server H_i to H_j. Therefore, the optimization goal can be written as:

$$\begin{array}{lll} & min & \sum_{i=1}^{|\mathcal{H}|} \sum_{u=1}^{|\mathcal{T}_i|} \sum_{j=1}^{|\mathcal{H}|} a_{i,j} E_{i,j}^u, \\ s.t. & C_1: & D_j^u < d_n^u,\ \forall H_j \in \mathcal{H},\ \forall T_n^u \in \mathcal{T}_n \\ & C_2: & r_j > \lambda_n^u,\ \forall H_j \in \mathcal{H},\ \forall T_n^u \in \mathcal{T}_n \\ & C_3: & cap_j > c_n^u,\ \forall H_j \in \mathcal{H},\ \forall T_n^u \in \mathcal{T}_n \\ & C_4: & a_{i,j} \in \{0,1\},\ \forall H_i \in \mathcal{H},\ \forall H_j \in \mathcal{H} \\ & C_5: & \sum_{j=0}^{|\mathcal{H}|} a_{i,j} \le 1\ . \end{array} \tag{5}$$

4. VM Migration Strategy Based on MADRL

For the above optimization objectives, taking into account the mobility of users and the dynamic changes of the server, as well as the sensitive delay requirements of tasks, it is difficult for traditional methods to solve them, so this paper uses the method of deep reinforcement learning to solve the migration strategy. In the existing DRL-based VM migration strategy research, it is often assumed that the global information of mobile devices and servers in the system can be obtained, but in practical applications, due to the delay of network transmission, it is difficult to achieve global information synchronization. Therefore, this paper uses MADRL to generate the VM migration strategy in the mobile edge environment. Each MEC server, as an independent agent, learns the migration strategy by interacting with the environment to minimize the migration cost.

In Reference [22], the system considered by the authors consists of a central control unit. The authors first assume that the tasks of User Equipment (UE) are successfully received to the central control unit of MEC as a UE request. The central control unit then detects the incoming UE request, selects the best Mobile Edge Server (MES) for the request and allocates resources, during which the authors assume the UE continuous transmission location. Finally, the central control unit can issue a command to migrate its VM to another MES and migrate the VM to that MES. Referring to the previous work related to virtual machine migration, we also assume that the VM migration mechanism between server hosts uses live VM migration technology. According to the concept of virtualization, VM is hosted by a virtual machine monitor (VMM), which runs on the server host. In modeling, we assume that the operational state of the VMM can be integrated into the operational state of the server. When the server makes a decision that the VM needs to be migrated, it triggers the live migration of the VM. The monitor identifies the destination server host to which this VM is to be migrated [23]. The live migration process is triggered by loading the VM instance and related user data from Network Area Storage (NAS) [24,25] to the destination server host by switching the current connection. After this process is completed, the VM can restart and continue to perform subsequent computing tasks on the server. In order to simplify the model migration process, we do not consider the unexpected failures in the process of data transfer between servers. These failures may be sudden network outages, memory errors at the source or destination, data packet corruption or loss in transmission [26].

4.1. Multi-Agent Deep Reinforcement Learning Model

(1) Agent Observation State

The observed state $o_n(i) \in \mathcal{S}$ represents the state that MEC server H_n can perceive during the time period SI_i, which consists of two parts; namely, the current running status of the server and the parameters of the VM to be migrated, so it can be expressed as $o_n(i) = \{H_n, T_n^1, T_n^2, \ldots, T_n^u\}$.

(2) Action Space

According to the observed state of time period SI_i and the migration strategy π_i, we can receive the action $a_n(i)$, which is a vector composed of u elements, each element represents the destination server of VM T_n^u migration, and $a_n(i) = (a_n^1, \cdots, a_n^u)$. For VM T_n^u, because the output $p_{n,m}$ of neural network is the probability of migration from VM T_n^u to server H_m, a_n^u chooses the server with the highest probability as the migration destination server through greedy algorithm, and migrates VM T_n^u to this server, so $a_n^u = \max(p_{n,1}, \cdots, p_{n,m})$. The action set of all agents is represented by $\mathcal{A}$, which is an n dimensional vector.

