Scheduling Virtual Machine Migration During Datacenter Upgrades with Reinforcement Learning

Chen Ying[1(✉)], Baochun Li[1], Xiaodi Ke[2], and Lei Guo[2]

[1] University of Toronto, Toronto, Canada
{chenying,bli}@ece.toronto.edu
[2] Huawei Canada, Markham, Canada
{xiaodi.ke,leiguo}@huawei.com

Abstract. Physical machines in modern datacenters are routinely upgraded due to their maintenance requirements, which involves 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, we propose a new scheduler, *Raven*, that uses an experience-driven approach with deep reinforcement learning to schedule the virtual machine migration process. 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 a choosing destination physical machine for each migrating virtual machine. We compare *Raven* with state-of-the-art heuristic algorithms in the literature, and our results show that *Raven* effectively leads to shorter time to complete the datacenter upgrade process.

Keywords: Reinforcement learning · Virtual machine migration

1 Introduction

In modern datacenters, it is routine for physical machines to be upgraded to newer versions of operating systems or firmware versions from time to time, as part of their maintenance process. 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.

X. Chu et al. (Eds.): QShine 2019, LNICST 300, pp. 102–117, 2020.
https://doi.org/10.1007/978-3-030-38819-5_7

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 carefully plan the schedule of migrating virtual machines. 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 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. It is common that existing work only migrated a small number of virtual machines to reduce the energy consumption of physical machines [1], or to balance the utilization of resources across physical machines [2]. 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 [3], which trains a deep neural network agent to learn the policy of making 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 [4], whether it is suitable for scheduling resource, 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 process 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 in 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, 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 existing work. *Second*, we design the state space, action space, and reward function for our deep reinforcement learning agent to schedule the migration process 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 heuristic methods with respect to minimizing the total migration time it takes to complete the datacenter upgrade, without any *a priori* knowledge of the datacenter network.

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 also 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 main memory of VMs on a PM can be accommodated by the total number of physical CPUs and the total size of physical memory on its hosting PM.

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 sizes of these PMs and VMs, with respect to the number of CPUs and the amount of memory, are shown in Fig. 1. We will upgrade these PMs 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 between the core switch to each PM is 1 GB/s.

In the first schedule, as shown in Fig. 1, we first upgrade PM #0 and migrate VM #0 to PM #2, and then upgrade PM #1 by migrating VM #1 to PM #0. Finally, we upgrade PM #2 with migrating VM #0 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 #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, here we cannot migrate VM #1 and VM #2 whose destination PM is PM #0. Since only VM #0 is

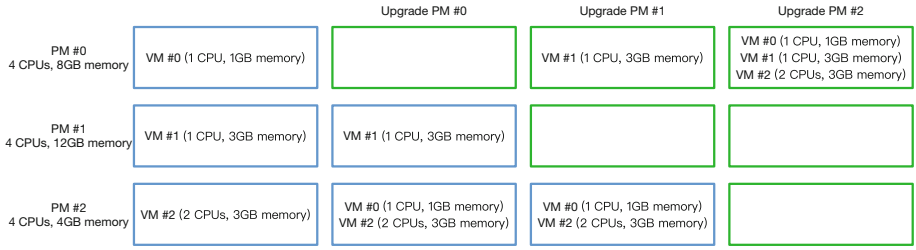


Fig. 1. Scheduling VM migration: a schedule that takes 10 s of total migration time.

being migrated, it can occupy all the network capacity through the link during its migration. Because the image size of VM #0 is 1 GB, this migration takes $1\,\text{GB}/(1\,\text{GB/s}) = 1\,\text{s}$.

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 #0;
♢ VM #2: PM #2 → PM #0.

Actually, all these three migration processes can be processed at the same time, because (1) for these three migration processes, no source PM is the same as any destination PMs and no destination PM is the same as any source PMs and (2) PM #0 has enough residual CPUs and memory to host these three VMs at the same time. Therefore, we can treat these three migration processes as a *migration batch*. The migration time of a migration batch is determined by the longest migration process in this batch. Since these three VMs share the link between the core switch and PM #0, each of them will get $\frac{1}{3}$ GB/s. The migration time of VM #0 is $1\,\text{GB}/(\frac{1}{3}\,\text{GB/s}) = 3\,\text{s}$. The migration time of VM #1 is $3\,\text{GB}/(\frac{1}{3}\,\text{GB/s}) = 9\,\text{s}$, which is the same as the migration time of VM #2. Thus, the migration time of this migration batch is 9 s. So the total migration time is $1\,\text{s} + 9\,\text{s} = 10\,\text{s}$.

