A Unified Approach to Virtual Machine Placement and Migration in the Cloud using Deep Reinforcement Learning

Zihao Yang[a], Muhammed Tawfiqul Islam[b], Aaron Harwood[c]

[a]*The University of Melbourne, Grattan Street, Parkville, 3010, Victoria, Australia*
[b]*The University of Melbourne, Grattan Street, Parkville, 3010, Victoria, Australia*
[c]*Streamsoft Technologies Pty Ltd, , Coburg, 3058, Victoria, Australia*

Abstract

Efficient virtual machine (VM) placement in cloud data centres is crucial for delivering real-time services, optimizing resource utilization, and ensuring energy conservation. While there has been extensive research on VM placement and migration to achieve goals like energy efficiency and load balancing, existing solutions often overlook the integration of both processes into a single model. Such an integrated approach could eliminate strategy conflicts and enhance overall performance.

In this paper, we formulate the online VM placement problem as a unified optimization model that facilitates both initial VM placement and VM migration. We propose a Reinforcement Learning (RL) model for the VM placement problem and provide an RL environment implementation. We also devise an approach to maximize physical machine (PM) utilization while maintaining balance based on Deep Reinforcement Learning. The evaluation results show that our approach with the Proximal Policy Optimization algorithm outperforms baselines, achieving a 0.8-3.4% higher PM utilization rate in a cluster with 100 machines and a high system load.

Keywords:
virtual machine placement, cloud computing, deep reinforcement learning

1. Introduction

Cloud computing is one of the most significant developments in information technology and is widely used across many industries. Nowadays, most online software applications are hosted on virtual machines (VMs) in the cloud. Through virtualization, cloud providers offer computing services to end-users using shared hardware resources over the Internet. The cloud infrastructure automatically instantiates and allocates VMs to users upon request. The online VM placement problem aims to pack the VMs into physical machines (PMs) following a continuous sequence of requests. An effective VM placement strategy may achieve specific goals, such as minimizing energy consumption, maintaining load balancing, reducing service-level agreement (SLA) violations, and reducing network latency. As the system scales up, improving the VM placement strategy can lead to significant progress toward achieving these goals.

The VM placement problem can be either online or offline, where the online problem requires VMs to be placed while new VM requests are continuously generated, and the offline problem has the entire VM request sequence beforehand. This work focuses on the online VM placement problem in the cloud environment, which can be formulated as a bin packing problem, widely known as an NP-hard problem [1]. In the cloud environment, it is also difficult to maintain an optimal VM placement for the long term. As most VMs are long-running, their placement could be further optimized by VM migrations. During the migration phase, the placement of VMs is adjusted to utilize the PMs better, allowing more VMs to be served in the cluster. Although VM placement optimization has been extensively researched, most works have only focused on either VM placement or VM migration [2–11]. The works that consider both VM placement and migration provide separate policies for initial placement and migration based on heuristics or event triggers [12–16].

Separate policies for placement and migration have different objectives so there could be decision conflicts, which leads to suboptimal hardware resource utilization and VM performance degradation. For example, the placement policy may request to place a new VM on a PM while the migration policy is migrating another VM to the same PM. In this case, one of the policies will fail if the PM is only available for one more VM.

To effectively address the aforementioned challenges, we propose a unified approach that integrates VM placement and migration into a single model based on deep reinforcement learning. As reinforcement learning (RL) can be seen as an automatic search for the optimal heuristics, it can often perform better than hand-crafted heuristics in combinatorial optimization problems [17]. Deep reinforcement learning (DRL) enhances this capability by modelling complex environments and

*Corresponding Author
Email addresses: zihao.yang@outlook.com.au (Zihao Yang), tawfiqul.islam@unimelb.edu.au (Muhammed Tawfiqul Islam), aaron@streamsoft.tech (Aaron Harwood)

decision-making policies using artificial neural networks as a function approximator [18]. Although convex optimization is commonly used for this type of problem, RL is often better suited to managing dynamic workloads. In our approach, we train a DRL agent to optimize hardware utilization while keeping the load balanced across the PMs. This agent is capable of selecting the most suitable PMs for VM placement and dynamically orchestrating migrations within a simulated cloud environment.

The main **contributions** of this paper are as follows:

- We formulate the problem into an optimization model, which allows for both initial placement and migrations.
- We devise a DRL-based VM placement and migration method.
- We provide an RL environment model implementation that is extensible to any placement and migration method.
- In the experimental evaluation, we demonstrate that our method outperforms the baseline algorithms in an environment with frequent VM requests, long VM service length, and large VM size.
- We conduct a sensitivity analysis of the environment parameters.

The rest of the paper is organised as follows. Section II summarises the related works. Section III describes our VM placement and migration problem formulation. Section IV presents the RL model designed for the problem. Section V evaluates the proposed method. Finally, Section VI concludes the paper and discusses future work.

2. Related Works

This section provides an overview of the relevant literature related to the VM placement problem.

2.1. The VM placement Problem

Online VM placement can be seen as a more complicated variant of the bin packing problem [3]. In the classical setting of this problem, the goal is to fit a list of fixed-size items into a list of fixed-capacity bins, while online VM placement has variable-size volumes arrive in a sequence, and the decision is where to pack them, and VMs will terminate and spare the resources.

The bin packing heuristics such as First Fit and Best Fit are commonly applied to optimize VM placement because they are simple and efficient. For instance, Bobroff et al. [19] proposed a server consolidation algorithm that combined time series forecasting and the First Fit heuristic. The consolidation algorithm can minimize the number of PMs by migrating servers into selected PMs. Shi and Hong [20] also used First Fit for VM packing to maximize the profit under the SLA. As Best Fit is a greedy approach, it may be beneficial in the case where VMs are better placed close. For example, Dong et al. [21] developed a method that places VMs that have large network traffic on the same PM or the same switch so that network congestion can be managed efficiently.

In the VM placement problem, there are many challenges other than fitting VMs into PMs, such as reducing energy consumption, the number of migrations, SLA violations, and network congestion [22]. Business cost, carbon emission, and quality of service are embedded in the aforementioned factors. Energy consumption and costs are related to optimizing the number of PMs running. VM migration may degrade the performance and incur overheads in available resources, while an optimal number and time of migrations can help reduce the overheads. SLA violations are downtime or performance degradation caused by migrations or resource contention, which can be a consequence of too many VMs on the same PM. Thus, inappropriate placement of VMs can cause significant problems in cloud operation.

