# Virtual Machines Scheduling using Reinforcement Learning in Cloud Data Centers

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

Virtual machine (VM) scheduling is the core of Infrastructure as a Service (IaaS), where the common practice is to adapt heuristic methods. However, a single heuristic method is known to generalize poorly to different contexts. In this paper, we propose an DRL-based method to choose among a candidate pool of heuristic methods at each step. We also developed a simulating framework based on OpenAI Gym for benchmarking. Despite the high-dimensional state/action space and poor data availability, we show that our RL method can outperform all heuristic candidates when the objective is to minimize the overall fragment rate (FR). The amount of improvement scales with the size of the problem, indicating promising potentials for industry-scaled applications.

## 1 Introduction

The introduction and proliferation of cloud services has revolutionized how computing resources are consumed. Providers allow end-users easy access to secure, elastic and state-of-the-art resources, while applying efficient management techniques in order to optimize their return on investment. In particular, resource virtualization is used to maximize the utilization of the underlying hardware. Consequently, one of the most crucial components in the cloud stack is the Virtual Machine (VM) allocator, which assigns VM requests to the physical hardware. Indeed, suboptimal placement decisions can result in fragmentation (and in turn, unnecessary over-provisioning of physical resources), performance impact and service delays, and even rejection of incoming requests and customer impacting allocation failures.

The use of optimized placement mechanisms proved to be successful in a broad set of use cases, including production quality scenarios [1]. A typical solution exploits heuristics based on bin packing [2]. In fact, VM placement can be modeled as a bin packing problem, where VMs and PMs are objects and bins, respectively. For instance, the first-fit heuristic allows to place VMs over PMs in an efficient manner, but at the price of too aggressive packings causing VMs to ignore the fragment rate characteristics. Other solutions exploit optimization to guarantee a more fine-grained scheduling over the trade-off between placement actions and performance objectives, e.g., used power, reliability of the hardware, and mitigation of information leakage between VMs [3].

To summarize, VM placement is an interplay of different objectives, constraints, and technological domains. Machine learning techniques can tame such a complexity, owing to their capability of finding "hidden" relationships among the available data and therefore generate placement actions that maybe difficult to be found using classical optimization tools or heuristics based on common sense. Machine learning can be used either to design new VM placement approaches or to enhance the capabilities of existing heuristics. Toward this end, in this paper, we propose a mechanism for VM placement based on deep reinforcement learning (DRL) [4]. Specifically, we consider a decision maker that, after training, is able to select the most suitable heuristic to compute the placement for

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

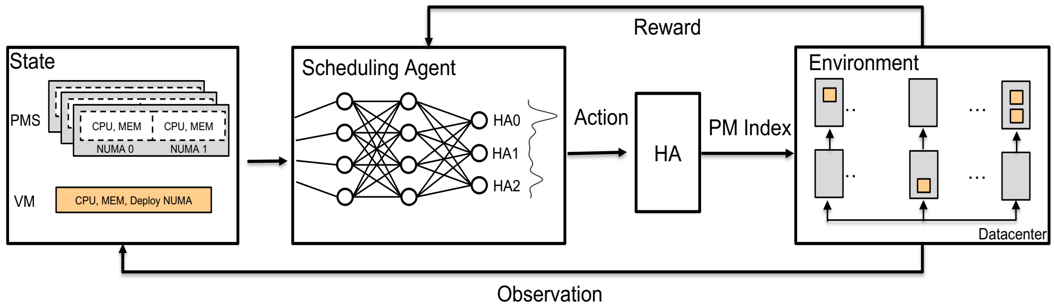



Figure 1: The overall VM scheduling framework proposed.

each VM requested by end users. To apply RL to complex VM scheduling problems, however, we had to solve several key challenges.

First, the cluster schedulers must scale to hundreds of PMs with thousands of VMs. In other words, the scheduling problem is significantly larger compared to typical RL applications such as game-play tasks, both in terms of the amount of information available to the scheduler (the state space), and the number of choices it must make (the action space). We therefore had to design a scalable RL formulation. Specifically, we design different heuristic algorithms as the set of actions for the RL agent. The heuristic algorithm selected by RL will determine the best PM to host the incoming VMs.