(3) Reward Function

According to the observed state of $o_n(i)$, the MEC server takes the action $a_n(i)$ and obtains a timely reward of $r_n(i)$. The optimization goal is to minimize the energy consumption of VM migration, so the reward function $r_n(i)$ is as follows:

$$r_n(i) = \begin{cases} \frac{1}{\sum_{u=1}^{|\mathcal{T}|} \sum_{j=1}^{|\mathcal{H}|} a_{i,j} E_{i,j}^u}, success \\ 0, fail \end{cases}. \tag{6}$$

In the multi-agent environment, we use *sar* to denote system status, joint action and joint reward, respectively, which are marked as $s = \{o_n(i)\}$, $a = \{a_n(i)\}$, $r = \{r_n(i)\}$, respectively. In order to simplify the representation, the state, action and reward of MEC server H_n in the current time period are expressed by using the o_n, a_n and r_n. o'_n, a'_n, r'_n are used to represent the status, action and reward of MEC server H_n in the next time period. Each agent expects to maximize its reward in the process of interacting with the environment. In this paper, the action value function of the action a_n^u taken by the MEC server H_n in the case of the observed state o_n in the time period SI_i is expressed by using $Q_n(o_n, a_n)$. Based on the Behrman equation, the recursive relation of the action value function $Q_n(o_n, a_n)$ is shown in the Formula (7),

$$Q_n(o_n, a_n) = \mathbb{E}[r_n + \gamma \mathbb{E}[Q_n(o'_n, a'_n)]], \tag{7}$$

where the discount factor $\gamma \in [0, 1)$ controls the ability of the agent to "look forward to the future". When $\gamma = 1$, the agent obtains a reward for Q_n without any discount in the future.

4.2. VM Migration Strategy Based on MADRL

Each MEC server H_n has its own agent, and each agent is composed of two parts, namely actor and critic network. The actor network includes two parts: the current actor network π_n and the target actor network π'_n. The two network parameters are θ_n and θ'_n, respectively. Similarly, the critic network also includes two parts: the current critic network Q_n and the target critic network Q'_n. The two network parameters are ω_n and ω'_n, respectively.

The VM migration strategy architecture based on MADRL is shown in Figure 2. In the process of model training, the current actor network π_n and the current critic network Q_n are trained online, and the target critic network Q'_n is trained offline. The current actor network π_n of each agent receives the current observation state o_n, and the MEC server takes the action a_n and obtains the reward r_n, and then the MEC server moves on to the next state o'_n. Then, the quaternion $\{s, a, r, s'\}$ is saved into the experience replay set D.

