Computer Networks 177 (2020) 107230

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

MARVEL: Enabling controller load balancing in software-defined networks with multi-agent reinforcement learning

Penghao Sun[a], Zehua Guo[b,*], Gang Wang[b,c], Julong Lan[a], Yuxiang Hu[a]

[a] National Digital Switching System Engineering and Technological Research Center, Zhengzhou 450002, China
[b] Beijing Institute of Technology, 5 Zhongguancun ST South, 100081, Beijing, China
[c] University of Minnesota Twin Cities, 117 Pleasant ST, 55455, Minneapolis, USA

ARTICLE INFO

Index Terms:
Multi-agent reinforcement learning
Neural networks
Software-defined networking
Switch migration

ABSTRACT

The control plane plays a significant role in Software-Defined Networking (SDN). A large SDN usually implements its control plane with several distributed controllers, each controlling a subset of switches and synchronizing with other controllers to maintain a consistent network view. Under the fluctuating network traffic, a static controller-switch mapping relationship could lead to imbalanced workload allocation. Controllers may getoverloaded and reject new requests, eventually reducing the control plane's request processing ability. Most existing schemes have relied heavily on iterative optimization algorithms to manipulate the mapping relationship between controllers and switches, which are either time-consuming or less satisfactory in terms of performance. In this paper, we propose a dynamic controller workload balancing scheme, that is termed MARVEL, based on multi-agent reinforcement learning for generation of switch migration actions. MARVEL works in two phases: offline training and online decision making. In the training phase, each agent learns how to migrate switches through interacting with the network. In the online phase, MARVEL is deployed to make decisions on migrating switches. Experimental results show that MARVEL outperforms competing existing schemes by improving the control plane's request processing ability at least 27.3% while using 25% less processing time.

1. Introduction

Due to the powerful programmability and flexible management on networks, Software-Defined Networking (SDN) [1] has attracted interests from both academia and industry. SDN decouples the control plane from the data plane, and the network operator can monitor and operate the network conveniently through a logically centralized controller with a global network view. To address the limited processing ability and single node failure of the single controller, several works have proposed to realize the control plane using multiple distributed controllers [2–4]. In the multi-controller control plane, each controller manages a subset of switches and synchronizes with other controllers to maintain a consistent network view.

The flow requests from switches may change in practice [6]. However, their static switch-controller relationship cannot accommodate to the dynamic traffic changes. Thus, the processing workload could be distributed among controllers in an unbalanced way, and some highly-loaded controllers are prone to be fully loaded and reject new requests, eventually reducing the control plane's request processing ability. To solve the problem, Dixit et al. [6] developed an elastic distributed controller architecture named ElastiCon that aims to balance the processing workload of controllers by dynamically establishing the mapping between switches and controllers. Also, OpenFlow v1.4 [7] enabled the distributed control plane by presenting a coordination mechanism among multiple controllers. Nonetheless, both ElastiCon and OpenFlow do not provide details in the problem of switch migration on how to migrate switches. The Switch Migration Problem (SMP) is usually formulated as an optimization problem [8,9], whose complexity is proved to be NP-hard. Solving the SMP either takes a long time to obtain the optimal result, which may not be acceptable under a dynamic traffic distribution, or produces heuristic solutions not good enough in migration performance.

Several attempts have been made to address this challenge, but they also come with limitations. Wang et al. [10,11] focused on realizing load balancing of the control plane for data center networks and neglected the communication cost of switch migration. Huang et al. [12] introduced a middle layer between the control plane and the data plane to distribute flow requests to multiple controllers, but this scheme may significantly increase the processing delay due to insertion of the middle layer. BalCon [13] addressed the SMP with a graph partition method, yet at the price of a complicated processing procedure, as it analyzes the communication pattern of all switches in the

* Corresponding author.
E-mail address: guo@bit.edu.cn (Z. Guo).

https://doi.org/10.1016/j.comnet.2020.107230
Received 21 November 2019; Received in revised form 26 February 2020; Accepted 21 March 2020
Available online 26 April 2020

control region of an overloaded controller to carry out one step of migration.

In this paper, building on recent advances in artificial intelligence, we propose a dynamic controller workload balancing scheme, that we term MARVEL based on Multi-Agent Reinforcement Learning (MARL). MARL is a distributed version of reinforcement learning (RL) and excels at generating dynamic control actions in distributed systems. The distributed processing nature of MARL makes it appealing for interactions and decision making of the distributed control plane in SDN. After a proper training phase, the MARL agents residing in controllers can make quick decisions on the control strategy of SMP.
The main contributions of this paper are summarized as follows:

1. We model the distributed control plane as a multi-agent system with the purpose of tackling the SMP in a distributed fashion. We design a zero-sum game mechanism for the multi-agent system model to reach an equilibrium as the ending signal of an SMP process.
2. We design a Deep Reinforcement Learning (DRL) framework for each agent in the MARL model. The DRL-based solution takes the workload pattern in the control plane as input and generates the migration decision as the output. After the training phase, the DRL agent can quickly and accurately decide how to migrate switches among the controllers.
3. We evaluate the performance of MARVEL with convincing simulations, and the results show that MARVEL improves the control plane request processing ability by at least 27.3% while reducing the processing time by about 25%.

The remainder of this paper is organized as follows. Section 2 introduces the background and motivation of this paper. Section 3 provides a brief overview of MARVEL. Section 4 formulates the resource utilization problem of the control plane. Section 5 illustrates the framework of MARVEL and elaborates on its training process, working procedure, and practical implementation details. Section 6 compares the performance of MARVEL with existing schemes. Related works are discussed in Section 7 with conclusions drawn in Section 8.

2. Background and motivation

In this section, we introduce the background of the switch migration problem, and illustrate scenarios where a switch migration is required.

2.1. Background

In an SDN, the control plane is in charge of calculating the network routing policies and deploying the policy into the network by installing/deleting/updating flow entries in related switches. Because of the decoupled control plane and data plane, the control plane can acquire a global view of the network and use the view to implement diverse manipulations on network operations. Typically, a controller has many functions, such as traffic prediction [14], and abnormal traffic detection [15].

Given the limited processing ability, a single controller is likely to suffer from resource exhaustion. Reference [16] shows that a single controller can only manage a limited flow arrival rate in proportion to its resources. To address the scalability issue, a distributed control plane is proposed, which uses multiple controllers to perform the control logic. In a distributed control plane, each controller is in charge of multiple switches in a domain by fixed mapping these switches on the controller. The fluctuation of network traffic brings different burdens to different controllers at different times, and a static mapping between switches and controllers leads to insufficient utilization of control resources. In this context, we are prompted to develop an adaptive mapping scheme between the control plane and the data plane based on the real-time network variations. The aim of this adaptive mapping is to handle more network requests with available network resources.

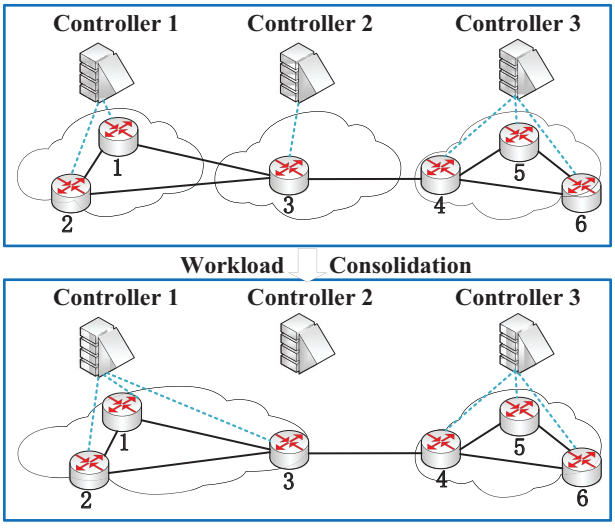

Fig. 1. An example of the workload consolidation.

