Deep Reinforcement Learning Enabled Adaptive Virtual Machine Migration Control in Multi-Stage Information Processing Systems

Yukinobu Fukushima
Faculty of Environmental, Life, Natural Science and Technology
Okayama University
Okayama, Japan
fukusima@okayama-u.ac.jp

Yuki Koujitani
Graduate School of Natural Science and Technology
Okayama University
Okayama, Japan
ppxw4aek@s.okayama-u.ac.jp

Kazutoshi Nakane
Graduate School of Information Science
Nagoya University
Nagoya, Japan
nakane@net.itc.nagoya-u.ac.jp

Yuya Tarutani
Graduate School of Engineering
Osaka University
Osaka, Japan
tarutani@comm.eng.osaka-u.ac.jp

Celimuge Wu
Graduate School of Informatics and Engineering
The Univ. of Electro-Commun.
Tokyo, Japan
celimuge@uec.ac.jp

Yusheng Ji
Information Systems Architecture Research Division
National Institute of Informatics
Tokyo, Japan
kei@nii.ac.jp

Tokumi Yokohira
Faculty of Interdisciplinary Science and Engineering in Health Systems
Okayama University
Okayama, Japan
yokohira@okayama-u.ac.jp

Tutomu Murase
Graduate School of Information Science
Nagoya University
Nagoya, Japan
tom@itc.nagoya-u.ac.jp

***Abstract*—This paper tackles a Virtual Machine (VM) migration control problem to maximize the progress (accuracy) of information processing tasks in multi-stage information processing systems. The conventional methods for this problem are effective only for specific situations, such as when the system load is high. In this paper, in order to adaptively achieve high accuracy in various situations, we propose a VM migration method using a Deep Reinforcement Learning (DRL) algorithm. It is difficult to directly apply a DRL algorithm to the VM migration control problem because the size of the solution space of the problem dynamically changes according to the number of VMs staying in the system while the size of the agent's action space is fixed in DRL algorithms. To cope with this difficulty, the proposed method divides the VM migration control problem into two problems: the problem of determining only the VM distribution (i.e., the proportion of the number of VMs deployed on each edge server) and the problem of determining the locations of all the VMs so that it follows the determined VM distribution. The former problem is solved by a DRL algorithm, and the latter by a heuristic method. This approach makes it possible to apply a DRL algorithm to the VM migration control problem because the VM distribution is expressed by a vector with a fixed number of dimensions and can be directly outputted by the agent. The simulation results confirm that our proposed method can adaptively achieve quasi-optimal accuracy in various situations with different link delays, types of the information processing tasks and the number of VMs.**

***Keywords*-Multi-stage information processing system; VM migration control; Deep reinforcement learning; Deep Deterministic Policy Gradient (DDPG)**

I. Introduction

This paper is an extended and improved version of an earlier paper presented at the IARIA International Conference on Networks (ICN 2024) [1] in Barcelona, Spain.

In recent years, ultra-real-time services, such as Cross Reality (XR) and automated driving, are expected to appear. In these services, information processing tasks requested by clients need to be executed immediately (e.g., on the order of milliseconds) and the progress (accuracy) of the processing results should be as high as possible.

A multi-stage information processing system [2] [3] is one of the promising candidates for the edge computing infrastructures for ultra-real-time services. In the system, information processing tasks requested by clients are executed in parallel by an edge server and a data center. The edge server prioritizes responsiveness over accuracy; it returns the highly responsive but low accurate processing results to the clients while the data center prioritizes accuracy over responsiveness; it return the highly accurate but low responsive processing results to the clients. When operating ultra-real-time services in a multi-stage information processing system, it is important to maximize the accuracy of information processing tasks executed by the edge servers while satisfying the responsiveness requested by clients.

Previous researches on multi-stage information processing systems focused on improving the accuracy of information processing tasks executed by edge servers through Virtual Machine (VM) migration control [2] [3]. VM migration control dynamically migrates VMs, which execute the information processing tasks requested by clients on edge servers, among multiple edge servers, which leads to effective use of CPU resources on edge servers, appropriate adjustment of CPU times allocated to the tasks and reduction of the communication delay between clients and VMs, thereby improving the

accuracy of the tasks. In the previous researches, as heuristic methods for VM migration control, VM sweeping method [3], VM number averaging method [3], early-blooming type priority processing method [2], and late-blooming type priority processing method [2] were proposed and their effectiveness were confirmed. These methods are, however, effective only in specific situations, such as when the system load is high and the type of information processing tasks is the late-blooming type. Since the system load and the type of information processing tasks change dynamically, VM migration control that can adaptively achieve high accuracy in a wide variety of situations is needed.

In this paper, in order to adaptively achieve high accuracy in a variety of situations, we propose a VM migration method using a Deep Reinforcement Learning (DRL) algorithm. DRL algorithms are expected to adaptively achieve a quasi-optimal performance in a variety of situations through interactions between a learning agent and a dynamically changing environment. On the other hand, it is difficult to directly apply a DRL algorithm to the VM migration control problem because, in the problem, the size of the solution space dynamically changes according to the dynamic changes in the number of VMs staying in the system while the size of the agent's action space is fixed in DRL algorithms, and consequently it is difficult for the agent to directly output an solution for the problem. To cope with this difficulty, in this paper, we divide the VM migration control problem into two problems: the problem of determining only the VM distribution (i.e., the proportion of the number of VMs deployed on each edge server) and the problem of determining the locations of all the VMs so that it follows the determined VM distribution. The former problem is solved by a DRL algorithm, and the latter by a heuristic method. This approach makes it possible to apply a DRL algorithm with a fixed action space size to the VM migration control problem because the VM distribution is expressed by a vector with a fixed number of dimensions and can be directly outputted by the agent.

The rest of this paper is organized as follows. Section II introduces related work on VM migration control. Section III describes the multi-stage information processing system and the VM migration control problem. In Section IV, we propose a VM migration method using a DRL algorithm. In Section V, we evaluate the effectiveness of our proposed method with computer simulations. In Section VI, we summarize the paper.