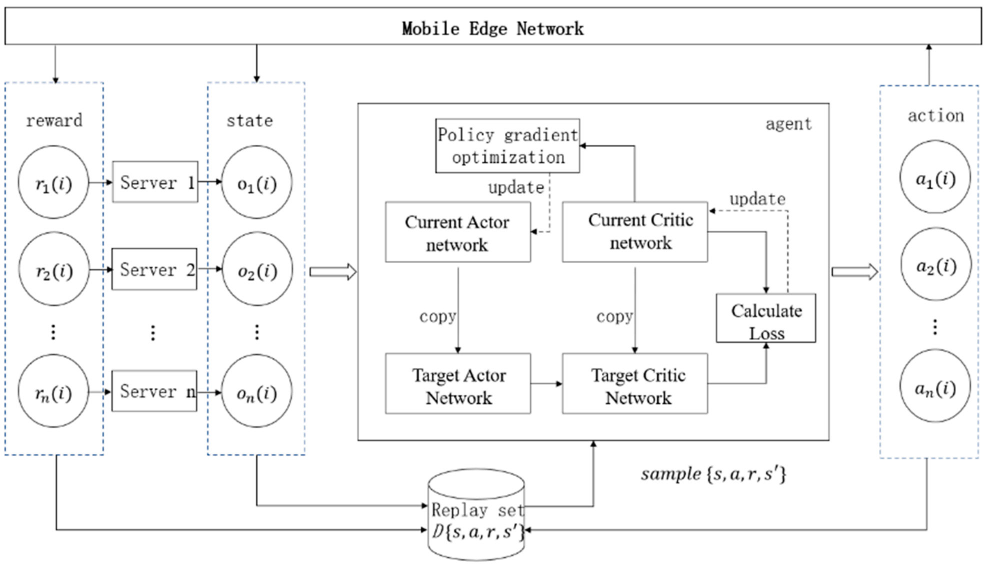

Figure 2. VM Migration Strategy Architecture based on MADRL.

Neural network training includes three steps. First update the current critic network, then update the current actor network, and finally update the target actor network and the target critic network.

(1) Update the Current Critic Network

At the beginning of each time period, each agent will randomly select s samples $\{s, a, r, s'\}$ from the experience replay set D, and use the mean square error loss function shown in Formula (8) to update the parameter ω_n of the current critic network through the gradient back propagation of the neural network,

$$L(\omega_n) = \frac{1}{s}\sum_{j=1}^{s}\left(y_j - Q_n(s,a)\right)^2, \tag{8}$$

where y_j is as shown in Equation (9),

$$y_j = r + \gamma Q'_n(s', \pi'_n(s')). \tag{9}$$

(2) Update the Current Actor Network

The current actor network updates parameters through policy gradient descent, as shown in Equation (10):

$$\nabla_{\theta_n} J \approx \frac{1}{s}\sum\nolimits_j \nabla_{\theta_n} log(\pi_n(o_n))\nabla_{a_n} Q_n(s,a). \tag{10}$$

(3) Update the Target Actor Network and Target Critic Network

The parameters of the target actor network and target critic network are updated as shown below:

$$\theta'_n = \tau\theta_n + (1-\tau)\theta'_n \,, \tag{11}$$

$$\omega'_n = \tau\omega_n + (1-\tau)\omega'_n \,, \tag{12}$$

where $\tau \in [0,1)$ represents the learning rate of the parameter.

The VM migration strategy based on MADRL is shown in Algorithm 1.

Algorithm 1 VM migration strategy based on MADRL

```
1.Initialize current actor network $\pi$ and critic network $Q$ with parameters $\theta$ and $\omega$
2.Initialize target actor network $\pi'$ and critic network $Q'$ with parameters $\theta'$ and $\omega'$
3.Initialize episode $K$, discount factor $\gamma$, learning rate $\tau$, the number of samples $s$, update
frequency $C$, experience replay set $D$
4.for episode $k = 1,2,\dots,K$ do
5.    for $SI_i$ $i = 1,2,\dots,T$ do
6.        choose action $a_n = \pi_n(o_n)$ for each MEC Server
7.        execute action $a = (a_1,\cdots,a_n)$, obtain reward $r$ and next state $s'$
8.        store transitions $\{s,a,r,s\prime\}$ into replay set $D$
9.        $s \leftarrow s\prime$
10.   end for
11.   if $k\%C == 0$ then
12.       for $agent$ $n = 1,2,\dots,n$ do
13.           sample $s$ transitions $\{s,a,r,s'\}$ from $D$
14.           update current critic network according to (8), (9)
15.           update current actor network according to (10)
16.       end for
17.       update target actor network and target critic network according to (11), (12)
18.   end if
19.end for
```

