A Multi-Objective Virtual Network Migration Algorithm Based on Reinforcement Learning

Desheng Wang, Weizhe Zhang, *Senior Member, IEEE*, Xiao Han, Junren Lin, and Yu-Chu Tian, *Senior Member, IEEE*

Abstract—Virtual network migration (VNM) helps improve network performance by remapping a subset of virtual nodes or links to physical infrastructure, aligning the resource allocation to the virtual network's changing conditions. However, existing VNM methods neglect integrating multiple objectives that affect network performance, such as energy, communication, migration, and service level agreement violation (SLAV). It is challenging to make VNM decisions to optimize the overall objective in a large-scale cloud environment. This article establishes a multi-objective optimization model and proposes a multi-objective VNM algorithm called MiOvnm. The MiOvnm employs the double deep Q-learning approach to cope with ample state space. It also applies an action selection method called actfilter to deal with large-scale action space. The MiOvnm finds the migration action with optimal potential reward from the candidate action set. Simulation results demonstrate the superiority of our MiOvnm to the state-of-the-art methods. More specifically, MiOvnm reduces average SLAV, communication cost, and total cost by 24.32%, 4.95%, and 12.45%, respectively. Furthermore, evaluation results in a real-world OpenStack platform reveal that making full use of computation and network resources, the MiOvnm reduces the completion time of computation- and network-intensive benchmarks by 11.35% and 10.31%, respectively, with a total cost reduction of 26.02%.

Index Terms—Virtual network, migration, reinforcement learning, OpenStack

1 Introduction

In recent years, cloud computing has experienced vigorous growth due to its capacity to efficiently utilize the information technology infrastructure while providing users with high quality of service [1]. Virtualization technologies, including hardware and lightweight virtualization, allow virtual networks (VNs) to reside on a physical network (PN). This frees the tasks from the user's local limitations. Furthermore, due to VNs' changing conditions, virtualization empowered by migration technology meets the ever-increasing demands of relocating virtual resources within VNs to help achieve various resource management objectives [2].

Virtual network migration (VNM) refers to remapping a subset of virtual nodes or links to physical infrastructure, thereby aligning the resource allocation to current network conditions [3]. Different optimization objectives lead to different migration decisions. An efficient VNM process should minimize both resource consumption and service level agreement violation (SLAV).

A SLAV may occur when virtual nodes are consolidated in over-utilized cloud servers. These nodes perform poorly under computation and network resource shortage [4]. The VNM is able to allocate sufficient resources for the VN to avoid excessive consolidation in some hosts, thereby bringing desired performance to each virtual node.

The optimization of resource consumption can be summarized into the following three aspects:

1) Energy saving: Energy consumption becomes prominent in cloud data centers [5]. Cloud servers consume a massive amount of energy even at an utterly idle state, contributing a significant portion of the overall operational costs in data centers. They possess under-utilization issues in some instances (e.g., when only a few tasks run on the server), resulting in a waste of high-cost resources [6]. Improving energy efficiency, the VNM helps save energy and thus reduces operational costs in data centers.
2) Communication cost reduction: In data-intensive distributed applications, massive data transfer occurs among multiple servers in general, resulting in high traffic load among virtual nodes [7]. The VNM can make virtual nodes to communicate with

- *Desheng Wang, Xiao Han, and Junren Lin are with the School of Cyberspace Science, Harbin Institute of Technology, Harbin, Heilongjiang 150001, China. E-mail: wangdesheng0821@hit.edu.cn, 1170301006@stu.hit.edu.cn, linponys@163.com.*
- *Weizhe Zhang is with the School of Cyberspace Science, Harbin Institute of Technology, Harbin, Heilongjiang 150001, China, and also with Cyberspace Security Research Center, Peng Cheng Laboratory, Shenzhen, Guangdong 518066, China. E-mail: wzzhang@hit.edu.cn.*
- *Yu-Chu Tian is with the School of Computer Science, Queensland University of Technology, Brisbane, QLD 4000, Australia. E-mail: y.tian@qut.edu.au.*

Manuscript received 12 October 2021; revised 23 May 2022; accepted 2 June 2022. Date of publication 9 June 2022; date of current version 7 June 2023.
This work was supported in part by the National Key Research and Development Program of China under Grant 2020YFB1406902, in part by the Key-Area Research and Development Program of Guangdong Province under Grant 2020B0101360001, in part by Shenzhen Science and Technology Research and Development Foundation under Grant JCYJ20190806143418198, in part by the Major Key Project of PCL under Grant PCL2021A02, and in part by the Fundamental Research Funds for the Central Universities under Grant HIT.OCEF.2021007.
(Corresponding author: Weizhe Zhang.)
Recommended for acceptance by Y. Yang.
Digital Object Identifier no. 10.1109/TCC.2022.3180784

closer nodes, which may even reside on the same physical host, with minimal data transfer delay, reducing the communication congestion in the PN.

3) Migration overhead optimization: To achieve the efficiency of the above objectives, sometimes unadvisable migration decisions are made. Numerous unnecessary migration actions cause much overhead. Since migration is a resource-intensive process, less migration reduces unnecessary resource consumption, including CPU and memory resources at the source and target hosts and bandwidth resources on the path from the source to the destination [8].

1.1 Motivations

VNs are initially deployed with multiple techniques, including fast provisioning [9]. However, the previous deployment of VNs may present limited performance due to VNs' changing workloads over time. This study focuses on solving the multi-objective VNM problem caused by VNs' changing workloads, thereby making full use of physical resources and improving the service performance of the cloud environment. The VNM problem is an NP-hard problem, in which each VN usually shows unpredictable variability [3]. There have been numerous efforts in the literature to deal with this problem. However, in a cloud environment with global shared state mechanism [10], dynamic VNM is a fundamental yet challenging problem.

Numerous VNM methods are formulated to optimize partial migration objectives [2], such as minimizing energy or improving the efficiency of virtual nodes in general. However, reducing communication cost is also a significant challenge. It takes much longer to communicate between two virtual nodes located on different hosts than on the same host because the data on the same host is transmitted through random access memory, which is usually faster than PN [11]. Furthermore, the migration process consumes additional physical resources to migrate virtual nodes, implying a migration cost. Recently, VNM research based on multiple objectives has also been rapidly developed [12]. However, due to multi-objective VNM modeling and method design complexities, these existing methods show a lack of generalizability and adaptability in tackling the VNM problem of optimizing energy, migration, communication, and SLAV, thus failing to build such a multi-objective VNM model. Therefore, the first challenge is **CH1:** *How to build a multi-objective optimization model that considers energy, communication, migration, and SLAV?*

Additionally, most heuristic and multi-objective VNM approaches obtain an immediate migration decision based on current system observation or future system prediction. However, these migration decisions can be sub-optimal due to the NP-hard complexity of VNM. Furthermore, once we have selected a policy and followed its decisions, it is still not clear how good the decisions will be. Therefore, the second challenge is **CH2:** *How to obtain migration decisions to optimize the overall VNM objective?*

Efforts have also been made in the research based on the Markov decision process (MDP) [3], which is proposed to tackle the VN reconfiguration problem. However, a large-scale VNM framework possesses high-dimensional state and action spaces. Conventional ways to solve large-scale MDP issues suffer from dimensionality [13]. Specifically, it is practically infeasible to traverse all migration decisions with an exhaustive search in high-dimensional action space and then choose the best. Furthermore, most VNM approaches also suffer from VN and PN scales. Therefore, the third challenge is **CH3:** *How to deal with high-dimensional state and action spaces in a VNM framework?*

1.2 Contribution

To address the above challenges, we have made in-depth work in this paper. Targeting **CH1**, we analyze the key factors existing in the VNM process, including energy, migration, communication, and SLAV. Furthermore, a VNM approach requires interaction with the cloud environment, where each migration decision is selected according to current state and is executed to transition to the next state during the interaction process. Therefore, the VNM problem obeys MDP. We finally build an MDP model for the multi-objective VNM problem.

Targeting **CH2**, we apply reinforcement learning to learn an optimization agent to make VNM decisions and adapt its decisions to align the different system states. The learned agent can maximize the potential reward for each state, i.e., the overall optimization of multi-objective VNM. Furthermore, the state and action spaces are discrete and high-dimensional in a large-scale VNM framework. A value-function-based deep reinforcement learning approach naturally satisfies our research problem. Therefore, we employ the recently proposed double deep Q-learning (DDQN) method [14] to train the model by interacting with ample state space in a cloud VNM environment. However, the DDQN method suffers from high-dimensional action space.

Targeting **CH3**, we propose a candidate action selection method for the DDQN agent. It selects candidate migration actions that may optimize one or more objectives. And then the learning agent selects the migration action with the most significant potential reward from the candidate action set.

In summary, we build an MDP model for the multi-objective VNM problem, employ the DDQN approach to adapt to the different system states, and propose a candidate action selection method to improve the model's efficiency. Our proposed algorithm focuses on finding the migration action with the maximized potential reward from the candidate action set. The main contributions of this paper are summarized as follows:

1) We build a novel multi-objective MDP-based VNM model regarding energy, migration, communication, and SLAV. Optimizing four objectives simultaneously for VNM is innovative.
2) We propose a multi-objective VNM algorithm called MiOvnm based on DDQN to cope with ample state space in a cloud VNM environment. Our method provides dynamic migration solutions to optimize the overall VNM objective.
3) We also propose an action selection algorithm called actfilter for the MiOvnm to deal with large-scale action space, thereby improving the efficiency of model operation.

We have conducted simulation studies and real-world OpenStack-based experiments to verify our algorithm. The simulation results demonstrate that our proposed MiOvnm is much more efficient than the state-of-the-art methods, reducing average SLAV, communication, and total costs by 24.32%, 4.95%, and 12.45%, respectively. Furthermore, real-world OpenStack evaluation demonstrates that the MiOvnm makes full use of computation and network resources and reduces the completion time of computation- and network-intensive benchmarks by 11.35% and 10.31%, respectively, with a 26.02% total cost reduction.

The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 describes the network model and problem formulation of the study. Our VNM algorithm based on reinforcement learning is presented in Section 4. This is followed by experimental studies in Section 5. Finally, Section 6 concludes the study.

2 Related Work

The VNM research aims to make VNM decisions with consideration of multiple objectives. In early research of this problem, the migration mechanism of the cloud data center was a strategy obtained based on the virtual machine (VM) deployment and VM merging during resource management. The migration research focused on when to trigger the migration mechanism and how to achieve efficient migration [2]. Xiao *et al.* [15] introduced the concept of skewness to measure the unevenness in multi-dimensional resource utilization. By minimizing skewness, a set of heuristics was proposed to prevent overload while saving energy effectively. Kansal *et al.* [16] proposed a real-time migration strategy based on the Firefly algorithm, which avoids wasting CPU and memory resources during the migration process. A binary graph matching-based bucket-code learning algorithm was designed to solve the dynamic migration of VMs [17]. In this algorithm, code learning and mutation helped improve the migration performance. However, these works actually possessed the limited performance in optimizing the overall VNM objectives.

In recent years, applications on cloud computing platforms have emerged endlessly. Therefore, the research of VM real-time migration algorithms based on diverse objectives has also been developed rapidly with the consideration of energy consumption, migration overhead, communication overhead, and SLAV optimizations [12].

With regard to energy consumption-related optimization, Hieu *et al.* [18] proposed a VM migration strategy based on the forecast of future resource utilization to reduce load and power consumption. Al *et al.* [19] provided a method to select VMs for migration and place the VMs based on the multiple choice and knapsack, improving energy efficiency and limiting the migration frequency. Han *et al.* [13] formulated the dynamic VM management as an MDP problem. They further exploited the particular structure of the problem and proposed an approximate MDP-based dynamic VM management method named MadVM to minimize power consumption. Sha *et al.* [20] presented a collective behavior-based meta-heuristics algorithm to reduce energy consumption via discrete bacterial foraging. Zhang *et al.* [5] developed a heuristic energy-aware VN migration algorithm called EA-VNM and presented an enhanced group-based VN migration algorithm to reduce time complexity. However, the competitiveness matrix of the EA-VNM took a significant overhead in large-scale VNs. Karthikeyan *et al.* [21] reported a naive Bayes classifier with hybrid optimization using an artificial bee colony-bat algorithm for migration to reduce energy consumption. Garg *et al.* [22] proposed a load-aware three-gear threshold algorithm and a modified best fit decreasing algorithm to minimize total energy consumption while reducing SLAV under dynamic VNs. Khaleel *et al.* [23] developed a VM allocation algorithm to achieve a tradeoff between energy and SLAV. The developed method over-utilized and under-utilized hosts, selected VMs for migration based on the modified best-fit decreasing algorithm, and turned off selected under-utilized hosts.