2.2. Motivation

Typically, we can categorize the SMP into two cases: workload consolidation and workload balancing. First, unbalanced network traffic allocation can lead to the idling in some controllers, which causes a waste of power. In this case, under an acceptable migration cost (e.g., the communication delay between a switch and a controller), previous works have proposed to design a strategy to consolidate the workload of lowly utilized controllers into several other controllers through which the power consumption of idle controllers can be saved [19,20]. Moreover, they also assume that the controllers can maintain consistent view of the network. Fig. 1 shows an example of workload balancing, in which there are three controllers in the control plane and six switches in the data plane. Each switch belongs to a control region managed by a controller. Controller 2′s load decreases since its controlled Switch 3′s load reduces. At the same time, its neighbor Controller 1 has the capability to take over the work of Controller 2. Under this situation, Switch 3 can be migrated to Controller 1 from Controller 2. We call this switch migration process a consolidation operation.

Subsequently, under the fluctuation of spatial-temporal distribution in network traffic, some switches may reach their peak workload, and this may consume most controller resources to handle the requests of such switches. In the worst case, the workload may exceed the capacity of this controller. Therefore, this calls for a strategy to dynamically distribute the workload in a balanced way to each controller to minimize the probability of overloading [13,21]. The overloaded controller should detach some switches from its controlled region to light-loaded controllers. For example, in Fig. 2, if the workload of Controller 1 reaches a certain threshold, it can release some of its burden to Controller 2, and Switch 4 may be picked for the migration. We call this scenario the overload prevention.

2.3. Challenges and opportunities

The two scenarios described above constitute the main background and motivation of studying SMP. However, when and where to carry out the migration is difficult to decide in practice. For example, Cello et al. [13] showed that the controller load balancing problem is NP-Complete. Traditional solutions to the NP-Complete problem employ heuristics to reduce the computation cost, which usually cannot reach a good performance. On the other hand, development of artificial intelligence in recent years has brought us to a set of new tools to tackle complex problems. As a kind of RL, DRL has gained a wide attention since its proposal

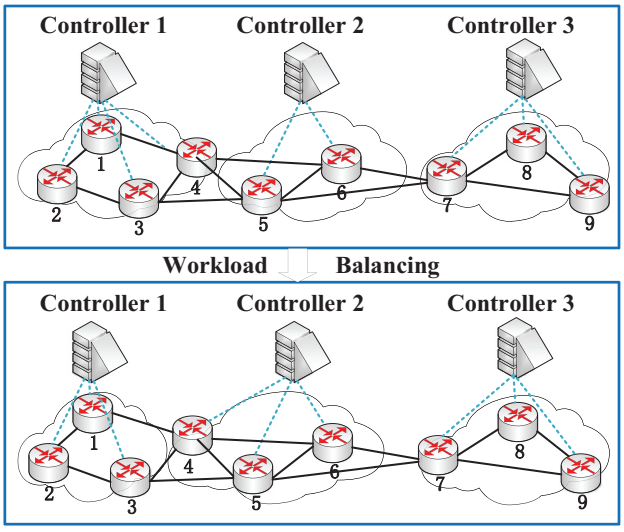

Fig. 2. An example of workload balancing to prevent overloading controllers.

[24]. DRL combines advantages of both Deep Neural Networks (DNN) and RL, and it is capable of dealing with huge input state spaces and efficiently generating a control action for the target system. Compared to supervised learning technologies such as deep learning, DRL can evolve based on interactions with the training environment without a large labeled data set. DRL has proven powerful in diverse fields, such as game playing [24], robot control [32], resource allocation in big data systems [33], and routing [34,35]. Also, a single-agent version of DRL is based on a centralized control logic, which is insufficient for distributed control in SDN. In this paper, we make the first attempt to solve the NP-Complete SMP in a distributed fashion using MARL.

3. Overview of MARVEL

This section starts with an overview of our design to handle the SMP. The system design of MARVEL is shown in Fig. 3. We model the control plane as a multi-agent system and invoke a multi-agent reinforcement

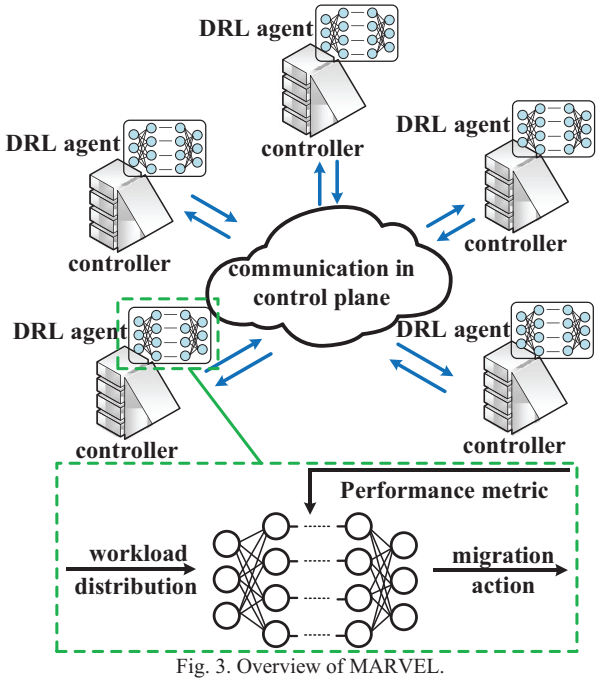

Fig. 3. Overview of MARVEL.

Fig. 3. Overview of MARVEL.

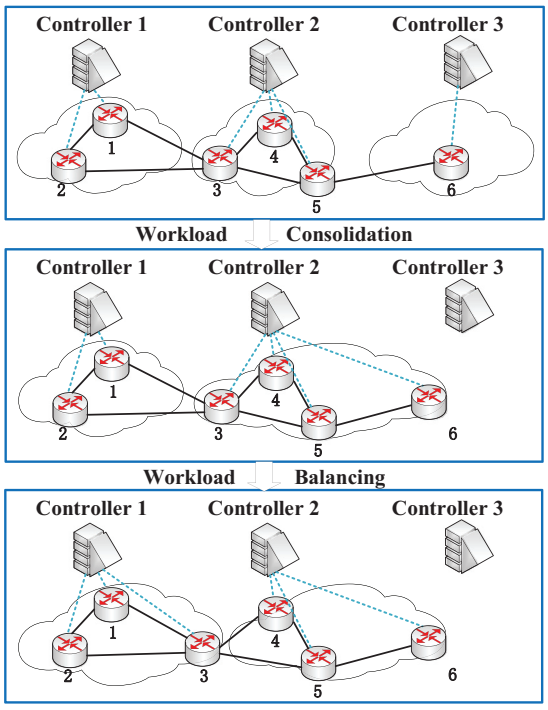

Fig. 4. An example of the MARVEL's working process.

learning model to solve the SMP in the control plane. In the distributed control plane, each controller is regarded as a DRL agent. All DRL agents constitute a multi-agent system. Each DRL agent uses its workload information in real time while keeping exchanging its switch migration intent with others agents. After a comparison of all intents in the agents, the multi-agent reinforcement system finally decides how a switch migration should take place.

To reach the migration goal described in Section 2.2, there are mainly two types of operations: workload consolidation and workload balancing, as illustrated in Fig. 4.

First, we evaluate the workload in the control plane. If there is a controller with a workload lower than the threshold, we consider migrating its workload to other controllers if there is enough capacity on the remaining controllers. For the example of Fig. 4, Controller 3 has just Switch 6 in charge, which can be migrated to other controllers. Considering the communication cost from Switch 6 to other controllers, Switch 6 is migrated to Controller 2.

Then, among the active controllers, we migrate switches to balance the workload distribution among them based on their available capacity in real time. To achieve a balanced workload distribution, we model the resource utilization in controllers as an optimization problem, which will be explained in Section 4. After the consolidation process, we can find that the workload between Controller 1 and Controller 2 is unbalanced in Fig. 4. Therefore, we then migrate Switch 3 from Controller 2 to Controller 1 to balance their workload. The details of the algorithms used in these two phases will be illustrated in Section 5.

4. Control plane workload balancing problem

In this section, we discuss the workload balancing problem in the control plane, and proposed a mathematical model to solve this problem.