Second, while model-free approaches in general, and RL techniques in particular, are very powerful, their main weakness is the amount of data required to train them properly. The amount of training data should be in proportion to the action space, where the latter is far larger in our case. This issue is very important since we cannot expect cloud stakeholders to have years of scheduling data readily available in order to train the VM scheduler. As an alternative, we build an instance-generator and a scheduling environment to generate training data on the fly. The instance-generator is used to generate any VMs client request data, while the scheduling environment simulates the effect of the scheduling decision on the pm states for the next time step.

# 2 Design of VMS

## 2.1 VMS Overview

Figure 1 summarizes the proposed architecture of VMS. In VMS, the RL agent takes in the request information from VM client as well as all the current PM states as input. Based on this, the agent selects the best heuristic algorithm as its action for the current time step. The chosen heuristic algorithm generates the best PM index in the cluster, on which the incoming VM is scheduled. Overall, the goal of VMS is to reduce the fragment rate while satisfying all hard constraints required. Note that to simulate the challenges of online deployment, we do not have control over the order that VMs arrive in, i.e., all VMs are scheduled at a first-come, first-served basis.

**Problem Formulation** We consider a data center running an IaaS made up of $N$ PMs, where each PM has 2 NUMAs with a bundle of CPU and memory resources. At time $t$, the scheduler receives a client request for a certain type of resources, where the request can involve either a single NUMA or double NUMAs. The primary goal of the scheduler is to select the most suitable PM to handle the VM requests and minimize the fragment rate (FR) as defined below:

$$\#blocks\_used = \sum_{i=0}^{N}\sum_{j=0}^{1}\lfloor NC_{i,j}/mem\_size_i \rfloor, \quad \text{FR} = 1 - \frac{mem\_size_i \cdot \#blocks\_used}{\sum_{i=0}^{N} F_{i,\text{cpu}}}. \quad (1)$$

Here, $NC_{i,j}$ is the amount of free CPU memory on NUMA $j$ of PM $i$, $mem\_size_i$ is the size of each memory block on PM $i$ which varies across different VM types, and $F_{i,\text{cpu}}$ is the amount of free CPU memory on PM $i$. Intuitively, even if a VM only occupies a portion of a memory block, the remaining portion on that memory block cannot be assigned to another VM. In summary, this equation quantifies the percentage of free CPU memory that cannot be used across all PMs.

### 2.2 Deep Reinforcement Learning in VMS

We propose to use deep RL for VM scheduling. Unlike previous attempts that rely on pre-defined rules with heuristic algorithms, our approach learns a scheduling policy from observations. In other words, we model the dynamic virtual machine scheduling problem under the Markov decision process (MDP) framework, where DRL learns an optimal scheduling policy through interacting with the environment. For each VM client request, the agent takes the current PM state as well as the current VM client request as input, and computes the best action, i.e., to select the most appropriate heuristic algorithm at the current state. The heuristic algorithm in turn calculates the target PM index, to where the incoming VM is deployed on. The selected PM along with the current PM states are fed into the scheduling environment to simulate the next state as well as the reward. The agent uses the reward to update its weights via gradient ascent. The full MDP formulation is in Section A. Next, we discuss how we tailor the design of each component for the task of VMS.

**Action Design** Our action set includes two designed heuristic algorithms suitable for different context, namely a *first-fit* heuristic algorithm and a *fragment-fit* heuristic algorithm. We also add a third action of *initializing a new PM*. Notably, we mask out all PMs that are infeasible and the selected heuristic algorithm is only allowed to choose among the rest. After that, another first-fit algorithm decides the final NUMA placement inside this PM.

**State Design** We design the input to the agent at each step to include: **1) PMS features** - five in total: the first one is a binary flag which indicates whether the PM contains any VMs, and we prefer to schedule VM request to PMs that do, as it avoids consuming new PMs. The other four are the remaining CPU and memory for each of the two NUMAs. **2) VM features** - also five in total: NUMA ID and the remaining CPU/memory on each NUMA. We choose this design to allow for fast comparison between a VM request and all PMs in order to efficiently mask out infeasible PMs. Also, note that if a single NUMA is requested, we use zeros as placeholders for the other NUMA. Lastly, min-max normalization is applied to all numerical features.