In contrast, a better schedule in terms of reducing the total migration time is shown in Fig. 2. 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 can not be in the current migration batch. For the

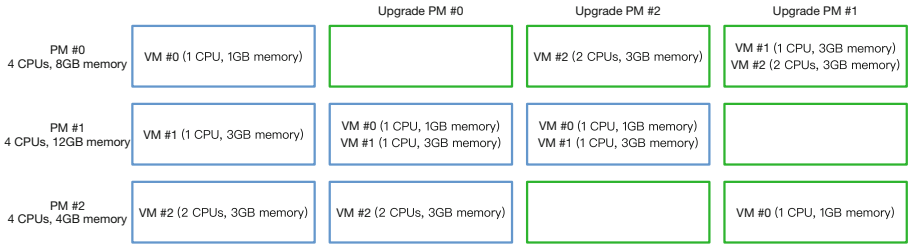


Fig. 2. Scheduling VM migration: a schedule that takes 8 s of total migration time.

same reason, the migration of VM #2 can not be in this batch either. Therefore, the migration time of this migration batch which only contains the migration of VM #0 from PM #0 to PM #1 is $1\,\text{GB}/(1\text{GB/s}) = 1\,\text{s}$. Then we come to VM #2 because it is on PM #2 which is upgraded next. In this migration batch, we can also have the migration of VM #1 from PM #1 to PM #0. These two VMs share the link between the core switch to PM #0 with each of them having $\frac{1}{2}$ GB/s. The migration time of this batch is $3\,\text{GB}/(\frac{1}{2}\text{GB/s}) = 6\,\text{s}$. At last, we compute the migration time of VM #0 from PM #1 to PM #2, which is 1 s. Therefore, the total migration time of this schedule is $1\,\text{s} + 6\,\text{s} + 1\,\text{s} = 8\,\text{s}$.

As we can observe from our example, 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 the topology and network capacities of the datacenter network is 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. 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 we advance to our proposed work in the *Raven* scheduler, 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 [3]. At each time step t, the agent observes the state s_t of the environment 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 then a new episode begins. 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 in the episode. Cumulative reward $R(x) = \sum_{t\in[T]} r_t$ measures how good the trajectory is by summing up the rewards received at each time step during this episode.

As the agent's behavior is defined by a policy $\pi(a_t = a|s_t = s)$, which maps state s to a probability distribution over all actions $a \in \mathcal{A}$, how to store the state-action pairs is an important problem of reinforcement learning. Since the number of state-action pairs of 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 [4–6]. One significant advantage of deep neural networks is that they do not need handcrafted features. 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 mapping a migrating VM to a destination PM, an optimal policy $\pi(a|s;\theta^*)$ with parameters θ^* is the mapping strategy we want to obtain.

To obtain the parameters θ^* of an optimal policy, we could use a basic but efficient method, cross-entropy [7], whose objective is to maximize the reward $R(x)$ received by 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 $\mathbf{1}_{\{R(x)\geq\xi\}}$ is the indicator function, that is, $\mathbf{1}_{\{R(x)\geq\xi\}} = 1$ 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

