Mobile Networks and Applications
https://doi.org/10.1007/s11036-020-01632-1

Raven: Scheduling Virtual Machine Migration During Datacenter Upgrades with Reinforcement Learning

Chen Ying[1] · **Baochun Li**[1] · **Xiaodi Ke**[2] · **Lei Guo**[2]

Abstract

Physical machines in modern datacenters are routinely upgraded due to their maintenance requirements, which involve migrating all the virtual machines they currently host to alternative physical machines. For this kind of datacenter upgrades, it is critical to minimize the time it takes to upgrade all the physical machines in the datacenter, so as to reduce disruptions to cloud services. To minimize the upgrade time, it is essential to carefully schedule the migration of virtual machines on each physical machine during its upgrade, without violating any constraints imposed by virtual machines that are currently running. Rather than resorting to heuristic algorithms as existing work, we propose a new scheduler, *Raven*, that uses an experience-driven approach with deep reinforcement learning to schedule the virtual machine migration. With our design of the state space, action space and reward function, *Raven* trains a fully-connected neural network using the cross-entropy method to approximate the policy of choosing a destination physical machine for each virtual machine before its migration. We compare *Raven* with state-of-the-art algorithms in the literature, and our results show that *Raven* can effectively shorten the time to complete the datacenter upgrade under different datacenter settings.

Keywords Virtual machine migration · Deep reinforcement learning

1 Introduction

In a modern datacenter, it is routine for its physical machines to be upgraded to newer versions of operating systems or firmware versions from time to time, as part of their maintenance. However, production datacenters are used for hosting virtual machines, and these virtual machines will have to be migrated to alternative physical machines during the upgrade process. The migration process takes time, which involves transferring the images of virtual machines between physical machines across the datacenter network.

To incur the least amount of disruption to cloud services provided by a production datacenter, it is commonly accepted that we need to complete the upgrade process as quickly as possible. Assuming that the time of upgrading a physical machine is dominated by the time it takes to migrate the images of all the virtual machines on this physical machine, the problem of minimizing the upgrade time of all physical machines in a datacenter is equivalent to minimizing the *total migration time*, which is the time it takes to finish all the virtual machine migrations during the datacenter upgrade.

In order to reduce the total migration time, we will need to plan the schedule of migrating virtual machines carefully. To be more specific, we should carefully select the best possible destination physical machine for each virtual machine to be migrated. However, as it is more realistic to assume that the topology of the datacenter network and the network capacity on each link are unknown to such a scheduler, computing the optimal migration schedule that minimizes the total migration time becomes more challenging.

With the objective of minimizing the migration time, a significant amount of work on scheduling the migration of

✉ Chen Ying
chenying@ece.toronto.edu

Baochun Li
bli@ece.toronto.edu

Xiaodi Ke
xiaodi.ke@huawei.com

Lei Guo
leiguo@huawei.com

[1] University of Toronto, Toronto, Canada

[2] Huawei Canada, Markham, Canada

Published online: 05 August 2020

virtual machines has been proposed. However, to the best of our knowledge, none of them considered the specific problem of migrating virtual machines during datacenter upgrades. Commonly, existing studies focus on migrating a small number of virtual machines to reduce the energy consumption of physical machines [1] [2] [3], or to balance the utilization of different resources across physical machines [4]. Further, most of the proposed schedulers are based on heuristic algorithms and a set of strong assumptions that may not be realized in practice.

Without such detailed knowledge of the datacenter network, we wish to explore the possibilities of making scheduling decisions based on deep reinforcement learning [5], which trains an agent to learn the policy with a deep neural network as the function approximator to make better decisions from its experience, as it interacts with an unknown environment. Though it has been shown that deep reinforcement learning is effective in playing games [6], whether it is suitable for scheduling resources, especially in the context of scheduling migration of virtual machines, is not generally known.

In this paper, we propose *Raven*, a new scheduler for scheduling the migration of virtual machines in the specific context of datacenter upgrades. In contrast to existing work in the literature, we assume that the topology and link capacities of the datacenter network are *not* known *a priori* to the scheduler, which is more widely applicable to realistic scenarios involving production datacenters. By considering the datacenter network as an unknown environment that needs to be explored, we seek to leverage reinforcement learning to train an agent to choose an optimal scheduling action, *i.e.,* the best destination physical machine for each virtual machine to be migrated, with the objective of achieving the shortest possible total migration time for the datacenter upgrade. By tailoring the state space, action space and reward function for our scheduling problem, *Raven* uses the off-the-shelf cross-entropy method to train a fully-connected neural network to approximate the policy of choosing a destination physical machine for each virtual machine before its migration, aiming at minimizing the total migration time.

Highlights of our original contributions in this paper are as follows. *First*, we consider the real-world problem of migrating virtual machines for upgrading physical machines in a datacenter, which is rarely studied in the previous work. *Second*, we design the state space, action space, and reward function for our deep reinforcement learning agent to schedule the migration of virtual machines with the objective of minimizing the total migration time. *Finally*, we propose and implement our new scheduler, *Raven*, and conduct a collection of simulations to show *Raven*'s effectiveness of outperforming the existing methods with respect to minimizing the total migration time, without any *a priori* knowledge of the datacenter network under different datacenter settings.

2 Problem formulation and motivation

To start with, we consider two different resources, CPU and memory, and we assume the knowledge of both the number of CPUs and the size of the main memory in each of the virtual machines (VMs) and physical machines (PMs). In addition, we assume that the current mapping between the VMs and PMs is known as well, in that for each PM, we know the indexes of the VMs that are currently hosted there. It is commonly accepted that the number of CPUs and the size of the main memory of every VM on a PM can be accommodated by the total number of physical CPUs and the total size of physical memory on this hosting PM.

Firstly, we use an example to illustrate the way of computing the migration time of a *migration batch*. Some migration processes can be conducted at the same time, if 1) for all these migration processes, no source PM is the same as any destination PM and no destination PM is the same as any source PM and 2) every destination PM in these migration processes has enough residual CPUs and memory to host all VMs that will be migrated to it. We treat migration processes that can process at the same time as a *migration batch*.

Fig. 1 shows the assumed network topology. There are 6 PMs, with PM #0 to PM #2 linked to AS (aggregation switch) # 0, and PM #3 to PM #5 linked to AS #1. AS #0 and AS #1 connect to the core switch.

Assume that the network capacity of each link between an aggregation switch and the core switch is 1GB/s, and the network capacity of each link between a PM and an AS is 2GB/s.