4.3. Simulation Experiment Results and Analysis

4.3.1. Experimental Environment and Data Set

In order to evaluate the performance of the VM migration strategy based on MADRL proposed in this paper, a simulation environment is developed based on Python language, and the tasks and servers in the environment are modeled. In the simulation process, time is divided into periods of equal length, and the interval is set as 1 s. Server running status is generated based on the machine_usage.csv file in the open source dataset cluster-trace-v2018 [27]. The dataset contains information about 4000 servers and corresponding online application containers as well as the operation of offline computing tasks in 8 days. A task model is generated based on the batch_task.csv file in the dataset. When considering the task mobility, this experiment uses real vehicle driving datasets to validate the proposed VM migration strategy. The dataset consists of 4316 cabs driving for 24 h in Shanghai, China on 20 February 2007 [28]. Each vehicle trajectory consists of a set of time-stamped GPS points, and each GPS point has seven fields, which are cab ID, time stamp, longitude and latitude of the cab's current location, instantaneous speed at this time, clockwise angle to north (unit: 2 degrees), and cab status (status: empty or with passengers). After pre-processing the raw data, five VMs were set to wait for migration in each time period, and 50 MEC servers were randomly deployed.

4.3.2. Experimental Scheme Design

(1) Simulation Experiment Parameters

The simulation environment is built based on the following parameters, as shown in Table 1.

Table 1. Simulation experiment parameter.

Parameter	Value
number of VM	5~20
size of VM	1000~10,000 bytes
tolerable delay	0.01~0.3 s
number of CPU cycles required for tasks	500~1500 cycles
speed	0~90 m/s
computational capacity of a server	1×10^6~6×10^6 cycles/s
bandwidth of a server	1~3 MHz
size of memory	1 TB
distance between servers	200~1000 m

(2) Neural Network Parameter Setting

The actor network consists of four fully connected layers and two hidden layers, which have 512 and 128 neurons, respectively. The critic network consists of three fully connected layers and two hidden layers, which have 512 and 256 neurons, respectively. The learning rate of actor network is 0.001. The learning rate of critic network is 0.01, and the discount factor is 0.95.

The experiment consists of three processes: preprocessing process, offline training process and online VM migration process. In the process of offline training, the processed data are input into actor network and critic network. During the online computing offload process, each vehicle chooses to offload its computing task to a MEC server.

5. Experimental Results and Analysis

Experiments compare the performance of the VM migration strategy based on MADRL strategy proposed in this paper with the following three VM migration strategies.

(1) Random migration strategy (Random): Randomly migrate VMs to a server.
(2) Nearest neighbor migration strategy (NN): Directly select the nearest MEC server to migrate the VM.
(3) Centralized migration strategy based on Actor–Critic algorithm (CON_AC): Based on the centralized network architecture of SDN, the migration strategy is generated by the SDN controller.
(4) Distributed migration strategy based on Actor–Critic algorithm (DIS_AC): Each agent interacts with the environment individually to learn the VM migration strategy, and the agents do not communicate with each other.

In the experiment, the neural networks used by CON_AC, DIS_AC and MADRL are all actor and critic networks based on the parameter configuration described in the previous section, and they use the same reward function. The experiment has 2000 iterations, and the maximum number of iterative steps in each round is 50. Experiments compare the convergence of the above three migration strategies based on reinforcement learning when the number of servers is 10 and the number of VMs is 20, as shown in Figure 3.

As can be seen from Figure 3, DIS_AC does not converge in this scenario. The DIS_AC method is based on the Actor–Critic method. During the training, each agent only makes the migration strategy according to the local information. When an agent makes a decision, the environment state has changed. It is precisely because the agents compete for resources and lack of awareness of the overall environment, which can easily lead to the fluctuation of the reward and the decrease of the reward value. The convergence of CON_AC is the best, and the MADRL strategy proposed in this paper is slightly inferior, but very close. This is because CON_AC can obtain the global server and VM status through the SDN controller, and only one agent exists in the SDN controller in this environment. On the other hand, MADRL has multiple agents, so it is necessary to construct the global state according to the observed state of each agent. Therefore, due to the interaction between agents, the convergence speed of the model is a little slower than that of CON_AC, and it begins to converge in about the 800th round.