Due to the complexity of dynamic resource utilization in data centres, VMs often need to be moved between PMs to maintain load balance so that no PMs are constantly over-utilized or under-utilized. However, VM migration could be expensive due to extra resource requests, performance degradation, and possible SLA violations [23]. Live migration is widely used in data centres to minimize service interruptions to an acceptable rate [24]. Large-scale live migrations may incur less than 300 milliseconds of downtime for most VMs, minimizing the impact on end users [25]. There are two problems in the task of VM migration: selecting the VM to migrate and selecting the migration time. Automatic migration can be triggered at specific threshold levels of resource usage [26–29]. Borgetto et al. [30] devised a framework of an automatic control loop to reallocate VMs, where Monte Carlo and vector packing algorithms are used to find a set of migrations iteratively based on the criteria of host utilization and migration jobs. However, the Monte Carlo method is unsuitable for large-scale systems as it will require too many iterations to find a better placement.

Notably, a multi-objective relaxed convex optimization method by Duong-Ba et al. [31] addressed both VM placement and migration. In contrast, our approach exploits the capabilities of RL, which can better handle non-stationary workloads compared to the static nature of convex optimization. Moreover, RL focuses on maximizing cumulative reward over time, so it inherently seeks the best long-term outcomes by continuously refining its decision-making strategies in response to operational conditions.

2.2. VM Placement based on RL

Reinforcement learning (RL) has been extensively researched and applied to decision-making problems in cloud computing. It provides a way to infer the optimal action based on the data or interactions with the cloud environment. SARSA (State-action-reward-state-action) [32] and Q-learning [33] are two common model-free tabular RL algorithms in the category of temporal difference learning. While SARSA is on-policy, Q-learning is an off-policy method [34]. The on-policy algorithms learn from the actions sampled from the current policy, while the off-policy algorithms learn from different actions. Based on

SARSA, Shaw et al. [35] developed an algorithm to allocate a list of VMs iteratively. Given a list of VM requests, the algorithm observes the utilization of hosts and selects a host to place each VM to achieve an optimal load balance. The agent is rewarded with fewer SLA violations and a lower energy consumption rate. Compared to the baseline policy, the algorithm significantly reduced energy consumption, VM migrations, and SLA violations. The model is also designed to work with Q-learning in their later work [8], which led to better performance.

VM placement becomes more difficult if formulated as a multi-objective problem. Long et al. [5] proposed a VM placement algorithm to optimize energy consumption and performance loss caused by resource contention, where both objectives appear conflicting in the factor of hardware utilization rate. As resource contention occurs when a PM runs too many VMs, there is a trade-off between high hardware utilization and low resource contention. The trade-off is controlled via two coefficients in its energy model. While it can be difficult to choose appropriate coefficients on multiple objectives, the concept of the Pareto set can be used to restrict the range of coefficients. An example is the Q-learning VM placement method, VMPMORL, developed by Qin et al. [4], where energy saving and resource usage are minimized within a Pareto approximate set. Aside from placement and migration, Q-learning can assist the VM migration by switching the host power mode between active and sleep [36, 37], which has a simpler action space.

SARSA and Q-learning are limited by the exponential growth of complexity in a large-scale cloud environment. An attempt to use Deep Q-learning (DQN) [38] has been developed for selecting a NUMA (Non-uniform Memory Access) or a cluster to allocate VMs [2]. While NUMA is a set of CPU and memory resources that can be used by large VMs, and multiple NUMAs are managed within a cluster, selecting the clustered index for a cross-NUMA VM request is similar to selecting a PM for a single-machine VM request. Despite the advantage over First Fit and Best Fit heuristics, this algorithm is limited by scalability, as the action space is the product of the number of clusters and NUMAs. Caviglione et al. [3] proposed a DRL model to tackle the scalability issue. Instead of designing an action of mapping hosts and VMs, the model only selects an action from a small set of bin-packing heuristics, which significantly simplifies the action space, allowing faster convergence. Another technique to reduce the state and action space is hierarchical DRL [39], where the list of PMs is divided into a decision tree. An RL agent is placed on each level of the decision tree, so the leaf nodes (PMs) will be chosen hierarchically.

In the cloud environment, it is difficult to maintain an indefinite long-term optimal by placing the VMs only once. VM migration is a major technique for load balancing and server migration. An appropriate migration strategy may improve the manageability, performance, and fault tolerance of systems [24]. Duggan et al. [6] proposed an RL approach for scheduling VM migration with an optimal data centre performance, measured by energy consumption, number of migrations, service level agreements, and performance degradation due to migration. For each host machine and its VMs, the agent observes its hardware utilization and the data centre bandwidth usage and decides if the VM should be migrated to an under-utilized host iteratively. This approach restricted the action space to only two discrete actions so that the agent could learn in fewer steps. VM migration may also be used to minimize the network access cost. Ren et al. [11] introduced an algorithm to achieve this based on Proximal Policy Optimization (PPO) [40] in conjunction with LSTM [41].

Related Works	RL Algorithm	Placement	Migration
SchedRL [2]	DQN	✓	✗
DRL-VMP [3]	DQN	✓	✗
VMPMORL [4]	Q-learning	✓	✗
RLVMP [5]	Q-learning	✓	✗
RL-LM [6]	Q-learning	✗	✓
Hummaida et al. [7]	Q-learning	✗	✓
ARLCA [8]	SARSA/Q	✗	✓
VMPACS [9]	DQN	✗	✓
Raven [10]	DQN	✗	✓
SMig-RL [11]	PPO	✗	✓
Our Work	PPO	✓	✓

Table 1: RL approaches for VM placement and migration

The continuity of the VM request sequence requires the initial VM placement to be processed serially. VM migration can further optimize placement on the fly. It remains an open issue for an RL-based VM placement approach to incorporate both initial placement and migration. Using separate models for placement and migration may cause conflicts. For instance, if the placement model initializes a VM on a PM while the migration model also moves a VM to this PM, it could lead to over-subscription. Additionally, the observations and actions of the two models could result in deadlocks, which add extra complexity to system design.

To address the aforementioned challenges, this work considers both initial placement and migration as a single problem. As shown in Table 2.2, our proposed approach stands out from the state-of-the-art methods. By unifying placement and migration, our approach aims to provide a more effective optimization in the cloud.

3. Problem Formulation

3.1. System Architecture

This paper addresses the problem of VM placement optimization in a cloud environment that includes a set of physical machines (PMs) hosting virtual machines (VMs). The system continuously generates VM requests, which specify the expected CPU and memory utilization and service length. Each VM terminates and spares the CPU and memory resources after reaching its service length.

As shown in Figure 3.1, the cloud manager is responsible for processing incoming VM requests and managing the PMs, which are assumed to be of equal size. The VMs may vary in size, as depicted by the grey boxes in the figure. The

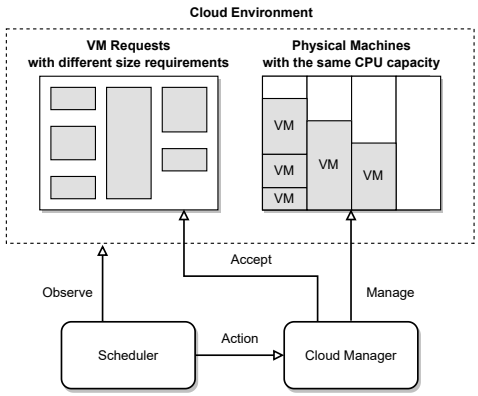


Figure 1: The proposed model for VM placement, where the scheduler creates and migrates VMs by operating the cloud manager

scheduler continuously monitors the cloud environment and optimizes VM placement on the PMs through the cloud manager. In certain situations, the scheduler may request the cloud manager to suspend VMs to migrate them across PMs. Our primary objective is to manage the location of VMs to ensure high utilization rates across all PMs while keeping them balanced.