Take computing the migration time of a migration batch with three migration processes as an example:

- ◇ VM #0 (with 1GB of image): PM #0 → PM #3
- ◇ VM #1 (with 2GB of image): PM #0 → PM #1
- ◇ VM #2 (with 3GB of image): PM #0 → PM #5

According to the network topology, the route of each migration process is:

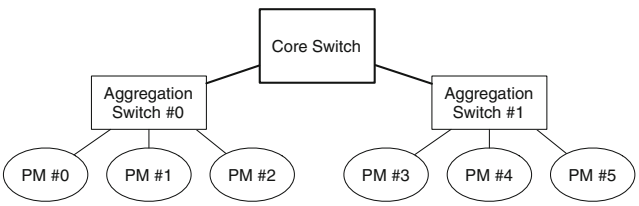

Fig. 1 The network topology of the example

- ◇ VM #0: PM #0 → AS #0 → core switch → AS #1 → PM #3
- ◇ VM #1: PM #0 → AS #0 → PM #1
- ◇ VM #2: PM #0 → AS #0 → core switch → AS #1 → PM #5

In the beginning, all these three migration processes proceed at the same time, and we assume that every migration process using the same bottleneck link gets equal throughput. So the bandwidth of each possible bottleneck link is:

PM #0 → AS #0: $\frac{2}{3}$GB/s;

AS #0 → core switch: $\frac{1}{2}$GB/s.

Thus the migration rate of VM #1 is $\frac{2}{3}$GB/s, while VM #0 and VM #2 are both $\frac{1}{2}$ GB/s.

To complete the migration with these rates, the time needed by each VM is:

VM #0: 1GB / ($\frac{1}{2}$GB/s) = 2s;

VM #1: 2GB / ($\frac{2}{3}$GB/s) = 3s;

VM #2: 3GB / ($\frac{1}{2}$GB/s) = 6s.

As VM #0 takes the minimal time of 2s, the bandwidth of every link might change after 2s.

In the first 2s, VM #1 transfers 2s $\times \frac{2}{3}$GB/s = $\frac{4}{3}$GB size of image. It needs to transfer $\frac{2}{3}$GB later. VM #2 finishes transferring 2s $\times \frac{1}{2}$GB/s = 1GB, with 2GB left.

Starting from 2s, the bandwidth of each possible bottleneck link is:

PM #0 → AS #0: 1GB/s;

AS #0 → core switch: 1GB/s.

So the rates of VM #1 and VM #2 are both 1GB/s.

VM #1 can finish its migration in $\frac{2}{3}$GB / (1GB/s) = $\frac{2}{3}$s. After that, VM #2 will be the only VM whose migration has not finished. In this $\frac{2}{3}$s, VM #2 transfers $\frac{2}{3}$GB of image, remaining $\frac{4}{3}$GB to be transferred.

Starting from 2s + $\frac{2}{3}$s, the bandwidth of each possible bottleneck link is:

PM #0 → AS #0: 2GB/s;

AS #0 → core switch: 1GB/s.

So the migration rate of VM #2 is 1GB/s, and thus VM #2 finishes its migration in $\frac{4}{3}$GB / (1GB/s) = $\frac{4}{3}$s.

Therefore, the migration time of this migration batch is 2s + $\frac{2}{3}$s + $\frac{4}{3}$s = 4s.

Now, we use the following example to illustrate the upgrade process of the physical machines, and to explain why the problem of scheduling VM migrations to minimize the total migration time may be non-trivial.

Assume there are three PMs in total, each hosting one of the three VMs. The information of these PMs and VMs, with respect to the number of CPUs and the amount of memory, are shown in Fig. 2. These PMs are updated one by one. To show the total migration times of different schedules, we assume the simplest possible datacenter network topology, where all of these three PMs are connected to the same core switch, and the network capacity of the link between the core switch and each PM is 1GB/s.

In the first schedule, as shown in Fig. 2, we first upgrade PM #0 and migrate VM #0 to PM #2, and then upgrade PM #1 by migrating VM #1 to PM #2. Finally, we upgrade PM #2 with migrating VM #0, VM # 1 and VM # 2 to PM #0.

From the perspective of VMs, the migration process of each VM is:

- ◇ VM #0: PM #0 → PM #2 → PM #0;
- ◇ VM #1: PM #1 → PM #2 → PM #0;
- ◇ VM #2: PM #2 → PM #0.

To calculate the total migration time, we start from the VM on the PM that is upgraded first. In this schedule, we start from the migration of VM #0 from PM #0 to PM #2. As PM #0 is being upgraded now, we cannot migrate VM #1 or VM #2, whose destination PM is PM #0. Since only VM #0 is being migrated, it can use all the network capacity through the link in its migration. This migration takes 1GB/(1GB/s) = 1s.

Then we come to VM #1 as PM #1 is upgraded next. Now the rest of migration processes whose migration times have not been calculated are:

- ◇ VM #0: PM #2 → PM #0;
- ◇ VM #1: PM #1 → PM #2 → PM #0;
- ◇ VM #2: PM #2 → PM #0.

As VM #1 is migrated to PM #2, here we cannot migrate VM #0 or VM #2, whose source PM is PM #2. Since only VM #1 is being migrated, it can occupy all the network capacity through the link during its migration. This migration takes 2GB/(1GB/s) = 2s.

At last, we come to PM #2. All unfinished migration processes are:

- ◇ VM #0: PM #2 → PM #0;
- ◇ VM #1: PM #2 → PM #0;
- ◇ VM #2: PM #2 → PM #0.

Apparently, all these three migration processes can be conducted in one migration batch. Since at first, these three VMs share the link from the core switch to PM #0, each of them will get the rate of $\frac{1}{3}$GB/s. To finish migrating each VM with this rate, VM #0 will take 1GB/($\frac{1}{3}$GB/s) = 3s, while VM #1 will take 2GB/($\frac{1}{3}$GB/s) = 6s, and VM #2 will take 3GB/($\frac{1}{3}$GB/s) = 9s. So the migration of VM #0 is completed in 3s.

After 3s, VM #1 and VM #2 share the link from the core switch to PM #0. Each of them get the rate of $\frac{1}{2}$GB/s. To finish migrating these two VMs under this rate, VM #1 will take 1GB/($\frac{1}{2}$GB/s) = 2s, while VM #2 will take

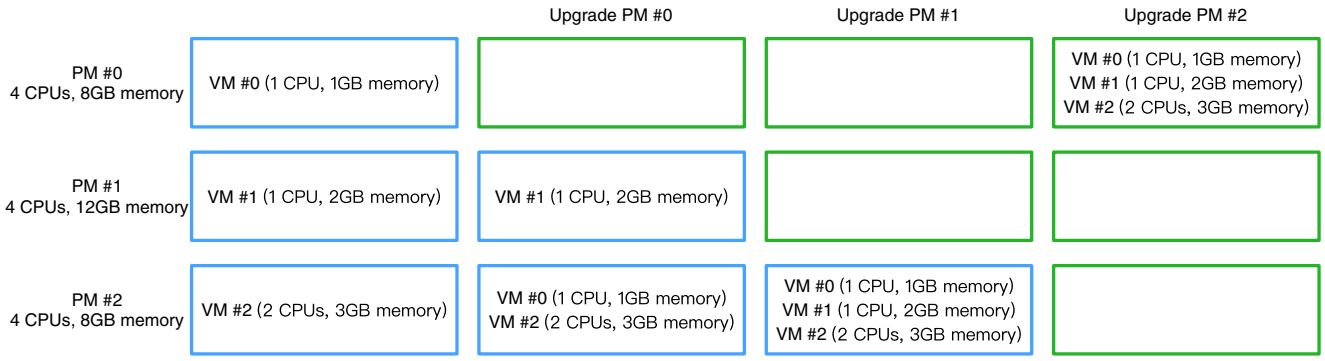

Fig. 2 Scheduling VM migration: a schedule that takes 9 seconds of total migration time

2GB/($\frac{1}{2}$GB/s) = 4s. So we can finish the migration of VM #1 in 2s. Now, we need to finish transferring 1GB of the image on VM #2 with the rate of 1GB/s, which can be done in 1s.