For communication cost-related optimization, Zhang *et al.* [24] proposed a genetic algorithm and artificial bee colony to make a tradeoff between communication and migration costs. Xue *et al.* [25] presented a communication-aware VM migration algorithm with the consideration of CPU, memory, and bandwidth constraints to minimize inter-VM traffic and migration costs. Cui *et al.* [26] investigated a progressive-decompose-rounding approach for VM migration in polynomial time to cut communication and migration costs. Gao *et al.* [3] developed a fully polynomial-time approximation scheme to minimize physical link utilization, physical node workload, and the migration number. The scheme selects the virtual nodes to be migrated and takes a random walk on a Markov chain to select new physical nodes for the migrated virtual nodes. However, a pre-setting geographical constraint of each virtual node simplified the problem with a limited size of target physical nodes. Flores *et al.* [27] designed heuristic policy-aware VM migration algorithms to minimize policy-aware data centers' communication and migration costs.

With regard to SLAV-related optimization, Saxena *et al.* [28] selected the largest resource capacity VM from the overloaded server to avoid SLAV caused by overload situations. The selected VMs were migrated to closer hosts to reduce inter-VMs' traffic and energy costs. Li *et al.* [29] formalized VN function migration and service-function-chain-reconfiguration problems and proposed an improved hybrid genetic evolution algorithm to minimize service delay and guarantee network load balancing. They also developed a multi-stage heuristic to reduce the computation overhead in large-scale networks.

In the field of migration overhead-related optimization, Wang *et al.* [8] proposed a fully polynomial-time approximation scheme to reduce the total migration time by maximizing effective transmission rate in the network, allowing multiple VMs to be migrated simultaneously via multiple routing paths. Nashaat *et al.* [30] designed two cooperative algorithms, SESA and AWFDVP, to arrange and migrate VMs based on the system evaluation, reducing the amount of memory migrated, SLAV, and power consumption.

To present the contribution of this paper more clearly, we classify the methods developed in the related work in terms of the four optimization objectives, as shown in Table 1. Most existing VNM methods traverse network information to perform complex and time-consuming calculations,

TABLE 1
Classification of the Relate Work

Ref.	Optimization goals	Approach
[15]	(1)(3)	Skewness minimization
[16]	(1)(3)	Firefly-based optimization
[17]	(1)(2)(4)	Bucket-code learning
[18]	(1)(3)(4)	Prediction-based consolidation
[19]	(1)(4)	Multiple Choice Knapsack
[13]	(1)(3)(4)	MDP-based optimization
[20]	(1)	Behavior-based metaheuristics
[5]	(1)	maximum weight matching
[21]	(1)	Artificial bee colony–bat
[22]	(1)(3)	Three-gear threshold and modified best fit decreasing
[23]	(1)(3)	Multi-step VM allocation
[24]	(2)(4)	Genetic algorithm and artificial bee colony
[25]	(2)(4)	Heuristic optimization for three problems regarding when, which, where
[26]	(2)(4)	Progressive-decompose-rounding approach
[3]	(2)(4)	Linear programming-based fully polynomial-time approximation scheme and random walk on a Markov chain
[27]	(2)(4)	Heuristic policy-aware VM migration
[28]	(1)(2)(3)	Online prediction and multi-objective VM migration
[29]	(2)(3)	Hybrid genetic evolution approach
[8]	(4)	Linear approximation plus fully polynomial time approximation
[30]	(1)(3)(4)	Adaptive worst fit decreasing VM placement

$*$ *Optimization goals column: (1) energy, (2) communication, (3) SLAV, (4) migration.*

suffering from VN and PN scales. Furthermore, the existing VNM methods deal with partial migration objectives and show limited performance in optimizing the overall VNM objective. Although the sampling-based estimation mechanism [31], [32], [33] can efficiently tackle the scalability problem for most existing VNM methods, it is still challenging to make migration decisions to optimize the overall VNM objective in a large-scale cloud environment. This study takes advantage of deep reinforcement learning and action filter to interact with complex environments efficiently and optimizes the overall VNM objective with the consideration of energy, communication, SLAV, and migration costs.

3 Network Model and Problem Formulation

This section formalizes the networks, the VNM problem, and VNM objectives. Key symbols used in this paper are tabulated in Table 2.

3.1 Migration System

The VNM process mainly migrates part of the virtual nodes to optimize energy, migration, communication, and SLAV. As shown in Fig. 1, a cloud migration system consists of a Resource Monitor Module (RMM), a Migration Decision Module (MDM), and a Migration Control Module (MCM). The RMM monitors system information through sensors in each server, including PN and VN information. The MDM periodically makes migration decisions based on the system information from the RMM. The VNM algorithm proposed in this study is the core of MDM. The MCM finally executes the migration process via effectors in each server.

Migration requirement/objective and system information affect the decisions of the VNM algorithm. As long as the MDM periodically retrieves system information and

TABLE 2
VNM Model Notations

Notation	Description
G^s	Physical network s
N^s	Set of physical nodes
L^s	Set of physical links
$n_i^s.v(t)$	Set of virtual nodes in physical node n_i^s at time t
$n_i^s.tc(t)$	Total CPU of physical node n_i^s at time t
$n_i^s.uc(t)$	Utilized CPU of physical node n_i^s at time t
$n_i^s.rc(t)$	Remaining CPU of physical node n_i^s at time t
$l_{i,j}^s.tb(t)$	Total bandwidth of physical link $l_{i,j}^s$ at time t
$l_{i,j}^s.ub(t)$	Utilized bandwidth of physical link $l_{i,j}^s$ at time t
$l_{i,j}^s.rb(t)$	Remaining bandwidth of physical link $l_{i,j}^s$ at time t
$l_{i,j}^s.l(t)$	Set of virtual links spanning over $l_{i,j}^s$ at time t
P^s	Set of all loop-free physical paths
$p_{i,j}^s.tb(t)$	Total bandwidth of physical path $p_{i,j}^s$ at time t
$p_{i,j}^s.ub(t)$	Utilized bandwidth of physical path $p_{i,j}^s$ at time t
G^v	Virtual network v
N^v	Set of virtual nodes
L^v	Set of virtual links
$n_i^v.loc(t)$	Physical node where virtual node n_i^v is located at time t
$n_i^v.m(t)$	Required memory of virtual nodes n_i^v at time t
$n_i^v.c(t)$	Required CPU of virtual nodes n_i^v at time t
$l_{i,j}^v.b(t)$	Required bandwidth of virtual link $l_{i,j}^v$ at time t
$l_{i,j}^v.p(t)$	Path length between $n_i^v.loc(t)$ and $n_i^v.loc(t)$ at time t
C_e	Energy cost
C_c	Communication cost
C_s	SLAV
$c_s^{cpu}(t)$	Immediate computation SLAV at time t
$c_s^{bd}(t)$	Immediate network SLAV at time t
C_m	Migration cost
$M(t)$	Set of virtual nodes to be migrated at time t
C	Total cost

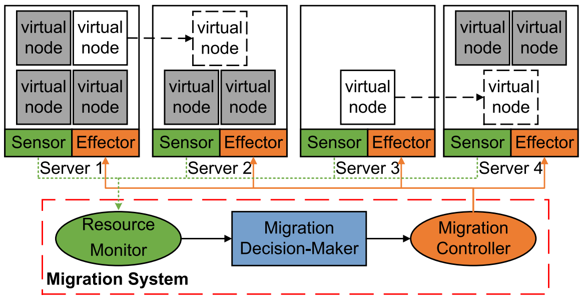


Fig. 1. Cloud migration system.

migration requirements from RMM and the cloud platform, our proposed VNM algorithm can easily make migration decisions for MCM to migrate virtual nodes, optimizing multiple objectives for a cloud platform. A migration example is presented in Fig. 1. Servers 1 and 2 are over-utilized and under-utilized servers, respectively. Some virtual nodes on server 1 are migrated to server 3. All virtual nodes on server 2 are migrated to server 4. Finally, there are no over-utilized or under-utilized servers in the entire system. In this case, we can reduce the energy consumption of server 2 and improve VM running performance in server 1.

3.2 Physical Network

The PN topology is defined as an undirected weighted graph $G^s = (N^s, L^s)$, where N^s and L^s represent the sets of physical servers and links, respectively. In this paper, superscripts s and v correspond to PN and VN, respectively. $|N^s|$ is the total number of physical nodes. Each physical node n^s_i $(i \in [0, |N^s| - 1])$ runs multiple virtual nodes. Define $n^s_i.v(t)$ as the set of all virtual nodes on n^s_i at time t. Current cloud frameworks, such as OpenStack and Google Cloud, limit the hosting capacity of each host by setting the maximum number of instances. We define $n^s_i.n$ as the maximum number of virtual nodes on n^s_i. The sets $n^s_i.tc(t), n^s_i.uc(t)$, and $n^s_i.rc(t)$ respectively represent the total, utilized, and remaining CPU resources of n^s_i at time t. To simplify the model, we focus on CPU usage in this paper and consider RAM and disk as sufficient resources.

If there is a link between any two different physical nodes, n^s_i and n^s_j, we define this link as $l^s_{i,j} \in L^s$. $|L^s|$ is the total number of physical links. Similarly, $l^s_{i,j}.tb(t)$, $l^s_{i,j}.ub(t)$, and $l^s_{i,j}.rb(t)$ represent the total, utilized, and the remaining bandwidth resource of $l^s_{i,j}$ at time t. $l^s_{i,j}.l(t)$ represents the set of virtual links spanning over $l^s_{i,j}$ at time t. Finally, we define variables related to the physical path. P^s is defined as the set of loop-free paths in a PN. A physical path $p^s_{i,j} \in P^s$ means the shortest path between n^s_i and n^s_j, consisting of a single or a sequence of physical links. The total bandwidth of the path $p^s_{i,j}$ at time t, denoted $p^s_{i,j}.tb(t)$, depends on the physical link with minimal total bandwidth capacity. The utilized bandwidth of the path $p^s_{i,j}$ at time t is $p^s_{i,j}.ub(t)$.

In this study, spanning is defined as the process of a virtual link communicating through a physical link. For example, Fig. 2 indicates that the virtual nodes a and b are located on physical nodes A and E, respectively. When the virtual nodes a and b communicate via path $\{A, F, E\}$, we consider that the virtual link between a and b spans over physical links $\{A, F\}$ and $\{F, E\}$. Furthermore, the shortest

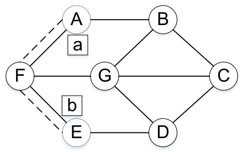


Fig. 2. An example of communication process.

path is defined as the communication path with the shortest length. For example (Fig. 2), when virtual nodes a and b communicate, the shortest path is $\{A, F, E\}$. The loop-free path is defined as the communication path with no cycles. For example (Fig. 2), path $\{A, F, E\}$ is a loop-free path, while path $\{A, F, G, B, A, F, E\}$, which contains a cycle $\{A, F, G, B, A\}$, is not a loop-free path.

3.3 Virtual Network

Similar to the PN topology, the VN topology is also defined as an undirected weighted graph $G^v = (N^v, L^v)$, where N^v and L^v are the sets of virtual nodes and links. $|N^v|$ represents the total number of virtual nodes. At time t, $n^v_i.loc(t)$ represents the physical node where the virtual node n^v_i $(i \in [0, |N^v| - 1])$ is located. Specifically, $n^v_i.loc(t) = n^s_j$ means n^v_i is located on n^s_j at time t. The amount of CPU resource required by n^v_i is denoted as $n^v_i.c(t)$. Additionally, $n^v_i.m(t)$ is defined as the amount of memory resource required by n^v_i at time t, related to migration cost (introduced in Section 3.4). Virtual link $l^v_{i,j} \in L^v$ represents the connection between two nodes, n^v_i and n^v_j, and $|L^v|$ represents the total number of virtual links. The $l^v_{i,j}.b(t)$ represents the required bandwidth of $l^v_{i,j}$. In addition, we denote $l^v_{i,j}.p(t)$ as the path length between $n^v_i.loc(t)$ and $n^v_j.loc(t)$.

3.4 Problem Formulation of Virtual Network Migration

As a necessary configuration in a PN, the network communication protocol finds a communication path for each virtual link. For simplicity, it is set to take the shortest path policy in this paper. Each virtual link is only mapped to one physical path. To optimize the communication cost, our algorithm needs to attain the communication path of virtual links. The shortest path policy relates to, and provides the basis for the calculation of, communication optimization.

The VNM in this study aims to change the original assignment of VNs in the PN to save energy and communication costs without performance degradation while incurring a low migration cost. More specifically, the dynamic VNM is described as a multi-objective optimization problem with the consideration of energy consumption, migration cost, SLAV, and communication cost.

3.4.1 *Energy Cost*

In existing research [4], [13], energy cost, denoted by C_e, is an integration of the total costs from all individual running nodes over time (Equation (1)):

$$C_e = \int \left(\sum_{i=1}^{|N^s|} e_i(t) \right) \mathrm{d}t, \tag{1}$$

where $e_i(t)$ represents the energy cost at time t when the physical node n_i^s is switched on. It combines idle state energy cost and computation energy cost (Equation (2)):

$$e_i(t) = P_{idle}(i) + (P_{max}(i) - P_{idle}(i)) \cdot n_i^s.uc(t), \tag{2}$$

where the $P_{idle}(i)$ and $P_{max}(i)$ represent the power consumption of 0% and 100% CPU utilization of n_i^s, respectively. A large amount of energy consumption results from the physical hosts with very low utilization. However, the energy consumption is nearly 0 when n_i^s is switched off. Therefore, switching off as many physical nodes as possible is an effective way to save energy.