4.1. Control plane resource utilization modeling

To reach a balanced workload among the working controllers, we design a resource utilization model to measure the requesting process-

ing ability of the control plane, since under limited resources a higher resource utilization results in a better request processing ability. The basic idea is to maximize the request processing ability of the control plane with limited resources as the requests are unbalanced distributed. In this model, we take three types of resources into consideration: bandwidth, CPU, and memory usage. Cost of such resources by different events may be different. For example, in a wide area network (WAN), the switches can be categorized into core switches and edge switches, and different switches have different patterns of events [19]. For example, some switches may produce many Packet_in (defined in OpenFlow) messages for flows with special routing request (e.g., QoS routing), so controllers corresponding to such switches should conduct more CPU-consuming operations such as routing calculations, and other controllers may not be tasked with many CPU-consuming operations, leading to different workload on controllers.

We set up our resource utilization model on such basic ideas: the control plane should try to handle all the requests from the data plane under fixed resources in each controller. Consider a distributed control plane with N controllers and M switches. The set of controllers is denoted as $C = \{c_1, c_2, ...c_N\}$ and the set of switches is denoted as $S = \{s_1, s_2, ...s_M\}$.x_i, y_i and z_i represent the usage of bandwidth, CPU, and memory per event in switch s_i respectively where $1 \le i \le M$, and X_j, Y_j and Z_j represent the bandwidth, CPU, and memory capacity in controller c_j, respectively, where $1 \le j \le N$. Let w_i be the number of event processing requests generated by switch s_i counted by the event number. Let the binary indicator $\delta_{i,j}$ to denote whether switch i is assigned to switch j and U_j denote the resource utilization in controller c_j. Following [19], we consider the following utility as a linear weighted sum of different resource

$$U_j = \frac{d_j^X}{X_j}\sum_{i=1}^{M}\delta_{i,j}w_i x_i + \frac{d_j^Y}{Y_j}\sum_{i=1}^{M}\delta_{i,j}w_i y_i + \frac{d_j^Z}{Z_j}\sum_{i=1}^{M}\delta_{i,j}w_i z_i \quad (1)$$

where d_j^X, d_j^Y, d_j^Z are the weights assigned for different types of resources, and $d_j^X + d_j^Y + d_j^Z = 1$. If we denote $U_j^X = \frac{\sum_{i=1}^{M}\delta_{i,j}w_i x_i}{X_j}$, (1) can be written as follows

$$U_j = d_j^X U_j^X + d_j^Y U_j^Y + d_j^Z U_j^Z \quad (2)$$

Note that when a controller's processing request exceeds its processing capability, this controller will be viewed as overloaded.

4.2. Control plane load balancing problem formulation

Our objective is to avoid hot spots among the controllers in use when the overall resources are adequate, i.e., to minimize $\max_{1\le j\le N} U_j$. Given that each controller may have a different capacity, the balance of workload in our model refers to a balanced resource utilization among the controllers. We can formulate the control plane balancing problem as follows

$$\min_{\delta}\ \max_{1\le j\le N} U_j \quad (3)$$

$$\text{s.t.}\ \sum_{i=1}^{M}\delta_{i,j}w_i x_i \le X_j, \forall j \quad (4)$$

$$\sum_{i=1}^{M}\delta_{i,j}w_i y_i \le Y_j, \forall j \quad (5)$$

$$\sum_{i=1}^{M}\delta_{i,j}w_i z_i \le Z_j, \forall j \quad (6)$$

$$\sum_{j=1}^{N}\delta_{i,j} = 1, \forall i \quad (7)$$

$$U_j = \frac{d_j^X}{X_j}\sum_{i=1}^{M}\delta_{i,j}w_i x_i + \frac{d_j^Y}{Y_j}\sum_{i=1}^{M}\delta_{i,j}w_i y_i + \frac{d_j^Z}{Z_j}\sum_{i=1}^{M}\delta_{i,j}w_i z_i$$

$$\delta_{i,j} \in \{0,1\}, \forall i \in [1,M], \forall j \in [1,N] \quad (8)$$

where X_j, Y_j, and Z_j are positive constants depending on the physical resources of controllers; and w_i, x_i, y_i, and z_i are positive constants determined by the network traffic. To optimize the objective function of (3), a possible solution is to adjust the mapping of switches to controllers in the network topology. Constraints (4–6) require that the resource utilization of each type should be within a certain physical capability, (7) ensures that one switch must only be assigned to only one controller at the same time, and (8) defines U_j.

4.3. Complexity analysis

The problem defined in (3) aims to allocate M switches to N controllers with a min-max load balancing such that the maximum load over all controllers isminimized. Such scheduling problem is a typical multi-dimensional vector scheduling problem, which is generally NP-Complete [36]. Solutions to NP-Complete problems are usually heuristic, which in general do not come with performance guarantees. Furthermore, many centralized algorithms that place the migration calculation on merely a single controller put too much calculation burden on the certain controller, thus a distributed manner is required to better address this problem. To our best knowledge, existing schemes are based on traditional optimization algorithms to solve the NP-hard problem of SMP, which are either time-consuming or not satisfactory in terms of performance. In this paper, we solve the problem in a distributed manner using MARL, which will be discussed in detail in the following section.

5. MARVEL system implementation

In this section, we illustrate our scheme to solve the workload balancing problem defined in Section 3. First, we introduce the model of MARL used in MARVEL. Then, we elaborate on how the MARL model is trained and works. Finally, we provide the implementation details of each agent in our MARL model.

5.1. Framework of MARVEL

RL algorithms learn by interacting with an unknown environment (e.g., the communication network). At each time step t, the algorithm (also called an agent) observes the state s of the environment, based on which it selects an action a. The action a is used to adjust the behavior of the network (e.g., adjust the mapping relationship between controllers and switches). The influence of the action on the environment is translated into a reward r revealed to the agent, based on which the agent can further improve the action generation policy. In MARL, each agent interacts with a Markov Decision Process (MDP), which is modeled as $M = (S, A, R, P, \gamma)$, where S is the space of state s, A is the space of action a, R is the space of reward r, P is the transition probability function $P(s_{t+1}, r|s_t, a_t)$, and $\gamma \in [0, 1)$ is the discount factor. Each agent aims to find an action policy under a certain environment state (e.g., a transition function T: $S \times A \times S \to [0, 1]$) to maximize the expected discounted sum $E(\sum_{k=0}^{\infty}\gamma^k r_i^{t+k}|s_t = s)$, where k denotes the action step. In MARVEL, the environment is the SDN, and the definition of state, action, reward will be illustrated in Section 5.4.

The form of discounted sum is widely adopted in RL because it can cater to both the overall performance of a strategy and the short-term profit. In the MARL model, each agent maintains an individual policy π_i: $S \times A_i \to [0, 1]$, and under such policy, an agent can have a state-value function to evaluate the quality of this policy given a certain state

$$V_i^{\pi}(s) = E_{\pi}\left(\sum_{k=0}^{\infty}\gamma^k r_i^{t+k}|s_t = s\right) \quad (9)$$

Since the policy π_i is actually an action choice function, the policy quality is usually denoted as $Q_i^{\pi}(s, a_i)$, which is defined as

$$Q_i^{\pi}(s, a_i) = E_{\pi}\left(\sum_{k=0}^{\infty}\gamma^k r_i^{t+k}|s_t = s, a_t = a\right) \quad (10)$$

This form of quality function evaluates the action value taken by a certain policy at a certain state. Compared to formula (9), formula (10) is more concrete in the measure of a policy.

However, the acquisition of future reward (from $k = 0$ to $k = \infty$) is not applicable in online learning. To solve this problem, the Q value can also be expressed in an iterative mode:

$$Q_i(s_t, a_{i,t}) = E\left[r_t, \gamma Q_i(s_{t+1}, a_{i,t+1})\right] \tag{11}$$

The quality function Q can be saved in a table if there are limited combinations of state s and action a. However, if there are too many (s, a), or even in a continuous environment where there are innumerable (s, a), such a tabular method will fail. In SMP, the network state is just beyond a tabular representation. Under such circumstances, we can represent the Q-value with a function approximator instead of storing all Q values in a table. In this paper, we use a neural network as the function approximator. Specifically, Q is specified as $Q(s, a|\theta)$, where θ is the set of parameters of the neural network. For optimization of the neural network, we need a loss function to carry out the backpropagation. As defined in [24], the loss function can be e.g., the following

$$L(\theta) = E\left[(Q(s_t, a_t|\theta) - y_t)^2\right] \tag{12}$$

of which y_t is defined as

$$y_t = \gamma \max_{a'} Q(s_{t+1}, a'|\theta) \tag{13}$$

The update of the neural network is based on backpropagation with the loss defined in (12).