Therefore, the migration time of this migration batch is 3s + 2s + 1s = 6s. So the total migration time is 1s + 2s + 6s = 9s.

In contrast, a better schedule in terms of reducing the total migration time is shown in Fig. 3. We first upgrade PM #0 and migrate VM #0 to PM #1, and then upgrade PM #2 by migrating VM #2 to PM #0. Finally, we upgrade PM #1 and migrate VM #0 to PM #2 and VM #1 to VM #0.

From the perspective of VMs, the migration process of each VM is:

- ◇ VM #0: PM #0 → PM #1 → PM #2;
- ◇ VM #1: PM #1 → PM #0.
- ◇ VM #2: PM #2 → PM #0.

We start from VM #0 which is initially on the first migrated PM, PM #0. Since the destination PM of VM #1 is PM #0, which is the source PM of VM #0, the migration of VM #1 cannot be proceeded now. For the same reason, the migration of VM #2 cannot be done either. Therefore, the migration time of migrating VM #0 from PM #0 to PM #1 is 1GB/(1GB/s) = 1s.

Then we come to VM #2 which is on PM #2, the next upgraded PM. The migration of VM #1 from PM #1 to PM #0 can be done at the same time. At first, These two VMs share the link between the core switch to PM #0, with $\frac{1}{2}$GB/s. To finish migrating these two VMs with this rate, VM #1 will take 2GB/($\frac{1}{2}$GB/s) = 4s, while VM #2 will take 3GB/($\frac{1}{2}$GB/s) = 6s. So we can finish the migration of VM #1 in 4s. Now, we need to finish transferring 1GB of image on VM #2 with the rate of 1GB/s, which can be done in 1s. Thus, the migration time of this batch is 4s + 1s = 5s.

At last, we compute the migration time of VM #0 from PM #1 to PM #2, which is 1s. Therefore, the total migration time of this schedule is 1s + 5s + 1s = 7s, which is shorter than that of the previous schedule taking 9s.

As we can observe from our examples, even though we schedule the migration of VMs one by one, the actual migration order cannot be determined until we have the schedule of all VM migrations that will take place during the datacenter upgrade. This implies that we will only be able to compute the total migration time when the datacenter upgrade is completed. To make the scheduling problem even more difficult to solve, it is much more practical to assume that, unlike our examples, the topology and network capacities of the datacenter network are not known *a priori*, which makes it even more challenging to estimate the total migration time when scheduling the migration of VMs one by one.

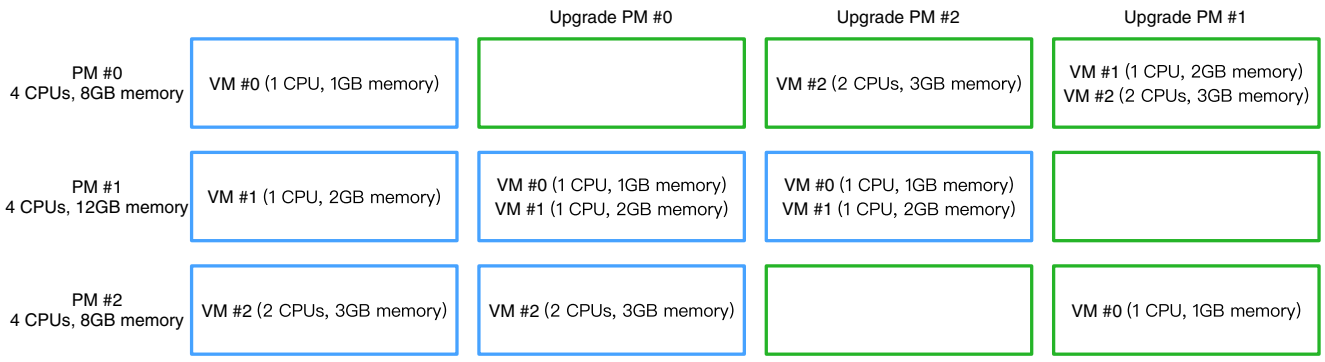

Fig. 3 Scheduling VM migration: a schedule that takes 7 seconds of total migration time

Therefore, we propose to leverage deep reinforcement learning to schedule the destination PM for each migrating VM, with the objective of minimizing the total migration time. The hope is that the agent of deep reinforcement learning can learn to make better decisions by iteratively interacting with an unknown environment over time.

3 Preliminaries

Before advancing to our proposed work, we first introduce some preliminaries on deep reinforcement learning, which applies deep neural networks as function approximators of reinforcement learning.

Under the standard reinforcement learning setting, an agent learns by interacting with the environment E over a number of discrete time steps in an episodic fashion [5]. During an episode, the agent observes the state s_t of the environment at each time step t and takes an action a_t from a set of possible actions $\mathcal{A}$ according to its policy $\pi : \pi(a|s) \rightarrow [0, 1]$, which is a probability distribution over actions. $\pi(a|s)$ is the probability that action a is taken in state s. Following the taken action a_t, the state of the environment transits to state s_{t+1} and the agent receives a reward r_t. The process continues until the agent reaches a terminal state. The states, actions, and rewards that the agent experienced during one episode form a trajectory $x = (s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_T, a_T, r_T)$, where T is the last time step. The cumulative reward $R(x) = \sum_{t=1}^{T} r_t$ is used to measure how good a trajectory is.

As the agent's behaviour is defined by a policy $\pi(a|s)$, which maps state s to a probability distribution over all actions $a \in \mathcal{A}$, the method of storing the state-action pairs should be carefully considered. Since the number of state-action pairs in complex decision-making problems would be too large to store in tabular form, it is common to use function approximators, such as deep neural networks [6] [7] [8] [9]. A function approximator has a manageable number of adjustable policy parameters θ. To show that the policy corresponds to parameters θ, we represent it as $\pi(a|s;\theta)$. For the problem of choosing a destination PM for a migrating VM, an optimal policy $\pi(a|s;\theta^*)$ with parameters θ^* is the migration strategy we want to obtain.

To obtain the parameters θ^* of an optimal policy, we could use a basic but efficient method, cross-entropy [10], whose objective is to maximize the cumulative reward $R(x)$ received in a trajectory x from an arbitrary set of trajectories $\mathcal{X}$. Denote x^* as the corresponding trajectory at which the cumulative reward is maximal, and let ξ^* be the maximum cumulative reward, we thus have $R(x^*) = \max_{x\in\mathcal{X}} R(x) = \xi^*$.