II. Related Work

The previous researches in [4]–[11] tackle VM migration control problems in server migration services, and propose heuristic methods [4] [6], mathematical programming methods [5], [7]–[9], [11], and Q-learning methods [10]. These methods, however, aim at improving the communication quality between clients and VMs and reducing network power consumption, and do not consider the accuracy of information processing tasks.

The previous researches in [2] [3] tackle VM migration control problems in multi-stage information processing systems,

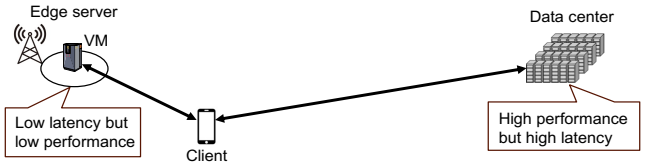

Figure 1. Multi-stage information processing systems.

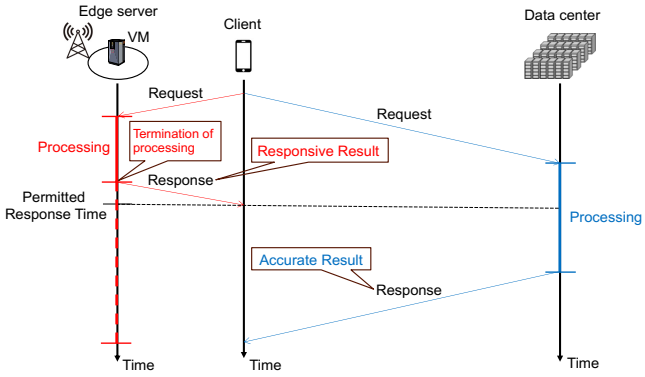

Figure 2. Flow of information processing in a multi-stage information processing system.

and propose the heuristic methods; VM sweeping method [3], VM number averaging method [3], early-blooming type priority processing method [2], and late-blooming type priority processing method [2]. These methods are, however, effective only in specific situations. For example, the VM sweeping method is shown to be effective only in situations where the system load is high and the type of information processing tasks is the late-blooming type. Since the system load and the type of information processing tasks change dynamically, VM migration control that can adaptively achieve high accuracy in a wide variety of situations is needed.

The previous researches in [12] [13] tackle VM migration control problems in mobile edge computing, and propose VM migration methods using Deep Q-Network (DQN) [14], which is a kind of DRL algorithms. These methods, however, can only be applied to VM migration control problems with a single VM because the size of an agent's action space is fixed in DQN, and cannot be applied to VM migration control problems with multiple VMs.

III. Multi-Stage Information Processing Systems

As shown in Figure 1, a multi-stage information processing system consists of edge servers located proximate (e.g., base stations) to clients and data centers located distant from them. The system provides clients with both highly responsive and highly accurate processing results by executing information processing tasks in parallel at the edge servers and the data centers.

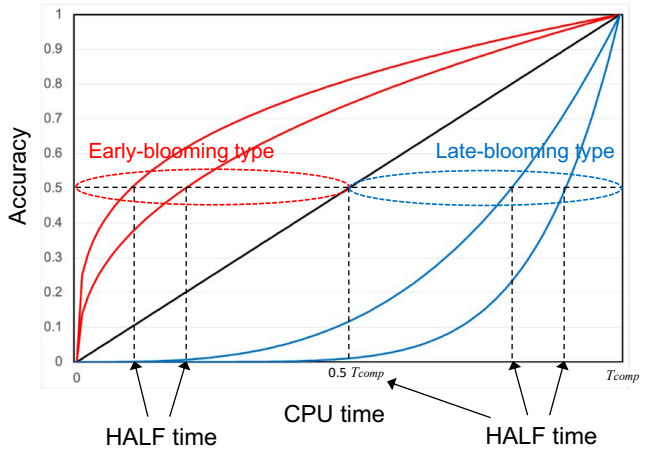

Figure 3. Relationship between CPU time allocated to a task and accuracy of the task.

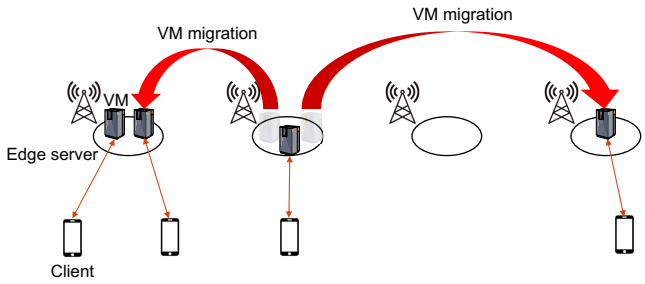

Figure 4. VM migration control in a multi-stage information processing system.

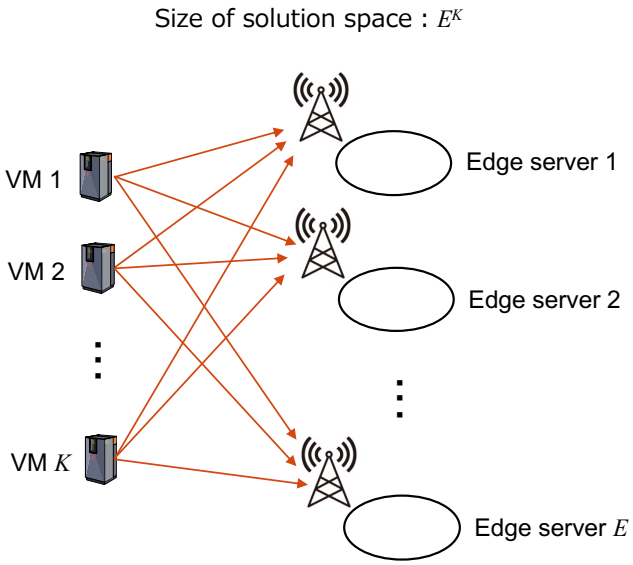

Figure 5. Size of solution space in VM migration control problem.

Figure 2 shows the flow of information processing in a multi-stage information processing system. A client requests both an edge server and a data center to process its task in parallel. When the response time permitted by the client approaches, the edge server terminates its processing to meet the permitted response time and returns the highly responsive processing result to the client. The data center, on the other hand, accomplishes its processing and returns the highly accurate processing result to the client.