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

The optimal importance sampling probability for a fixed level ξ is given by

$$f(x;v^*) \propto |\mathbf{1}_{\{R(x)\geq\xi^\}}| f(x;u), \tag{2}$$

which is generally difficult to obtain. Thus the idea of the cross-entropy method is to choose the importance sampling probability density $f(x;v)$ in a specified class of densities such that the distance between the optimal importance sampling density $f(x;v^*)$ and $f(x;v)$ is minimal. The distance $D(f_1, f_2)$ between two

probability densities f_1 and f_2 could be measured by the Kullback-Leibler (KL) divergence which is defined as follows:

$$\begin{aligned}\mathcal{D}(f_1, f_2) &= \mathbb{E}_{x\sim f_1(x)}\left[\log \frac{f_1(x)}{f_2(x)}\right] \\ &= \mathbb{E}_{x\sim f_1(x)}\big[\log f_1(x)\big] - \mathbb{E}_{x\sim f_1(x)}\big[\log f_2(x)\big],\end{aligned} \tag{3}$$

where the first term $\mathbb{E}_{x\sim f_1(x)}\big[\log f_1(x)\big]$ is called entropy, which does not reflect the distance between $f_1(x)$ and $f_2(x)$ and could be omitted during the minimization, while the second term $-\mathbb{E}_{x\sim f_1(x)}\big[\log f_2(x)\big]$ is called *cross-entropy*, which is a common optimization objective in deep learning. It turns out that the optimal parameters v^* is the solution to the maximization problem

$$\max_v \int \mathbf{1}_{\{R(x)\geq\xi\}} f(x;u)\log f(x;v)\mathrm{d}x, \tag{4}$$

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

$$\hat{v} = \arg\max_v \frac{1}{N}\sum_{n\in[N]} \mathbf{1}_{\{R(x_n)\geq\xi\}} \frac{f(x_n;u)}{f(x_n;w)}\log f(x_n;v), \tag{5}$$

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

At the beginning of the deep neural network training, the parameters $u = \hat{v}_0$ are initialized randomly. By sampling with the current importance sampling distribution in each iteration k, we create a sequence of levels $\hat{\xi}_1, \hat{\xi}_2, \cdots$ which converges to the optimal performance ξ^*, and the corresponding sequence of parameter vectors $\hat{v}_0, \hat{v}_1, \cdots$ which converges to the optimal parameter vector. 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^*. Typically, a smoothed updating rule with a smoothing parameter α is used, in which the parameter vector $\tilde{v}_k$ within the importance sampling density $f(x;v)$ after k-th iteration is $\tilde{v}_k = \alpha\hat{v}_k + (1-\alpha)\tilde{v}_{k-1}$.

The probability of a trajectory $x \in \mathcal{X}$ 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. Therefore, the parameter estimator at iteration k could be represented as

$$\hat{\theta}_k = \arg\max_{\theta_k} \sum_{n\in[N]} \mathbf{1}_{\{R(x_n)\geq\xi_k\}}\bigg(\sum_{a_t,s_t\in x_n} \pi(a_t|s_t;\theta_k)\bigg), \tag{6}$$

where $x_1, \cdots, x_N$ are sampled from policy $\pi(a|s; \tilde{\theta}_{k-1})$, and $\tilde{\theta}_k = \alpha\hat{\theta}_k + (1 - \alpha)\tilde{\theta}_{k-1}$. The Eq. (6) 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, the action space, and the reward function.

4.1 Architecture

Figure 3 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 the upgrade process of 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 which indicates whether the action a_t is good or not 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 decide a reward boundary, and train 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.

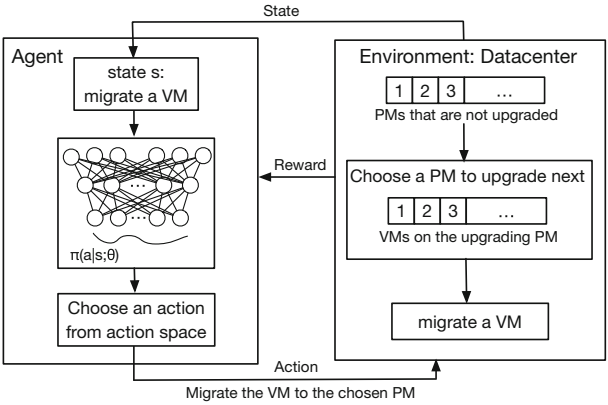


Fig. 3. The architecture of *Raven*.