Assume x has the probability density $f(x;u)$ with parameters u on $\mathcal{X}$, and the estimation of the probability that the cumulative reward of a trajectory is greater than a fixed level ξ is $l = \mathbb{P}(R(x) \geq \xi) = \mathbb{E}[\mathbf{1}_{\{R(x)\geq\xi\}}]$, where $\mathbf{1}_{\{R(x)\geq\xi\}} = 1$ if $R(x) \geq \xi$, and 0 otherwise. If ξ happens to be set closely to the unknown ξ^*, $R(x) \geq \xi$ will be a rare event, which requires a large number of samples to estimate the expectation of its probability accurately.

A better way to perform the sampling is to use importance sampling. Let $f(x;v)$ be another probability density with parameters v such that for all x, $f(x;v) = 0$ implies that $\mathbf{1}_{\{R(x)\geq\xi\}} f(x;u) = 0$. Using the probability density $f(x;v)$, we can represent l as

$$\begin{aligned} l &= \int \mathbf{1}_{\{R(x)\geq\xi\}} \frac{f(x;u)}{f(x;v)} f(x;v)\mathrm{d}x \\ &= \mathbb{E}_{x\sim f(x;v)} \left[\mathbf{1}_{\{R(x)\geq\xi\}} \frac{f(x;u)}{f(x;v)} \right]. \end{aligned} \tag{1}$$

Thus the optimal importance sampling probability for a fixed level ξ is given by $f(x;v^*) \propto |\mathbf{1}_{\{R(x)\geq\xi\}}| f(x;u)$, which is generally difficult to obtain. So the idea of the cross-entropy method is to choose an importance sampling probability density $f(x;v)$ in a specified class of densities so that the distance between the optimal importance sampling density $f(x;v^*)$ and $f(x;v)$ is minimal. The distance between two probability densities f_1 and f_2 could be measured by the Kullback-Leibler divergence, $\mathbb{E}_{x\sim f_1(x)}\left[\log f_1(x)\right] - \mathbb{E}_{x\sim f_1(x)}\left[\log f_2(x)\right]$, The second term $-\mathbb{E}_{x\sim f_1(x)}\left[\log f_2(x)\right]$ is called *cross-entropy*, a common optimization objective in deep learning. While the first term $\mathbb{E}_{x\sim f_1(x)}\left[\log f_1(x)\right]$ could be omitted as it does not reflect the distance between $f_1(x)$ and $f_2(x)$. It turns out that the optimal parameters v^* is the solution to the maximization problem

$$\max_{v} \int \mathbf{1}_{\{R(x)\geq\xi\}} f(x;u) \log f(x;v)\mathrm{d}x, \tag{2}$$

which can be estimated via sampling by solving a stochastic counterpart program with respect to parameters v:

$$\hat{v} = \arg\max_{v} \frac{1}{N} \sum_{n\in[N]} \mathbf{1}_{\{R(x_n)\geq\xi\}} \frac{f(x_n;u)}{f(x_n;w)} \log f(x_n;v), \tag{3}$$

where $x_1, \cdots, x_N$ are random samples from $f(x;w)$ for any reference parameter w.

To train the deep neural network, randomly initialize the parameters $u = \hat{v}_0$. By sampling with the importance sampling distribution in each iteration k, we create a sequence of levels $\hat{\xi}_1, \hat{\xi}_2, \cdots$ converging to the optimal performance ξ^*, and the corresponding sequence of parameter vectors $\hat{v}_0, \hat{v}_1, \cdots$, converging to the optimal parameter vector v^*. Note that $\hat{\xi}_k$ is typically chosen as the $(1-\rho)$-quantile of performances of the sampled trajectories, which means that we will leave the top ρ of episodes sorted by cumulative reward. Sampling from an importance sampling distribution that is close to the theoretically

optimal importance sampling density will produce optimal or near-optimal trajectories x^*.

The probability of a trajectory x is determined by the transition dynamics $p(s_{t+1}|s_t, a_t)$ of the environment and the policy $\pi(a_t|s_t;\theta)$. As the transition dynamics is determined by the environment and cannot be changed, the parameters θ in policy $\pi(a_t|s_t;\theta)$ are to be updated to improve the importance sampling density $f(x;v)$ of a trajectory x with $R(x)$ of high value. Thus the parameter estimator at iteration k is

$$\hat{\theta}_k = \arg\max_{\theta_k} \sum_{n\in[N]} \mathbf{1}_{\{R(x_n)\geq\xi_k\}} \left(\sum_{a_t, s_t \in x_n} \pi(a_t|s_t;\theta_k) \right), \quad (4)$$

where $x_1, \cdots, x_N$ are sampled from policy $\pi(a|s;\tilde{\theta}_{k-1})$, and $\tilde{\theta}_k = \alpha\hat{\theta}_k + (1-\alpha)\tilde{\theta}_{k-1}$. α is a smoothing parameter. The equation (4) could be interpreted as maximizing the likelihood of actions in trajectories with high cumulative rewards.

4 Design

This section presents the design of *Raven*. It begins with illustrating an overview of the architecture of *Raven*. We then formulate the problem of VM migration scheduling for reinforcement learning, and show our design of the state space, action space, and reward function.

4.1 Architecture

Figure 4 shows the architecture of *Raven*. The upgrade process starts from choosing a PM among PMs that have not been upgraded to upgrade. Then we have a queue of VMs that are on the chosen PM to be migrated. At each time step, one of the VMs in this queue is migrated. The key idea of *Raven* is to use a deep reinforcement learning agent to perform scheduling decision of choosing a destination PM for the migrating VM. The core component of the agent is the policy $\pi(a|s;\theta)$, providing the probability distribution over all actions given a state s. The parameters θ in $\pi(a|s;\theta)$ are learned from experiences collected when the agent interacts with the environment E.

An episode here is to finish upgrading all physical machines in the datacenter. At time step t in an episode, the agent senses the state s_t of the environment, recognizes which VM is being migrated right now, and takes an action a_t, which is to choose a destination PM for this migrating VM based on its current policy. Then the environment will return a reward r_t to the agent and transit to s_{t+1}.

We play a number of episodes with our environment. Due to the randomness of the way that the agent selects actions to take, some episodes will be better, *i.e.,* have higher cumulative rewards, than others. The key idea of the cross-entropy method is to throw away bad episodes and train the policy parameters θ based on good episodes. Therefore, the agent will calculate the cumulative reward of every episode and then decide a reward boundary, and train a deep neural network based on episodes whose cumulative reward is higher than the reward boundary by using each state as the input and the issued action as the desired output.

4.2 Deep reinforcement learning formulation

The design of the state space, action space and reward function is one of the most critical steps when applying deep reinforcement learning to a practical problem. To train an effective policy within a short period of time, the deep reinforcement learning agent should be carefully designed such that it will be able to master the key components of the problem without useless or redundant information.