In this paper, we adopt the accuracy model (i.e., the relationship between the CPU time (t_{CPU}) allocated to a task and the accuracy ($f(t_{CPU})$) of the task) in [3]. Figure 3 shows the accuracy model. In the model, the accuracy of the task is calculated as follows.

$$f(t_{CPU}) = \left(\frac{t_{CPU}}{T_{comp}}\right)^{\frac{log(0.5)}{log(\frac{HALFtime}{T_{comp}})}} \quad (1)$$

where T_{comp} represents the time for the task to be completed (i.e., accuracy reaches 1.0) and HALF time represents the time for the task to reach accuracy of 0.5. Tasks are classified based on their HALF time. The tasks with HALF time shorter than 0.5 T_{comp} are classified into early-blooming type, those with HALF time of 0.5 T_{comp} are classified into linear type, and those with HALF time longer than 0.5 T_{comp} are classified into late-blooming type.

In this paper, we tackle a VM migration control problem among multiple edge servers for maximizing the accuracy of information processing tasks executed by edge servers while satisfying the responsiveness requested by the clients (Figure 4). The objective of the problem is to maximize the sum of accuracies of all the information processing tasks. VM migration enables effective use of CPU resources on edge servers, appropriate adjustment of CPU times allocated to the tasks and reduction of the communication delay between clients and VMs, thereby improving the accuracy of the tasks. On the other hand, VM migration stops the execution of the tasks during the VM migration time, which may decrease the accuracy of the tasks. We need to carefully determine the locations of the VMs with consideration of the pros and cons of VM migration.

IV. Proposed Method

In this paper, in order to adaptively achieve high accuracy in a variety of situations, we propose a VM migration method using a Deep Reinforcement Learning (DRL) algorithm. In reinforcement learning, an agent learns policy (i.e., how to map a situation to an action) from interactions with an environment in discrete timesteps. At each timestep t, the agent observes state s_t of the environment, takes action a_t and receives reward r_t. The objective of the agent is to acquire the policy that maximizes the discounted cumulative reward $R_t = \sum_{i=t}^{T} \gamma^{i-t} r_i$ where $\gamma \in [0, 1]$ is the discount rate. We believe that DRL is promising for VM migration control because the agent can adaptively learn an appropriate policy in accordance with the dynamically changing environment.

With regard to applying a DRL algorithm to a VM migration control problem in multi-state information processing systems,

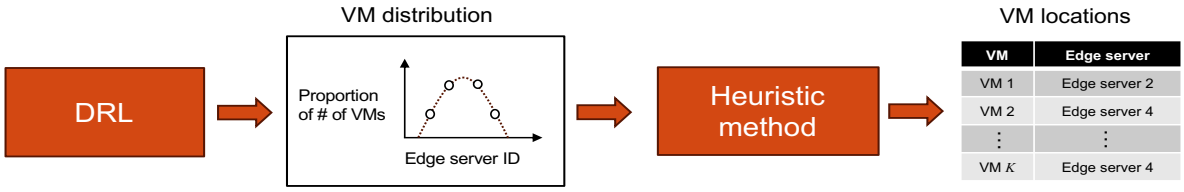

Figure 6. Outline of our proposed method.

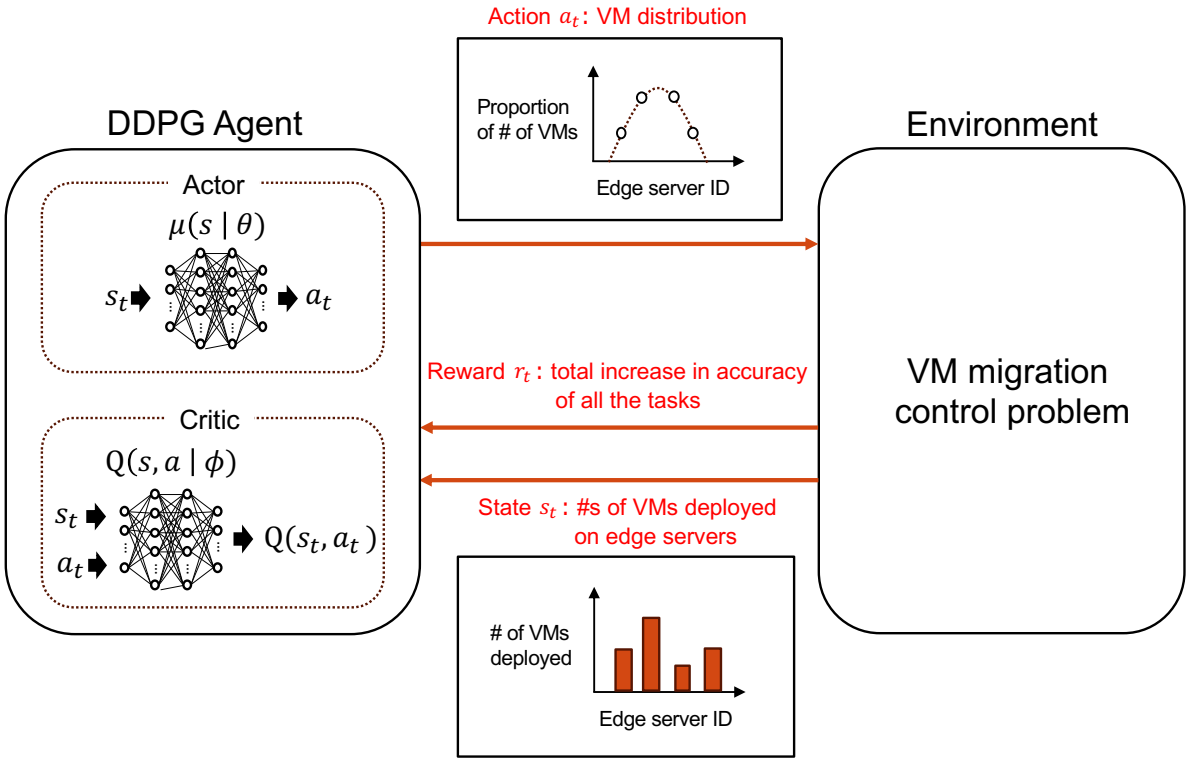

Figure 7. Interaction between the DDPG agent and the environment.

it should be noted that the size of the solution space (i.e., the total number of all possible solutions) of the problem dynamically changes according to the dynamic changes in the number of VMs staying in the system. As shown in Figure 5, the size of the solution space is E^K where E is the number of edge servers and K is the number of VMs, and the size of the solution space E^K dynamically changes according to the number of VMs K. On the other hand, the size of the agent's action space in DRL algorithms is fixed. For example, an agent in Deep Deterministic Policy Gradient (DDPG) [15] outputs a vector with a fixed number of dimensions. Therefore, it is difficult for an agent to directly output an solution for the VM migration control problem.