3.4.2 Communication Cost

As presented in existing research [17], communication cost, denoted by C_c, results from network communication between virtual nodes. When two virtual nodes are on the same physical machine, their communication process is accessed through random memory, of which the communication cost is considered as 0. When two virtual nodes reside on different physical machines, their communication consumes bandwidth in each physical path. Thus, the communication cost is formalized as the total utilized bandwidth of all physical links, as shown in Equation (3):

$$C_c = \int \left(\sum_{i=1}^{|N^s|} \sum_{j=1}^{|N^s|} l_{i,j}^s.ub(t) \right) \mathrm{d}t. \tag{3}$$

The length of the communication path of each virtual link influences the communication cost significantly. Therefore, making the virtual link communication path as close as possible or even communicating through random access memory on the same host will reduce communication costs.

3.4.3 Service Level Agreement Violation

As presented in [4], SLAV denoted by C_s characterizes the total CPU and bandwidth shortages of the VN. Insufficient computation and network resources may be provided to virtual nodes and links, resulting in poor working performance. The SLAV is formalized as follows:

$$C_s = \int \left(c_s^{cpu}(t) + c_s^{bd}(t) \right) \mathrm{d}t, \tag{4}$$

where $c_s^{cpu}(t)$ and $c_s^{bd}(t)$ are the immediate SLAVs caused by the shortages of CPU and bandwidth. For each physical node n_i^s, the difference between the required CPU of virtual nodes on n_i^s and the utilized CPU of n_i^s is the CPU shortage of n_i^s. The $c_s^{cpu}(t)$ is formalized as the total CPU shortage of all physical nodes, as shown in Equation (5). Similarly, the $c_s^{bd}(t)$ is formalized as the total bandwidth shortage of all physical links, as shown in Equation (6).

$$c_s^{cpu}(t) = \sum_{i=1}^{|N^s|} \left(\sum_{n^v \in n_i^s.v(t)} (n^v.c(t)) - n_i^s.uc(t) \right), \tag{5}$$

$$c_s^{bd}(t) = \sum_{i=1}^{|N^s|} \sum_{j=1}^{|N^s|} \left(\sum_{l^v \in l_{i,j}^s.l(t)} (l^v.b(t)) - l_{i,j}^s.ub(t) \right). \tag{6}$$

It is worth mentioning that when the CPU (bandwidth) resources of each virtual node (link) are insufficient, the CPU (bandwidth) resource utilization of the PN becomes smaller, as well as the energy (communication) cost. Although a proper resource allocation causes energy and communication consumption, it may effectively reduce the SLAV because the resource shortage problem can be fixed.

3.4.4 Migration Cost

As presented in existing research [17], [34], migration cost denoted by C_m mainly refers to the transfer data size during the migration of the virtual nodes. Live migration technologies, such as pre-copy, post-copy, and hybrid-copy, allow VMs to be migrated without loss of computation and storage tasks, with a short switching time closely related to memory utilization [12], [35]. They require to synchronize the memory and storage data of the virtual nodes being migrated. As the storage devices in the same cloud data center are shared, storage migration is not required [1]. We assume all PMs share backend storage in this study. Thus, the migration cost is formalized as the total amount of memory data to be migrated, as shown below:

$$C_m = \int \left(\sum_{n^v \in M(t)} n^v.m(t) \right) \mathrm{d}t, \tag{7}$$

where $M(t)$ represents the set of virtual nodes to be migrated at time t. It is seen from the migration cost definition that migration cost can be minimized by reducing the migration size and times.

3.4.5 Problem Definition

Our dynamic VNM problem is defined as minimizing multi-objective costs by migrating virtual nodes. The objective function of the VNM problem is formulated to minimize the total cost C, which is expressed as a weighted sum of the four individual costs discussed above:

$$\min C = \alpha \cdot C_e + \beta \cdot C_c + \theta \cdot C_s + \zeta \cdot C_m, \tag{8}$$

where α, β, θ, and ζ are importance coefficients, which are initialized from the environmental parameters.

The four objectives mentioned above can never reach their optimal values simultaneously because they are conflicting in nature. If we try to aggregate hosts as much as possible to reduce energy consumption, the communication delay and SLAV will increase. If we aim to minimize the communication overhead, we may make the virtual nodes with the communication traffic reside on the same physical host as much as possible. But this will often lead to resource shortage of the physical host, thereby increasing the SLAV. Moreover, the importance of each objective is different in different cloud computing platforms. Therefore, one can set the weight parameters of these four objectives. For example, considering the PN is likely to cause communication congestion, it is better to apply a high weight of communication cost and set a high β value.

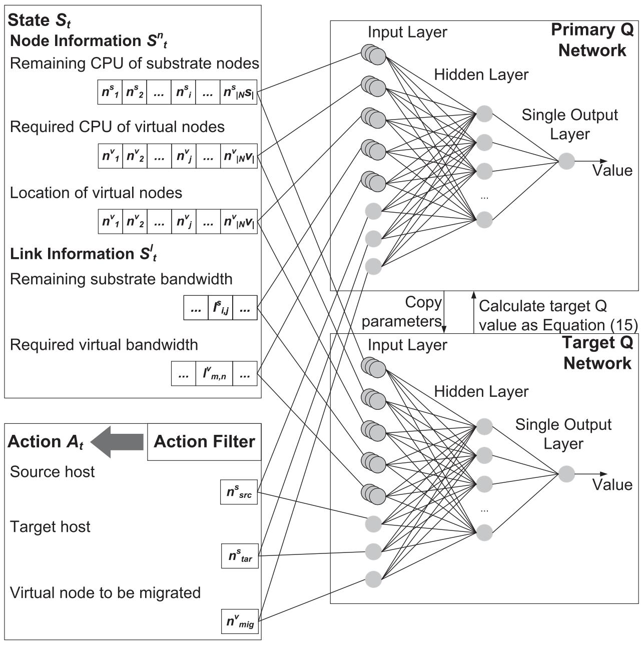

State S_t
Node Information S^n_t
Remaining CPU of substrate nodes
Required CPU of virtual nodes
Location of virtual nodes
Link Information S^l_t
Remaining substrate bandwidth
Required virtual bandwidth
Action A_t
Action Filter
Source host
Target host
Virtual node to be migrated
Input Layer
Hidden Layer
Single Output Layer
Value
Primary Q Network
Target Q Network
Copy parameters
Calculate target Q value as Equation (15)

Fig. 3. Learning scheme of the proposed algorithm.

4 Multi-Objective Virtual Network Migration Algorithm

In order to cope with the complex VNM in a large-scale cloud environment, we propose a VNM algorithm based on DDQN, named MiOvnm. The architecture of the MiOvnm learning algorithm is shown in Fig. 3. To deal with ample state space and avoid excessively optimistic value estimation, the algorithm uses double Q networks in the DDQN model to perform action selection and target Q value calculation, separating the selection and evaluation of migration actions. In response to the inefficiency caused by the considerable action space in a large-scale environment, we propose a heuristic action selection algorithm named actfilter to filter and reduce the action space. Our proposed algorithm finds the migration action with the maximized potential reward from the candidate action set. Since the reinforcement learning algorithm satisfies the contraction mapping principle, the migration decision can be replicated to get the optimal solution from the candidate action set.

4.1 Markov Decision Process Model

In the MDP model, the agent chooses to perform an action A_t with the maximum potential reward according to the current state S_t, obtains an immediate reward R_t after performing the action, and enters a new state S_{t+1} in the next time. Therefore, we define VNM as a quadruple $< S_t, A_t, S_{t+1}, R_t >$.

The VNM environment information is complex, including workload, communication relationships, and distribution. Especially in large-scale networks, it is difficult to use all these information for model calculation. We define the state with node information S^n_t and link information S^l_t. S^n_t includes the remaining CPU resources of each physical node, required CPU resources, and the location of each virtual node. S^l_t includes the remaining bandwidth of each physical link, and the required bandwidth of each virtual link. Therefore, the state at time t is represented as a one-dimensional vector:

$$S_t = \left[S^n_t, S^l_t\right], \tag{9}$$

where S^n_t and S^l_t are defined as follows:

$$\begin{cases} S^n_t = \left[n^s_i.rc(t), n^v_j.c(t), n^v_j.loc(t) | \forall n^s_i \in N^s, \forall n^v_j \in N^v\right], \\ S^l_t = \left[l^s_{i,j}.rb(t), l^v_{m,n}.b(t) | \forall l^s_{i,j} \in L^s, \forall l^v_{m,n} \in L^v\right]. \end{cases} \tag{10}$$

In the VNM problem, when the number of virtual nodes to be migrated is not limited, there are $|N^s|^{|N^v|}$ discrete actions for selection each time. When both $|N^s|$ and $|N^v|$ are large, it is almost practically impossible to make migration decisions in such a considerable action space. In order to reduce the action space, instead of no limit on the number of virtual nodes to be migrated, we choose to migrate one virtual node as an action. Then, the action space will become $|N^s| \cdot |N^v|$. In this way, we can significantly reduce the dimensionality of the action space to speed up training. The information in action includes the source host, the target host, the virtual node to be migrated, as expressed below:

$$A_t = \left[n^s_{src}, n^s_{tar}, n^v_{mig} | n^s_{src}, n^s_{tar} \in N^s, n^v_{mig} \in N^v\right]. \tag{11}$$

The goal of reinforcement learning is to get the most significant reward for the agent after performing the action. Our research problem is to get the least total cost after the agent performs the migration action. Therefore, the immediate reward function can be set to the negative sum of the immediate cost so that the maximum cumulative reward represents the minimal total system costs, as shown below:

$$R_t = -(\alpha \cdot c_e(t) + \beta \cdot c_c(t) + \theta \cdot c_s(t) + \zeta \cdot c_m(t)), \tag{12}$$

where the imdmediate cost of energy c_e, communication c_c, SLAV c_s, and migration c_m are defined as follows:

$$\begin{cases} c_e(t) = \sum_{i=1}^{|N^s|} e_i(t), c_c(t) = \sum_{i=1}^{|N^s|} \sum_{j=1}^{|N^s|} l^s_{i,j}.ub(t), \\ c_s(t) = c^{cpu}_s(t) + c^{bd}_s(t), c_m(t) = \sum_{n^v \in M(t)} n^v.m(t). \end{cases} \tag{13}$$

4.2 DDQN-Based Virtual Network Migration Algorithm

This section proposes the MiOvnm algorithm to achieve the multi-objective VNM method based on the DDQN model [14]. Our MiOvnm algorithm contains two deep neural networks (DNNs), the primary Q network and the target Q network. The former generates the Q value for each action, and the latter generates the Q value used to train the primary Q network. The specific definition is as follows:

1) The primary Q network, which takes the current state s_t and the action a_t as input, can obtain potential reward (i.e., Q value) for each action. We define the primary Q network as $Q_\omega(S, A)$.
2) The target Q network, which is used to train the primary Q network, periodically copies model parameters from the primary Q network. We define the target Q network as $Q'_{\omega'}(S, A)$, from which we

calculate the potential rewards of a_{t+1} and s_{t+1}, namely $Q'_{\omega'}(s_{t+1}, a_{t+1})$.

We usually use the experience replay method to train the primary Q network. We store the quadruple $< s_t, a_t, s_{t+1}, r_t >$ generated during the training process in replay memory and then randomly extract empirical data to train the network model parameters. We define the loss function of the primary Q network with the mean square error function ((14)).

$$J(\omega) = E\Big[\big(y_t - Q_\omega(s_t, a_t)\big)^2\Big], \tag{14}$$

where $Q_\omega(s_t, a_t)$ is the potential reward of Q network ((15)). The y_t is the target Q value ((16)), where $\gamma \in [0, 1]$ is the discount rate.

$$Q_\omega(s_t, a_t) = E[r_t + \gamma \cdot Q_\omega(s_{t+1}, a_{t+1})], \tag{15}$$

$$y_t = r_t + \gamma \cdot Q'_{\omega'}(s_{t+1}, \arg\max_a Q_\omega(s_{t+1}, a)). \tag{16}$$

With a random mini-batch experience from replay memory $< s_t, a_t, s_{t+1}, r_t >$, $(t \in \{1, \ldots, X\})$, the parameter ω is updated through gradient as Equation (17). The parameter ω of the primary Q network is copied periodically to target Q network ω'.

$$\bigtriangledown_\omega J(\omega) = E[2(y_t - Q_\omega(s_t, a_t)) \bigtriangledown_\omega Q_\omega(s_t, a_t)]. \tag{17}$$

At each specific training step, ω is updated as follows:

$$\omega = \omega - \frac{\alpha_Q}{X} \cdot \sum_{t=1}^{X} [2(y_t - Q_\omega(s_t, a_t)) \bigtriangledown_\omega Q_\omega(s_t, a_t)], \tag{18}$$

where α_Q is the learning rate.