State Space. To describe the environment correctly and concisely for the agent, the state should include the knowledge of the upgrade status, the total number and usage of all resources of each PM and information of the migrating VM. So we have the following design of the state.

At time step t, we denote the detailed information of PM $\#j$, $j = 1, 2 \ldots, J$, as

$$\begin{aligned} s_{tj} = \{ & s_{tj}^{\text{status}}, s_{tj}^{\text{total cpu}}, s_{tj}^{\text{total mem}}, \\ & s_{tj}^{\text{used cpu}}, s_{tj}^{\text{used mem}} \}, \end{aligned} \quad (5)$$

where J is the total number of PMs in this datacenter and s_{tj}^{status} is the upgrade status of PM $\#j$. There are three possible upgrade statuses: not yet, upgrading and upgraded. $s_{tj}^{\text{total cpu}}$ and $s_{tj}^{\text{total mem}}$ represent the total number of CPUs and the total size of main memory of PM $\#j$, respectively. $s_{tj}^{\text{used cpu}}$ and $s_{tj}^{\text{used mem}}$ denote the number of used CPUs and the size of used memory of PM $\#j$, respectively. We represent the state of the environment at time step t as

$$s_t = \{s_{t1}, s_{t2}, \ldots, s_{tJ}, v_t^{\text{cpu}}, v_t^{\text{mem}}, v_t^{\text{pm id}}\}, \quad (6)$$

where v_t^{cpu}, v_t^{mem} and $v_t^{\text{pm id}}$ denote the number of CPUs, the size of memory of and the source PM index of the migrating VM, respectively.

Action Space. Since the agent is trained to choose a destination PM for each migrating VM, the action a_t should be set as the index of the destination PM.

Even though it is intuitive to set the action space as $\mathcal{A} = \{1, 2, \ldots, J\}$, this setting has two major problems that 1) the destination PM may not have enough residual resources to host the migrating VM and 2) the destination PM maybe the source PM of the migrating VM. To avoid these two problems, we dynamically change the action space for

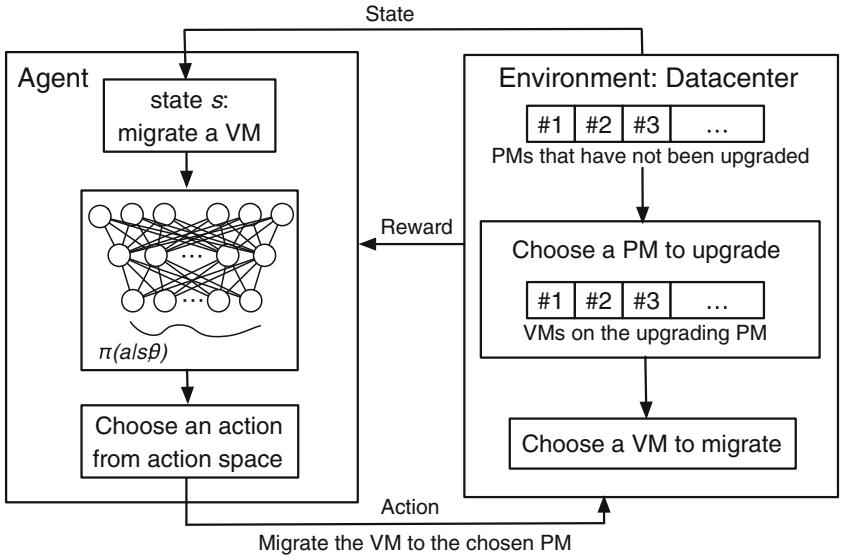

Fig. 4 The architecture of *Raven*

each migrating VM, instead of keeping the action space unchanged as traditional reinforcement learning methods.

When the migrating VM is decided at time step t, the subset of PMs that are not the source PM of the migrating VM and have enough number of residual CPUs and enough size of residual memory to host this VM can be determined. We denote the set of indexes of PMs in this subset as $\mathcal{A}_t^{\text{eligible}}$. Thus the action space at this time step t is $\mathcal{A}_t^{\text{eligible}}$.

State Transition. Assume at time step t, the migrating VM on PM #m takes v_t^{cpu} number of CPUs and v_t^{mem} size of memory, and the agent takes action $a_t = n$. Therefore, we have the state of PM #m at time step $t+1$ as

$$\begin{aligned} s_{(t+1)m} =&\{\text{upgraded}, s_{tm}^{\text{total cpu}}, s_{tm}^{\text{total mem}}, \\ & s_{tm}^{\text{used cpu}} - v_t^{\text{cpu}}, s_{tm}^{\text{used mem}} - v_t^{\text{mem}}\}, \end{aligned} \tag{7}$$

and the state of PM #n at time step $t+1$ as

$$\begin{aligned} s_{(t+1)n} =&\{s_{tn}^{\text{status}}, s_{tn}^{\text{total cpu}}, s_{tn}^{\text{total mem}}, \\ & s_{tn}^{\text{used cpu}} + v_t^{\text{cpu}}, s_{tn}^{\text{used mem}} + v_t^{\text{mem}}\}. \end{aligned} \tag{8}$$

Thus the state of the environment at time step $t+1$ will be

$$\begin{aligned} s_{t+1} =&\{s_{t1}, \ldots, s_{t(m-1)}, s_{(t+1)m}, s_{t(m+1)}, \ldots, \\ & s_{t(n-1)}, s_{(t+1)n}, s_{t(n+1)}, \ldots, s_{tJ}, \\ & v_{t+1}^{\text{cpu}}, v_{t+1}^{\text{mem}}, v_{t+1}^{\text{pm id}}\}, \end{aligned} \tag{9}$$

where v_{t+1}^{cpu}, v_{t+1}^{mem} and $v_{t+1}^{\text{pm id}}$ are the number of CPUs, size of memory and source PM index of the migrating VM at time step $t+1$.

Reward. The objective of the agent is to find out a scheduling decision for each migrating VM to minimize the total migration time. Since for our scheduling problem, we cannot know the total migration time until the schedule of all VM migrations is known. To be more specific, we can only know the total migration time after the episode is finished. This is because although we make the scheduling decision for VMs one by one, the VM migrations in the datacenter network could be conducted in a different order, which is not determined until we finish scheduling all the migrations, as we have discussed in Section 2.

Therefore, we design the reward r_t after taking action a_t at state s_t as $r_t = 0$ when $t = 1, 2, \ldots, T-1$, where T is the number of time steps to complete this episode. At time step T, as the schedule of all VM migrations is determined, the total migration time can be computed. We set r_T as the negative number of the total migration time. So the cumulative reward of an episode will be the negative number of the total migration time of this episode. Therefore, by maximizing the cumulative reward received by the agent, we can minimize the total migration time.

5 Performance evaluation

We conduct simulations of *Raven* to show its effectiveness in scheduling the migration process of virtual machines during the datacenter upgrade in terms of shortening the total migration time.