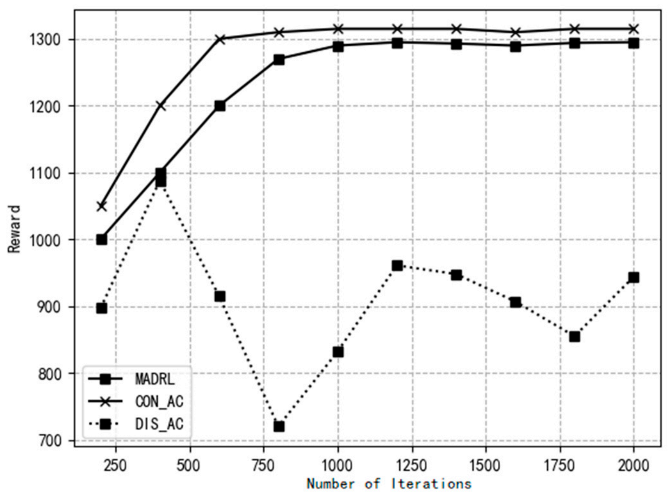

Figure 3. Convergence of different strategy.

Figure 4 shows the time it takes for MADRL and CON_AC to obtain strategies as the number of servers increases with the number of VMs 10 and 20, respectively. The time it takes to obtain a strategy is expressed as the time it takes from the start of the triggered migration to the time before the migration strategy is executed. The experiments are all carried out on the same server, and the parameters are fixed.

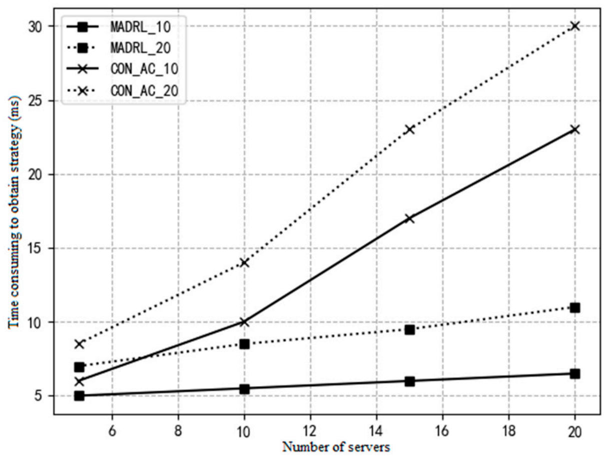

Figure 4. Time needed to obtain migration strategy in different number of servers.

As we can see from Figure 4, MADRL has the lowest time consumption. The increase in the number of servers has little impact on MADRL, and only when the number of VMs increases, it will increase the time it takes to obtain strategies. However, CON_AC is very affected by the number of servers, and when the number of VMs is 10, the time spent increases 2.8 times as the number of servers increases from 5 to 20. The reason is that MADRL only needs local observation information to obtain the migration action, and does not need to communicate with other servers, which speeds up the perception of the current environment and generates the migration strategy more quickly. On the other hand, CON_AC needs to communicate with all servers to obtain the current global information. As the number of servers increases, the complexity of the network increases exponentially, increasing the time it takes to generate a migration strategy.

Figure 5 shows the number of VMs migrated per hour within 24 h. As can be seen from the figure, the number of NN is the least, followed by NN, DIS_AC, MADRL and

CON_AC, which are the most effective. This is because the migration strategies between servers coordinate with each other to avoid centralized migration to the same server, which can speed up the processing speed of the system. Compared to Random, NN, and DIS_AC, MADRL migrated 93.57%, 25.59%, and 21.61% more VMs per hour on average, respectively.