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 $\#j$, $j = 0, 1, \ldots, J-1$, as $s_{tj} = \{s_{tj}^{\text{status}}, s_{tj}^{\text{total-cpu}}, s_{tj}^{\text{total-mem}}, s_{tj}^{\text{used-cpu}}, s_{tj}^{\text{used-mem}}\}$, 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

$$s_t = \{s_{t0}, s_{t1}, \ldots, s_{t(J-1)}, v_t^{\text{cpu}}, v_t^{\text{mem}}, v_t^{\text{pm-id}}\}, \tag{7}$$

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 $\mathcal{A} = \{0, 1, \ldots, J-1\}$, this setting has two major problems that (1) the destination PM may not have enough residual resources for the migrating VM and (2) the destination PM may be the source PM of the migrating VM. To avoid these two problems, we dynamically change the action space for 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 can be determined. We denote the

set of indexes of PMs in this subset as $\mathcal{A}_t^{\text{eligible}}$. The action space at this time step will be $\mathcal{A}_t^{\text{eligible}}$.

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 $a_t = n$, then we have the state of PM $\#m$ at time step $t+1$ as

$$s_{(t+1)m} = \{\text{upgraded}, s_{tm}^{\text{total-cpu}}, s_{tm}^{\text{total-mem}}, s_{tm}^{\text{used-cpu}} - v_t^{\text{cpu}}, s_{tm}^{\text{used-mem}} - v_t^{\text{mem}}\}, \tag{8}$$

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

$$s_{(t+1)n} = \{s_{tn}^{\text{status}}, s_{tn}^{\text{total-cpu}}, s_{tn}^{\text{total-mem}}, s_{tn}^{\text{used-cpu}} + v_t^{\text{cpu}}, s_{tn}^{\text{used-mem}} + v_t^{\text{mem}}\}. \tag{9}$$

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

$$s_{t+1} = \{s_{t0}, \ldots, s_{(t+1)m}, \ldots, s_{(t+1)n}, \ldots, s_{t(J-1)}, v_{t+1}^{\text{cpu}}, v_{t+1}^{\text{mem}}, v_{t+1}^{\text{pm-id}}\}, \tag{10}$$

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 to minimize the total migration time for each migrating VM. 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 are conducted in a different order which is not determined until we finish scheduling all the migrations, as we have discussed in Sect. 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 actually 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 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.

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.

Here we compare *Raven* with the state-of-the-art virtual machine migration scheduler, Min-DIFF [2], which uses a heuristic based method to balance the usage of different kinds of resources on each physical machine. We also come up with a simple heuristic method to minimize the total migration time.

5.2 Simulation Results

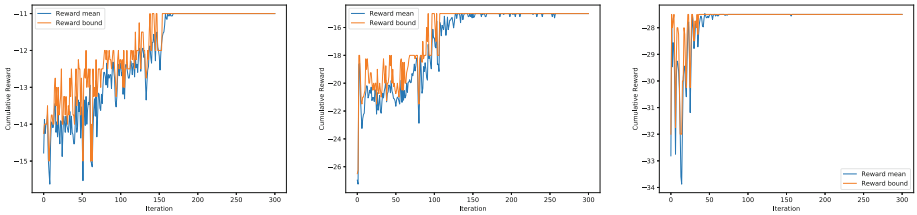
Convergence Behaviour. The convergence behaviour of a reinforcement learning agent is a useful indicator to show that if the agent successfully learns the policy or not. A preferable convergence behaviour is that the cumulative rewards can gradually increase through the training iterations and converge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

We plot the figures of number of training iterations and cumulative rewards in different datacenters with 3, 5, 10 physical machines as Figs. 4, 5 and 6, respectively. The network topology here is that all PMs are connected to the same core switch.

As shown in these figures, *Raven* is able to learn the policy of scheduling the virtual machine migration in datacenter with various number of physical machines and virtual machines. As the number of total physical machines and virtual machines increases, the agent normally needs more iterations to converge, which is reasonable since the environment becomes more complex for the agent to learn.