To optimize energy efficiency and load balancing, the system must maintain a high PM utilization rate among the used machines. This allows the unused machines to be switched off to save energy. Since overly high hardware utilization may cause performance degradation, the utilization of PMs needs to be balanced. Instead of pushing the CPU and memory usage to 100%, we allow the usage target to be set at a specific level.

3.2. System Model

We formally define the VM placement problem as follows. For time steps $t = 1, \ldots, T$:

- $v \in \{0, 1, \ldots N-1\}$ is a discrete variable denoting a VM. N is the total number of VM requests in the task.
- $p \in \{0, 1, \ldots, P-1\}$ is a discrete variable denoting a PM; P is the total number of PMs.
- $a_{v,t} \in \{0, 1, \ldots, P+1\}$ is the discrete status value of VM v at time t. If a VM request has not yet arrived or the VM has terminated, $a_{v,t} = P + 1$. If a VM is waiting to be placed, $a_{v,t} = P$. If a VM has been deployed on the PM p, $a_{v,t} = p$;
- $V_t : \{v | a_{v,t} \leq P\}$ is the set of VMs and requests;
- $V_{p,t} : \{v | a_{v,t} = p\}$ is the set of VMs placed on PM p;
- $V'_t : \{v | a_{v,t} < P\}$ is the set of VMs placed on PMs;
- C is the PM resource capacity;
- $s \in \{0, 1\}$ is the resource type, where 0 represents CPU, and 1 represents memory;
- $r_{s,v} \in [0, 1]$ is the continuous-value CPU or memory resource requested or consumed by v, representing a percentage of C;
- $\bar{r}_s = \frac{1}{N} \sum_{v \in V_{p,t}} r_{s,v}$ is a mean resource requirement;
- $z_v \in \{1, 2, \ldots, T\}$ is the service length of v;
- $Y_t : \{(z_0, r_{0,0}, r_{1,0}), \ldots (z_{N-1}, r_{0,N-1}, r_{1,N-1})\}$ is the set of VM requests arriving at step t;
- $u_{p,s,t} = \sum_{v \in V_{p,t}} r_{s,v}$ is the utilization rate of resource type sg on PM p at step t;
- $\mu_{s,t} = \frac{1}{P} \sum_{v \in V'_t} r_{s,v}$ is the empirical mean of the CPU or memory utilization rate at step t;
- $\tilde{\mu}_{s,t} = \frac{1}{P} \sum_{v=0}^{N-1} r_{s,v}$ is the target mean of the CPU or memory utilization rate at step t;
- $\sigma^2_{s,t} = \frac{1}{P} \sum_{p=0}^{P-1} (u_{p,s,t} - \mu_{s,t})^2$ is the empirical variance of the CPU or memory utilization rate at step t;
- $\tilde{\sigma}^2_s = \frac{1}{V} \sum_{v=0}^{V-1} (r_{s,v} - \bar{r}_s)^2$ is the target covariance matrix of the CPU and memory utilization rate at step t;

The objective function $J_t^{(1)}$ is defined as

$$J_t^{(1)} = -\boldsymbol{\beta} \sum_{s=0}^{1} \mu_{s,t} \tag{1}$$

where $\boldsymbol{\beta} = \begin{bmatrix} \beta & 1-\beta \end{bmatrix}$ are the coefficients that control the importance of resources.

Given a sequence of VM arrivals $S_{request} = \langle Y_1, \ldots, Y_T \rangle$ and the PM capacity C, the objective is to find the optimal sequence of actions for $v = 0, \ldots, N-1$, $S_v = \langle a_{v,1}, a_{v,2}, \ldots, a_{v,T} \rangle$, subject to:

$$\sum_{t=1}^{T} |Y_t| = N \tag{2}$$

$$C \geq \sum_{v \in V_{p,t}} r_{s,v}, \forall p = 0, \ldots, P-1, \forall s = 0, 1, \forall t \in [1, T] \tag{3}$$

such that the objective function $J_t^{(1)}$ is minimized.

Minimizing $J_t^{(1)}$ encourages the PMs to be highly utilized. We also consider two alternative objective functions:

$$J_t^{(2)} = \frac{w_t}{|V_t|}, \tag{4}$$

$$w_t = |\{v | a_{v,t} = P\}| \tag{5}$$

$J_t^{(2)}$ is the number of waiting VMs w_t is to be minimized at time step t.

$$J_t^{(3)} = \mathrm{KL}(\mathcal{N}_1, \mathcal{N}_2), \tag{6}$$

$$\mathcal{N}_1 = \mathcal{N}(\begin{bmatrix} \tilde{\mu}_{0,t} & \tilde{\mu}_{1,t} \end{bmatrix}, \begin{bmatrix} \tilde{\sigma}^2_0 & 0 \\ 0 & \tilde{\sigma}^2_1 \end{bmatrix}) \tag{7}$$

$$\mathcal{N}_2 = \mathcal{N}(\begin{bmatrix} \mu_{0,t} & \mu_{1,t} \end{bmatrix}, \begin{bmatrix} \sigma^2_0 & 0 \\ 0 & \sigma^2_1 \end{bmatrix}) \tag{8}$$

$J_t^{(3)}$ is the KL divergence [42] between the actual and the target normal distributions of the CPU and utilization rate. To save energy, VMs should be grouped into a series of highly-utilized PMs, allowing spare PMs to be switched off. With an appropriate target variance, minimizing the objective function is equivalent to placing VMs in such a way that the physical resource requests are fulfilled as many as possible while the utilization rate is balanced across all PMs. This objective may possibly be better than linear functions that incorporate multiple goals because it describes the goals with a single value.

We utilize the negative objective functions as the reward functions in our proposed RL model discussed in the next section.

4. RL Design

The online VM placement problem consists of successive VM requests and termination, which can be formulated into a Markov decision process [2]. In this section, we discuss the design of the observation space, the action space, and the reward functions.

4.1. Observation Space

The observation space is a vector containing information for RL agents to read as input. The agent will then decide on actions based on the observed information. At each time step, the agent will observe the following values:

- V_{status}: the VM status vector. This vector stores the current status of each VM $a_{v,t}$, including unarrived or terminated (P), waiting to be placed ($P-1$), and placed (p);
- V_{life}: the vector of the remaining number of steps that each VM will run until terminated;
- V_{size}: the matrix of the CPU and memory resource $r_{s,v}$ that each VM requests;
- U: the vector of each PM's utilization rate $u_{p,t}$ at the current time step.