We have designed the MiOvnm algorithm to determine migration actions. Due to the vast and discrete state and action space in large-scale networks, it is practically infeasible to traverse all actions to make the best decision. So we design a method for selecting candidate actions in the next section. In this way, we can significantly reduce the action space and improve the model efficiency. The training process of the MiOvnm algorithm is shown in algorithm 1.

Algorithm 1. MiOvnm Training Procedure

```
1:  Initialize the primary $Q$ network $Q_\omega(S, A)$ and target $Q$ network $Q'_{\omega'}(S, A)$;
2:  $Alist \leftarrow \varnothing$, create replay;
3:  for each episode do
4:    for each $i \in [1, K]$ do
5:      Initialize the environment and obtain state information;
6:      $Alist \leftarrow actfilter(G^s, G^v)$   //Algorithm 3;
7:      $a_t \leftarrow \arg\max Q_\omega(s_t, a \in Alist)$, execute action $a_t$;
8:      Calculate immediate reward $r_t$ ( (12)), and observe the next state$s_{t+1}$;
9:        Store the quadruple $< s_t, a_t, s_{t+1}, r_t >$ in replay memory;
10:     Sample mini-batch from the replay memory;
11:     Update primary $Q$ network based on Equations (17);
12:     Update target $Q$ network by copying from primary $Q$ network periodically;
```

In each episode, we first obtain the current resource state information (line 5). For the current state, we obtain a set of candidate actions $Alist$ (line 6) according to the actfilter algorithm proposed in the next section and select the action a_t with the most significant Q value from $Alist$ (line 7). We perform a_t, calculate the immediate reward r_t, observe the next state s_{t+1}, and save the obtained experience quadruple $< s_t, a_t, s_{t+1}, r_t >$ to replay memory (lines 8-9). After that, by randomly sampling the previous experience in the replay memory, we train the primary Q network parameters ω (lines 10-11). Finally, we periodically update the target Q network (line 12).

The training process also faces the exploration-exploitation dilemma. Since the ϵ-greedy strategy can solve this problem effectively, the ϵ-greedy strategy is used to solve the exploration-discovery dilemma in the model training phase (line 7). This step is omitted in our algorithm. We can select an action randomly from candidate actions with the probability of ϵ and select the action with the most significant Q value of $1 - \epsilon$.

After the MiOvnm model training is completed, it is used to decides the migration actions (Algorithm 2). We first obtain the current state information and calculate the immediate reward r_t (lines 3-4). Then, we obtain candidate action set $Alist$ (line 5) according to the actfilter algorithm for the current state and select the action a_t with the most significant Q value from $Alist$ (line 6). If r_t increases, we consider a_t an executable action and complete the migration action (lines 8). Otherwise, the model terminates the decision-making process.

Algorithm 2. MiOvnm

```
1:  $Alist \leftarrow \varnothing$, $R_{\max} \leftarrow -Inf$;
2:  while True do
3:    Obtain current state information $s_t$;
4:    Calculate immediate reward $r_t$ ( (12));
5:    $Alist \leftarrow actfilter(G^s, G^v)$   //Algorithm 3;
6:    $a_t \leftarrow \arg\max Q_\omega(s_t, a \in Alist)$;
7:    if $r_t > R_{\max}$ then
8:      Execute action $a_t$;
9:      $R_{\max} \leftarrow r_t$;
10:   else
11:     break;
```

4.3 Action Filter Algorithm

In a large-scale cloud environment, even if we have adopted the scheme to migrate a single virtual node as an action, the discrete action space is still huge (i.e., $|N^v| \cdot |N^s|$), and the training speed is languid. We propose a heuristic action selection algorithm to obtain a candidate action set. Under each system state, numerous actions are meaningless. For example, if the source physical node is the target physical node or the virtual node migrates from a normal physical node to an overutilized one, the ideal agent should abandon these evil actions. Therefore, we have designed a heuristic algorithm named actfilter for the agent to filter the original action space. This will reduce the action space and consequently speed up the training process.

The goal of actfilter (Algorithm 3) is to select a set of actions that may reduce one or more types of costs and then use the neural network to select the action with the most significant cost reduction among these actions. As mentioned in Equation (8), the four costs are conflicting in nature. So we cannot find actions to minimize all of them simultaneously. The actfilter starts from optimizing a single objective and selects a series of actions that will not excessively drag down other costs.

Algorithm 3. Actfilter

```
Input: $G^s$, $G^v$;
Output: candidate action set $Alist$;
1:  $Alist, A_e, A_c, A^s_{cpu}, A^s_{bd} \leftarrow \varnothing$;
2:  Get the physical node $n^s$ with the lowest CPU utilization;
3:  Get the virtual node $n^v$ on $n^s$ with the smallest $\frac{n^v.m(t)}{n^v.c(t)}$ value;
4:  for each $n^s_i \in N^s$ do
5:      if $|n^s_i.v(t)| < n^s_i.n$ then
6:          add $< n^s, n^s_i, n^v >$ to $A_e$;
7:  Get virtual link $l^v_{m,n}$ with the highest communication cost,
    where $n^v_m.m(t) < n^v_n.m(t)$;
8:  for each $n^s_i \in N^s$ do
9:      if $P_{n^v_n.loc(t),n^s_i} < l^v_{m,n}.p(t)$ AND $|n^s_i.v(t)| < n^s_i.n$ then
10:         add $< n^v_m.loc(t), n^s_i, n^v_m >$ to $A_c$;
11: for each $n^s_i \in N^s$ do
12:     if $n^s_i.rc(t) = 0$ then
13:         Get virtual node $n^v \in n^s_i.v(t)$ with smallest $\frac{n^v.m(t)}{n^v.c(t)}$ value;
14:         for each $n^s_j \in N^s$ do
15:             if $n^s_j.uc(t) + n^v.rc(t) < n^s_j.tc(t)$ AND $|n^s_j.v(t)| < n^s_j.n$
                then
16:                 add $< n^s_i, n^s_j, n^v >$ to $A^s_{cpu}$;
17: for each $l^s_{i,j} \in L^s$ do
18:     if $l^s_{i,j}.rb(t) = 0$ then
19:         Get each virtual link $l^v_{m,n}$ spanning over $l^s_{i,j}$ and find the
            virtual node $n^v_m$ with smallest $\frac{n^v_m.m(t)}{l^v_{m,n}.b(t)}$ value;
20:         for each $n^s_k \in N^s$ do
21:             if $p^s_{n,k}.ub(t) + l^v_{m,n}.b(t) < p^s_{n,k}.tb(t)$ AND $|n^s_k.v(t)| <$
                $n^s_k.n$ then
22:                 add $< n^v_m.loc(t), n^s_k, n^v_m >$ to $A^s_{bd}$;
23: Add each action in $A_e \cup A_c \cup A^s_{cpu} \cup A^s_{bd}$ to $Alist$ with proba-
    bility $\frac{1}{\sqrt{|N^s|}}$;
24: return $Alist$;
```

First, we select candidate action set A_e for energy optimization (lines 2-6). Since running a physical host consumes energy, we try to migrate the virtual nodes on the physical host when the physical host utilizes only a few resources. The physical node can be finally shut down, saving energy. We choose the physical host with the lowest CPU utilization as the source host and select the virtual node n^v with the smallest $\frac{n^v.m(t)}{n^v.c(t)}$ ratio as the virtual node to be migrated, considering migration cost. Without reaching the maximum number of virtual nodes, we select under-used hosts as target hosts. Add these actions to the candidate action set A_e. The size of the action set A_e is less than $|N^s| - 1$.

Then, we select candidate action set A_c for communication optimization (lines 7-10). For a pair of virtual nodes that possess large communication requests with each other, we can migrate one of the virtual nodes to the physical host where the other virtual node is located or to a physical host closer to the other virtual node to reduce communication costs. We select the virtual link $l^v_{m,n}$ with the highest communication cost (line 7), i.e., $l^v_{m,n}.b(t) \cdot l^v_{m,n}.p(t)$. Considering migration cost, we choose the virtual node (assuming n^v_m) with the smallest $n^v_m.m(t)$ as the virtual node to be migrated. Consequently, the physical host $n^v_m.loc(t)$ is the source host. We choose the under-used hosts that make the communication path between n^v_m and n^v_n shorter as the target host and add these actions to the candidate action set A_c. The size of action set A_c is less than $|N^s| - 1$.

Next, we select candidate action set A^s_{cpu} for computation SLAV optimization (lines 11-16). When the CPU utilization of all virtual nodes on the physical host exceeds the host capacity, the physical node cannot provide enough computation resources for each VM, resulting in the computation SLAV C^{cpu}_s. If virtual nodes in these physical hosts are migrated out, C^{cpu}_s will be reduced. Therefore, we choose the over-utilized physical hosts as the source hosts. Considering migration cost, we select the virtual node n^v with the smallest $\frac{n^v.m(t)}{n^v.c(t)}$ ratio in each source host as the virtual node to be migrated. We select the under-used host with adequate resources to run the virtual node as the target host and add these actions to the candidate action set A^s_{cpu}. The size of the action set A^s_{cpu} is less than $|N^s| - 1$.

Last, we select candidate action set A^s_{bd} for network SLAV optimization (lines 17-22). When the network resources of a physical link are over-utilized, the physical link cannot provide sufficient bandwidth for virtual links, resulting in network SLAV C^{bd}_s. Each over-utilized physical link $l^s_{i,j}$ is responsible for the communication of each virtual link $l^v_{m,n}$ in $l^s_{i,j}.l(t)$. If the virtual nodes communicating through these physical links are migrated, the communication path will also change, reducing the C^{bd}_s. When selecting virtual nodes to be migrated, we get all virtual nodes (take n^v_m as an example) connected to these virtual links, calculate $\frac{n^v_m.m(t)}{l^v_{m,n}.b(t)}$ value, and select the virtual node with the smallest value as the virtual node to be migrated (assuming n^v_m). The physical host $n^v_m.loc(t)$ is the source host. We select the under-used host that can provide sufficient bandwidth resource for the virtual link $l^v_{m,n}$ as the target host, and add these actions to the candidate action set A^s_{bd}. The size of the action set A^s_{bd} is less than $|N^s| - 1$.

In order to further reduce the action space, we put each action into the final candidate action set $Alist$ with a probability of $\frac{1}{\sqrt{|N^s|}}$ (line 23). The time complexity quantities of the above four action selection processes are $O(|N^s|)$, $O(|L^v| + |N^s|)$, $O(|N^s|^2)$, and $O(|L^s| \cdot |N^s|)$, respectively. Thus, the time complexity of Algorithm 3 is $O(|L^v| + |N^s|^2 + |L^s| \cdot |N^s|)$.

4.4 Mathematical Analysis

The proposed algorithm applies DDQN to calculate the Q values for migration actions in the $|N^v||N^s|$-size discrete action space and select an action with the most significant Q value. The training speed is languid in large-scale cloud environments. With the action filter, the agent reduces the action space. The total size of the candidate action sets A_e, A_c, A^s_{cpu}, and A^s_{bd} is smaller than $4 \cdot |N^s| - 4$, indicating that the "bad" migration size is greater than $|N^v| \cdot |N^s| - 4 \cdot$

$|N^s| + 4$. Considering the selection probability of $\frac{1}{\sqrt{|N^s|}}$, the actfilter can speed up the training process via reducing action space from $|N^v| \cdot |N^s|$ to less than $\frac{4 \cdot |N^s| - 4}{\sqrt{|N^s|}}$. Furthermore, the proposed method provides a performance guarantee to make a migration decision that can reduce one or more types of costs.

5 Experimental Evaluation

In order to verify the performance of our migration strategy in terms of multiple aspects, including the energy cost, communication cost, migration cost, SLAV, and total cost, we conduct simulation experiments. We choose to compare with three existing migration strategies in the simulation experiment. To verify the generalizability of the proposed MiOvnm, we design static and dynamic simulation experiments, where the VN state information stays constant in static simulation while the VN state information changes dynamically in dynamic simulation.

Furthermore, we conduct a real-world OpenStack experiment to verify the migration performance of MiOvnm.

5.1 Experimental Setup

5.1.1 Simulation Experimental Setup

In the small-scale experiment, the numbers of physical nodes and links are 10 and 20, respectively, while the numbers of virtual nodes and links are 50 and 100, respectively. In the medium-scale experiment, the numbers of physical nodes and links are 200 and 800, respectively, while the numbers of virtual nodes and links are 1000 and 2000, respectively. In the large-scale experiment, the numbers of physical nodes and links are 1000 and 2000, respectively, while the numbers of virtual nodes and links are 5000 and 10000, respectively The maximum number of virtual nodes on a host is set at 50. Both PNs and VNs are generated with BRITE [36]. Physical node resources and link resources are randomly generated through normal distribution. The CPU resource of each physical node obeys a normal distribution with a mean of 120 and a variance of 20. The bandwidth resource of each physical link obeys a normal distribution with a mean of 150 and a variance of 20.