To cope with the dynamic change in the size of solution space, we divide the VM migration control problem into two problems (Figure 6): the problem of determining only the VM distribution (i.e., the proportion of the number of VMs deployed on each edge server) and the problem of determining the locations of all the VMs so that it follows the determined VM distribution. The former problem is solved by a DRL algorithm, and the latter problem is solved by a heuristic method. This approach makes it possible to apply a DRL algorithm with a fixed action space size to the VM migration control problem because the VM distribution is expressed by a vector with a fixed number of dimensions and can be directly outputted by an agent.

We adopt DDPG [15] as a DRL algorithm. DDPG approximates both a policy function $\mu(s|\theta)$ (Actor) and an action-value function $Q(s, a|\phi)$ (Critic) with deep neural networks. Actor $\mu(s|\theta)$ maps a given state to an action to be taken. Critic $Q(s, a|\phi)$ maps a given state-action pair to the expected value of the discounted cumulative reward if the action is taken under the state. During the training phase, Critic $Q(s, a|\phi)$ and Actor $\mu(s|\theta)$ are updated using experiences, which are

expressed with the tuple (s_t, a_t, r_t, s_{t+1}), obtained in interactions with the environment. As for Critic $Q(s, a|\phi)$, weights ϕ of $Q(s, a|\phi)$ are updated with a gradient decent method so that the following loss L is minimized:

$$L = \mathbb{E}\Big[(y_t - Q(s, a|\phi))^2\Big] \tag{2}$$

where $y_t = r_t + \gamma Q(s_{t+1}, \mu(s_{t+1}|\theta)|\phi)$. As for Actor $\mu(s|\theta)$, weights θ of $\mu(s|\theta)$ are updated with a gradient ascent method so that the performance (J) of the actor (i.e., expected value of the discounted cumulative reward) is maximized. In the gradient ascent method, the policy gradient $\nabla_\theta J$ is calculated by applying the chain rule to J with respect to weights θ as follows.

$$\begin{aligned}\nabla_\theta J &\approx \mathbb{E}\Big[\nabla_\theta Q(s, a|\phi)\Big] \\ &= \mathbb{E}\Big[\nabla_a Q(s, \mu(s|\theta)|\phi)\nabla_\theta \mu(s|\theta)\Big]\end{aligned} \tag{3}$$

In DDPG, Actor can output the VM distribution (i.e., the proportion of the number of VMs deployed on each edge server) as an action because it can operate over continuous action space. As well as DQN [14], DDPG adopts experience replay and target network techniques in order to train Actor and Critic in a stable and robust way.

Figure 7 depicts an interaction between a DDPG agent and an environment, which corresponds to the VM migration control problem. When applying a DRL algorithm to the VM migration control problem, we need to define action, state, and reward in accordance with the problem. Action a_t of the agent is defined as the VM distribution (i.e., the proportion of the number of VMs deployed on each edge server), and is expressed with the following equation.

$$a_t = (p_1, p_2, \ldots, p_E) \tag{4}$$

where p_i is the proportion of the number of VMs deployed on edge server i. State s_t of the environment is defined as the numbers of VMs deployed on edge servers for observing the load of each edge server, and is expressed with the following equation.

$$s_t = (d_1, d_2, \ldots, d_E) \tag{5}$$

where d_i is the number of VMs deployed on edge server i. Reward r_t is defined as the total increase in accuracy of all the tasks during the period from the last VM migration control to the current one.

Algorithm 1 in Figure 8 shows the procedure of our proposed method. In line 1, we generate Actor $\mu(s|\theta)$ and Critic $Q(s, a|\phi)$, and randomly initialize weights θ and ϕ. In lines 2 and 3, we generate the target networks of Actor and Critic, initialize their weights with those of Actor and Critic, and initialize replay buffer R, which stores a set of experiences for experience replay. The procedures in lines 4 to 17 and those in lines 7 to 16 are repeated for each episode and for each timestep of the episode, respectively. In lines 8 to 11, the agent selects action a_t (i.e., VM distribution), determines locations of all the VMs by the heuristic, observes reward r_t and next state s_{t+1}, and stores the obtained experience (s_t, a_t, r_t, s_{t+1}) to the replay buffer. Please note that a random noise $\mathcal{N}$ is added to the output by Actor for action exploration. In lines 12 to 15, we train Actor, Critic and target networks. In line 13, we update weights ϕ of Critic $Q(s, a|\phi)$ with a gradient descent method. Please note that we use target networks $Q'(s, a|\phi')$ and $\mu'(s|\theta')$ instead of Critic $Q(s, a|\phi)$ and Actor $\mu(s|\theta)$ for calculating target value y_i. In line 14, we update weights θ of Actor $\mu(s|\theta)$ with a gradient ascent method. In line 15, we update weights of target networks.