5.2. Training phase of MARVEL

In this section, we discuss how the agents are trained in a MARL model and present the detailed interaction among agents of multi-agent reinforcement learning. The goal of the training process is to train each controller whether to export a switch to other controllers or import one from other controllers without human experience. The process is shown in Algorithm I. For each DRL agent, the state, action and reward are illustrated in detail in Section 5.4 Algorithm I mainly works as an iteration, through which the DRL agents get trained. Line 1 decides the iterations of the whole training process. In lines 2–7, each controller calculates the resource utilization of all controllers in use and pick up a controller as the master. In line 8–16, in each iteration one controller is selected by the master as an actor (lines 9–11), and then the actor controller generates a switch migration action (lines 13–16). Since the actor controller is selected according to a probability based on the resource utilization (line 10), as the training goes on, a more balanced resource utilization will lead to a more balanced chance for each controller to be selected as an actor. Therefore, each actor can be guaranteed with enough training probability.

Algorithm I
MARVEL training process.

Input: X_j, Y_j, Z_j and $\sum_{s_i \in S_{C_j}} w_i x_i$, $\sum_{s_i \in S_{C_j}} w_i y_i$, $\sum_{s_i \in S_{C_j}} w_i z_i$, $(1 \le j \le N)$;
Output: switch migration decision and trained parameters of DRL;
1: **for** epi_num←1 to EPISODE:
2: **for** each controller c_j:
3: collect X_j, Y_j, Z_j and $\sum_{s_i \in S_{C_j}} w_i x_i$, $\sum_{s_i \in S_{C_j}} w_i y_i$, $\sum_{s_i \in S_{C_j}} w_i z_i$;
4: calculate U_j;
5: **if** this controller c_{j_1} satisfies $c_{j_1} = \arg\max_{c_j \in C} U_j$:
6: c_{j_1} serves as the master;
7: set other controllers as followers;
8: **for** step_num←1 to STEP:
9: **if** this controller is a master:
10: pick controller C_x with probability $\frac{U_x}{\sum_{c_j} U_j}$ as an actor;
11: inform C_x.;
12: **if** this controller is an actor:
13: input state to the MARVEL agent, get action;
14: broadcast action to other controllers;
15: calculate reward and update the MARVEL agent;
16: **else:**
17: wait for action from the actor;
16: update $\sum_{s_i \in S_{C_j}} w_i x_i$, $\sum_{s_i \in S_{C_j}} w_i y_i$, $\sum_{s_i \in S_{C_j}} w_i z_i$ according to action;
17: save the trained parameters in MARVEL agents;
18: **End**

5.3. Working phase of the MARVEL

In this section, we discuss how the trained MARL model works. To apply the MARL model in the resource utilization problem in the control plane, we must first design a game model so the agents can know how to carry out the switch migration among each other and when to stop the migration. The goal of the game model is to reach an equilibrium that stands for optimal resource allocation in the control plane, and during this process a MARL agent is implemented in each controller to calculate the advantage of each step in the game playing.

We can map the SMP to the commodity transaction model. In our design, switches and controllers are treated as commodities and players in a market, respectively. Switches are traded among controllers, during which process each controller tries to maximize its profit. Generally, a switch could only be traded to the player near to its temporary owner, which can be realized by the punishment of communication delay (e.g., it may introduce much delay for controllers to control a switch that is far away in the network). Also, a switch migration process takes place only under certain circumstances, i.e., a workload lower than a threshold or a workload of a switch is higher than a threshold.

Definition 1. Game Engagement (GE): We call a switch transaction process a Game Engagement (GE).

To ensure the state consistency of the controller, one controller can only engage in one GE at the same time. This requirement simplifies the MARL training process since there is no need to consider the confliction of two switch migration action.

As mentioned in the above, the goal of game playing is to find an equilibrium that stands for an optimal workload allocation scenario that can maximize the utilization of controlling resources. Suppose that in a GE, an action tuple $P = (p_1, p_2, \ldots p_n)$ is the joint action of controllers in $C = \{c_1, c_2, \ldots c_n\}$, where p_i stands for the action policy of controller c_i.

Definition 2. (MARVEL equilibrium) In a GE, an action tuple $p* = (p_1 *, p_2 *, \ldots p_n *)$ is a Nash equilibrium if for any $p_i' \neq p_i$ $*\rightarrow p' = (p_1 *, p_2 *, \ldots p_i', \ldots p_n *)$, we have $\sum_{c_j} U_j(p*) \ge \sum_{c_j} U_j(p')$. We name this equilibrium in this paper as an MARVEL equilibrium.

The working procedure contains two main stages: consolidation decision and balancing decision, as shown in Fig. 5. In the MARL model, each controller continuously monitors the network status, and then decides whether a consolidation decision and a balancing decision is required.

The process of consolidation detection in a controller is shown in Algorithm II. Since the process is carried out in a parallel way that each controller executes its independent logic, the algorithm of Algorithm II should be run at each controller. A consolidation actually takes place only when the workload of a certain controller can be migrated to other controllers (e.g., workload below a certain level) and at the same time, there is another controller that can take the workload of this controller. Therefore, a consolidation does not always take place. In lines 1–3 of Algorithm II, each controller collects the resource utilization of all other controllers, lines 5–6 select the controller with the minimum workload for consolidation if its workload is below a certain threshold, lines 7–8 decide if there is another controller that can adopt the workload, and line 9 executes the migration of switches.

When a trial of the consolidation process is finished, the balancing process follows. During the gaming process, all agents work together to reach an equilibrium, which is shown in Algorithm III. In Algorithm III, lines 1–5 start a GE, and lines 6–13 operate the iteration of GE toward a balanced workload.

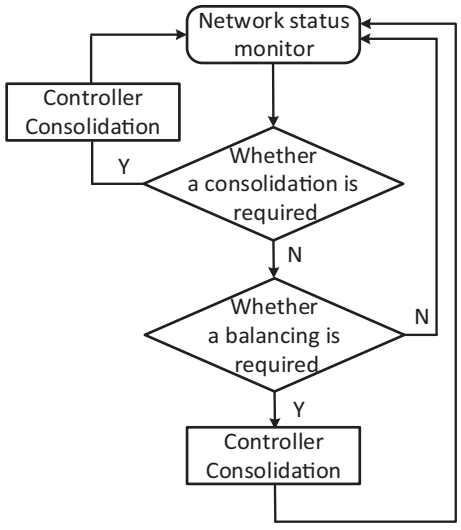

Fig. 5. Working phase of MARVEL.

Algorithm II
Switch migration based on workload consolidation.

Input: X_j, Y_j, Z_j and $\sum_{s_i \in S_{c_j}} w_i x_i$, $\sum_{s_i \in S_{c_j}} w_i y_i$, $\sum_{s_i \in S_{c_j}} w_i z_i$, $(1 \le j \le N)$
Output: switch migration decision
1: **Initialization:**
2: **for** controller c_j $(1 \le j \le N)$:
3: calculate U_j;
4: **Consolidation:**
5: **if** this controller c_{j_1} satisfies $c_{j_1} = \arg\min_{c_j \in C} U_j$:
6: **if** $U_{j_1} \le THRESHOLD\_LOW$:
7: find $c_{j_2} = \arg\min_{c_j \in C_{-j_1}} U_j$;
8: **when** constraints $\{\sum_{s_i \in S_{c_{j_1}}} w_i x_i + \sum_{s_i \in S_{c_{j_2}}} w_i x_i \le X_{j_2}$, $\sum_{s_i \in S_{c_{j_1}}} w_i y_i + \sum_{s_i \in S_{c_{j_2}}} w_i y_i \le Y_{j_2}$, $\sum_{s_i \in S_{c_{j_1}}} w_i z_i + \sum_{s_i \in S_{c_{j_2}}} w_i z_i \le Z_{j_2}$ are guaranteed:
9: and migrate $S_{c_{j_1}}$ to c_{j_2};
10: **End**

Algorithm III
Switch migration based on load balancing among controllers.