**Reward Design** To quantify the change in FR of one PM before and after an incoming VM is added, we define the following:

$$\mathrm{R} = \mathrm{S}_{\text{before}} - S_{\text{after}}, \tag{2}$$

$$S = \begin{cases} 0 & \sum_{j=0}^{1} NC_j = 0, \\ [\sum_{j=0}^{1}(NC_j \% mem\_size_i)] / \sum_{j=0}^{1} NC_j & \sum_{j=0}^{1} NC_j > 0. \end{cases} \tag{3}$$

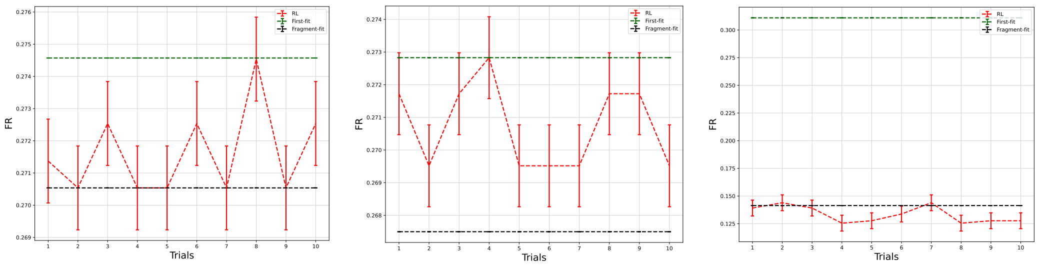
We omit the subscript $i$ indicating PM $i$ for simplicity. The reward of an action is the change in $S$ of the target PM. Note that here we only consider the memory size of VM type 4 as listed in Table 2.

## 3 Data Training Requirements

While DRL can be very powerful, its main drawback is the amount of training data required [5]. In light of this, we design an instance-generator (IG) and a high-fidelity scheduling simulator (SS). **Instance-generator** is designed to generate dummy VM client requests. The parameters include the VM type, the percentage of each VM type, and the migration status (able or disable) following predefined probability. The VM request information is saved as a JSON file and can be readily used by SS. **Scheduling Simulator** can reflect the real situations of cloud computing. The simulator follows the OpenAI Gym environments [6] including specific file hierarchy and function abstractions. We welcome researchers to use our framework to easily train their models and compare against different heuristic methods.

## 4 Experiments

To showcase the effectiveness of the proposed approach, we build an instance-generator to generate the synthetic data. We consider a data center composed of 279 PMs and 2089 VMs, where the VMs are shuffled randomly to simulate the order of client request arrivals. Seven classes of VMS were considered as shown in Table 2 in the Appendix. We compare VMS against two heuristic baselines.




(a) Small(500vms, 60pms) (b) Middle(1000vms, 120pms) (c) Big(2089vms, 279pms)

Figure 2: FR of Methods under Three Different Dataset.

Table 1: VM scheduling Results

| Methods | Scheduled | Failed | PMs Used | Fragment-rate |
|---|---|---|---|---|
| First-fit | 2089 | 0 | 265 | 0.311 |
| Fragment-fit | 2080 | 9 | 279 | 0.1414 |
| RL | 2089 | 0 | 279 | 0.1276 |

**First-fit method [7]:** Traverse all the PMs and place the VM on the first PM that can meet the cpu and memory requirement of the client request.

**Fragment-fit method:** Sort all PMs that can meet the requirements of the current VM according to the amount of the fragmentation rate reduction before and after this VM is added, and return the PM with the largest fragmentation rate reduction.

#### 4.0.1 Results:

We evaluate the three methods under three different dataset. The results are shown in Figure 2. We can see that the more big dataset, the FR from RL reduces. Note that in the middle datasett, though the FR of Fragment-fit method is better, it has 23 fails. The RL has no fails which means that all the 1000 can be scheduled. To be more specific, the big dataset experiments results are shown in Table 1. Compared to *first-fit*, the proposed RL can reduce FR by 58.9% with only 4.9% more PMs. Compared to *fragment-fit*, RL uses the same number of PMs but lowered FR by 9.6%. Additionally, RL have no failed VM requests, while fragment-fit has 9. This is because fragment-fit is a greedy algorithm that selects the PM with the largest FR reduction at the current step. As a result, all PMs will quickly be at least partially occupied, leaving no space for a large (64c, 88c) VM request that might come later. On the other hand, RL learns to sacrifice the current reward for better long-term reward.