Algorithm 1 Procedure of our proposed method

1: Randomly initialize weights θ of Actor $\mu(s|\theta)$ and weights ϕ of Critic $Q(s, a|\phi)$
2: Initialize weights of Actor's target network $\mu'(s|\theta')$ and Critic's target network $Q'(s, a|\phi')$: $\theta' \leftarrow \theta$, $\phi' \leftarrow \phi$
3: Initialize replay buffer R
4: **for** episode = 1, M **do**
5: Initialize a random noise $\mathcal{N}$ for action exploration
6: Observe initial state s_1 from the environment
7: **for** t = 1, T **do**
8: Select VM distribution $a_t = \mu(s_t|\theta) + \mathcal{N}_t$ as action
9: Determine locations of all the VMs by the heuristic method among the VM locations that follow the determined VM distribution a_t, and migrates the VMs
10: Observe reward r_t and the next state s_{t+1}
11: Store experience (s_t, a_t, r_t, s_{t+1}) in R
12: Sample a random minibatch of N experiences (s_i, a_i, r_i, s_{i+1}) from R
13: Learning of Critic:
Calculate target value y_i:
$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta')|\phi')$
Update weights ϕ with a gradient descent method so that loss $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\phi))^2$ is minimized
14: Learning of Actor:
Calculate policy gradient $\nabla_\theta J$:
$\nabla_\theta J \propto \frac{1}{N}\sum_i \nabla_a Q(s_i, \mu(s_i|\theta)|\phi)\nabla_\theta \mu(s_i|\theta)$
Update weights θ with a gradient ascent method so that performance of Actor J is maximized
15: Update weights of target networks:
$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$
$\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
16: **end for**
17: **end for**

Figure 8. Procedure of our proposed method.

After determining the VM distribution, we determine the locations of all the VMs by a heuristic method so that it follows the determined VM distribution. In this paper, we adopt a minimum client-VM delay method as the heuristic method. The minimum client-VM delay method selects the VM location with the minimum sum of the delays between clients and VMs in a brute force manner among the VM

locations that follow the VM distribution determined by the DDPG agent.

V. Performance Evaluation

In this section, we evaluate our proposed method with computer simulations. Section V.A explains the simulation model. Section V.B shows the evaluation results.

A. Simulation Model

We developed the VM migration control simulator and the DDPG agent with OpenAI Gym [16] and Keras-rl [17], respectively. Table I summarizes the parameter settings as to the DDPG agent. We adopt the same parameter values as those used by the DDPG agent in Keras-rl [17] because the previous research [15] reports that a DDPG agent with the same parameter setting successfully solved various physics tasks.

The left side of Figure 9 shows the network model. The network consist of four edge servers, which are connected in a full mesh manner. An edge server equally allocates its CPU time to all the VMs located on it. A VM is individually generated for each client, that is, the number of VMs is equal to the number of clients. We set the response time permitted by a client to 110 [ms] and the completion time of an information processing task (T_{comp}) to 110 [ms].

In order to evaluate whether our proposed method can adaptively cope with various situations, we change 1) link delay, 2) task type, and 3) total number of clients as follows.

1) link delay
   We assume that the delays of all the links are identical. We set the delay of each link to one of the following values: 1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 [ms].
2) task type
   We assume that task type (i.e., HALF time) of all the information processing tasks are identical. We set HALF time of each task to either (1) 11 [ms] (= 0.1 $\times T_{comp}$) assuming the task type is the early-blooming type, (2) 55 [ms] (= 0.5 $\times T_{comp}$) assuming the task type is the linear type, or (3) 99 [ms] (= 0.9 $\times T_{comp}$) assuming the task type is the late-blooming type.
3) total number of clients
   We set the total number of clients that join the system (and the corresponding VMs) to either four or eight.

During an episode of the simulation, the following events occur (right side of Figure 9). When an episode starts, four or eight clients join the system in turn at time 0.1 [ms] with the interval of 0.1 [ms]. The locations of all the clients are fixed at edge server 1 during the episode. The initial locations of all the VMs are set to edge server 1. At time 3 [ms], we perform the first VM migration control. Then, at time 103 [ms], we perform the second VM migration control. Lastly, all the clients leave the system in turn at time 110.1 [ms] with the interval of 0.1 [ms]. The first VM migration control aims at determining the locations of the VMs during the episode and the second VM migration control aims at obtaining the reward and the experience for training the DDPG agent.

TABLE I
Parameter settings

Parameter	Value
Number of training episodes (M)	10,000
Discount rate (γ)	0.99
Number of hidden layers	Actor : 2, Critic : 5
Number of neurons in a hidden layer	Actor : 256, 256, Critic : 16, 32, 32, 256, 256
Activation function of hidden layers	Actor : relu, Critic : relu
Learning rate (α)	Actor : 0.001, Critic : 0.002
Noise process for action exploration ($\mathcal{N}$)	Ornstein-Uhlenbeck process
Size of replay buffer	10,000
Minibatch size (N)	64
Weights of updated parameters when updating the weights of target networks (τ)	0.005

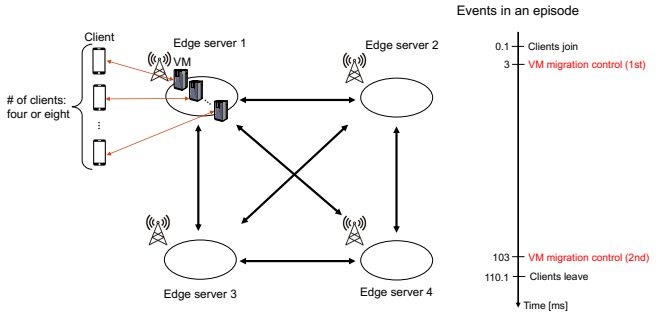

Figure 9. Network model and events in an episode.

We compare our proposed method with the following methods.

- VM sweeping method [3]
  It classifies all the edge servers into a congested edge server and working edge servers. On each of the working edge servers, a single VM with higher accuracy increase rate is individually deployed so that the VM can occupy the CPU time on the edge server. On the congested edge server, the remaining VMs are aggregated.
- VM number averaging method [3]
  It equally distributes all the VMs to all the edge servers for load balancing.
- Non-migration method
  It fixes all the VMs at their initial edge server (i.e., edge server 1).
- Minimum client-VM delay method
  It locates each of the VMs on the edge server most proximate to its client.