Each virtual node includes CPU resources and memory resources, and each virtual link includes bandwidth resources. In the experimental VNs, to simulate the dynamic workloads in the cloud center environment more realistically, we select the Alibaba Cluster Data[1] open-sourced by Alibaba in 2017 as the experimental data set. The cluster-trace-v2017 contains CPU, memory, and other resources' changing information of 1313 machines and 11275 tasks in the 12 hours. Each task's current CPU request amount and memory request amount are recorded every 5 minutes. We randomly choose a resource state for 5 minutes as an experimental environment in the static simulation. While in dynamic simulation experiments, we choose resource information as the resource variation of each virtual node during the experiment and update each virtual node's CPU and memory resources every 5 minutes for 20 times.

1. https://github.com/alibaba/clusterdata

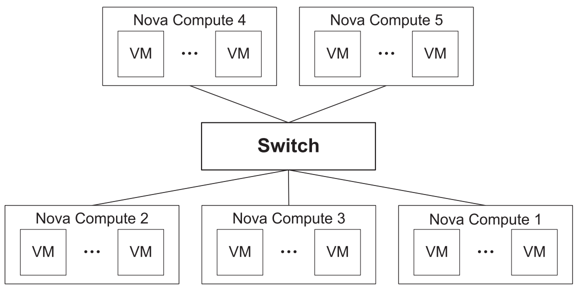


Fig. 4. The OpenStack-base physical network framework.

In order to fully illustrate the performance of our MiOvnm, we compare with the following three algorithms.

1) VMCUP-M algorithm [18] is a dynamic migration algorithm based on prediction. The core idea is to determine the source host by predicting the resource usage in the next six moments, selecting the virtual node to be migrated, and finding the target host to avoid frequent migration of virtual nodes.
2) DRAUVM algorithm [15] is based on exponentially weighted moving average parameters. The algorithm introduces the concept of resource skewness to measure the unevenness of server resource utilization. It uses skewness to determine the target host for migration. Thereby resource allocation is more even.
3) NSGA-II algorithm [37], as the static genetic-based solution for multi-objective optimization, stratifies the population through non-dominated sort, retains the excellent individuals to the offspring, and continuously selects and iterates the excellent offspring, wherein the mapping of each virtual node acts as the encoding method.

5.1.2 OpenStack Experimental Setup

We use the open-source cloud service platform OpenStack as the real-world experimental platform in this study. The platform framework is shown in Fig. 4. Each server possesses two Intel(R) Xeon(R) Silver 4116 processors, providing 128GB of memory and a disk size of 3.3TB. A measured network environment of approximately 300Mbps for each host. The operating system is Centos-release-7-5 x86_64. We use a VN composed of VMs to verify the proposed migration algorithm. The nova service in the framework is a project where KVM is deployed, and each node has a vCPU quota of 192. The maximum number of instances on a host is set to 50. The platform includes five nova computing nodes and a switch (the controller node is omitted in Fig. 4). Each nova node has two network interface controllers (NICs) in this architecture. One, named tunnel-nic, provides the tunnel network used for communication between VMs; and the other, named manage-nic, is used for management, including VM migration. In addition, we set the coefficients $\alpha = \beta = \zeta = 1$ and $\theta = 5$, indicating the optimization objectives of our cloud platform.

We designed the VN in Fig. 5, used to verify and analyze the migration performance of the MiOvnm algorithm on the real-world platform. We deployed a VN with 100 VMs on the OpenStack platform, and the initial vCPU, memory, and disk configurations were 1, 2GB, and 40GB, respectively.

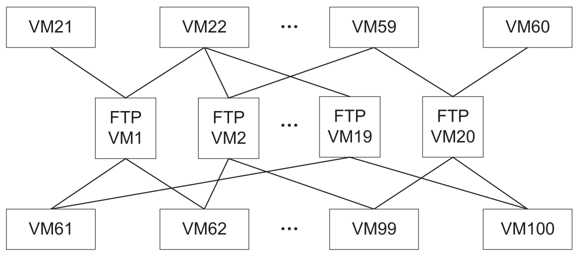


Fig. 5. The FTP-base virtual network topology.

The memory and disk resources of each host are sufficient in general. We evaluate the performance with DaCapo [38] benchmarks on each VM and select computation-intensive benchmarks (including h2, eclipse, avrora, jython, sunflow, lusearch, lusearch-fix, pmd, luindex, batik, fop, xalan). During the experiment, we package these programs together and execute them sequentially. In addition, we also choose a network-intensive test, setting 20 VMs (VM1-VM20) as FTP servers, while the remaining VMs (VM21-VM100) randomly extract fixed-size files from the FTP server document. We set the bandwidth requirement of each virtual node to be proportional to the size of the file to be transmitted, 10Mbps for each 1GB file. We set the migration scheduling period in the platform to 30 minutes (not including the migration process time). Let the workload of the VM also change every 30 minutes to evaluate the program execution performance and resource usage.

5.2 Pre-Training Process and Parameter Setting

Before training, we set training data regarding VNs and PNs, where each PN is set the same as experimental PNs. Each VN shares the same topology structure as experimental VNs. The CPU and memory of each virtual node obey normal distributions with a mean of 10 and a variance of 3, and with a mean value of 5 and a variance of 2, respectively. Moreover, the bandwidth of each virtual link obeys a normal distribution with a mean value of 10 and a variance of 3.

The training processes for small-scale, medium-scale, large-scale, and OpenStack experiments are performed according to algorithm 1, where both small-scale and OpenStack experiments experience 10,000 episodes, medium- and large-scale experiments experience 30,000 episodes. The Q networks' input layer is a $|N^s|+2|N^v|+|L^s|+|L^v|+3$ dimensional vector, and the output layer is a one-dimensional vector. In the small-scale simulation and the OpenStack experiments, we apply a four-layer fully connected network as the structure of the DNN, wherein the two hidden layers have 128 and 16 dimensions. In the medium- and large-scale simulation experiments, a five-layer fully connected network is used as the structure of the DNN, wherein the three hidden layers have dimensions of 1024, 128, and 16, respectively. All networks use the Relu function as the activation function. During the training process, the ϵ value in the ϵ-greedy strategy is 0.2, and the Adam optimizer is used to update the neural network parameters. Generally, the discount factor γ ranges from 0 to 1. The larger the γ is, the broader the algorithm's vision on global optimization; the smaller the γ is, the more the algorithm focuses on the current performance [39]. Here, we suggest $\gamma = 0.9$ to attain a broader vision of global optimization in

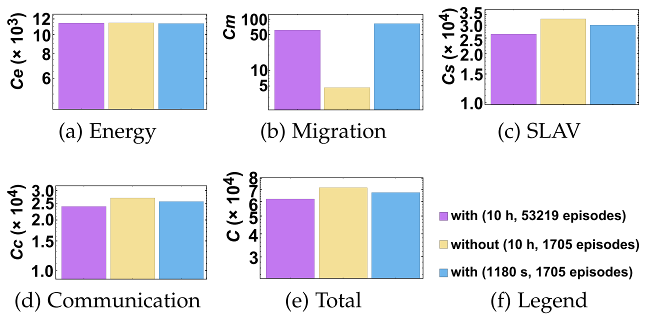


Fig. 6. Comparison between three MiOvnm models.

the proposed algorithm. As for the learning rate α_Q, which is generally set at a small value (such as value from 10^{-6} to 10^{-1}), based on the observation in numerous experiments, we find that the MiOvnm attains better performance during the training process when $\alpha_Q = 10^{-3}$.

5.3 MiOvnm Analysis

We analyze the MiOvnm in terms of efficiency and performance and present each migration step in a small-scale dynamic experiment (where $\alpha = \beta = \theta = \zeta = 1$). We train a MiOvnm model with and without actfilter for 10 hours (h), respectively. The MiOvnm trains for 53219 episodes when it runs with actfilter while only 1705 episodes for no actfilter. Additionally, we train a MiOvnm model with actfilter for 1705 episodes, of which the completion time is only 1180 seconds (s). These results indicate that actfilter improves the efficiency of MiOvnm significantly. We compare the trained models in a small-scale dynamic experiment.

Fig. 6 presents the comparison results. The MiOvnm with actfilter achieves smaller energy, communication, and SLAV costs. For the total cost, the MiOvnm with actfilter shows 13.06% and 5.65% less cost than the MiOvnm without actfilter. The comparison between "with (1180s, 1705 episodes)" and "without (10h, 1705 episodes)" indicates that actfilter helps MiOvnm make better decisions when the early agent learns relatively little information.

We also present the relation between immediate cost and migration steps under the "with (10h, 53219 episodes)" situation (Fig. 7). The immediate cost is the negative of the immediate reward (i.e., $-R_t$). As stated in Section 5.1.1, the VN experiences 20-time resource variation stages during the experiment. Empty results (such as Figs. 7f, 7g, 7h, 7m, 7p, 7q, 7s, and 7t) mean that our MiOvnm decides not to migrate. While the other results present that our MiOvnm chooses migration actions that reduce immediate cost. The experimental results indicate that the proposed MiOvnm always makes the migration decisions that reduce the immediate cost, achieving the overall optimization of multiple VNM objectives.

5.4 Small-Scale Static Simulation

In a small-scale static simulation experiment, we evaluate the performance of MiOvnm under $\alpha = \beta = \theta = \zeta = 1$.

Fig. 8 presents the results of MiOvnm and the comparison algorithms. According to the results, the MiOvnm achieves optimal SLAV and total cost. In total cost result (Fig. 8e), the MiOvnm presents 6.30%, 9.22%, and 15.52% less cost than NSGA-II, DRAUVM, and VMCUP-M,

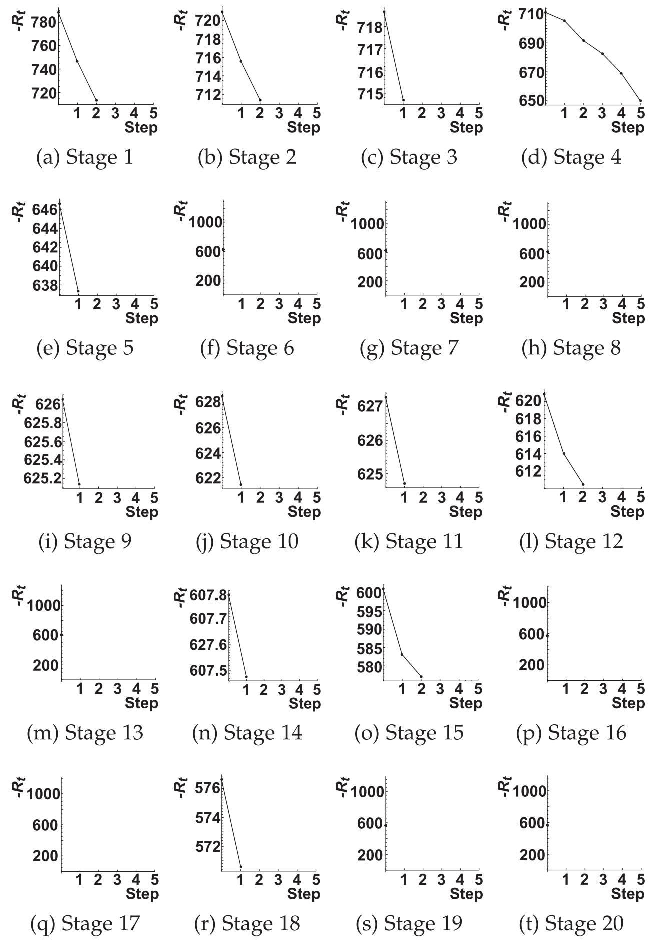
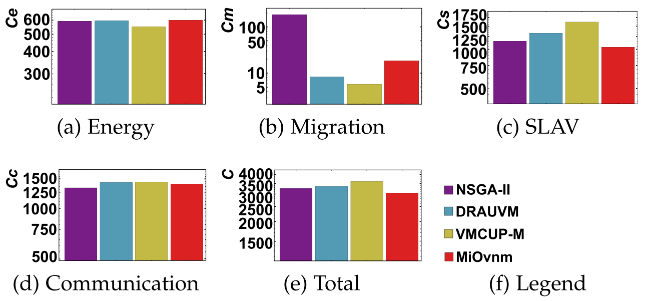

(a) Stage 1 (b) Stage 2 (c) Stage 3 (d) Stage 4 (e) Stage 5 (f) Stage 6 (g) Stage 7 (h) Stage 8 (i) Stage 9 (j) Stage 10 (k) Stage 11 (l) Stage 12 (m) Stage 13 (n) Stage 14 (o) Stage 15 (p) Stage 16 (q) Stage 17 (r) Stage 18 (s) Stage 19 (t) Stage 20

Fig. 7. Relation between immediate cost and migration step under the "with (10h, 53219 episodes)" situation.