The status vector is a crucial component of both the observation space and the action space. At each step, the RL agent can place or suspend VMs by updating the status number in the vector. The length of the status vector represents the maximum number of VMs the system can handle at a time step. If all the VMs in the vector are either awaiting or placed, new incoming VM requests will be discarded.

4.2. Action Space

Since VM placement is achieved by updating the status vector, the action space is simply the status vector. To perform VM migrations, running VMs are suspended and placed on the wait status (P) so that they can be reallocated to the same or different PMs in later steps. The wait status is an important design feature that enables the system to place and migrate VMs using a single model.

Large action space has been a challenge for RL in our system model. Given P PMs and $V = |V_{status}|$ status slots, each time step has $(P+1) \times V$ possible operations for all VMs, such as placing VM v on PM p, suspending VM v, etc. To reduce the complexity, we employ a multi-discrete action space where the agent can update all elements in the status vector in a time step, and each element in the status vector has $P+1$ possible values.

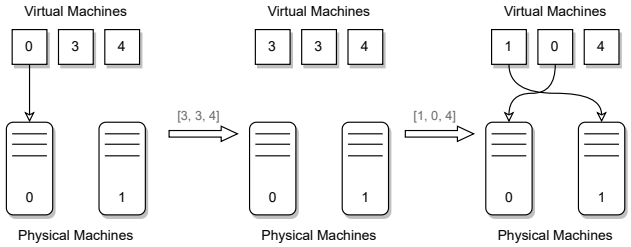


Figure 2: Status vector in VM placement and migration

Figure 2 provides an example of how VMs are placed and migrated in response to actions. Initially, the VM status is $[0, 3, 4]$, where a VM is running on PM 0, and another VM is waiting to be placed. The action $[3, 3, 4]$ would suspend the first VM. In the subsequent step, the first VM is placed on PM 1, and the second VM is placed on PM 0. At this point, the first VM has been migrated from PM 0 to PM 1. It is crucial to note that migration requires at least two steps.

A valid multi-discrete action may contain invalid status changes, such as allocating a VM to a full PM, moving VMs without suspension, or allocating an unarrived VM request. When the environment receives an action, only valid status changes will be executed, while invalid status changes will be disregarded.

4.3. Reward Function

The reward functions are the negative of the objective functions:

$$R_t^{(1)} = \beta \sum_{s=0}^{1} \mu_{s,t} \tag{9}$$

$$R_t^{(2)} = -\frac{w_t}{|V_t|} \tag{10}$$

$$R_t^{(3)} = -\mathrm{KL}(\mathcal{N}_1, \mathcal{N}_2) \tag{11}$$

$R_t^{(1)}$ would encourage the agent to increase CPU and memory utilization for all PMs. $R_t^{(2)}$ focuses on another aspect of the system: VMs. Minimizing average VM waiting time should produce a similar effect as maximizing PM utilization. $R_t^{(3)}$ minimizes the distance from the actual to the target distribution of the CPU and memory utilization rate.

4.4. DRL Algorithms

In this paper, we utilized the DRL-based approach to training RL agents, as DRL can model complex environments and policies [18]. Proximal Policy Optimization (PPO) [40] is adopted as the main family of algorithms. PPO and its predecessor Trust

Region Proximal Optimization (TRPO) [43] have the same motivation as natural gradient descent [44]: move towards the greedy optimal direction (best action), but only make sufficiently small updates to the policy without breaking the monotonic improvement condition. The details of our DRL algorithms are as follows.

Algorithm 1 PPO for VM Placement and Migration

Initialize neural networks for actor π_θ and critic V_ϕ
for each episode **do**
  Collect a batch of transitions B
  for each state s_t in transition $t = 1 \dots T$ in B **do**
    Compute the generalized advantage estimate (GAE) using the critic:
      $\delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$
      $\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}$
  end for
  for K epochs **do**
    Sample a minibatch M from B
    Normlaise $\hat{A}_t$ for t in M
    for each transition $t = 1 \dots T$ in M **do**
      Compute the entropy bonus $S_\theta(s_t)$ using actor
      Compute the clipped surrogate loss $L_t^{CLIP}(\phi) = \max(-r_t(\phi)\hat{A}_t, -\text{clip}(r_t(\phi), 1-\epsilon, 1+\epsilon)\hat{A}_t)$
      Compute the value function loss $L_t^{VF}(\phi) = (V_\phi(s_t) - r_t)^2$
      Compute the objective: $J_t(\phi) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\phi) + c_1 L_t^{VF}(\phi) - c_2 S_\theta(s_t)]$
    end for
    Optimize the actor and critic networks with the minibatch to minimize the objective
  end for
end for

Algorithm 1 describes the PPO-based training algorithm, where r_t is the reward at time step t. γ, ϵ, c_1, and c_2 are hyperparameters. The algorithm adopts the actor-critic architecture, where the actor and the critic are separate feed-forward neural networks with the same structure. The actor chooses actions and computes the entropy bonus, and the entropy bonus ensures sufficient exploration. The critic is used to estimate the value function. The generalized advantage estimate (GAE), indicating the advantage of an action a in a state s_t, is calculated and normalized in each minibatch. Then, the algorithm computes the minibatch's objective function values and uses them to optimize the actor and critic.

4.5. Convex Optimization

To compare with approaches using relaxed convex optimization, we devise an algorithm that fits in our system model based on Multi-level Join VM Placement and Migration (MJPM) proposed by Duong-Ba et al. [31]. Let $X^{P\times N}$ be a matrix of placement where $X(i, j) \in \{0, 1\}$ and $X(i, j) = 1$ means that VM i is placed on PM j. Let $R_s^{2\times N} = \begin{bmatrix} r_{s,0} & \cdots & r_{s,N-1} \end{bmatrix}$ be the resource requirement of VMs. Mathematically, the multi-objective optimization is written as:

$$\text{Minimize} \quad -\sum_{v=0}^{N-1}\sum_{p=0}^{P-1} X(v,p) \tag{12}$$

$$\text{subject to} \quad X(v,p) \in \{0,1\},$$
$$\forall v \in \{0,\dots,N-1\}, p \in \{0,\dots,P-1\} \tag{13}$$
$$R_s X \le 1, \quad \forall s \in \{0,1\} \tag{14}$$
$$X\mathbf{1} = \mathbf{1} \tag{15}$$

where the objective Eq. (12) measures the number of VMs placed on PMs. Contrain Eq. (13) ensures that the placement matrix X is binary. Constrain Eq. (14) ensures that the total resources requested by VMs are within the PM capacities. Constrain Eq. (15) ensures a VM is placed on only one PM.