B. Evaluation Results

Figure 10 shows the average accuracy as a function of link delay for all the VM migration methods when the task type is the early-blooming type (HALF time = 11 [ms]) and the total number of clients (and the corresponding VMs) is four. The average accuracy of our proposed method (DDPG + Minimum client-VM delay method) is plotted with 95% confidence interval of 50 trials because it varies trial-by-trial

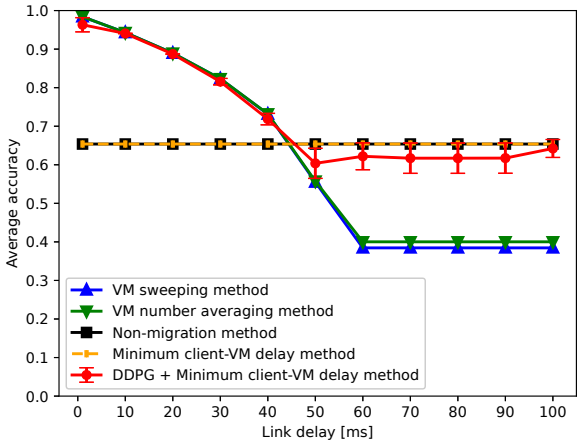

Figure 10. Average accuracy as a function of link delay (HALF time: 11 [ms], Total number of clients: 4).

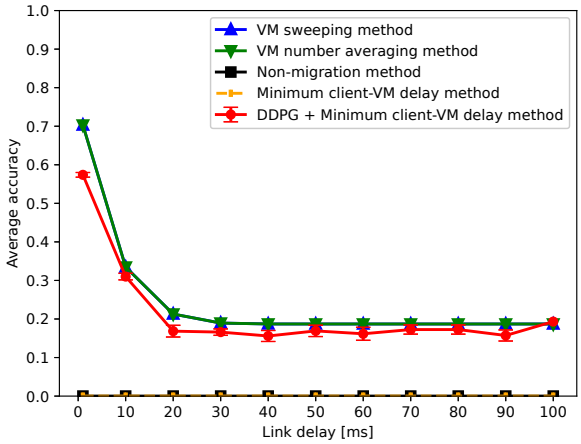

Figure 12. Average accuracy as a function of link delay (HALF time: 99 [ms], Total number of clients: 4).

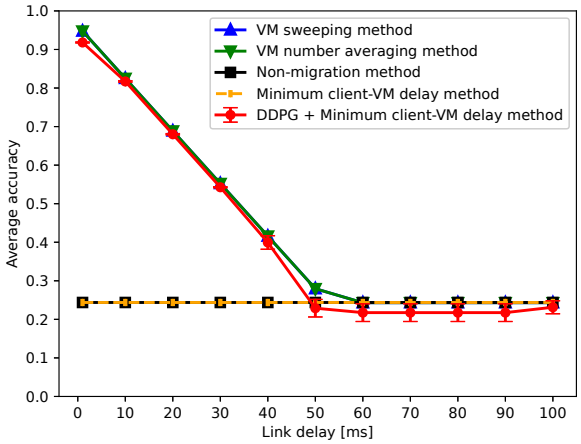

Figure 11. Average accuracy as a function of link delay (HALF time: 55 [ms], Total number of clients: 4).

depending on the initial weights of Actor and Critic, and the noises for action exploration.

Both non-migration method and minimum client-VM delay method show the constant accuracy of about 0.65 regardless of the link delay. This is because these methods always fix all the VMs at their initial edge server (edge server 1) regardless of the link delay. Both VM sweeping method and VM number averaging method achieve the maximum accuracy of about 0.98 when the link delay is 1 [ms], and the accuracy decreases as the link delay increases. This is explained as follows. These methods always distribute the VMs to all the edge servers so that a VM is individually located at an edge server regardless of the link delay. As the link delay increases, the VM migration time and the communication delay between the client and the VM increases, and consequently the CPU time allocated to the task decreases after completing the VM migration.

We compare the performances of non-migration method, minimum client-VM delay method, VM sweeping method, and VM number averaging method. When the link delay is shorter than or equal to 40 [ms], VM sweeping method and VM number averaging method achieve 12 to 50% higher accuracy than non-migration method and minimum client-VM delay method. Therefore, in this case, it is desirable to distribute all the VMs to different edge servers. When the link delay is longer than or equal to 50 [ms], non-migration method and minimum client-VM delay method achieve 17 to 70 % higher accuracy than VM sweeping method and VM number averaging method. Therefore, in this case, it is desirable to fix all the VMs at their initial edge server.

We focus on the performance of our proposed method. When the link delay is shorter than or equal to 40 [ms], our proposed method 1) achieves 10 to 49% higher accuracy than non-migration method and minimum client-VM delay method, and 2) achieves almost as high accuracy (at most 2% lower accuracy) as VM sweeping method and VM number averaging method, by successfully learning the policy that distributes all the VMs to all the edge servers similarly to VM sweeping method and VM number averaging method in most trials. When the link delay is longer than or equal to 50 [ms], our proposed method 1) achieves 9 to 68% higher accuracy than VM sweeping method and VM number averaging method, and 2) achieves almost as high accuracy (at most 6% lower accuracy) as non-migration method and minimum client-VM delay method, by successfully learning the policy that fixes all the VMs at their initial edge servers similarly to non-migration method and minimum client-VM delay method in most trials.

Figure 11 shows the average accuracy as a function of link delay for all the VM migration methods when the task type is changed to the linear type (HALF time = 55 [ms]) and the total number of clients is four. All of the conventional methods

select the same VM locations as those in Figure 10 because they determine VM locations without considering the task type; non-migration method and minimum client-VM delay method fix all the VMs at their initial locations while VM sweeping method and VM number averaging method distribute all the VMs to all the edge servers.

When the link delay is shorter than or equal to 50 [ms], VM sweeping method and VM number averaging method achieve 14 to 289% higher accuracy than non-migration method and minimum client-VM delay method because distributing the VMs to all the edge servers leads to more efficient use of CPU resources of all the edge servers thanks to short VM migration time. When the link delay is longer than or equal to 60 [ms], all the conventional methods shows the identical accuracy because only the VMs fixed at the initial edge server can execute the tasks due to long VM migration time, and those VMs achieve the identical accuracy in total regardless of their numbers for the linear model.