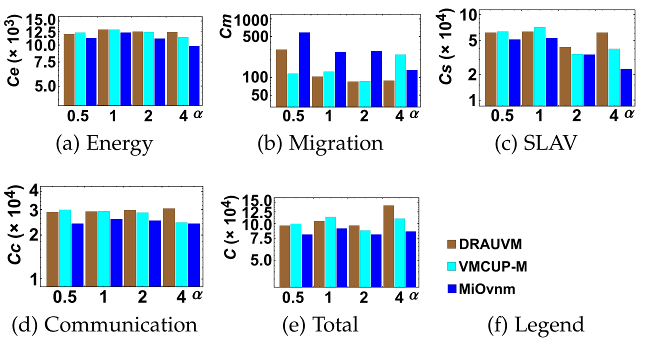
respectively. In terms of SLAV overhead (Fig. 8c), the MiOvnm algorithm is 10.04%, 22.03%, and 35.99% less than the NSGA-II, DRAUVM, and VMCUP-M, respectively. The NSGA-II presents more SLAV and migration costs to reduce communication consumption, whereas NSGA-II presents 90.10% and 10.04% higher than MiOvnm in migration and SLAV overhead. Furthermore, the MiOvnm presents nearly the exact energy cost as NSGA-II and DRAUVM, while 8.81% worse than the VMCUP-M (Fig. 8a). However, the VMCUP-M results in more SLAV and communication costs to reduce energy and migration overhead, resulting in a higher total cost.

Compared with three benchmark algorithms, MiOvnm reduces the total cost by the average rate of 10.35%, with



(a) Energy (b) Migration (c) SLAV (d) Communication (e) Total (f) Legend

Fig. 8. Cost comparison in small-scale static simulation.



(a) Energy (b) Migration (c) SLAV (d) Communication (e) Total (f) Legend

Fig. 9. Cost comparison under different energy parameters in small-scale dynamic simulation.

SLAV cost reduced by 22.68% on average. The results verify that the MiOvnm can improve static VNM evaluation performance.

5.5 Small-Scale Dynamic Simulation

The NSGA-II needs to repeat the time-consuming genetic-mutation process for the new network state to find an optimal individual when VN state information changes in a small-scale dynamic simulation experiment. By numerous attempts, the time-consuming computation processes reveal that NSGA-II is challenging to tackle the situation in which the VN state information changes dynamically. Therefore, the NSGA-II is absent in the dynamic simulation evaluations. In this section, we set $\zeta = 1$ and demonstrate the performance of MiOvnm under different α, β, and θ values.

We first take different α values, indicating different weights are given to the energy optimization. Fig. 9 presents the results of MiOvnm and the comparison algorithms. The MiOvnm achieves smaller costs in energy, communication, and SLAV. In total cost result (Fig. 9e), when α takes $\{0.5, 1, 2, 4\}$, the MiOvnm presents 15.42%, 12.90%, 15.45%, 38.92% less cost than DRAUVM, and 18.26% 19.33% 7.11% 21.51% lower than VMCUP-M. In energy cost result (Fig. 9a), when α changes in $\{0.5, 1, 2, 4\}$, the energy cost of our MiOvnm performs 6.64%, 4.92%, 11.51%, 21.18% smaller than DRAUVM, and 8.52%, 4.65%, 10.78%, 14.23% smaller than VMCUP-M. With the weight (i.e., α) of energy cost in the reward function increasing, our MiOvnm reduces energy consumption during decision-making to obtain lower energy costs. In terms of communication cost (Fig. 9d), the MiOvnm presents 16.29% and 11.42% less than the DRAUVM and VMCUP-M on average, respectively. In terms of the SLAV (Fig. 9c), the MiOvnm result is 28.05% and 21.81% less than the DRAUVM and VMCUP-M on average, respectively.

Fig. 10 presents the results of our MiOvnm and two comparison algorithms under different β values, indicating different weights are given to the communication optimization. The MiOvnm can achieve better energy, communication SLAV, and total costs. In terms of total cost, when β changes in $\{0.5, 1, 2, 4\}$, the MiOvnm result is 21.51%, 12.90%, 18.12%, 19.47% lower than the DRAUVM, and 10.87%, 19.33%, 24.23%, 34.39% lower than the VMCUP-M. When β takes values in $\{0.5, 1, 2, 4\}$, the MiOvnm can achieve better results in communication cost, presenting 18.34%, 11.68%, 12.04%, 22.90% less costs than DRAUVM, and 8.53%, 11.87%, 13.56%,

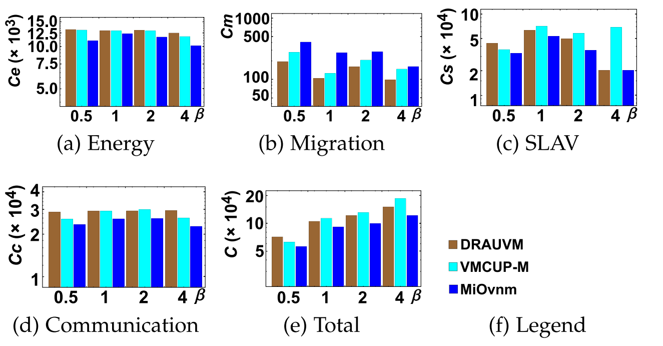


Fig. 10. Cost comparison under different communication parameter in small-scale dynamic simulation.

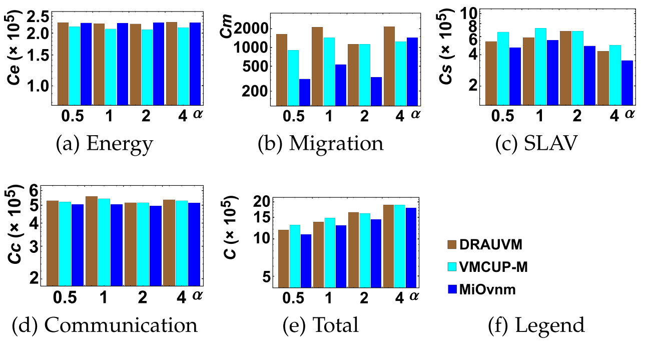


Fig. 12. Cost comparison under different energy parameters in medium-scale dynamic simulation.

12.91% less than that of VMCUP-M. Except for the comparison with DRAUVM under $\beta = 0.5$ and VMCUP-M under $\beta = 4$, the rest is consistent with the fact that our MiOvnm can reduce more communication costs with the increase of β. Since the communication cost and the network SLAV possess a specific correlation, the action that reduces the communication cost may also reduce the network SLAV, so the SLAV can also achieve better results with the increase of β. Similar to experiments with different α, the migration cost takes more than DRAUVM and VMCUP-M because more decisions need to be made to reduce the communication cost and network SLAV. In terms of energy consumption, the MiOvnm is 12.81% and 11.11% less than the DRAUVM and VMCUP-M on average, respectively. In terms of SLAV, the MiOvnm algorithm is 16.95% and 36.21% less than the DRAUVM and VMCUP-M on average, respectively.

Fig. 11 shows the results when θ takes different values. That is, different weights are given to SLAV in the reward function. The MiOvnm achieves better results than two comparison algorithms in energy cost, communication cost, and SLAV. In total cost (Fig. 11a), with θ value increasing in $\{0.5, 1, 2, 4\}$, our MiOvnm can often achieve better results, presenting 5.29%, 12.90%, 44.26%, 19.16% less cost than the DRAUVM, and 15.66%, 19.33%, 32.34%, 41.94% lower than the VMCUP-M. When θ increases, the MiOvnm considers more decisions to reduce SLAV, reducing by -2.24%, 15.41%, 52.29%, 20.08% compared to the DRAUVM, and 24.36%, 25.31%, 40.20%, 46.72% than VMCUP-M. The MiOvnm will produce higher migration overhead under different θ. Since the MiOvnm considers the energy cost and computation SLAV in the VMCUP-M and DRAUVM algorithms, communication cost, and network SLAV, the

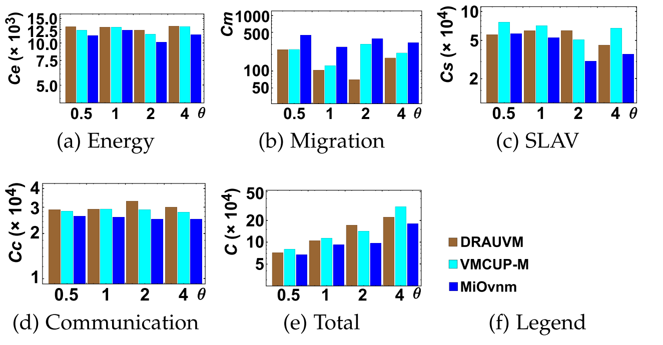


Fig. 11. Cost comparison under different SLAV parameters in small-scale dynamic simulation.

MiOvnm needs to make more migration decisions than the other two algorithms to reduce communication cost and network SLAV. Furthermore, the additional migration cost in MiOvnm is less than the reduced communication cost and SLAV, thereby achieving better results in total cost. In terms of energy consumption, the MiOvnm is 12.44% and 9.47% less than the DRAUVM and VMCUP-M on average, respectively. The communication cost of MiOvnm is 15.67% and 10.81% less than that of the DRAUVM and VMCUP-M. These results show that the MiOvnm also reduces energy and communication costs under different θ values.

Based on the above experimental results, we can conclude that the MiOvnm can achieve less cost than two comparison algorithms in terms of energy cost, communication cost, and SLAV in small-scale experiments. Compared with the comparison algorithms, MiOvnm reduces the SLAV, communication cost, and energy cost by 26.43%, 13.69%, and 11.07% on average, respectively, with total cost reduced by 20.86% on average. The results demonstrate superiority of the MiOvnm in multi-objective VNM optimization. Although the migration costs are higher than the comparison algorithms, the MiOvnm can achieve the slightest results in the total cost. Furthermore, with different parameters in the reward function, the MiOvnm adapts the migration decision to align the resource allocation to different objectives.

5.6 Medium-Scale Dynamic Simulation

In medium-scale experiments, we also set $\zeta = 1$ and evaluate the performance of MiOvnm under different α, β, and θ values in the reward function.

According to the results (Fig. 12), the MiOvnm achieves optimal migration cost, communication cost, SLAV, and total cost. In terms of total cost (Fig. 12e), when α takes $\{0.5, 1, 2, 4\}$, the MiOvnm presents 8.07%, 6.24%, 12.49%, 5.74% less than the DRAUVM, and 15.86%, 12.75%, 10.69%, 5.25% less than the VMCUP-M. Regarding energy cost (Fig. 12a), MiOvnm is the same as DRAUVM, while 7.35% worse than the VMCUP-M on average. However, the VMCUP-M presents more SLAV to reduce energy consumption, resulting in a higher total cost. In terms of migration cost (Fig. 12b), the MiOvnm is 64.88% and 45.67% less than the DRAUVM and VMCUP-M on average, respectively. Regarding communication cost (Fig. 12d), the MiOvnm is 5.52% and 4.24% less than the DRAUVM and VMCUP-M on average, respectively. In terms of SLAV

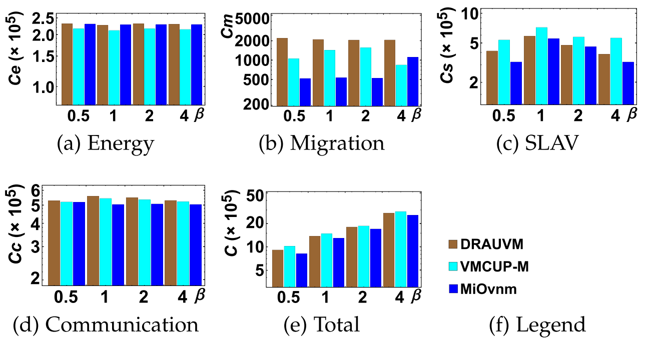


Fig. 13. Cost comparison under different communication parameters in medium-scale dynamic simulation.

(Fig. 12c), the MiOvnm is 16.32% and 27.27% less than the DRAUVM and VMCUP-M on average, respectively.

Fig. 13 shows the results of the MiOvnm and two comparison algorithms when the parameter β takes different values in $\{0.5, 1, 2, 4\}$, indicating different weights are given to the communication optimization. According to the results, the MiOvnm achieves optimal migration cost, communication cost, SLAV, and total cost. In terms of the total cost, when β changes in $\{0.5, 1, 2, 4\}$, the MiOvnm possesses stable improvements compared to the DRAUVM and VMCUP-M, presenting 11.06%, 6.24%, 5.62%, 6.12% less cost than the DRAUVM, and 20.11%, 12.75%, 8.58%, 10.23% less than the VMCUP-M. Regarding energy cost, MiOvnm is similar to that of the DRAUVM but worse than the VMCUP-M. However, the VMCUP-M causes more SLAV to reduce energy cost, resulting in a large gap between the VMCUP-M and the MiOvnm regarding the total cost. In terms of migration cost, the MiOvnm is 67.93% and 37.16% less than the DRAUVM and VMCUP-M on average, respectively. In the medium-scale evaluation environment, the migrations of virtual nodes can make link state change significantly. When an action reduces a specific cost, it may lead to the increase of other costs. Therefore, our MiOvnm reduces many migration actions to reduce the total cost. In terms of SLAV, the MiOvnm is 12.06% and 31.70% lower than the DRAUVM and the VMCUP-M on average, respectively. In terms of communication cost, MiOvnm is 1.94%, 9.38%, 7.49%, 4.76% less than DRAUVM, and 0.48%, 6.94%, 5.01%, 3.20% less than VMCUP-M.