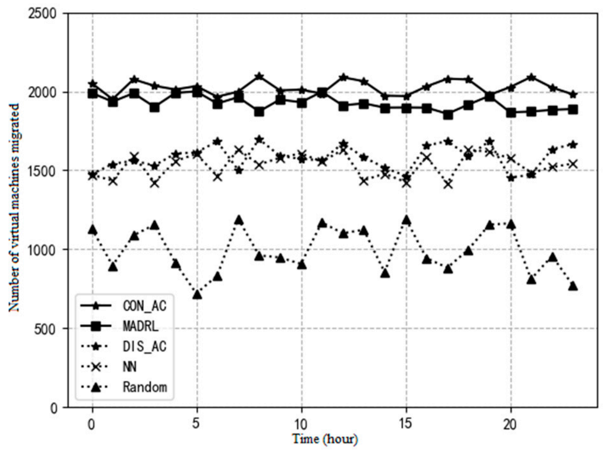

Figure 5. Number of VMs per hour with different migration strategies.

In this paper, the average energy consumption and migration success rate of the five migration strategies are compared. The number of servers set in the experiment is 10, and there are 20 VMs to be migrated in each time period, with a total of 50 iterations. The experimental results are shown in Figure 6.

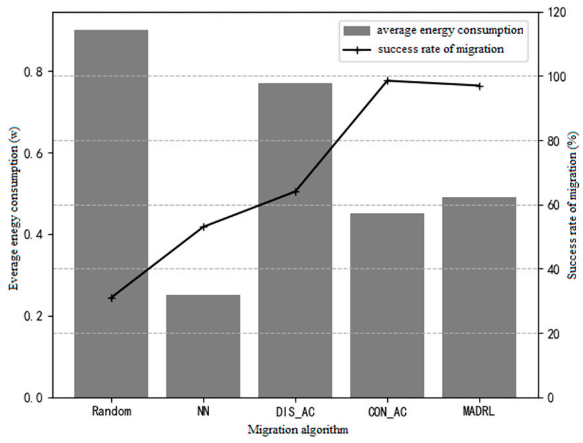

Figure 6. Average energy consumption and migration success rate of different migration strategies.

As can be seen from Figure 6, Random has the highest energy consumption and the lowest migration success rate because it migrates randomly and does not optimize energy consumption. NN has the lowest energy consumption, but the migration success rate is 45.4% lower than that of MADRL. The main reason is that the NN strategy always migrates VMs to the nearest server, so the total energy consumption is the lowest. However, the nearest server may not meet the migration conditions, resulting in low success rate of the migration. The migration result of MADRL is similar to that of CON_AC. While optimizing the energy consumption of migration, we can take into account the status of other servers, reduce the probability of migration failure, and obtain better migration performance.

6. Conclusions

This paper uses VM migration to solve the problem of service degradation as the distance increases. Considering the difficulty of synchronizing information in a distributed computing environment, a VM migration strategy based on MADRL is proposed. Firstly, the mobile edge network model is given, then the energy consumption of the migration problem is modeled, and the optimization goal is established. Then, a solving method based on MADRL is given. Finally, a simulation experiment of the VM migration strategy based on MADRL proposed in this paper is carried out. Through the comparison of convergence between different strategies, the number of migrated VMs, and energy consumption, it can be seen that the strategy proposed in this paper has better results. This paper assumes that the task cannot be divided. However, in the actual application, the task may be divided into multiple parts. Then, how to solve the problem in this scenario will be the focus of future work.