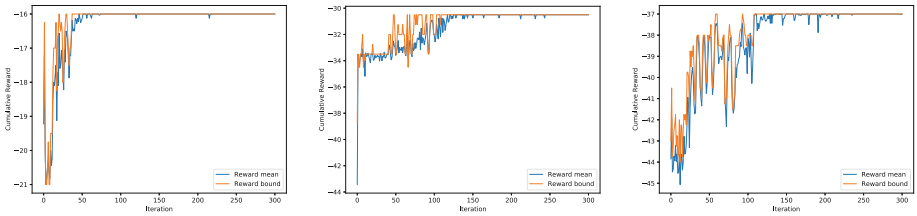
However, in Fig. 4, we find that it takes fewer iterations to converge for a datacenter with 12 virtual machines than for a datacenter with 6 virtual machines or 9 virtual machines. This is due to the randomness of the initial mapping between physical machines and virtual machines before the upgrade process. We can see that some initial mappings are easier to learn for the deep reinforcement learning agent that the others.

The same situation happens when there are 10 physical machine in Fig. 5. It takes longer to converge for a datacenter with 30 virtual machines than for a datacenter with 40 virtual machines. Also, it seems difficult to converge when there are 30 virtual machines. But we find that the agent can still generate schedules with shorter total migration time than the methods we compare with before it converges, which will be presented next.



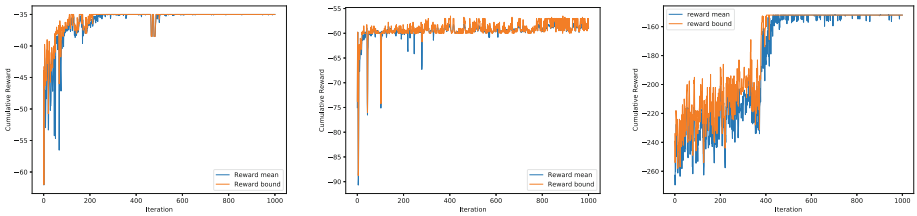
(a) 6 virtual machines. (b) 9 virtual machines. (c) 12 virtual machines.

Fig. 4. The learning curve of *Raven* in datacenter with 3 physical machines.



(a) 10 virtual machines. (b) 15 virtual machines. (c) 20 virtual machines.

Fig. 5. The learning curve of *Raven* in datacenter with 5 physical machines.



(a) 20 virtual machines. (b) 30 virtual machines. (c) 40 virtual machines.

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

Total Migration Time. Even though we demonstrate that *Raven* is able to learn the scheduling policy of virtual machine migration, we still need to evaluate if this scheduling policy can generate the schedule that reduces the total migration time. 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. Different network topologies are also applied. Besides the two-layer network topology where all the physical machines are connected to the same core switch, we also have the three-layer network topology where physical machines are connected to different aggregation switches and all aggregation

switches are connected to the same core switch. If the number of aggregation switches is 0, it means that the network topology is two-layer.

As we have discussed that the initial mapping between virtual machines and physical machines before the datacenter upgrade will affect the result, we conduct the simulation of each datacenter setting for 10 times and show the average total migration time in Table 1.

From Table 1, we can see that *Raven* is able to achieve shorter migration time than the other two schedulers. As the network topology becomes more complex and the number of physical machines and virtual machines increases below a certain number, *i.e.,* 50 physical machines, the superiority of *Raven* in terms of shortening the total migration time becomes more obvious. This indicates

Table 1. 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*
3	6	0	13.00	12.95	12.20
3	6	2	32.05	32.05	31.35
3	9	0	18.60	18.45	17.54
3	9	2	51.50	51.55	47.10
3	12	0	28.35	28.85	27.65
3	12	2	73.00	73.25	70.95
3	12	3	85.55	85.00	81.35
5	10	0	18.35	17.55	16.25
5	10	2	48.35	52.35	46.25
5	15	0	32.30	32.15	31.80
5	15	2	59.80	60.85	59.05
5	15	3	61.80	64.60	52.75
5	20	0	39.85	38.05	36.75
5	20	2	98.65	104.60	92.50
5	20	3	116.45	115.35	100.05
5	20	4	133.75	130.65	119.60
10	20	0	39.00	39.50	38.20
10	30	0	59.85	58.35	53.25
10	40	0	76.60	75.25	72.80
20	100	0	195.00	205.00	175.50
50	100	0	198.60	198.50	216.25
50	150	0	298.50	296.50	377.25
100	200	0	416.85	430.30	403.50
100	300	0	598.75	598.75	796.75

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 number 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.