Fig. 14 shows the results of the MiOvnm and the comparison algorithms when different θ weights are assigned to the SLAV optimization. According to the results, the MiOvnm also achieves optimal migration cost, communication cost, SLAV, and total cost. In terms of total cost, when θ takes $\{0.5, 1, 2, 4\}$, the MiOvnm presents 7.03%, 6.24%, 13.25%, 9.64% less than the DRAUVM, and 5.51%, 12.75%, 19.09%, 32.49% less than the VMCUP-M. Similar to the previous results in energy cost, the MiOvnm shows similar results to the DRAUVM but worse than the VMCUP-M. The VMCUP-M performs worse in the total cost, proving the VMCUP-M optimizes the energy cost while ignoring the impact of other costs. In terms of migration cost, the MiOvnm is 75.95% and 60.18% less than the DRAUVM and VMCUP-M on average, proving that our MiOvnm reduces many migration actions to obtain a continuous reduction of the total cost. In terms of communication cost, the MiOvnm is 7.02% and 4.06% less than the DRAUVM and VMCUP-M on average, respectively. In terms of SLAV, when θ takes $\{0.5, 1, 2, 4\}$, the MiOvnm is 4.40%, 5.64%, 18.27%, 14.53% less than the DRAUVM, and 16.81%, 23.08%, 29.18%, 44.27% less than the VMCUP-M. With the SLAV parameter θ increasing, the MiOvnm can can significantly reduce SLAV.

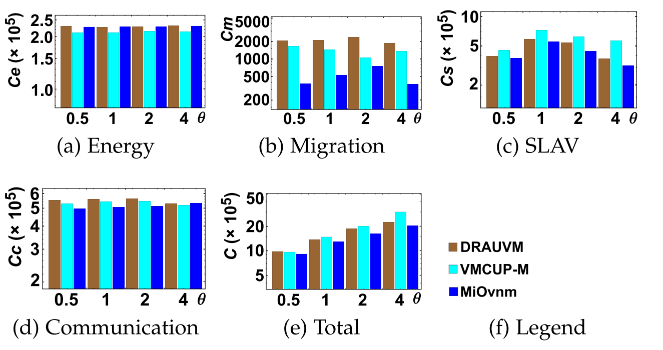


Fig. 14. Cost comparison under different SLAV parameters in medium-scale dynamic simulation.

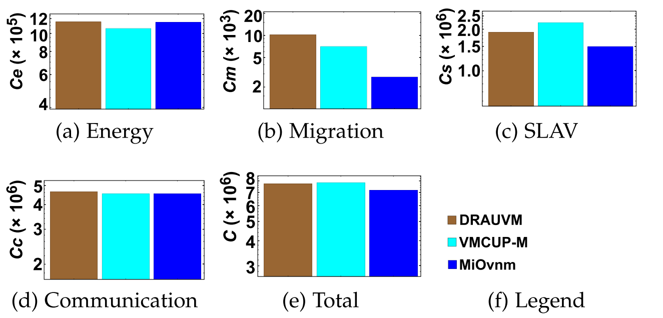


Fig. 15. Cost comparison in large-scale dynamic simulation.

Based on the above experimental results, we can conclude that the MiOvnm can achieve the best results in migration, communication, SLAV, and total cost in medium-scale experiments. Compared with two comparison algorithms, MiOvnm reduces the SLAV, communication cost, and migration cost by average rates of 21.07%, 5.11%, and 58.63%, respectively, with total cost reduced by 10.99% on average. Regarding energy cost, the MiOvnm presents close results to the DRAUVM, while slightly larger than the VMCUP-M. However, the VMCUP-M sacrifices more SLAV to reduce the energy cost, increasing the total cost. When the weight of the SLAV parameter θ changes, the MiOvnm makes different decisions. The larger the weight θ, the smaller the SLAV. However, the energy cost parameter α has little effect on the migration result. In medium-scale evaluation, the migration actions affect communication cost and SLAV significantly but have less impact on energy cost. The results of VMCUP-M verify that migration actions that try to reduce energy costs may significantly increase SLAV. In comparison, our MiOvnm tries to avoid numerous actions to reduce energy consumption due to the consideration of multiple objectives.

5.7 Large-Scale Dynamic Simulation

In a large-scale experiment, we evaluate the performance of MiOvnm under $\alpha = \beta = \theta = \zeta = 1$ situation. Fig. 15 presents the results of MiOvnm and the comparison algorithms.

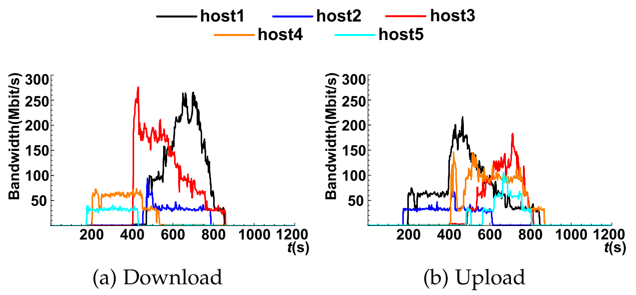


Fig. 16. Migration traffic through manage-nic of compute nodes.

According to the results, the MiOvnm achieves optimal migration cost, SLAV, and total cost. In total cost (Fig. 15e), the MiOvnm presents 6.86% and 8.36% less cost than DRAUVM and VMCUP-M. In terms of energy cost (Fig. 15a), MiOvnm presents slightly less than DRAUVM, while 7.89% worse than the VMCUP-M. However, the VMCUP-M presents more SLAV and migration costs to reduce energy consumption, resulting in a higher total cost. The MiOvnm performs slightly less than comparison methods regarding communication cost (Fig. 15d). In terms of migration cost (Fig. 15b), the MiOvnm is 73.34% and 61.53% less than the DRAUVM and VMCUP-M, respectively. In terms of SLAV overhead (Fig. 15c), the MiOvnm is 21.38% and 32.80% less than the DRAUVM and VMCUP-M, respectively.

Compared with the two comparison algorithms, MiOvnm reduces the total cost by the average rate of 7.61%, with SLAV and migration costs reduced by 27.09% and 67.43% on average, respectively. The results verify that the MiOvnm can improve performance in large-scale scenarios.

5.8 OpenStack-Based Experiment

The OpenStack-based experiment process is divided into three stages. Each stage contains 30 minutes after the migration is completed. The bandwidth requests and the FTP tasks of the VMs do not change at each stage. While the CPU requests and computation-intensive benchmarks of VMs change: i) stage 1, 1vCPU, FTP task; ii) stage 2, 2vCPU, FTP task, 1-round DaCapo benchmarks; and iii) stage 3, 4vCPU, FTP task, 2-round DaCapo benchmarks. We test the three stages separately under the default deployment of OpenStack (called Default) and under MiOvnm migration, respectively. We observe the tasks' completion time and resource usage in each stage, whereas we analyze the migration cost, SLAV, communication cost, and energy cost.

5.8.1 *Migration Cost*

Before the test program of the first stage start, we perform a migration based on our MiOvnm. The virtual nodes are migrated through the manage-nic NICs in each host. We obtain the upload and download traffic information during the migration process (Fig. 16). Before the beginning of the remaining two phases, no migration occurs according to our MiOvnm algorithm.

During the VM migration process, the target hosts receive the copy-on-write image file and memory data of the VM from the source host and may also need to receive the original backing image from the control node. Therefore, these five compute nodes' total upload and download traffic behave inconsistently. The migration process is complete through manage-nic and does not compete with the communication tunnel-nic NICs, demonstrating the feasibility of setting a relatively low migration parameter in evaluation.

TABLE 3
Completion Times of Benchmarks in Three Stages

Stage	Default (s)		MiOvnm (s)	
	DaCapo	FTP	DaCapo	FTP
1	—	175.460265	—	176.184192
2	694.77783	208.99754	652.5817	189.56458
3	1092.734154	279.1179752	911.0479614	217.5921428

5.8.2 *Service Level Agreement Violation*

We obtain the completion time of the benchmarks in the three stages, as shown in table 3.

Compared with the default deployment of OpenStack, the MiOvnm significantly improves the performance of VMs, reducing the completion time of the FTP and DaCapo tasks by average rates of 10.31% and 11.35%, respectively.

In the default deployment of OpenStack, the distribution of VMs is relatively even. At the same time, the computing resource requests and workloads of VMs are the same. Therefore, as the amount of computing resource requests increases, resource allocation of default deployment is relatively reasonable for VMs. However, default deployment ignores bandwidth resource allocation and causes VMs to suffer from bandwidth restrictions, resulting in a relatively large network SLAV. After our MiOvnm migration process, communication between VMs can be improved, reducing competition for PN resources and the completion time of network-intensive tasks. The experimental results in Table 3 demonstrate that the MiOvnm reduces the FTP task completion time by an average rate of 10.31% compared with the default deployment of OpenStack.

In addition, the IO events in the host NIC are reduced by our MiOvnm migration, which can reduce the waiting time for responding to IO events and the competition for computation resources. Figs. 17, 18, and 19 show the CPU resource usage of each compute node in the three stages. The CPU resources of the physical hosts in each stage have been more efficiently used, and the completion time of the computing process is shortened. According to the results in Table 3, the MiOvnm migration reduces the completion

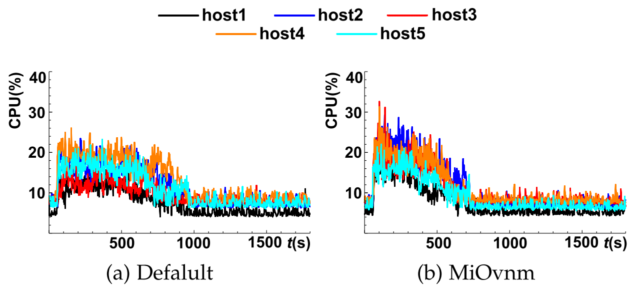


Fig. 17. CPU utilization during the first stage.

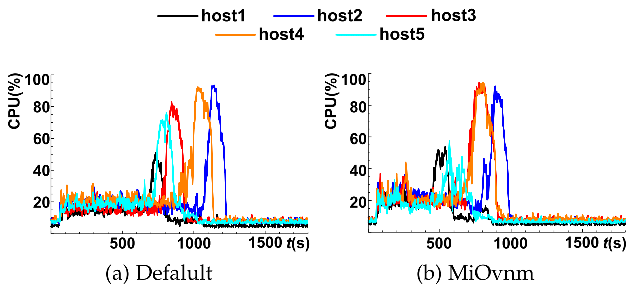


Fig. 18. CPU utilization during the second stage.

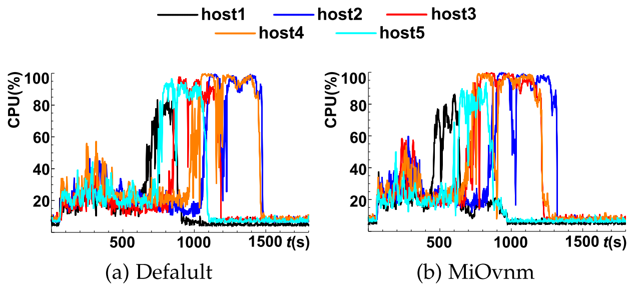


Fig. 19. CPU utilization during the third stage.

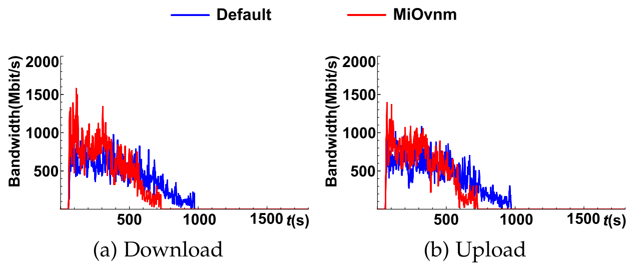


Fig. 20. Bandwidth utilization during the first stage.

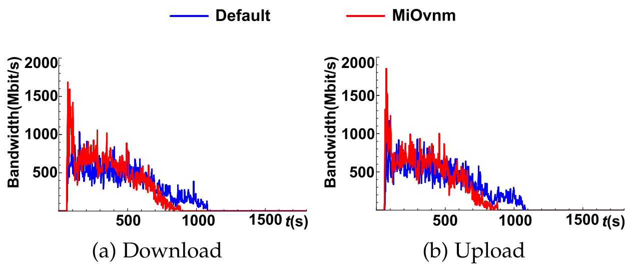


Fig. 21. Bandwidth utilization during the second stage.