Our proposed method achieves almost as high accuracy (at most 16% lower accuracy) as VM sweeping method and VM number averaging method when the link delay is shorter than or equal to 50 [ms]. In most trials, our proposed method successfully learns the policy that distributes all the VMs to all the edge servers similarly to VM sweeping method and VM number averaging method. In addition, our proposed method achieves almost as high accuracy (at most 9% lower accuracy) as all the conventional methods when the link delay is longer than or equal to 60 [ms]. Although our proposed method learns various policies that determine different VM locations, most policies deploy at least one VM on the initial edge server, which leads to achieving the comparable accuracy as all the conventional methods.

Figure 12 shows the average accuracy as a function of link delay for all the VM migration methods when the task type is changed to the late-blooming type (HALF time = 99 [ms]) and the total number of clients is four. All of the conventional methods select the same VM locations as those in Figures 10 and 11.

Both non-migration method and minimum client-VM delay method show the accuracy close to zero regardless of link delay. This is because these methods fix all of the four VMs at the initial edge server and each of the VM is assigned only one fourth of the CPU time of the edge server, which is not enough for the late-blooming tasks to increase the accuracy. Both VM sweeping method and VM number averaging method achieve the accuracy of 0.70 when the link delay is 1 [ms], and the accuracy decreases as the link delay increases because the accuracies achieved by the three VMs distributed to edge servers 2, 3 and 4 decrease due to longer VM migration time. Our proposed method achieves almost as high accuracy (at most 18% lower accuracy) as VM sweeping method and VM number averaging method by successfully learning the policy that distributes all the VMs to all the edge servers similarly to VM sweeping method and VM number averaging method in most trials.

The average accuracy as a function of link delay for all the VM migration methods when the total number of clients is changed to eight are depicted in Figures 13, 14 and 15. Please note that VM sweeping method selects VM locations different from those by VM number averaging method; VM sweeping method distributes a single VM to each of edge servers 2, 3 and 4 and fixes the remaining five VMs at the initial edge server while VM number averaging method distributes two VMs to each of edge servers 2, 3 and 4 and fixes the remaining two VMs at the initial edge server. Non-migration method and minimum client-VM delay method fix all the VMs at their initial edge server.

In Figure 13 where the task type is the early-blooming type (HALF time = 11 [ms]), when the link delay is shorter than or equal to 20 [ms], VM number averaging method achieves higher accuracy than VM sweeping method because the VMs distributed to edge servers 2, 3 and 4 in the former method gain higher accuracy than the VMs fixed at the initial edge server in the latter method. For example, when the link delay is 1 [ms], all the VMs gain accuracy of about 0.80 in VM number averaging method while the five VMs fixed at the initial edge server gain only accuracy of about 0.61 and the three VMs distributed to edge servers 2, 3 and 4 gain accuracy of about 0.98 in VM sweeping method. When the link delay is longer than or equal to 40 [ms], VM sweeping method conversely achieves higher accuracy than VM number averaging method. This is explained as follows. As the link delay gets longer, the accuracy gained by the VMs distributed to edge servers 2, 3 and 4 decrease and the accuracy is dominated by those gained by the VMs fixed at the initial edge server. Because VM number averaging method fixes more VMs than VM number averaging method, the former method achieves higher accuracy than the latter.

Our proposed method achieves almost as high accuracy (at most 2% lower accuracy) as the best conventional methods by successfully learning the same policies as 1) VM number averaging method when the link delay is shorter than or equal to 30 [ms], 2) VM sweeping method when the link delay is 40 [ms], and 3) non-migration method and minimum client-VM delay method when the link delay is longer than or equal to 50 [ms], in most trials.

In Figure 14 where the task type is the linear type (HALF time = 55 [ms]), our proposed method also achieves almost as high accuracy (at most 14% lower accuracy) as the best conventional methods by successfully learning the same policies as them in most trials.

In Figure 15 where the task type is the late-blooming type (HALF time = 99 [ms]), VM sweeping method achieves higher accuracy than other conventional methods when the link delay is shorter than or equal to 10 [ms]. This is because only the VM that is individually deployed at an edge server and occupies the CPU time on it can obtain high accuracy for the late-blooming type tasks. In VM sweeping method, each of the three VMs distributed to edge servers 2, 3 and 4 can occupy the CPU time while in other conventional methods, no VM occupies the CPU time. In VM sweeping method, the accuracy decreases as the link delay increases due to longer

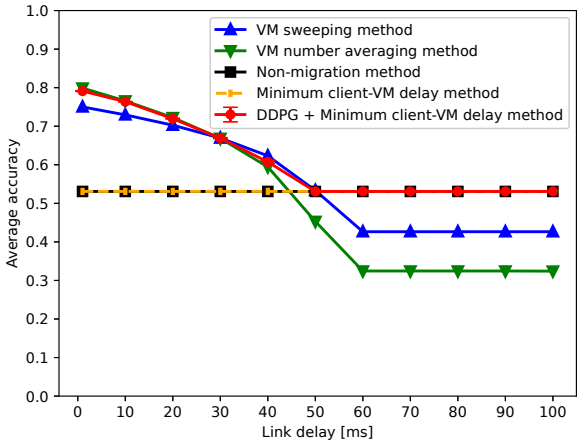

Figure 13. Average accuracy as a function of link delay (HALF time: 11 [ms], Total number of clients: 8).

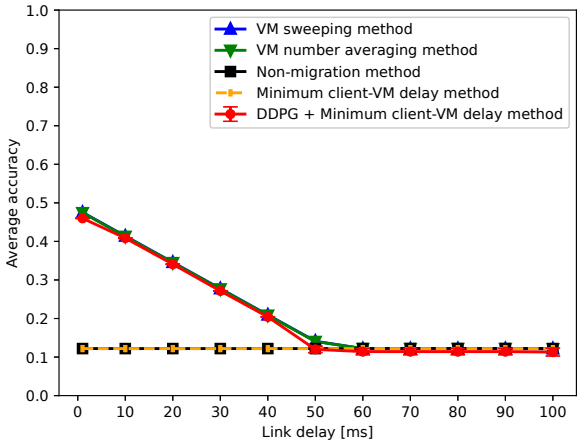

Figure 14. Average accuracy as a function of link delay (HALF time: 55 [ms], Total number of clients: 8).