5.1 Simulation settings

We evaluate the performance of *Raven* under various datacenter settings, where the network topology, the total number of physical machines and the total number of virtual machines are different. Also, we randomly generate the mapping between the virtual machines and physical machines before the datacenter upgrade to make the environment more uncertain.

We simulate two types of network topology. One is of two-layer, where all physical machines are linked to a core switch. The other one is of three-layer, where several physical machines are linked to an aggregation switch, and all aggregation switches are linked to a core switch. Fig, 1 is an example of the three-layer topology. The network topology is set by the number of aggregation switches. The two-layer topology is applied with 0 aggregation switches, and the three-layer topology is used when the number is no less than 2.

Since to the best of our knowledge, there is no existing work that studies the same scheduling problem of virtual machine migration for the datacenter upgrade as we do, we can only compare our method with existing scheduling methods of virtual machine migration designed for other migration reasons.

We compare *Raven* with the state-of-the-art virtual machine migration scheduler, Min-DIFF [4], which uses a heuristic-based method to balance the usage of different resources on each physical machine. We also come up with a heuristic method to minimize the total migration time. The main idea of this heuristic method is to migrate a VM to the PM with the least residual memory among PMs that have enough resources to host this VM. Before migrating a VM, we pick up PMs with enough residual CPU and memory to host this VM and choose the PM with the least residual memory among these PMs to be the destination PM.

5.2 Simulation results

Convergence behaviour. The convergence behaviour of a reinforcement learning agent is a useful indicator to show whether the agent successfully learns the policy or not. A preferable convergence behaviour is that the cumulative rewards can gradually increase through the training iterations and converge to a high value.

We plot the figures of the number of training iterations and cumulative rewards under different datacenter settings. Each iteration has 16 episodes. The reward mean is the average cumulative rewards of the 16 episodes in one iteration, and the reward bound is the 70-th percentile of cumulative rewards in one iteration. Figure 5 is for 9 physical machines and 30 virtual machines, while Fig. 6 is for 10 physical machines and 40 virtual machines. The network topology here is of two-layer, where all physical machines are connected to the same core switch.

As shown in these two figures, *Raven* is able to learn the policy of scheduling the virtual machine migration in a datacenter with various numbers of physical machines and virtual machines, as the cumulative rewards gradually increase through the training. However, we find that when the number of physical machines and virtual machines becomes large, for example, 50 physical machines and 100

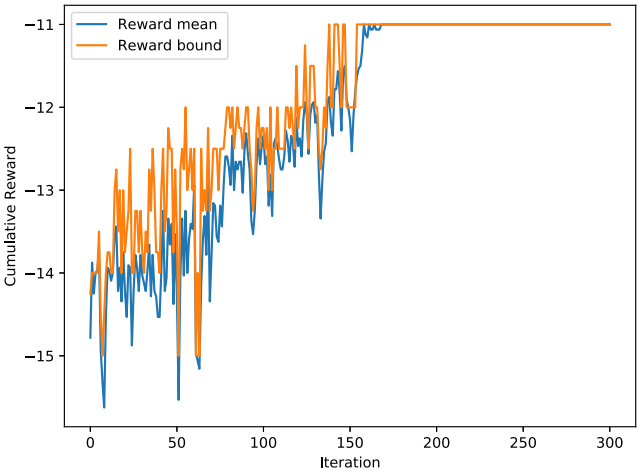

Fig. 5 The learning curve of *Raven* in datacenter with 9 physical machines and 30 virtual machines

virtual machines, as shown in Fig. 7, it seems difficult for *Raven* to converge. Figure 8 shows the learning curve of *Raven* within a datacenter of 260 physical machines and 520 virtual machines. The tendency of this curve is similar to the curve in Fig. 7. The cumulative rewards first gradually increase then start to bounce from a specific point but seem unlikely to converge.

As the number of physical machines and virtual machines becomes even larger, for example, 260 physical machines and 1158 virtual machines, the learning curve is as Fig. 9. It starts to decrease when it reaches a high cumulative reward and then jitters. In terms of convergence, this learning curve is not satisfactory. And the root cause behind it is the sparse reward of this scheduling problem. Before the last time step of a training episode, the reward of every time step is 0, which makes the agent hard to tell whether an action is good or not. To be more specific, for

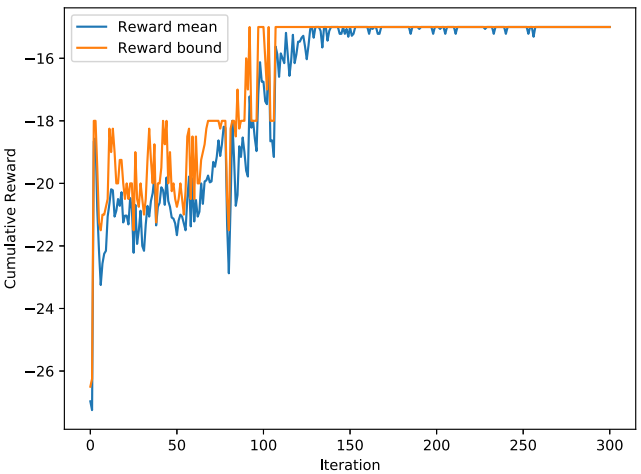

Fig. 6 The learning curve of *Raven* in datacenter with 10 physical machines and 40 virtual machines

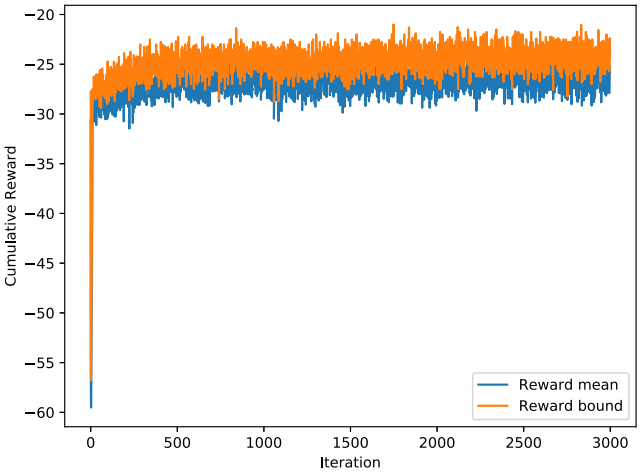

Fig. 7 The learning curve of *Raven* in datacenter with 50 physical machines and 100 virtual machines

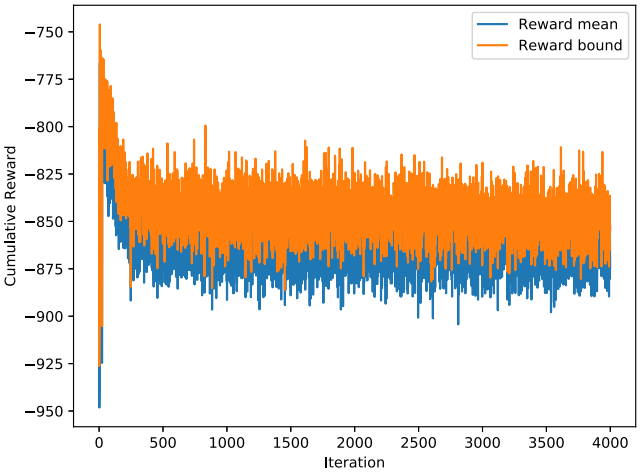

Fig. 9 The learning curve of *Raven* in datacenter with 260 physical machines and 1158 virtual machines

one training episode, only the reward of the last time step contributes to the cumulative reward, and the agent cannot learn much from the actions it takes before the last time step.