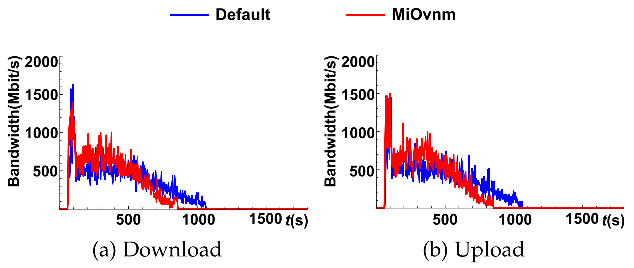


Fig. 22. Bandwidth utilization during the third stage.

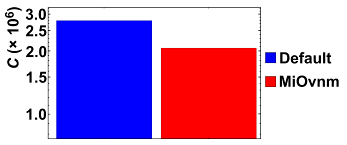


Fig. 23. Total cost comparison.

time of the DaCapo benchmarks by 11.35% compared to the default deployment of OpenStack.

5.8.3 *Communication Cost*

We obtain the traffic information of the communication tunnel-nic NIC of each physical host during the experiment and accumulate it. The traffic information of the communication process is shown in Figs. 20, 21, and 22. Firstly, similar to CPU utilization, the completion time of the network-intensive tasks has been reduced in all stages by our MiOvnm migration. Secondly, our MiOvnm migration reduces the traffic size between physical nodes, optimizing the performance of network tasks and alleviating the congestion in the PN.

5.8.4 *Energy Cost*

Based on the above results, our MiOvnm migration can fully use computation resources, which is beneficial for energy optimization. As for optimizing the number of active computing nodes, due to the limited size of the PN in this experiment, it is challenging to release resources in the limited five compute nodes, and energy has not been significantly optimized. Therefore, we set a relatively low energy parameter (i.e., α) in the OpenStack experimental model.

5.8.5 *Total Cost*

Finally, we obtain the total cost for the three stages via our VNM program, as shown in Fig. 23.

Compared with the default deployment of OpenStack, the MiOvnm presents 26.02% less total cost, indicating that our MiOvnm can significantly optimize the overall VNM performance in terms of multiple objectives.

6 Conclusion

This paper has modelled the multi-objective VNM problem with the consideration of energy cost, communication cost, migration cost, and SLAV. Then, it has proposed a MiOvnm algorithm based on deep reinforcement learning, which uses DNNs to process large-scale network state space and applies an action selection method called actfilter to deal with large-scale action space. The MiOvnm finds the migration action with optimal potential reward from the candidate action set. We have conducted two types of experiments to verify the effectiveness of MiOvnm. Then, we have compared the proposed MiOvnm with existing algorithms in the simulation experiment. Small-, medium- and large-scale simulation experiments reveal that the MiOvnm performs better than the comparison methods. In particular, SLAV and communication costs are reduced by 24.32% and 4.95% on average, respectively. The total cost is reduced by 12.45% on average. Furthermore, the experimental results of the real-world

OpenStack show that the MiOvnm migration can make full use of the computation and network resources of the compute nodes. The completion time of computation- and network-intensive programs is reduced by 11.35% and 10.31%, respectively, in comparison with the default OpenStack deployment. The total cost reduction reaches as high as 26.02%.

REFERENCES

[1] B. Shi and H. Shen, "Memory/disk operation aware lightweight VM live migration across data-centers with low performance impact," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 334–342.

[2] R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shiraz, A. Yousafzai, and F. Xia, "A survey on virtual machine migration and server consolidation frameworks for cloud data centers," *J. Netw. Comput. Appl.*, vol. 52, pp. 11–25, 2015.

[3] L. Gao and G. N. Rouskas, "Virtual network reconfiguration with load balancing and migration cost considerations," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 2303–2311.

[4] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration modeling and learning algorithms for containers in fog computing," *IEEE Trans. Services Comput.*, vol. 12, no. 5, pp. 712–725, Sep./Oct. 2019.

[5] Z. Zhang, H. Cao, S. Su, and W. Li, "Energy aware virtual network migration," *IEEE Trans. Cloud Comput.*, vol. 10, no. 2, pp. 1173–1189, Apr.–Jun. 2022.

[6] M. Masdari and M. Zangakani, "Green cloud computing using proactive virtual machine placement: Challenges and issues," *J. Grid Comput.*, vol. 18, no. 4, pp. 727–759, 2020.

[7] P. K. Sahoo, C. K. Dehury, and B. Veeravalli, "LVRM: On the design of efficient link based virtual resource management algorithm for cloud platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 4, pp. 887–900, Apr. 2017.

[8] H. Wang, Y. Li, Y. Zhang, and D. Jin, "Virtual machine migration planning in software-defined networks," *IEEE Trans. Cloud Comput.*, vol. 7, no. 4, pp. 1168–1182, Oct.–Dec. 2019.

[9] J. L. Chen, D. Liaqat, M. Gabel, and E. de Lara, "Starlight: Fast container provisioning on the edge and over the WAN," in *Proc. 19th {USENIX} Symp. Netw. Syst. Des. Implementation*, 2022, pp. 35–50.

[10] L. Zeno *et al.*, "SwiSh: Distributed shared state abstractions for programmable switches," in *Proc. 19th {USENIX} Symp. Netw. Syst. Des. Implementation*, 2022, pp. 171–191.

[11] W. Zhang, D. Wang, S. Yu, H. He, and Y. Wang, "Repeatable multi-dimensional virtual network embedding in cloud service platform," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2021.3102016.

[12] F. Zhang, G. Liu, X. Fu, and R. Yahyapour, "A survey on virtual machine migration: Challenges, techniques, and open issues," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 2, pp. 1206–1243, Apr.–Jun. 2018.

[13] Z. Han, H. Tan, R. Wang, G. Chen, Y. Li, and F. C. M. Lau, "Energy-efficient dynamic virtual machine management in data centers," *IEEE/ACM Trans. Netw.*, vol. 27, no. 1, pp. 344–360, Feb. 2019.

[14] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.

[15] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1107–1117, Jun. 2013.

[16] N. J. Kansal and I. Chana, "Energy-aware virtual machine migration for cloud computing-a firefly optimization approach," *J. Grid Comput.*, vol. 14, no. 2, pp. 327–345, 2016.

[17] F. Tao, C. Li, T. W. Liao, and Y. Laili, "BGM-BLA: A new algorithm for dynamic migration of virtual machines in cloud computing," *IEEE Trans. Services Comput.*, vol. 9, no. 6, pp. 910–925, Nov./Dec. 2016.

[18] N. T. Hieu, M. Di Francesco, and A. Ylä-Jääski, "Virtual machine consolidation with multiple usage prediction for energy-efficient cloud data centers," *IEEE Trans. Services Comput.*, vol. 13, no. 1, pp. 186–199, Jan./Feb. 2020.

[19] A. Al-Dulaimy, W. Itani, R. Zantout, and A. Zekri, "Type-aware virtual machine management for energy efficient cloud data centers," *Sustain. Comput. Informat. Syst.*, vol. 19, pp. 185–203, 2018.

[20] J. Sha, A. G. Ebadi, D. Mavaluru, M. Alshehri, O. Alfarraj, and L. Rajabion, "A method for virtual machine migration in cloud computing using a collective behavior-based metaheuristics algorithm," *Concurrency Comput. Pract. Experience*, vol. 32, no. 2, 2020, Art. no. e5441.

[21] K. Karthikeyan *et al.*, "Energy consumption analysis of virtual machine migration in cloud using hybrid swarm optimization (abc–ba)," *J. Supercomput.*, vol. 76, no. 5, pp. 3374–3390, 2020.

[22] V. Garg and B. Jindal, "Energy efficient virtual machine migration approach with sla conservation in cloud computing," *J. Central South Univ.*, vol. 28, no. 3, pp. 760–770, 2021.

[23] M. I. Khaleel and M. M. Zhu, "Adaptive virtual machine migration based on performance-to-power ratio in fog-enabled cloud data centers," *J. Supercomput.*, vol. 77, no. 10, pp. 11 986–12 025, 2021.

[24] W. Zhang, S. Han, H. He, and H. Chen, "Network-aware virtual machine migration in an overcommitted cloud," *Future Gener. Comput. Syst.*, vol. 76, pp. 428–442, 2017.

[25] W. Xue, W. Li, H. Qi, K. Li, X. Tao, and X. Ji, "Communication-aware virtual machine migration in cloud data centres," *Int. J. High Perform. Comput. Netw.*, vol. 10, no. 4/5, pp. 372–380, 2017.

[26] Y. Cui, Z. Yang, S. Xiao, X. Wang, and S. Yan, "Traffic-aware virtual machine migration in topology-adaptive DCN," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3427–3440, Dec. 2017.

[27] H. Flores, V. Tran, and B. Tang, "Pam & pal: Policy-aware virtual machine migration and placement in dynamic cloud data centers," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 2549–2558.

[28] D. Saxena, A. K. Singh, and R. Buyya, "OP-MLB: An online VM prediction based multi-objective load balancing framework for resource management at cloud datacenter," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2021.3059096.

[29] B. Li, B. Cheng, X. Liu, M. Wang, Y. Yue, and J. Chen, "Joint resource optimization and delay-aware virtual network function migration in data center networks," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 3, pp. 2960–2974, Sep. 2021.

[30] H. Nashaat, N. Ashry, and R. Rizk, "Smart elastic scheduling algorithm for virtual machine migration in cloud computing," *J. Supercomput.*, vol. 75, no. 7, pp. 3842–3865, 2019.

[31] A. Jajoo, Y. C. Hu, and X. Lin, "Your coflow has many flows: Sampling them for fun and speed," in *Proc. USENIX Annu. Tech. Conf.*, 2019, pp. 833–848.

[32] A. Jajoo, Y. C. Hu, X. Lin, and N. Deng, "A case for task sampling based learning for cluster job scheduling," in *Proc. 19th USENIX Symp. Netw. Syst. Des. Implementation*, 2022, pp. 19–33.

[33] A. Jajoo, Y. C. Hu, and X. Lin, "A case for sampling based learning techniques in coflow scheduling," *IEEE/ACM Trans. Netw.*, to be published, doi: 10.1109/TNET.2021.3138923.

[34] J. Hao, K. Ye, and C.-Z. Xu, "Live migration of virtual machines in openstack: A perspective from reliability evaluation," in *Proc. Int. Conf. Cloud Comput.*, 2019, pp. 99–113.

[35] R. Zolfaghari and A. M. Rahmani, "Virtual machine consolidation in cloud computing systems: Challenges and future trends," *Wireless Pers. Commun.*, vol. 115, no. 3, pp. 2289–2326, 2020.

[36] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: Universal topology generation from a user's perspective," Boston University Computer Science Department, Tech. Rep. BUCS-TR-2001-003, 2001.

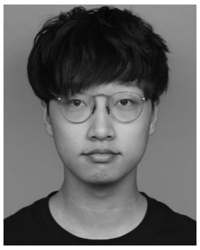
[37] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[38] S. M. Blackburn *et al.*, "The dacapo benchmarks: Java benchmarking development and analysis," in *Proc. 21st Annu. ACM SIGPLAN Conf. Object-Oriented Program. Syst. Lang. Appl.*, 2006, pp. 169–190.

[39] L. Wang, W. Mao, J. Zhao, and Y. Xu, "DDQP: A double deep q-learning approach to online fault-tolerant SFC placement," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 1, pp. 118–132, Mar. 2021.

Desheng Wang received the BEng degree in computer science and engineering from Harbin Engineering University, Harbin, China, in 2015, and the PhD degree in cyberspace security from the Harbin Institute of Technology, Harbin, China, in 2022. His research interests include virtualization techniques for cloud-edge computing and machine learning.

Weizhe Zhang (Senior Member, IEEE) received the BEng, MEng and PhD degrees all in computer science and technology from the Harbin Institute of Technology, China, in 1999, 2001 and 2006, respectively. He is currently a professor with the School of Computer Science and Technology, Harbin Institute of Technology, China, and the director with the Cyberspace Security Research Center, Peng Cheng Laboratory, Shenzhen, China. His research interests are primarily in parallel computing, distributed computing, cloud and grid computing, and computer networks.



Xiao Han received the BEng degree in information security from the Harbin Institute of Technology, China, where he is currently working toward the MEng degree with the School of Cyberspace Science. His research interests include cloud computing and edge computing.

Junren Lin received the BEng degree in information security from the Harbin Institute of Technology, China, where he is currently working toward the MEng degree with the School of Cyberspace Science. His research interests include edge computing and machine learning.

Yu-Chu Tian (Senior Member, IEEE) received the PhD degree in computer and software engineering from the University of Sydney, Sydney NSW, Australia, in 2009, and the PhD degree in industrial automation from Zhejiang University, Hangzhou, China, in 1993. He is currently a professor with the School of Computer Science, Queensland University of Technology, Brisbane QLD, Australia. His research interests include Big Data computing, cloud computing, computer networks, optimization and machine learning, networked control systems, and cyber-physical system security.