5. Performance Evaluation

5.1. Baseline Algorithms

To measure the quality of the decisions generated by the RL models, First Fit and Best Fit are chosen as the baseline heuristic policies because they are simple and efficient. In First Fit, the first PM that can fit the VM is selected to allocate new VMs, while Best Fit places the VMs on the available PMs with the highest usage level. In this work, both First Fit and Best Fit generate multi-discrete actions.

To compare DRL against convex optimization, we implement an MJPM-based policy described in the previous section [31]. To evaluate the effectiveness of incorporating both placement and migration, we train a Rainbow DQN based on the DRL-VMP algorithm devised by Caviglione et al. [3].

5.2. Experiment Setup

To cope with the low sampling efficiency of current RL techniques [45], models are typically trained in a simulated environment that can generate a large number of samples. Thus, we developed an Gymnasium [46] (formerly Open AI Gym) environment based on the proposed model [1]. The Gym environment provides an evaluation framework that is extensible to any VM placement method that fits in the mathematical model described in section 3. The following parameters are controlled in the environment:

- $P = 100$: the number of PMs;
- $|V_{status}| = 300$: the length of the status vector;
- $T = 150,000$: the total number of steps in an experiment;
- γ: the mean of VM lifespan Poisson distribution;
- λ: the mean of arrival rate Poisson distribution;
- r_s: the mean of expected VM resource requirement;
- $\boldsymbol{\beta} = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}$: CPU and memory's coeffcients in $R_t^{(1)}$.

[1] https://github.com/yzh503/vm-placement-migration-gym

The system load is measured by a percentage of the upper threshold C. By default, a 100% load system has 100 PMs while the CPU and memory requirements are sampled from Unif(0.1, 1.0). The default service length is 1000 steps, which results in an arrival rate of $\lambda = \frac{P}{r \times \gamma} \times 100\% \approx 1.828$. Under this setting, the cluster utilization would reach 100% assuming all VMs could fit.

During the evaluation, the policies run for 150,000 steps so the data reaches the statistical equilibrium. Due to the stochastic nature of the environment and RL, the statistics are obtained by an average of multiple runs. A list of statistical metrics below is used to measure the performance of the agents. To prevent an excessive number of migrations, only 0.2 % of migrations were executed for the PPO agents. This ratio is chosen based on experiments.

The definitions used in the metrics are:

- $T_v = |\{a_{v,t} | a_{v,t} \geq -1\}|$: the lifespan of the VM v;
- $m_v = min(\{t | a_{v,t-1} = -1, a_{v,t} \geq 0, 2 \leq t \leq T\})$: the step when VM v is placed for the first time;
- $b_v = |\{a_{v,t} | a_{v,t-1} \geq 0, a_{v,t} = -1, 2 \leq t \leq T\}|$: the number of suspensions on VM v;
- $w_v = |\{a_{v,t} | a_{v,t} = -1, 1 \leq t \leq m_v\}|$: the number of waiting steps of the VM v before it is initially placed on a PM;
- $w'_v = |\{a_{v,t} | a_{v,t} = -1, m_v \leq t \leq T\}|$: the number of waiting steps of the VM v after it is initially placed on a PM;

The metrics are:

- **Average Waiting rate**: $\frac{1}{T}\sum_{t=1}^{T} w_t$, the average ratio of waiting steps to all steps, where w_t is defined in section 3.2;
- **Average Pending rate**: $\frac{1}{N}\sum_{v=0}^{N-1} \frac{w_v}{T_v}$, the average ratio of waiting steps before the initial placement for each VM;
- **Average Slowdown rate**: $\frac{1}{N}\sum_{v=0}^{N-1} \frac{w'_v}{T_v}$, the average ratio of waiting steps caused by suspension in the VM lifespan;
- **Suspensions per VM**: $\frac{1}{N}\sum_{v=0}^{N-1} b_v$, the average number of suspensions on VMs;
- **Drop rate**: the ratio of VM requests cannot be handled due to the VM status vector size limitation;
- **Average resource utilization rate**: $\mu_s = \frac{1}{T}\sum_{t=1}^{T} \mu_{s,t}$;
- **Resource utilization variance**: $\sigma_s^2 = \frac{1}{T}\sum_{t=1}^{T} \sigma_{s,t}^2$.

Figure 5.2 depicts the episodic return during the training of the PPO and DRL-VMP agents. DRL-VMP runs in a single-discrete action space because it is computationally expensive to approximate the Q-values for every possible combination of actions in a multi-discrete action space. The PPO agent runs in a multi-discrete action space because it optimizes the categorical distributions of actions instead of the Q-values. The Episodic return first drops quickly and then grows until stable.

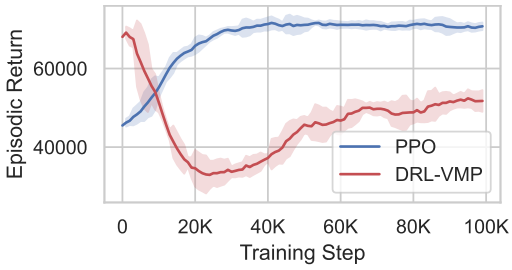


Figure 3: DRL episodic return in training with $R^{(1)}$

Agent	Policy	VM Served	μ_0	σ_0^2
PPO	$R_t^{(1)}$	13934	0.770	0.049
PPO	$R_t^{(2)}$	13923	0.770	0.049
PPO	$R_t^{(3)}$	13923	0.769	0.049
DRL-VMP	$R_t^{(1)}$	13564	0.747	0.054
DRL-VMP	$R_t^{(2)}$	13234	0.728	0.049
DRL-VMP	$R_t^{(3)}$	13234	0.728	0.049

Table 2: PPO and DRL-VMP performance in a 100-PM environment with different reward functions

Then the agents run again in a new environment for evaluation. It is worth noting that the episodic return is not an accurate metric for comparison between policies because the VM sequence may be different after the first dropped request.

5.3. Effect of Reward Functions

Table 2 provides a comparison of the reward functions defined in equations (7), (8), and (9) in section 4.3. With $R_t^{(1)}$, both PPO and DRL-VMP reach higher resource utilization rates than with the other two reward functions. Given that $R_t^{(1)}$ serves more VMs and utilizes PMs more efficiently, it is selected as the primary reward function for the subsequent experiments. The resource utilization rates for CPU and memory are similar, so only the CPU is included in the table.