Input: X_j, Y_j, Z_j and $\sum_{s_i \in S_{c_j}} w_i x_i$, $\sum_{s_i \in S_{c_j}} w_i y_i$, $\sum_{s_i \in S_{c_j}} w_i z_i$, $(1 \le j \le N)$
Output: switch migration decision
1: check GE requests from other controllers;
2: **if** this controller follows $U \ge THRESHOLD\_HIGH$:
3: send signal to other controllers for a GE;
4: **if** a GE is required:
5: set EQUILIBRIUM to 0;
6: **while** not EQUILIBRIUM:
7: **for** each controller c_j:
8: collect X_j, Y_j, Z_j and $\sum_{s_i \in S_{c_j}} w_i x_i$, $\sum_{s_i \in S_{c_j}} w_i y_i$, $\sum_{s_i \in S_{c_j}} w_i z_i$;
9: input state to the MARVEL agent, broadcast action and reward;
10: wait for action and reward from other controllers;
11: execute the action with the maximum reward;
12: **if** the selected action moves no switch:
13: EQUILIBRIUM = 1;
14: **End**

5.4. Details of MARVEL model implementation

In this section, the detail of the MARL model implementation is discussed, which mainly includes the implementation details of the neural network in the agents of MARL.

As illustrated in Section 4.1, the agent is implemented with DRL. In our scheme, the neural network in DRL is implemented with an RNN (Recurrent Neural Network) [25]. The structure of the neural network is shown in Fig. 6. RNN is designed to process sequence data and has now various popular versions such as LSTM (long short-term memory) and GRU (Gated Recurrent Unit) [26]. After each time step, the output data also serve as the inputs in the next time step. Then, the output of all RNN units is connected to a feedforward neural network, which produces the final outcome. In the multi-agent reinforcement learning model, the network state is the resource utilization of each controller, which is normalized and sent to the RNN as the input data. Therefore, the number of input layer neurons corresponds to the number of controllers. The output layer has three neurons, which correspond to three types of actions, namely, import, stay still, and export.

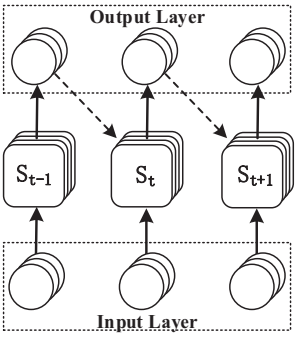

Fig. 6. Neural network structure in the DRL agent of MARVEL.

If the action is stay still, there is no switch migration. If the action is import, then the agent picks the controller with the maximum resource utilization U from whom to import a switch. If the action is export, the agent picks the controller with the minimum resource utilization U to whom to export a switch. The choosing of a certain switch relies on the calculation of resource utilization distribution, which is based on the principle that the chosen switch should compensate the resource utilization gap between the exporting controller and importing controller as much as possible. For example, if controller c_i should export a switch to controller c_j, then the switch to be migrated could be

$$s = \arg\min_{s_h \in S_{c_i}} \left\{ d_j^x \left[X_i (U_i^X - U_j^X) - w_h x_h \right]^2 + d_j^y \left[Y_i (U_i^Y - U_j^Y) - w_h y_h \right]^2 + d_j^z \left[Z_i (U_i^Z - U_j^Z) - w_h z_h \right]^2 \right\} \quad (14)$$

The reward is computed mainly based on the improvement of resource utilization balancing, while the change of communication delay from the migrated switch to different controllers can also be taken into consideration. Suppose that the resource utilization before and after switch migration is U_j , U_j' for controller c_j and U_i, U_i' for controller c_i respectively, and the extra communication delay caused by the migration operation can be denoted with μ_{ji}. With μ_{ji}, the unexpected controlling over distant switches can be avoided. Therefore, even though we do not explicitly impose a constraint in the problem formulation that the switches controlled by one controller should form a connected subgraph, the controlling relationship between a controller and the switches can still cater to the practical need. Specifically, the reward value can be expressed as $reward = (U_i - U_j)^2 - (U_i' - U_j')^2 - \mu_{ji}$.

6. Simulation and performance evaluation

In this section, we introduce our simulation environment to test MARVEL and compare the performance of MARVEL with other state-of-the-art schemes.

6.1. Simulation setup

The performance of the proposed model is simulated with six controller instances [9] on a host machine with Intel E5-2600 processor,

32GB DDR4 memory and two GTX1080Ti graphics cards. The DRL agent of MARVEL is implemented with Keras based on TensorFlow. The statistical characteristics of the request follow a Poisson process added to a periodical fluctuation [9,13,34] which can ensure both the adaption to randomness and the reproducibility of performance of the proposed scheme.

6.2. Comparison schemes

To evaluate the performance of MARVEL, we make comparisons among the following schemes:

- **DHA-LB** [9]: DHA-LB uses a distributed hopping algorithm to migrate a switch among controllers. In DHA-LB, there is a random selection of switch, and then the selected switch continuously chooses a controller at random until the chosen controller satisfies the resource utilization constraints.
- **BalCon** [13]: BalCon analyzes the switch behavior and uses a clustering algorithm to migrate a cluster of switches from the overloaded controller to another controller.
- **MARVEL (this paper)**: MARVEL runs an DRL agent at each controller. Each DRL agent can make their own decision of switch migration until an equilibrium is reached.

6.3. Simulation results

In this section, we show and compare the performance of MARVEL in the following aspects: effectiveness of MARVEL, comparison of flow request processing capability, improvement of resource utilization, and resource consumption of different schemes.

6.3.1. Effectiveness of MARVEL

Fig. 7 shows the original workload distribution over the six controllers within 200 mins. Each controller is denoted by a line with a certain color, and the workload is shown in the form of the percentage of the maximum capacity of a controller. As shown in the figure, the workload distribution on controllers varies with time, and different controller meets their peak workload value at different times. When the workload of a controller exceeds 100% (such as the red line), such controller is overloaded, we discard the requests that exceed the processing ability of such controller. In practice, there should always be some resource reserved for other incidents and affairs (e.g., when the dynamic switch migration process also requires some calculation and communication resource), thus the ideal workload should not exceed a certain threshold, which will also serve as a trigger for a balancing process as mentioned in Algorithm III.

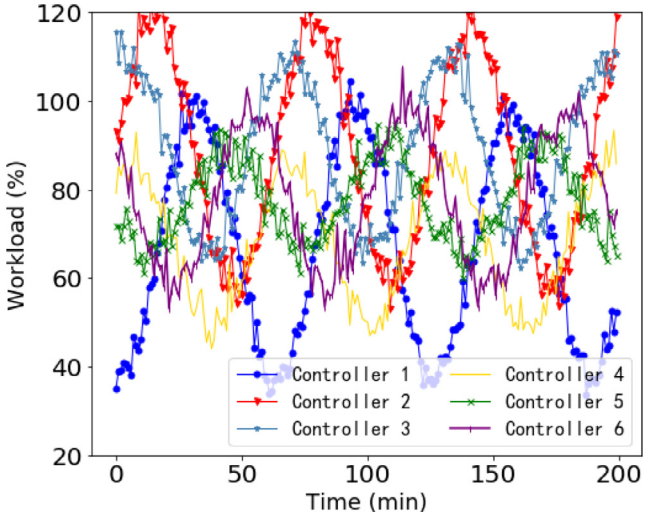

Fig. 7. Original distribution of workload in each controller (Each color denotes one controller).

Fig. 8 shows the workload distribution in MARVEL under the same request pattern from switches within 200 mins. Comparing Figs. 7 and 8 we can see that when the DRL agents of MARVEL start to work, there is an apparent improvement on the balance of workload distribution. First, it is clear that there is no overloaded controller in MARVEL. The improvement of workload distribution can reduce the failure rate of the event processing request from switches and improve the resource utilization to process more events with fixed resources.