Author Contributions: Conceptualization, L.Y.; methodology, Y.D. and Q.Z.; software, Y.D., Q.Z. and L.Y.; validation, Y.D., Q.Z. and L.Y.; formal analysis, Y.D., Q.Z. and L.Y.; investigation, Y.D., Q.Z. and L.Y.; data curation, Y.D., Q.Z. and L.Y.; writing—original draft preparation, Y.D., Q.Z. and L.Y.; writing—review and editing, Y.D., Q.Z. and L.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by Chinese national key research plan: 2019YFB1405402.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: In the simulation process, server running status is generated based on the machine_usage.csv file in the open source dataset cluster-trace-v2018 [17]. When considering the task mobility, this experiment uses real vehicle driving datasets to validate the proposed VM migration strategy. The dataset consists of 4316 cabs driving for 24 h in Shanghai, China on 20 February 2007 [18]. Each vehicle trajectory consists of a set of time-stamped GPS points, and each GPS point has seven fields, which are cab ID, time stamp, longitude and latitude of the cab's current location, instantaneous speed at this time, clockwise angle to north (unit: 2 degrees), and cab status (status: empty or with passengers).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Allam, H.; Nassiri, N.; Rajan, A.; Ahmad, J. A critical overview of latest challenges and solutions of mobile cloud computing. In Proceedings of the 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC), Valencia, Spain, 8–11 May 2017; pp. 225–229.
2. Moussa, M.I.; Elgendy, M.A.; Hawish, A.S. An enhanced version of the MCACC to augment the computing capabilities of mobile devices using cloud computing. *Int. J. Adv. Comput. Sci. Appl.* **2014**, *4*, 1–10.
3. Patel, M.; Naughton, B.; Chan, C.; Sprecher, N.; Abeta, S.; Neal, A. *Mobile-Edge Computing Introductory Technical White Paper*; ETSI: Valbonne, France, 2014; Volume 29, pp. 854–864.
4. Kumar, K.; Liu, J.; Lu, Y.; Bhargava, B. *A Survey of Computation Offloading for Mobile Systems*; Springer: Berlin, Germany, 2013; Volume 18, pp. 129–140.
5. Cox, J.H.; Chung, J.; Donovan, S.; Ivey, J.; Clark, R.J.; Riley, G.; Owen, H.L. Advancing software-defined networks: A survey. *IEEE Access* **2017**, *5*, 25487–25526. [CrossRef]
6. Bi, Y.; Han, G.; Lin, C.; Deng, Q.; Guo, L.; Li, F. Mobility support for fog computing: An SDN approach. *IEEE Commun. Mag.* **2018**, *56*, 53–59. [CrossRef]
7. Chen, X.; Pu, L.; Gao, L.; Wu, W.; Wu, D. Exploiting massive D2D collaboration for energy-efficient mobile edge computing. *IEEE Wirel. Commun.* **2017**, *24*, 64–71. [CrossRef]
8. Chen, X.; Li, W.; Lu, S.; Zhi, Z.; Fu, X. Efficient resource allocation for on-demand mobile-edge cloud computing. *IEEE Trans. Veh. Technol.* **2018**, *67*, 8769–8780. [CrossRef]
9. Kikuchi, J.; Wu, C.; Ji, Y.; Murase, T. Mobile edge computing based VM migration for QoS improvement. In Proceedings of the 2017 IEEE 6th Global Conference on Consumer Electronics (GCCE), Nagoya, Japan, 24–27 October 2017; pp. 1–5.
10. Clark, C.; Fraser, K.; Hand, S.; Hansen, J.G.; Warfield, A. Live migration of virtual machines. In Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation, Boston, MA, USA, 2–4 May 2005; pp. 273–286.
11. Fang, P.; Zhao, Y.; Liu, Z.; Gao, J.; Chen, Z. Resource allocation strategy for MEC system based on VM migration and RF energy harvesting. In Proceedings of the 2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring), Antwerp, Belgium, 25–28 May 2020; pp. 1–6.
12. Cong, S.; Tekin, C.; Schaar, M. A non-stochastic learning approach to energy efficient mobility management. *IEEE J. Sel. Areas Commun.* **2016**, *34*, 3854–3868.