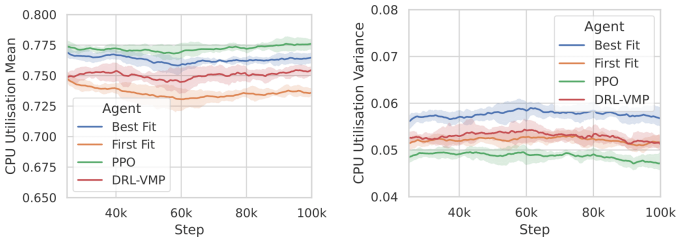
5.4. Experiment Results

5.4.1. CPU Utilization Rate

Table 3 compares First Fit, Best Fit, DRL-VMP, convex optimization, and PPO policies on 100% system load. When the environment contains 100 PMs, the problem size causes excessive long computation for convex optimization, so it is not included in the 100-PM experiment. As shown in the table, PPO

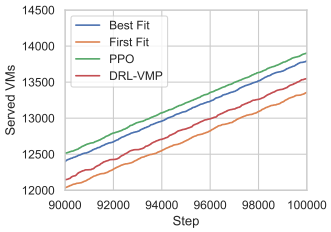
P	Policy	μ_0	σ_0^2	Pending	Slowdown	Suspension	Drop Rate
10	Convex	0.611	0.063	0.163	0.292	6413	0.339
10	DRL-VMP	0.674	0.061	0.348	0	0	0.275
10	First Fit	0.697	0.051	0.183	0	0	0.241
10	Best Fit	0.699	0.053	0.178	0	0	0.242
10	PPO	0.713	0.049	0.207	0.067	2000	0.236
100	First Fit	0.737	0.052	0.070	0	0	0.203
100	DRL-VMP	0.747	0.053	0.252	0	0	0.207
100	Best Fit	0.763	0.057	0.068	0	0	0.182
100	PPO	0.771	0.049	0.109	0.091	14053	0.187

Table 3: Performance of the VM placement and migration agents in 10-PM and 100-PM environments

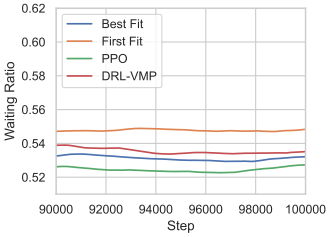


(a) CPU utilization rate mean

(b) CPU utilization variance



(c) VM Served



(d) Waiting Rate

Figure 4: Comparison between PPO, DRL-VMP, Best Fit and First Fit in a 100% load 100-PM environment on (a) CPU utilization rate mean μ_t (higher is better) (b) CPU utilization variance σ_t^2 (lower is better) (c) VM served (higher is better) (d) waiting rate (lower is better)

outperforms the other policies, achieving a 1.4–10.2% higher CPU utilization rate in the 10-PM environment and a 0.8%-3.4% higher rate in the 100-PM environment. As the second-best algorithm, Best Fit outperforms Convex optimization and DRL-VMP in both small and large PM clusters.

All policies exhibit non-zero variance due to the stochastic nature of VM arrival and size. Notably, Best Fit results in a higher average PM utilization rate and variance than First Fit. This highlights a trade-off between load balancing and PM utilization when employing the heuristics. In contrast, PPO achieves both a higher mean and a lower variance, suggesting that reinforcement learning can enhance load balancing and hardware utilization beyond this trade-off.

Figure 4a and 4b show that PPO achieved the highest average CPU utilization rate and the lowest CPU utilization variance throughout the episode in the 100-PM environment with full system load. The plots commence at step 25,000 to show the trend after the system reaches the statistical equilibrium. The coloured area around the lines represents the variation of $\mu_{0,t}$ and $\sigma_{0,t}^2$ across multiple runs. Since CPU and memory have the same independent distribution, memory utilization is similar to CPU.

5.4.2. Number of VMs Served

Figure 4c shows that PPO outperforms others in the 100-PM environment by serving the highest number of VMs. Figure 4d indicates that the VM requests have shorter waiting times.

5.4.3. Waiting Rate

In Figure 4d, PPO has the lowest waiting rate, indicating that fewer VMs are waiting during the operation of the cloud. Even though PPO causes waiting time when suspending VMs for migrations, the overall waiting rate is reduced because more VMs are placed on PMs.

5.4.4. Pending Time and Slowdown

According to Table 3, PPO achieves the highest CPU utilization rate at the cost of increased pending time and slowdown rate due to VM migration. However, the cost is acceptable, as the average pending rate of PPO is only 4% higher than that of Best Fit, and the average slowdown rate of PPO is 9% in a 100-PM environment.

5.5. Effects of Environment Parameters

By changing the VM arrival rate with the same service length, we can adjust the system load from 10% to 100%. Similarly, we may also vary service length on the same load level. To evaluate the behaviour of PPO in different environments, we ran the PPO agent on a series of environments with various load levels and service lengths.

5.5.1. Suspensions per VM

Figure 5a and 5b reveal the relationship between the number of suspensions and the environment parameters. As the system load grows, the number of suspensions gradually increases over 1. When the system load is 100%, each VM is migrated once per 1000 steps.

5.5.2. Pending Rate

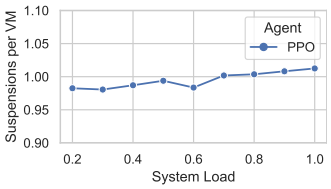
The pending rate is the proportion of steps when the VMs are waiting for instantiation. With a shorter pending rate, VMs will be allocated faster. Figure 5c and 5d compares the VM pending rate of PPO with First Fit and Best Fit over different load levels and service lengths. The results show that PPO starts to sacrifice pending rate for higher utilization from around 70% system load. An 80% load serves as a distinct dividing line of the demand for VM migration. With a 100% system load, the pending rate of PPO decreases as service length grows. Thus, VM requests can be fulfilled within a shorter time if they have a long lifespan.

5.5.3. Slowdown Rate

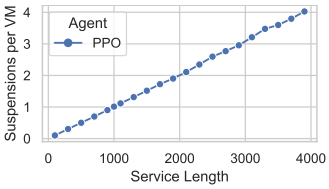
The slowdown rate represents the proportion of waiting steps caused by migration. If this rate is too high, the interruptions for running VMs could be disruptive. Thus, we only allow a portion of migrations to be executed, and the portion is controlled by the migration ratio. As illustrated in Figure 5e, the slowdown rate increases to around 0.1 when the migration ratio is 0.002. In Figure 5f, PPO overtakes Best Fit when the migration ratio grows to 0.002, which is therefore chosen as the optimal migration ratio for PPO.