VM migration time.

When the link delay is 1 [ms], our proposed method achieves almost as high accuracy (about 23% lower accuracy) as VM sweeping method by learning the policies that individually deploy a single VM on three of all the four edge servers in most trials. When the link delay is longer than or equal to 10 [ms], our proposed method achieves higher accuracy of about 0.12 by learning the policies that fix only a single VM at the initial edge server and making it occupy the CPU time of it without the VM migration time while all the conventional methods show the accuracy close to zero.

VI. Conclusions

In this paper, we proposed a VM migration method using a DRL algorithm in order to adaptively achieve high accuracy

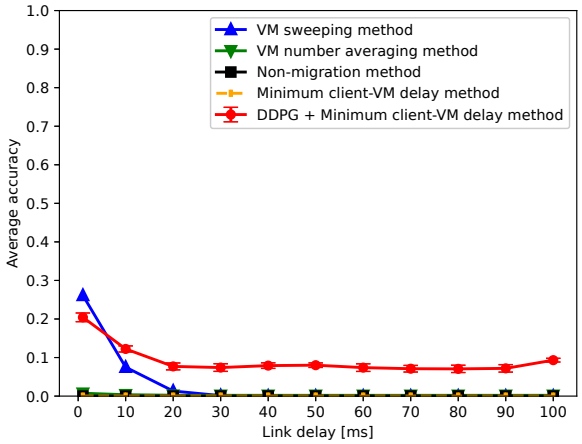

Figure 15. Average accuracy as a function of link delay (HALF time: 99 [ms], Total number of clients: 8).

of information processing tasks in various situations for multi-stage information processing systems. Our proposed method divides the VM migration control problem into two problems: the problem of determining only the VM distribution and the problem of determining the locations of all the VMs so that it follows the determined VM distribution. Our proposed method solves the former problem by a DRL algorithm and the latter problem by the minimum client-VM delay method. In order to evaluate whether our proposed method can adaptively cope with various situations, we performed simulation evaluations with different 1) link delays, 2) types of the tasks and 3) the number of VMs. The simulation results confirm that our proposed method can adaptively achieve quasi-optimal accuracy in those situations.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number JP23K11065 and JP24K02937.

References

[1] Y. Fukushima, Y. Koujitani, K. Nakane, Y. Tarutani, C. Wu, Y. Ji, T. Yokohira, and T. Murase, "Application of a Deep Reinforcement Learning Algorithm to Virtual Machine Migration Control in Multi-Stage Information Processing Systems," in *Proc. of ICN*, May 2024, pp. 13–18.

[2] K. Nakane, T. Anjiki, J. Xie, Y. Fukushima, and T. Murase, "VM Migration Considering Downtime for Accuracy Improvement in Multi-Stage Information Processing System," in *Proc. of IEEE ICCE*, Jan. 2022, pp. 335–336.

[3] T. Anjiki, K. Nakane, and T. Murase, "Performance Improvement by Controlling VM Migration between Edge Nodes in a Multi-Stage Information Processing System," in *Proc. of WSCE*, Sept. 2022, pp. 53–58.

[4] A. Yamanaka, Y. Fukushima, T. Murase, T. Yokohira, and T. Suda, "Destination selection algorithm in a server migration service," in *Proc. of CFI*, Sept. 2012, pp. 15–20.

[5] Y. Fukushima, T. Murase, T. Yokohira, and T. Suda, "Optimization of server locations in server migration service," in *Proc. of ICNS*, March 2013, pp. 200–206.

[6] Y. Hoshino, Y. Fukushima, T. Murase, T. Yokohira, and T. Suda, "An online algorithm to determine the location of the server in a server migration service," in *Proc. of IEEE CCNC*, Jan. 2015, pp. 740–745.
[7] Y. Fukushima, T. Murase, T. Yokohira, and T. Suda, "Power-Aware Server Location Decision in Server Migration Service," in *Proc. of ICTC*, pp. 150–155, Oct. 2016.
[8] Y. Fukushima, T. Murase, G. Motoyoshi, T. Yokohira, and T. Suda, "Determining Server Locations in Server Migration Service to Minimize Monetary Penalty of Dynamic Server Migration," Journal of Network and Systems Management, Vol. 26, Iss. 4, pp. 993–1033, Oct. 2018.
[9] Y. Fukushima, T. Murase, and T. Yokohira, "Link Capacity Provisioning and Server Location Decision in Server Migration Service," in *Proc. of IEEE CloudNet*, pp. 1–3, Oct. 2018.
[10] R. Urimoto, Y. Fukushima, Y. Tarutani, T. Murase, and T. Yokohira, "A Server Migration Method Using Q-Learning with Dimension Reduction in Edge Computing," in *Proc. of ICOIN*, pp. 301–304, Jan. 2021.
[11] Y. Fukushima, T. Suda, T. Murase, Y. Tarutani, and T. Yokohira, "Minimizing the Monetary Penalty and Energy Cost of Server Migration Service," Transactions on Emerging Telecommunications Technologies, Vol. 33, Iss. 9, pp. 1–34, Sept. 2022.
[12] D. Zeng, L. Gu, S. Pan, J. Cai, and S. Guo, "Resource Management at the Network Edge: A Deep Reinforcement Learning Approach," IEEE Network, Vol. 33, Iss. 3, pp. 26–33, May/June 2019.
[13] C. Zhang and Z. Zheng, "Task Migration for Mobile Edge Computing Using Deep Reinforcement Learning," Future Generation Computer Systems, Vol. 96, pp. 111–118, 2019.
[14] M. Volodymyr, et al. "Playing Atari with Deep Reinforcement Learning," arXiv preprint arXiv:1312.5602, 2013.
[15] T. P. Lillicrap, et al. "Continuous Control with Deep Reinforcement Learning," arXiv preprint arXiv:1509.02971, 2015.
[16] G. Brockman, et al. "OpenAI Gym," arXiv preprint arXiv:1606.01540, 2016.
[17] M. Plappert, "Keras-rl," GitHub repository, 2016.