## 5 Conclusion

We propose a framework for the VMS task and show that RL-based methods can achieve competitive results against widely-adapted heuristic algorithms, albeit the disproportionately large action space and the scarcity of data available. Extensive results reveal that letting the RL agent choose among multiple heuristic algorithms can lead to better results than any single heuristics, especially as the size of the problem grows.

## References

[1] Raja Wasim Ahmad, Abdullah Gani, Siti Hafizah Ab Hamid, Muhammad Shiraz, Abdullah Yousafzai, and Feng Xia. A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *Journal of network and computer applications*, 52:11–25, 2015.

[2] Rina Panigrahy, Kunal Talwar, Lincoln Uyeda, and Udi Wieder. Heuristics for vector bin packing. *research. microsoft. com*, 2011.

[3] Luca Caviglione and Mauro Gaggero. Multiobjective placement for secure and dependable smart industrial environments. *IEEE Transactions on Industrial Informatics*, 17(2):1298–1306, 2020.

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[5] Vincent Mai, Kaustubh Mani, and Liam Paull. Sample efficient deep reinforcement learning via uncertainty estimation. In *International Conference on Learning Representations*, 2022.

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[7] Mohammed Rashid Chowdhury, Mohammad Raihan Mahmud, and Rashedur M. Rahman. Implementation and performance analysis of various vm placement strategies in cloudsim. *Journal of Cloud Computing*, 4:1–21, 2015.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[9] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning, 2017.

[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

# A MDP and DRL for Scheduling

We model the dynamic virtual machine scheduling problem under the markov decision process (MDP) framework. For each VM client request, the RL agent schedules its PM destination. The PM states in the next step are determined by the current PM states and the VM request. Such a sequential decision-making problem can be formulated as a Markov Decision Process (MDP), modeled as $\langle S, A, T, R\rangle$, where $S$ is a finite set of states, which includes the remaining CPU and memory of all PMS, VM request. $A$ is a finite set of heuristic algorithms actions. $T$ is the state transition function defined as $T : S \times A \rightarrow S$. The PMS remaining information at next time step is determined by PMS remaining information and the VM request. $R$ is the reward function defined as $S \times A \rightarrow \mathbb{R}$, which qualifies the performance of a scheduling action. Based on the MDP-based VM scheduling problem formulation, we will find an optimal scheduling policy $\pi(s)^* : S \rightarrow A$, which maximizes the accumulative reward $R$.

In this paper, we consider an RL-based approach to generating VM scheduling algorithms. Unlike previous approaches that use pre-defined rules in heuristic algorithms, our approach will learn a scheduling policy from observations. DRL learns an optimal scheduling policy through interacting with the environment. At each time step $t$, the scheduling agent selects an action $A_t = a$, given the current state $S_t = s$, based on its policy $\pi_\theta$.

$$a \sim \pi_\theta(a|s) = \mathbb{P}(A_t|S_t = s; \theta) \tag{4}$$

In DRL, the scheduling policy is approximated by a neural network parameterized by $\theta$ [8]. When the scheduling agent takes the action $a$, a state transition $S_{t+1} = s'$ occurs based the system dynamics $f_\theta$ (Equation 5), and the scheduling agent receives a reward $R_{t+1} = r$.

$$s' \sim f_\theta(s, a) = \mathbb{P}(S_{t+1}|S_t = s, A_t = a) \tag{5}$$
$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, \mathbb{E}_{\pi_\theta}[r] \tag{6}$$

Table 2: VM Types

| Type | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| VM Classes | large(2) | xlarge(4) | 2xlarge(8) | 4xlarge(16) | 8xlarge(32) | 16xlarge(64) | 22xlarge(88) |

Due to the Markov property, both reward and state transition depend only on the previous state. DRL then finds a policy $\pi_\theta$ that maximizes the expected reward (Equation 6).

# B Model Details

We implement our RL model using RLlib [9]. We use a batch size of $4000$ and a learning rate of $0.0005$ for training. The neural network consists of two stacked fully-connected layers with $256$ hidden units each and ReLU activation functions. The network is trained using PPO [10] with $\gamma = 0.99$. Critic and GAE are used with kl coefficient being $0.2$.