5.5.4. VM Size

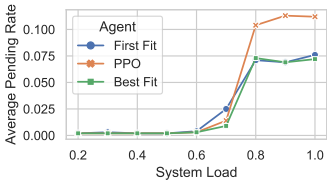
Table 4 compares PPO and the heuristics in two environments, where the VM size $r_{s,v} \sim \text{Unif}(0.1, 0.65)$ produces smaller VMs and $r_{s,v} \sim \text{Unif}(0.25, 1.0)$ produces larger VMs. A smaller mean of VM size represents small items in large bins in the bin packing context. The result shows that First Fit and Best Fit



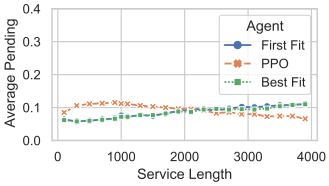
(a) Suspensions per VM with 1000-step service length over different load levels



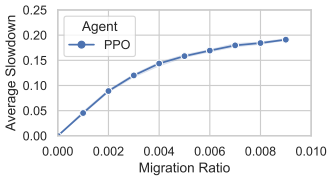
(b) Suspensions per VM with 100% system load over with different service lengths



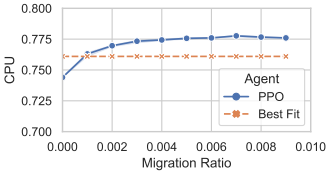
(c) Pending rate with 1000-step service length over different load



(d) Pending rate with 100% system load over with different service lengths



(e) Effect of migration ratio on slowdown rate



(f) Effect of migration ratio on CPU utilization

Figure 5: Effect of load level and service length on suspensions per VM, pending rate (lower is better), and effect of migration ratio for First Fit, Best Fit, and PPO.

perform better than PPO with small items and large bins. PPO only has an advantage over the heuristics on large items with small bins, where VM migrations significantly improve packing efficiency.

Policy	**VM size**	**Suspension**	μ_0	μ_1	σ_0^2	σ_1^2
First Fit	Unif(0.1, 0.65)	0	0.850	0.850	0.018	0.018
Best Fit	Unif(0.1, 0.65)	0	0.868	0.867	0.019	0.019
PPO	Unif(0.1, 0.65)	22981	0.863	0.862	0.015	0.015
First Fit	Unif(0.25, 1.0)	0	0.718	0.717	0.044	0.044
Best Fit	Unif(0.25, 1.0)	0	0.727	0.727	0.046	0.047
PPO	Unif(0.25, 1.0)	11677	0.734	0.734	0.044	0.043

Table 4: PPO performance with small and large VMs

6. Conclusions and Future Work

In this paper, we consider the problem of online VM placement in the form of bin packing. First, we formulate the problem into a mathematical model that allows for both initial placement and migration. Then, we propose a placement and migration approach based on RL, in which the observation space and the reward functions are designed to maximize the hardware utilization while keeping them balanced. A simulated environment based on the problem model is developed to evaluate the approach against the baselines in a series of experiments.

In the experiments, we provide a detailed analysis of the strength of our PPO agent against First Fit, Best Fit, a placement policy based on DRL-VMP [3], and a placement-migration policy based on MJPM [31]. As the system model allows migration, our method with PPO achieves the best performance on a high system load. Additionally, the experimental study shows that PPO performs better with large VMs. As a trade-off, the limitation of our approach is that VM migration may interrupt the VMs a few times during operation.

In the future, we plan to extend this work to include more resource types, such as network bandwidth. Other RL techniques may be employed to improve performance. For example, Monte Carlo Tree Search (MCTS) has been proven effective in a highly complex domain [47]. RNN and Transformers [48] can capture the dependencies between past actions.

References

[1] K. Jansen, S. Kratsch, D. Marx, I. Schlotter, Bin packing with fixed number of bins revisited, Journal of Computer and System Sciences 79 (2013) 39–49. doi:10.1016/j.jcss.2012.04.004.

[2] J. Sheng, Y. Hu, W. Zhou, L. Zhu, B. Jin, J. Wang, X. Wang, Learning to schedule multi-NUMA virtual machines via reinforcement learning, Pattern Recognition 121 (2022) 108254. doi:10.1016/j.patcog.2021.108254.

[3] L. Caviglione, M. Gaggero, M. Paolucci, R. Ronco, Deep reinforcement learning for multi-objective placement of virtual machines in cloud datacenters, Soft Computing 25 (2021) 12569–12588. doi:10.1007/s00500-020-05462-x.

[4] Y. Qin, H. Wang, S. Yi, X. Li, L. Zhai, Virtual machine placement based on multi-objective reinforcement learning, Applied Intelligence 50 (2020) 2370–2383. doi:10.1007/s10489-020-01633-3.

[5] S. Long, Z. Li, Y. Xing, S. Tian, D. Li, R. Yu, A Reinforcement Learning-Based Virtual Machine Placement Strategy in Cloud Data Centers, in: 2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), IEEE, Yanuca Island, Cuvu, Fiji, 2020, pp. 223–230. doi:10.1109/HPCC-SmartCity-DSS50907.2020.00028.

[6] M. Duggan, J. Duggan, E. Howley, E. Barrett, A reinforcement learning approach for the scheduling of live migration from under utilised hosts, Memetic Computing 9 (2017) 283–293. doi:10.1007/s12293-016-0218-x.

[7] A. R. Hummaida, N. W. Paton, R. Sakellariou, Scalable Virtual Machine Migration using Reinforcement Learning, Journal of Grid Computing 20 (2022) 15. doi:10.1007/s10723-022-09603-4.

[8] R. Shaw, E. Howley, E. Barrett, Applying Reinforcement Learning towards automating energy efficient virtual machine consolidation in cloud data centers, Information Systems 107 (2022) 101722. doi:10.1016/j.is.2021.101722.

[9] Z. Gao, Q. Jiao, K. Xiao, Q. Wang, Z. Mo, Y. Yang, Deep reinforcement learning based service migration strategy for edge computing, in: 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE), 2019, pp. 116–1165. doi:10.1109/SOSE.2019.00025.

[10] C. Ying, B. Li, X. Ke, L. Guo, Raven: Scheduling Virtual Machine Migration During Datacenter Upgrades with Reinforcement Learning, Mobile Networks and Applications 27 (2022) 303–314. doi:10.1007/s11036-020-01632-1.

[11] H. Ren, Y. Wang, C. Xu, X. Chen, SMig-RL: An Evolutionary Migration Framework for Cloud Services Based on Deep Reinforcement Learning, ACM Transactions on Internet Technology 20 (2020) 43:1–43:18. doi:10.1145/3414840.

[12] W. Chen, X. Qiao, J. Wei, T. Huang, A Profit-Aware Virtual Machine Deployment Optimization Framework for Cloud Platform Providers, in:

2012 IEEE Fifth International Conference on Cloud Computing, 2012, pp. 17–24. doi:10.1109/CLOUD.2012.60.

[13] E. Feller, C. Rohr, D. Margery, C. Morin, Energy Management in IaaS Clouds: A Holistic Approach, in: 2012 IEEE Fifth International Conference on Cloud Computing, 2012, pp. 204–212. doi:10.1109/CLOUD.2012.50.

[14] S. Kirthica, R. Sridhar, CIT: A Cloud Inter-operation Toolkit to enhance elasticity and tolerate shut down of external clouds, Journal of Network and Computer Applications 85 (2017) 32–46. doi:10.1016/j.jnca.2016.12.009.

[15] J. T. Piao, J. Yan, A Network-aware Virtual Machine Placement and Migration Approach in Cloud Computing, in: 2010 Ninth International Conference on Grid and Cloud Computing, 2010, pp. 87–92. doi:10.1109/GCC.2010.29.