13. Nademnega, A.; Hafid, A.S.; Brisebois, R. Mobility prediction model-based service migration procedure for follow me cloud to support QoS and QoE. In Proceedings of the IEEE International Conference on Communications (ICC), Kuala Lumpur, Malaysia, 23–27 May 2016; pp. 1–6.
14. Wang, S.; Urgaonkar, R.; Chan, K.; He, T.; Zafer, M.; Leung, K.K. Dynamic service placement for mobile micro-clouds with predicted future costs. In Proceedings of the IEEE International Conference on Communications (ICC), London, UK, 8–12 June 2015; pp. 5504–5510.
15. Aissioui, A.; Ksentini, A.; Gueroui, A.; Taleb, T. On enabling 5G automotive systems using follow me edge-cloud concept. *IEEE Trans. Veh. Technol.* **2018**, *67*, 5302–5316. [CrossRef]
16. Taleb, T.; Ksentini, A. An analytical model for follow me cloud. In Proceedings of the 2013 IEEE Global Communications Conference (GLOBECOM), Atlanta, GA, USA, 9–13 December 2013; pp. 1291–1296.
17. Ksentini, A.; Taleb, T.; Min, C. A markov decision process-based service migration procedure for follow me cloud. In Proceedings of the IEEE International Conference on Communications (ICC), Sydney, Australia, 10–14 June 2014; pp. 1350–1354.
18. Wang, S.; Urgaonkar, R.; He, T.; Zafer, M.; Chan, K.; Leung, K.K. Mobility-induced service migration in mobile micro-clouds. In Proceedings of the IEEE Military Communications Conference (MILCOM), Baltimore, MD, USA, 6–8 October 2014; pp. 835–840.
19. Taleb, T.; Ksentini, A.; Frangoudis, P.A. Follow-me cloud: When cloud services follow mobile users. *IEEE Trans. Cloud Comput.* **2016**, *7*, 369–382. [CrossRef]
20. Wang, S.; Urgaonkar, R.; Zafer, M.; He, T.; Chan, K.; Leung, K.K. Dynamic service migration in mobile edge-clouds. In Proceedings of the IFIP Networking Conference (IFIP Networking), Toulouse, France, 20–22 May 2015; pp. 1–9.
21. Hai, J.; Li, D.; Song, W.; Shi, X.; Pan, X. Live virtual machine migration with adaptive, memory compression. In Proceedings of the 2009 IEEE International Conference on Cluster Computing, New Orleans, LA, USA, 31 August–4 September 2009; pp. 1–10.
22. Ali, Z.; Kahf, S.; Abbas, Z.H.; Abbas, G.; Kim, S. A Deep learning approach for mobility-aware and energy-efficient resource allocation in MEC. *IEEE Access* **2020**, *8*, 179530–179546. [CrossRef]
23. Nguyen, T.A.; Kim, D.S.; Park, J.S. Availability modeling and analysis of a data center for disaster tolerance. *Future Gener. Comput. Syst.* **2016**, *56*, 27–50. [CrossRef]
24. Kapil, D.; Pilli, E.S.; Joshi, R.C. Live virtual machine migration techniques: Survey and research challenges. In Proceedings of the 2013 3rd IEEE International Advance Computing Conference (IACC), Ghaziabad, India, 22–23 February 2013; pp. 963–969.
25. Ma, F.; Liu, F.; Liu, Z. Live virtual machine migration based on improved pre-copy approach. In Proceedings of the 2010 IEEE International Conference on Software Engineering and Service Sciences, Beijing, China, 19 August 2010; pp. 230–233.
26. Dong, Y.; Zhu, H.; Peng, J.; Wang, F.; Mesnier, M.; Wang, D.; Chan, S. RFS: A Network File System for Mobile Devices and the Cloud. In *ACM SIGOPS Operating Systems Review*; Association for Computing Machinery: New York, NY, USA, 2001; Volume 45, pp. 101–111.
27. Clusterdata. Available online: https://github.com/alibaba/clusterdata/blob/v2018/cluster-trace-v2018/trace_2018.md (accessed on 1 June 2021).
28. Smart City Research Group. Available online: https://www.cse.ust.hk/scrg/ (accessed on 3 May 2021).