6.3.2. Flow request processing capability

Fig. 9 compares the performance of MARVEL with DHA-LB and Balcon under the same request pattern. The three schemes are tested under the same simulation network with the same event generation source. In the figure, we take the same time interval of running of the three schemes and place their performance into the same coordinate for comparison. In our simulation, overloaded controller discards extra requests, so different schemes show different actual request processing rates. As shown in the figure, MARVEL can process the most events, while DHA-LB performs the worst. The reason of the difference in the event processing number is that when some controllers are overloaded and not relieved in time, some processing requests are dropped in our controllers. During our whole experiment time, the average event processing rate versus the overall capacity in a period of MARVEL is 85.3%, BalCon is 80.1%, and DHA-LB is 73.3%, which means that MARVEL can process 16.3% and 6.4% more events than DHA-LB and BalCon respectively under our test traffic.

6.3.3. Resource utilization

Fig. 10 shows the resource utilization improvement of different schemes under different intensities of workload over static mapping of switches. In this figure, the workload is measured by the average rate of requests in the experiment network, and the resource utilization is calculated according to the capacity of the whole network. As Fig. 9 shows, with the growth of the intensity of workload from 8,000 to 16,000, the improvement of network resource utility of MARVEL rises from 10.3% to 20.5%, which is better than BalCon (from 9.6% to 16.1%) and DHA-LB (from 6.1% to 12.2%). We can also notice in the figure that as the rate of request rises, the improvement of BalCon and DHA-LB rises slower than MARVEL. This is because that in both BalCon and DHA-LB, the switch migration process concentrates on only one controller at a time. In this way, only after the workload processing of one certain controller has been finished can the load balancing process of other controllers take place. In contrast, in MARVEL, the load balancing process takes place among all controllers at the same time. When the request rate rises, more controllers tend to be overloaded at the same time, thus making the working scheme of BalCon and DHA-LB less efficient than MARVEL.

6.3.4. Resource consumption

In Fig. 11, we compare the time and resource consumption of MARVEL and DHA_LB. Since the switch migration process itself also takes up some resources, too much occupancy of resource of the migration will undermine the controller's capacity to deal with regular packet events. Under different workloads and in different controllers, the consumption varies a lot, thus we use a normalized value with the average consumption of DHA_LB as a metric. The run time, bandwidth consumption and memory consumption of DHA_LB are set to 1 in Fig. 10, and the ratio of consumption on different resources of MARVEL and BalCon compared to DHA_LB is shown in the bar. From the figure, we can see that the run time of MARVEL and BalCon is 45% and 27% less than DHA_LB respectively. BalCon takes up the most memory resources because it needs to store the workload changes in switches. The bandwidth consumption of MARVEL is a little higher than BalCon, and this is mainly because there is more interaction and information sharing among different controllers in MARVEL during the balancing process.

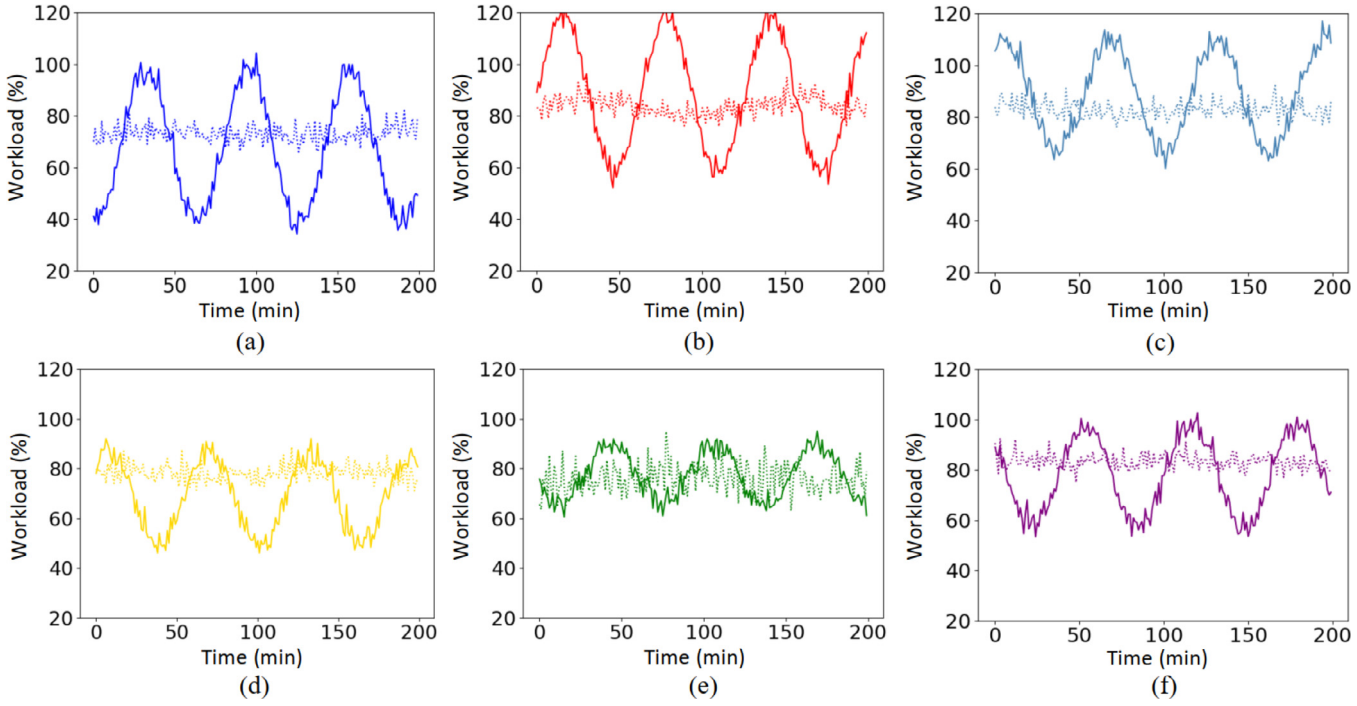

Fig. 8. Workload comparison in each controller without (denoted in solid line) and with (denoted in dotted line) the use of MARVEL. Fig. 8 (a)~7(f) correspond to controller 1 to controller 6 respectively.As shown in the figure, the original workload distribution (without MARVEL) fluctuates greatly, while after applying MARVEL to the controllers the workload is more evenly distributed.

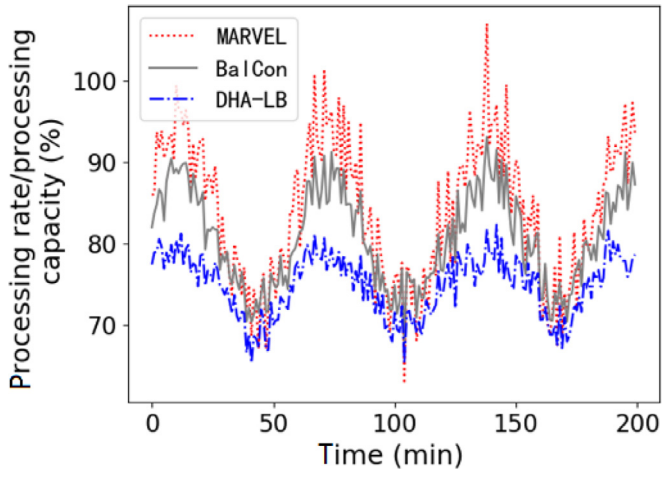

Fig. 9. Comparison of flow request processing rate in the control plane. The Y axis shows the actual request processing rate versus the overall processing capacity in the control plane measured by corresponding resources.

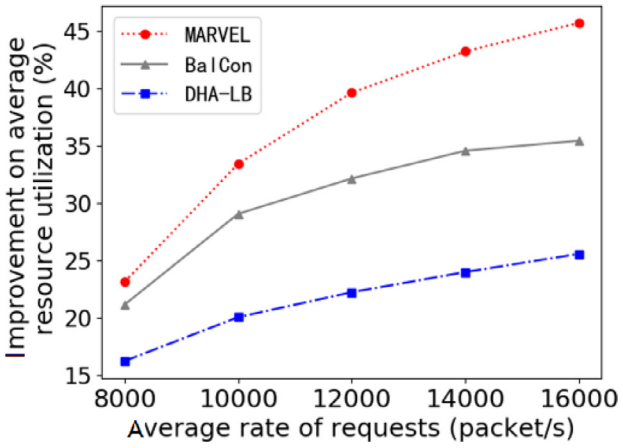

Fig. 10. Average improvement of resource utilization and event handling number under different workloads.