The results of 10 physical machines are the total migration times generated at the 1000th training iteration. Those results indicate that even when it may take a large number of iterations for *Raven* to converge its cumulative reward, *i.e.,* when there are 30 virtual machines, the schedule that *Raven* generates before the convergence can still outperform the other two methods.

However, when the number of physical machines becomes 50, we find that it is difficult for *Raven* to learn a schedule that has shorter total migration time than the other two methods. The cumulative rewards can gradually increase through training iterations, but it is hard to converge to a high value. One possible reason could be that so far we have only used fully-connected neural networks as policy approximators. It might be easier for the agent to converge when more complex and powerful deep neural networks are used. It should be noted that under the datacenter setting of 100 physical machines and 200 virtual machines, *Raven* outperforms again, which again shows the effect of the randomness of the initial mapping between physical machines and virtual machines to the result.

6 Related Work

Within a datacenter, it is common to process virtual machine migration due to various reasons, such as for the maintenance of the hosting physical machines or for the load balancing among physical machines in the datacenter. To ensure high quality of service of the datacenter, how to efficiently schedule the virtual machine migration to achieve different objectives has been extensively studied.

In order to reduce the amount of data transferred over the network during the virtual machine migration, Sapuntzakis *et al.* [8] designed a capsule-based system architecture, *Collective*.

To reduce the number of migration, Maurya *et al.* [1] proposed a minimum migration time policy which is capable of reducing the number of migration and the energy consumption of virtual machine migration proceeded to optimize resource usage and lower energy consumptions of physical machines in the datacenter.

Virtual machine migration is also studied in the field of over-committed cloud datacenter [2,9,10]. Within this kind of datacenter, the service provider allocates more resources to virtual machines than it actually has to reduce resource wastage, as study 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 it is capacity limitation. Ji *et al.* [2] proposed a virtual machine migration algorithm which 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.

7 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 to complete the datacenter upgrade. With our careful design of the state space, the action space and the 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 heuristic scheduling methods under different datacenter settings with various number of physical machines and virtual machines and different network topology. However, as the number of physical machines and virtual machines becomes large, it is difficult for *Raven* to converge and outperform other methods. We discuss the possible reasons behind it and will improve it in our future work.

References

1. Maurya, K., Sinha, R.: Energy conscious dynamic provisioning of virtual machines using adaptive migration thresholds in cloud data center. Int. J. Comput. Sci. Mob. Comput. **2**(3), 74–82 (2013)
2. Ji, S., Li, M.D., Ji, N., Li, B.: An online virtual machine placement algorithm in an over-committed cloud. In: 2018 IEEE International Conference on Cloud Engineering, IC2E 2018, Orlando, FL, USA, 17–20 April 2018, pp. 106–112 (2018)
3. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, vol. 1, no. 1. MIT Press, Cambridge (1998)
4. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015)
5. Mao, H., Alizadeh, M., Menache, I., Kandula, S.: Resource management with deep reinforcement learning. In: Proceedings of the 15th ACM Workshop on Hot Topics in Networks, pp. 50–56. ACM (2016)
6. Silver, D., et al.: Mastering the game of go without human knowledge. Nature **550**, 354–359 (2017)
7. Rubinstein, R., Kroese, D.: The Cross-Entropy Method. Springer, Heidelberg (2004). https://doi.org/10.1007/978-1-4757-4321-0
8. Sapuntzakis, C.P., Chandra, R., Pfaff, B., Chow, J., Lam, M.S., Rosenblum, M.: Optimizing the migration of virtual computers. In: 5th Symposium on Operating System Design and Implementation (OSDI 2002), Boston, Massachusetts, USA, 9–11 December 2002 (2002)

9. Zhang, X., Shae, Z.-Y., Zheng, S., Jamjoom, H.: Virtual machine migration in an over-committed cloud. In: Proceedings of the IEEE Network Operations and Management Symposium (NOMS) (2012)
10. Dabbagh, M., Hamdaoui, B., Guizani, M., Rayes, A.: Efficient datacenter resource utilization through cloud resource overcommitment. In: Proceedings of the IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) (2015)