However, even when the convergence behaviour is not satisfying, we find that the agent can still generate schedules with shorter total migration time than the methods we compare to, which will be presented next.

Total migration time. In this part, we evaluate the schedule generated by *Raven*. Since we set the cumulative reward as the negative number of the total migration time, the negative number of the cumulative reward is the total migration time of the schedule generated by *Raven*.

We compute the total migration times of using Min-DIFF and the heuristic method for comparison. Both two-layer and three-layer network topologies are applied, as we mentioned at the beginning of this section.

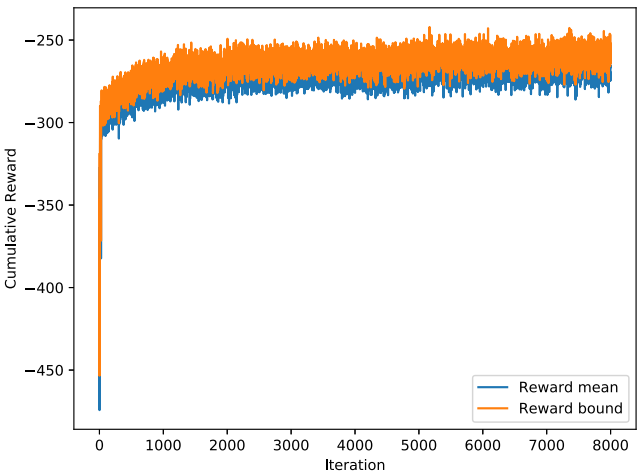

Fig. 8 The learning curve of *Raven* in datacenter with 260 physical machines and 520 virtual machines

The results are indicated in Table 1. We also calculate the percentage that *Raven* shortens the total migration time compared to Min-DIFF and the heuristic method. In this table, the number of aggregation switches indicates the network topology. If it is 0, the network topology is two-layer; otherwise the network topology is three-layer.

As we can see from this table, *Raven* is able to come up with schedules with shorter total migration time than that of the other two methods. And as the number of physical machines and virtual machines become larger, and the network topology becomes more complex, *e.g.,* with more aggregation switches, *Raven* performs even better than the other two methods. We believe that this is because *Raven* can learn the information of datacenter through its training and adaptively take different and possibly better actions in the next iterations. While the other two methods always generate deterministic schedules based on simple and unchanged rules. These rules may good for some datacenter settings but are bad for others. Therefore, they cannot outperform *Raven* under our simulated settings.

Test the trained model of *Raven*. Since we train *Raven* under one initial mapping of virtual machines to physical machines, it is necessary to test if the trained model can also work well under other initial mappings.

For each datacenter setting, we apply the trained model to ten different initial mappings, and average the total migration time of the schedule for each initial mapping. The results are shown in Table 2. The percentage of the total migration time reduced by *Raven* is the number in brackets.

From this table, we can see that *Raven* can achieve shorter migration time than the other two methods in most cases. As the network topology becomes more complex, and the number of physical machines and virtual machines increases, the superiority of *Raven* in terms of shortening

Table 1 Total migration time within different datacenters

Datacenter Setting			Total Migration Time		
Number of PMs	Number of VMs	Number of Aggregation Switches	Min-DIFF	Heuristic	*Raven*
50	100	0	35.56 (11%)	57.50 (45%)	31.50
50	100	2	130.5 (27%)	110.0 (14%)	94.00
50	150	0	55.50 (31%)	103.00 (63%)	38.00
50	150	2	168.00 (16%)	159.00 (11%)	140.50
100	200	0	43.50 (28%)	91.50 (66%)	31.00
100	200	2	227.00 (35%)	221.50 (34%)	146.00
100	300	0	65.50 (2%)	192.50 (66%)	63.99
100	300	2	339.00 (15%)	378.00 (24%)	286.00
260	520	6	791.00 (58%)	510.00 (34%)	332.14
260	520	7	681.00 (35%)	445.78 (1%)	440.15
260	520	8	651.25 (58%)	440.00 (39%)	268.00
260	1158	6	944.31 (7%)	924.42 (5%)	869.97
260	1158	7	903.00 (10%)	910.50 (11%)	805.85
260	1158	8	918.50 (15%)	870.95 (11%)	774.23

the total migration time becomes more obvious. This again indicates that it may be difficult for heuristic methods to handle the scheduling of virtual machine migration with minimal total migration time when the datacenter has large numbers of physical machines and virtual machines under complex network topology, while *Raven* can gradually learn to come up with schedules of shorter total migration time.

However, for some small number of physical machines and virtual machines and simple network topology, for example, with 50 physical machines, 100 physical machines and under the two-layer topology, *Raven* has slightly longer average total migration time than Min-DIFF. We check the total migration time of each initial mappings, as we conduct 10 different initial mappings here. It turns out that for most

Table 2 Average total migration time within different datacenter

Datacenter Setting			Average Total Migration Time		
Number of PMs	Number of VMs	Number of Aggregation Switches	Min-DIFF	Heuristic	*Raven*
50	100	0	36.20 (0%)	57.55 (36%)	36.39
50	100	2	117.00 (1%)	107.38 (-7%)	115.13
50	150	0	47.25 (-1%)	94.50 (49%)	47.94
50	150	2	172.95 (9%)	165.83 (5%)	157.00
100	200	0	53.05 (17%)	100.16 (56%)	43.54
100	200	2	248.24 (15%)	214.80 (2%)	209.14
100	300	0	66.80 (7%)	156.65 (60%)	61.96
100	300	2	335.95 (12%)	299.65 (1%)	296.32
260	520	6	725.55 (46%)	451.52 (14%)	388.94
260	520	7	667.70 (35%)	465.76 (7%)	433.17
260	520	8	622.65 (45%)	420.08 (19%)	338.65
260	1158	6	971.67 (12%)	1020.25 (17%)	846.06
260	1158	7	912.01 (14%)	987.62 (20%)	782.55
260	1158	8	836.54 (10%)	969.42 (22%)	748.93

initial mappings, *Raven* has shorter total migration time, while for the rest, Min-DIFF has shorter total migration time. This result is reasonable as the network topology here is simple for Min-DIFF and the heuristic method to handle. But we should be noted that this kind of small-scale datacenter with simple network topology is not that common in reality. Larger numbers of physical machines and virtual machines, for example, 260 physical machines and 1158 virtual machines, and the three-layer topology is more common and widely used in the real world. And under these datacenter settings, *Raven* outperforms the two other methods.