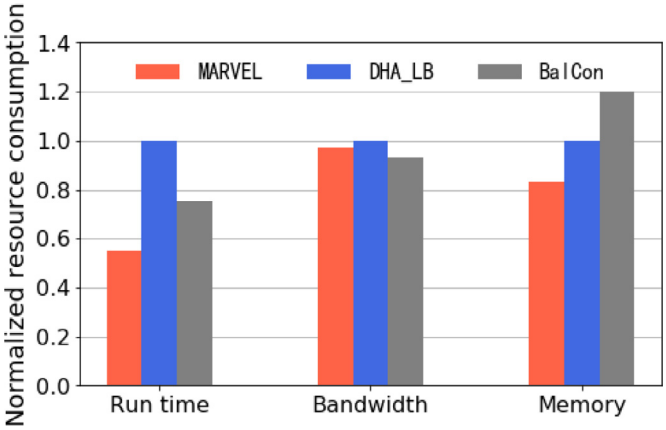

Fig. 11. Run time and resource consumption comparison among different schemes.

7. Related work

7.1. Elastic control in SDN

A static mapping between the multi-controller control plane and data plane in SDN is hard to adapt to the frequent change of spatial-temporal distribution of network traffic [2,5]. To solve this problem, some works suggested offloading some workload from the controllers to switches [37–39]. On the other hand, several works concentrated on the workload processing techniques within the control plane. ElastiCon [6] is proposed to dynamically change the number of controllers and migrate switches among different controllers according to traffic load; Yazıcı1 et al. [17] proposes a framework for the scalability and reliability of distributed control planes; B. Heller et al. [11] propose a scheme to place controllers based on the propagation latency. However, the works men-

tioned above do not solve the SMP about how to select a switch for migration and how to pick a target controller based on the workload. Infact, such solutions on dynamic controller provisioning are based on the change of the amount and location of controllers via reassigning the switches to controllers, which may lead to network instability and broadcast storm due to a large number of state synchronizations among controllers. Papers [9,18,20] try to find a switch migration path without changing the distribution of controllers. Such schemes use optimization arithmetic to model the SMP (e.g., Boolean integer linear program) and try to find a near-optimal solution based on heuristic methods or extra constraints, which are time-consuming and cannot ensure the total performance. Guo et al. [21–23] also study the switch allocation problems in the control plane, which use heuristic algorithms to find solutions.

7.2. MARL

Deep neural networks have recently boosted the notion of "learning from data" with field-changing performance improvements on different tasks [24,48,49]. RL is an area of machine learning, aiming at designing agents which can learn to achieve a certain goal in a long term by interacting with an unknown environment [39]. Combining deep learning with RL [24,40], deep reinforcement learning has created the first artificial agents capable of achieving human-level performance across a number of challenging domains, including AlphaGo [24], intelligent transportation systems [41], resource allocation in cyber-physical networks [42–45], traffic engineering [46,47].

There is now an emerging line of research of RL called MARL [40], which is designed to solve the problem in a Multi-Agent System (MAS), dealing with a group of agents that can sense and interact with an environment, and target a certain goal. MARL has been successfully used in many fields such as QoS routing in ATM [27]. There are many types of MARL techniques such as [28,29,50], among which equilibrium-based MARL [30,31] are the most important approaches.

8. Conclusions

In this paper, we have made the first attempt to introduce artificial intelligence into load balancing of the control plane in SDN. The SMP was modeled as an MARL among controllers, and the DRL agents in controllers make switch migration decisions without any human experience. After a period of training, the DRL agents gather enough knowledge through interactions with the network, and make automatic decisions on switch migration. According to our experiments, the MARL-based scheme performs better than other competing alternatives, and it is less time-consuming. This corroborates that artificial intelligence tools can "learn" to solve the challenging NP-Complete optimization problem of network resource utilization, thus considerably saving human labor. Our future research will leverage other distributed AI models to further boost the performance.

Declaration of Competing Interest

The authors declare that they have no known competing financial interestsor personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The research of this paper is supported by the National Key Researchand Development Plan (grant number 2017YFB0803204),the National Natural Science Fund (grant numbers 61521003,61872382) and Beijing Institute of Technology Research Fund Program for Young Scholars.

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.comnet.2020.107230.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, et al., OpenFlow: enabling innovation in campus networks, in: Processings of ACM SIGCOMM Computer Communication Review, 38, ACM, 2008, pp. 69–74.
[2] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, et al., Onix: a distributed control platform for large-scale production networks, in: Processings of 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2010), 2010, pp. 351–364.
[3] ZehuaGuo, WendiFeng, SenLiu, WenchaoJiang, YangXu, and Zhi-LiZhang, RetroFlow: maintaining Control Resiliency and Flow Programmability for Software-Defined WANs",IEEE/ACM International Symposium on Quality of Service 2019 (IWQoS'19).
[4] A.W. Tam, K. Xi, H. Chao, Use of devolved controllers in data center networks, in: Processings of the IEEE Computer Communications Workshops, IEEE, 2011, pp. 596–601.
[5] S. Hassas Yeganeh, Y. Ganjali, Kandoo: a framework for efficient and scalable offloading of control applications, in: Proceedings of 1st Workshop on Hot Topics in Software Defined Networking (HotSDN 2012), HotSDN, 2012, pp. 19–24.
[6] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, R. Kompella, Towards an elastic distributed SDN controller, in: Proceedings of 1st Workshop on Hot Topics in Software Defined Networking (HotSDN 2013), HotSDN, 2013, pp. 7–12.
[7] OpenFlow. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf.
[8] H. Chen, G. Cheng, Z. Wang, A game-theoretic approach to elastic control in software-defined networking, China Commun. 13 (5) (2016) 103–109.
[9] X. Ye, G. Cheng, X Luo, Maximizing SDN Control Resource Utilization via Switch Migration, Comp. Netw. (2017) S1389128617302669.
[10] T. Wang, F. Liu, J. Guo, H. Xu, Dynamic SDN controller assignment in data center networks: stable matching with transfers, in: Proc. 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM), Apr. 2016, pp. 1–9.
[11] T. Wang, F. Liu, H. Xu, An Efficient online algorithm for dynamic SDN controller assignment in data center networks, IEEE/ACM Trans. Netw. 25 (5) (Oct. 2017) 2788–2801.
[12] V. Huang, Q. Fu, G. Chen, E. Wen, J. Hart, BLAC: a bindingless architecture for distributed SDN controllers, in: Proc. IEEE 42nd Conf. Local Comput. Netw. (LCN), Oct. 2017, pp. 146–154.
[13] M. Cello, Y. Xu, A. Walid, et al., BalCon: a Distributed Elastic SDN Control via Efficient Switch Migration, IEEE International Conference on Cloud Engineering, IEEE, 2017.
[14] A. Azzouni, G. Pujolle, Neutm: a neural network-based framework for traffic matrix prediction in sdn, in: NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium, IEEE, 2018, pp. 1–5.
[15] T.Wang, Z.Guo, H.Chen, and W.Liu, "Bwmanager: mitigating denial of service attacks in software-defined networks through bandwidth prediction," IEEE Transactions on Network.
[16] S. Mallon, V. Gramoli, G. Jourjon, Are today's sdn controllers ready for prime-time? in: 2016 IEEE 41st Conference on Local Computer Networks (LCN), Nov 2016, pp. 325–332.
[17] V. Yazıcı1, M.O. ˘.guz Sunay1, Ö. Ali, 1. Ercan, Controlling a software-defined network via distributed controllers, Process. NEM (2012) submit 2012, arXiv:1401.7651.
[18] B. Heller, Rob Sherwood, Nick McKeown, The controller placement problem, in: Proc. 1st Workshop on Hot Topics in Software Defined Networking (HotSDN 2012), New York, ACM Press, 2012, pp. 7–12.
[19] G. Cheng, et al., DHA: distributed decisions on the switch migration toward a scalable SDN control plane, 2015 IFIP Networking Conference (IFIP Networking), IEEE Computer Society, 2015.
[20] T. Wang, F. Liu, J. Guo, H. Xu, Dynamic SDN controller assignment in data center networks: stable matching with transfers, Proceedings of IEEE INFOCOM, IEEE, 2016.
[21] Z.Guo, M.Su, Y.Xu, Z.Duan, L.Wang, S.Hui and H.Chao. Improving the performance of load balancing in software-defined networks through load variance-based synchronization. Comp. Netw., 68, pp. 95–109.
[22] Z. Guo, W. Chen, Y. Liu, Y. Xu, Z. Zhang, Joint Switch Upgrade and Controller Deployment in Hybrid Software-Defined Networks, IEEE J. Select. Areas Commun. 37 (5) (2019) 1012–1028.
[23] T.Hu, P.Yi, Z.Guo, J.Lan and Y.Hu. Dynamic slave controller assignment for enhancing control plane robustness in software-defined networks. Future Gen. Comp. Syst., 95, pp. 681–693.
[24] V. Mnih, K. Kavukcuoglu, D. Silver, Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.
[25] S. Hochreiter, Jürgen Schmidhuber, Long Short-Term Memory, Neural Comput. 9 (8) (1997) 1735–1780.
[26] ChoK., Van MerriënboerB., GulcehreC., BahdanauD., BougaresF., SchwenkH., and Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv:1406.1078.
[27] A.F. Atlasis, N.H. Loukas, A.V. Vasilakos, The use of learning algorithms in ATM networks call admission control problem: a methodology, Comp. Netw. 34 (3) (2000) 341–353 2000.