[16] J. Wang, C. Huang, K. He, X. Wang, X. Chen, K. Qin, An Energy-Aware Resource Allocation Heuristics for VM Scheduling in Cloud, in: 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing, 2013, pp. 587–594. doi:10.1109/HPCC.and.EUC.2013.89.

[17] N. Mazyavkina, S. Sviridov, S. Ivanov, E. Burnaev, Reinforcement learning for combinatorial optimization: A survey, Computers & Operations Research 134 (2021) 105400. doi:10.1016/j.cor.2021.105400.

[18] G. Zhou, W. Tian, R. Buyya, Deep Reinforcement Learning-based Methods for Resource Scheduling in Cloud Computing: A Review and Future Directions, 2021. arXiv:2105.04086.

[19] N. Bobroff, A. Kochut, K. Beaty, Dynamic Placement of Virtual Machines for Managing SLA Violations, in: 2007 10th IFIP/IEEE International Symposium on Integrated Network Management, 2007, pp. 119–128. doi:10.1109/INM.2007.374776.

[20] W. Shi, B. Hong, Towards Profitable Virtual Machine Placement in the Data Center, in: 2011 Fourth IEEE International Conference on Utility and Cloud Computing, 2011, pp. 138–145. doi:10.1109/UCC.2011.28.

[21] J. Dong, H. Wang, X. Jin, Y. Li, P. Zhang, S. Cheng, Virtual Machine Placement for Improving Energy Efficiency and Network Performance in IaaS Cloud, in: 2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops, 2013, pp. 238–243. doi:10.1109/ICDCSW.2013.48.

[22] M. C. Silva Filho, C. C. Monteiro, P. R. Inácio, M. M. Freire, Approaches for optimizing virtual machine placement and migration in cloud environments: A survey, Journal of Parallel and Distributed Computing 111 (2018) 222–250. doi:10.1016/j.jpdc.2017.08.010.

[23] R. W. Ahmad, A. Gani, S. H. Ab. Hamid, M. Shiraz, F. Xia, S. A. Madani, Virtual machine migration in cloud data centers: A review, taxonomy, and open research issues, The Journal of Supercomputing 71 (2015) 2473–2515. doi:10.1007/s11227-015-1400-5.

[24] W. Voorsluys, J. Broberg, S. Venugopal, R. Buyya, Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation, in: M. G. Jaatun, G. Zhao, C. Rong (Eds.), Cloud Computing, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2009, pp. 254–265. doi:10.1007/978-3-642-10665-1_23.

[25] A. Ruprecht, D. Jones, D. Shiraev, G. Harmon, M. Spivak, M. Krebs, M. Baker-Harvey, T. Sanderson, VM Live Migration At Scale, ACM SIGPLAN Notices 53 (2018) 45–56. doi:10.1145/3296975.3186415.

[26] T. Wood, P. Shenoy, A. Venkataramani, M. Yousif, Sandpiper: Black-box and gray-box resource management for virtual machines, Computer Networks 53 (2009) 2923–2938. doi:10.1016/j.comnet.2009.04.014.

[27] D. Gmach, J. Rolia, L. Cherkasova, A. Kemper, Resource pool management: Reactive versus proactive or let's be friends, Computer Networks 53 (2009) 2905–2922. doi:10.1016/j.comnet.2009.08.011.

[28] A. Beloglazov, R. Buyya, Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers., MGC@ Middleware 4 (2010) 1890799–803.

[29] K. Maurya, R. Sinha, Energy conscious dynamic provisioning of virtual machines using adaptive migration thresholds in cloud data center, International Journal of Computer Science and Mobile Computing 2 (2013) 74–82.

[30] D. Borgetto, M. Maurer, G. Da-Costa, J.-M. Pierson, I. Brandic, Energy-efficient and SLA-aware management of IaaS clouds, in: 2012 Third International Conference on Future Systems: Where Energy, Computing and Communication Meet (e-Energy), 2012, pp. 1–10. doi:10.1145/2208828.2208853.

[31] T. Duong-Ba, T. Tran, T. Nguyen, B. Bose, A Dynamic Virtual Machine Placement and Migration Scheme for Data Centers, IEEE Transactions on Services Computing 14 (2021) 329–341. doi:10.1109/TSC.2018.2817208.

[32] G. A. Rummery, M. Niranjan, On-Line Q-learning Using Connectionist Systems, volume 37, University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[33] C. J. C. H. Watkins, Learning from delayed rewards, 1989.

[34] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, MIT press, 2018.

[35] R. Shaw, E. Howley, E. Barrett, An advanced reinforcement learning approach for energy-aware virtual machine consolidation in cloud data centers, in: 2017 12th International Conference for Internet Technology and Secured Transactions (ICITST), 2017, pp. 61–66. doi:10.23919/ICITST.2017.8356347.

[36] F. Farahnakian, P. Liljeberg, J. Plosila, Energy-Efficient Virtual Machines Consolidation in Cloud Data Centers Using Reinforcement Learning, in: 2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2014, pp. 500–507. doi:10.1109/PDP.2014.109.

[37] O. Rolik, E. Zharikov, A. Koval, S. Telenyk, Dynamie management of data center resources using reinforcement learning, in: 2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET), 2018, pp. 237–244. doi:10.1109/TCSET.2018.8336194.

[38] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (2015) 529–533. doi:10.1038/nature14236.

[39] L. Wang, Q. Weng, W. Wang, C. Chen, B. Li, Metis: Learning to Schedule Long-Running Applications in Shared Container Clusters at Scale, in: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, 2020, pp. 1–17. doi:10.1109/SC41405.2020.00072.

[40] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal Policy Optimization Algorithms, 2017. arXiv:1707.06347.

[41] S. Hochreiter, J. Schmidhuber, Long Short-Term Memory, Neural Computation 9 (1997) 1735–1780. doi:10.1162/neco.1997.9.8.1735.

[42] S. Kullback, R. A. Leibler, On Information and Sufficiency, The Annals of Mathematical Statistics 22 (1951) 79–86. arXiv:2236703.

[43] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, P. Abbeel, Trust Region Policy Optimization, 2017. arXiv:1502.05477.

[44] S. Kakade, A natural policy gradient, in: Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic, NIPS'01, Cambridge, MA, USA, 2001, pp. 1531–1538.

[45] Y. Yu, Towards Sample Efficient Reinforcement Learning, in: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence Organization, Stockholm, Sweden, 2018, pp. 5739–5743. doi:10.24963/ijcai.2018/820.

[46] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, O. G. Younis, Gymnasium, 2023. URL: https://zenodo.org/record/8127025. doi:10.5281/zenodo.8127026.

[47] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton, A Survey of Monte Carlo Tree Search Methods, IEEE Transactions on Computational Intelligence and AI in Games 4 (2012) 1–43. doi:10.1109/TCIAIG.2012.2186810.

[48] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention Is All You Need, 2017. arXiv:1706.03762.