6 Related work

For a modern datacenter, providing users with stable and reliable cloud services is always one of its top priorities, and migrating virtual machines is a common, sometimes essential way to reach this goal [11]. Therefore, scheduling virtual machine migration, with various focuses and optimization goals, has been extensively studied in recent years.

Energy consumption is one of the critical factors stuied in previous work [1]. Borgetto et al. [12] formulated the virtual machine migration and physical machine power management as a NP-hard resource allocation problem, and designed a multi-level action approach to break down and solve this problem. Another energy-conscious migration strategy for datacenters was proposed by Maurya et al. [2]. This strategy considers higher and lower thresholds for virtual machine migration. Virtual machines will be migrated when the load is higher or lower than defined upper or lower thresholds. Luo et al. [3] devised a communication-aware allocation algorithm in which virtual machines with high traffic volumes are preferred to be migrated. As this algorithm reduces the data traffic, energy consumption of datacenters could be decreased.

Another optimization goal in the literature is to maximize resource utilization. Garg et al. [13] proposed a virtual machine scheduling strategy to maximize the utilization of cloud datacenter and allow the execution of heterogeneous application workloads. Hieu et al. [14] designed a virtual machine management algorithm that can maximize the resource utilization and balance the usage of different resources. This algorithm is based on multiple resource-constraint metrics to find the most suitable physical machine for deploying virtual machines in large cloud datacenters.

Virtual machine migration is also studied in the field of over-committed cloud datacenter [4] [15] [16]. Within this kind of datacenter, the service provider allocates more resources to virtual machines than it has to reduce resource wastage, as studies indicated that virtual machines tend to utilize fewer resources than reserved capacities. Therefore, it is necessary to migrate virtual machines when the hosting physical machines reach their capacity limitations. Ji et al. [4] proposed a virtual machine migration algorithm that can balance the usage of different resources on activated physical machines and also minimize the number of activated physical machines in an over-committed cloud. Dabbagh et al. [16] designed an integrated energy-efficient prediction-based virtual machine placement and migration framework for cloud resource allocation with overcommitment. This framework reduces the number of active physical machines and decreases migration overheads, and thus makes significant energy savings.

7 Future work

This paper proposes a virtual machine migration scheduler which aims at minimizing the total migration time. However, other metrics such as load balancing and energy efficiency could also affect the performance of virtual machines and physical machines in a datacenter. In future work, we may study and design other schedulers to optimize these metrics.

Moreover, our work focuses on cold migration, where the virtual machine is shut-off before its migration. There is another type of virtual machine migration, live migration, which is also widely used in the real world. In live migration, the virtual machine keeps running during its migration. Therefore, the migration would involve an iterative copy of dirty pages in successive rounds. This is different from the cold migration, which only needs a one-time copy. We are also interested in live migration and may explore the scheduling problem of this kind of virtual machine migration in the future.

8 Conclusion

In this paper, we study a scheduling problem of virtual machine migration during the datacenter upgrade without a priori knowledge of the topology and network capacity of the datacenter network. We find that for this specific scheduling problem which is rarely studied before, it is difficult for previous schedulers of virtual machine migration using heuristics to reach the optimal total migration time.

Inspired by the success of applying deep reinforcement learning in recent years, we develop a new scheduler, *Raven*, which uses an experience-driven approach with deep reinforcement learning to decide the destination physical machine for each migrating virtual machine with the objective of minimizing the total migration time it takes to

complete the datacenter upgrade. With our careful design of the state space, action space and reward function, *Raven* learns to generate schedules with the shortest possible total migration time by interacting with the unknown environment.

Our extensive simulation results show that *Raven* is able to outperform existing scheduling methods under different datacenter settings with various number of physical machines and virtual machines and different network topologies. Although it is difficult for *Raven* to converge under some datacenter settings during training due to the sparse reward problem, it can still come up with schedule with shorter total migration time than that of the existing methods.

References

1. Beloglazov A, Abawajy J, Buyya R (2012) Energy-Aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing. Future generation computer systems 28(5):755–768
2. Maurya K, Sinha R (2013) Energy conscious dynamic provisioning of virtual machines using adaptive migration thresholds in cloud data center. International Journal of Computer Science and Mobile Computing 2(3):74–82
3. Luo J, Fan X, Yin L (2019) Communication-Aware and Energy Saving Virtual Machine Allocation Algorithm in Data Center. In: 21st IEEE International Conference on High Performance Computing and Communications. IEEE, pp 819–826
4. Ji S, Li MD, Ji N, Li B (2018) An online virtual machine placement algorithm in an over-committed cloud. In: 2018 IEEE International Conference on Cloud Engineering, IC2E 2018, Orlando, FL, USA, April 17-20, 2018, pp 106–112
5. Sutton RS, Barto AG (1998) Reinforcement learning: An introduction. MIT Press Cambridge, MA, USA.
6. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D (2015) Human-level control through deep reinforcement learning. Nature 518:529–533
7. Mao H, Alizadeh M, Menache I, Kandula S (2016) Resource management with deep reinforcement learning. In: Proceedings of the 15th ACM Workshop on Hot Topics in Networks, ACM, pp 50–56
8. Silver D, Huang A, Maddison CJ et al (2016) Mastering the game of go with deep neural networks and tree search. nature 529(7587):484–489
9. Silver D, Schrittwieser J, Simonyan K, Antonoglou I et al (2017) Mastering the game of go without human knowledge. Nature 550:354–359
10. Rubinstein RY, Kroese DP (2004) The cross-entropy method. Springer New York, NY, USA.
11. Ahmad RW, Gani A, Hamid SHA, Shiraz M, Xia F, Madani SA (2015) Virtual Machine Migration in Cloud Data Centers: a Review, Taxonomy, and Open Research Issues. The Journal of Supercomputing 71(7):2473–2515
12. Borgetto D, Maurer M et al (2012) Energy-Efficient and SLA-Aware Management of IaaS Clouds. In: Proceedings of the 3rd International Conference on Energy-Efficient Computing and Networking, Madrid, Spain, May 9-11, 2012. ACM, p 25
13. Garg SK, Toosi AN, Gopalaiyengar SK, Buyya R (2014) SLA-Based Virtual Machine Management for Heterogeneous Workloads in a Cloud Datacenter. J. Netw. Comput. Appl. 45:108–120
14. Hieu NT, Francesco MD et al (2014) A Virtual Machine Placement Algorithm for Balanced Resource Utilization in Cloud Data Centers. In: 2014 IEEE 7th International Conference on Cloud Computing, Anchorage, AK, USA, June 27 - July 2, 2014. IEEE Computer Society, pp 474–481
15. Zhang X, Shae Z-Y, Zheng S, Jamjoom H (2012) Virtual Machine Migration in an Over-Committed Cloud. In: Proc. IEEE Network Operations and Management Symposium (NOMS)
16. Dabbagh M, Hamdaoui B, Guizani M, Rayes A (2015) Efficient Datacenter Resource Utilization Through Cloud Resource Overcommitment. In: Proc. IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)