[28] A.V. Vasilakos, G.I. Papadimitriou, A new approach to the design of reinforcement schemes for learning automata: stochastic estimator learning algorithm, Neurocomputing 7 (3) (1995) 275–297.
[29] L. MacDermed, K.S. Narayan, L. Weiss, Quick polytope approximation of all correlated equilibria in stochastic games, in: Proc. 25th AAAI Conf. Artif. Intell, 2011, pp. 707–712.
[30] J. Hu, Nash Q-learning for general-sum stochastic games, J. Mach. Learn. Res. 4 (4) (2003) 1039–1069.
[31] Y. Hu, Y. Gao, An B. Multiagent Reinforcement Learning with Unshared Value Functions, IEEE Trans. Cybern. 45 (4) (2015) 647–662.
[32] A. Cully, J. Clune, D. Tarapore, J.-.B. Mouret, Robots that can adapt like animals, Nature 521 (7553) (2015) 503.
[33] A. Mirhoseini, A. Goldie, H. Pham, B. Steiner, Q.V. Le, J. Dean, A hierarchical model for device placement, in: Proc. of ICLR, 2018.
[34] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C.H. Liu, D. Yang, Experience-driven networking: a deep reinforcement learning based approach, in: Proc. of IEEE INFOCOM, 2018.
[35] P. Sun, Y. Hu, J. Lan, L. Tian, M. Chen, TIDE: time-relevant deep reinforcement learning for routing optimization, Future Gen. Comp. Syst. 99 (2019) 401–409.
[36] C. Chekuri, S. Khanna, On multi-dimensional packing problems, in: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 1999, pp. 185–194.
[37] A.R. Curtis, J.C. Mogul, J. Tourrilhes, et al., DevoFlow: scaling flow management for high-performance networks, in: ACM SIGCOMM Computer Communication Review, 41, ACM, 2011, pp. 254–265.
[38] J. Yang, X. Yang, Z. Zhou, et al., Focus: function offloading from a controller to utilize switch power, in: 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE, 2016, pp. 199–205.
[39] S. Zhu, J. Bi, C. Sun, et al., "Sdpa: enhancing stateful forwarding for software-defined networking, in: 2015 IEEE 23rd International Conference on Network Protocols (ICNP), IEEE, 2015, pp. 323–333.
[40] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT press, Cambridge, MA, 2018.
[41] M.Bojarski, D.Del Testa, D.Dworakowski, B.Firner, B.Flepp, P.Goyal, L.D.Jackel, M.Monfort, U.Muller, J.Zhanget al., "End to end learning for self-driving cars," arXiv:1604.07316, 2016.
[42] Alireza Sadeghi, Gang Wang, Georgios Giannakis, Deep Reinforcement Learning for Adaptive Caching in Hierarchical Content Delivery Networks, IEEE Transactions on Cognitive Communications and Networking 5 (4) (2019) 1024–1033.
[43] Qiuling Yang, Gang Wang, Alireza Sadeghi, Georgios Giannakis, Jian Sun, Two-timescale voltage control in distribution grids using deep reinforcement learning, IEEE Transactions on Smart Grid 11 (3) (2020) 2313–2323, doi:10.1109/TSG.2019.2951769.
[44] Penghao Sun, Julong Lan, Zehua Guo, Di Zhang, Xianfu Chen, Yuxiang Hu, Zhi Liu, DeepMigration: flow Migration for NFV with Graph-based Deep Reinforcement Learning, IEEE International Conference on Communications, 2020.
[45] PenghaoSun, ZehuaGuo, SenLiu, JulongLan, and YuxiangHu, QoS-aware Flow control for Power-Efficient Data Center Networks with Deep Reinforcement Learning, IEEE International Conference on Acoustics, Speech, and Signal Processing 2020 (ICASSP'20).
[46] Junjie Zhang, Minghao Ye, Zehua Guo, Chen-Yu Yen, H.Jonathan Chao, CFR-RL: traffic Engineering with Reinforcement Learning in SDN, IEEE J. Select. Areas Commun. (2020).
[47] Penghao Sun, Junfei Li, Zehua Guo, Yang Xu, Julong Lan, and Yuxiang Hu, SINET: enabling Scalable Network Routing with Deep Reinforcement Learning on Partial Nodes", ACM Special Interest Group on Data Communication (SIGCOMM'19 poster).
[48] Liang Zhang, Gang Wang, Georgios Giannakis, Real-time power system state estimation and forecasting via deep unrolled neural networks, IEEE Transactions on Signal Processing 67 (15) (2019) 4069–4077.
[49] Gang Wang, Georgios Giannakis, Jie Chen, Learning ReLU networks on linearly separable data: Algorithm, optimality, and generalization, IEEE Transactions on Signal Processing 67 (9) (2019) 2357–2370.
[50] Jun Sun, et al., Finite-Time Analysis of Decentralized Temporal-Difference Learning with Linear Function Approximation, Proc. of Intl. Conf. on Artificial Intelligence and Statistics (2020).

Penghao Sun is currently a Ph.D candidate in National Digital Switching System Engineering and Technological R&D Center (NDSC), China. He received the B.S and M.S degrees from NDSC in 2014 and 2017 respectively. His current research interests include network architecture, edge computing and machine learning on networking.

Zehua Guo received the B.S. degree from Northwestern Polytechnical University, Xi'an, China, the M.S. degree from Xidian University, Xi'an, China, and the Ph.D. degree from Northwestern Polytechnical University. He was a Research Fellow with the Department of Electrical and Computer Engineering, New York University Tandon School of Engineering, New York, NY, USA, and a Research Associate with the Department of Computer Science and Engineering, University of Minnesota Twin Cities, Minneapolis, MN, USA. His research interests include software-defined networking, network function virtualization, data center network, cloud computing, network security, machine learning, and Internet exchange. Dr. Guo is an Associate Editor for IEEE ACCESS and the EURASIP Journal on Wireless Communications and Networking (Springer), and an Editor for the KSII Transactions on Internet and Information Systems. He was the Session Chair for the IEEE International Conference on Communications 2018 and the Technical Program Committee Member of Computer Communications (Elsevier), ICCCN 2020, ICA3PP 2020, CSCloud 2020, SmartCloud 2020. He is a Senior Member of IEEE.

Gang Wang is a postdoctoral research associate at University of Minnesota Twin Cities.

Julong Lan is currently a professor and the chief engineer in National Digital Switching System Engineering and Technological R&D Center (NDSC), China. His is also the Chief scientist of China National Program on Key Basic Research Project (973 Program). His current research interests include 5 G communication system, next generation of computer network and artificial intelligence.

Yuxiang Hu received his Ph.D.degree in 2011 in National Digital Switching System Engineering and Technological Research Center ofChina. He is currently a associate professor, and hisresearch interests include Internet architecture, novelswitching and routing, multimedia network technologyand so on.