Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI-22)
Demonstrations Track

# VMAgent: A Practical Virtual Machine Scheduling Platform

**Junjie Sheng**[1], **Shengliang Cai**[1], **Haochuan Cui**[1], **Wenhao Li**[1], **Yun Hua**[1], **Bo Jin**[1], **Wenli Zhou**[2], **Yiqiu Hu**[2], **Lei Zhu**[3], **Qian Peng**[3], **Hongyuan Zha**[4], **Xiangfeng Wang**[1*]
[1]School of Computer Science and Technology, East China Normal University, Shanghai, China
[2]Algorithm Innovation Lab, Cloud BU, Huawei Technologies Co
[3]Alkaid Lab, Cloud BU, Huawei Technologies Co
[4]The Chinese University of Hong Kong (Shenzhen), Shenzhen, China
xfwang@cs.ecnu.edu.cn

## Abstract

Virtual machine (VM) scheduling is one of the critical tasks in cloud computing. Many works have attempted to incorporate machine learning, especially reinforcement learning, to empower VM scheduling procedures. Although improved results are shown in several demo simulators, the performances in real-world scenarios are still underexploited. In this paper, we design a practical VM scheduling platform, i.e., `VMAgent`, to assist researchers in developing their methods on the VM scheduling problem. `VMAgent` consists of three components: *simulator*, *scheduler*, and *visualizer*. The *simulator* abstracts three general realistic scheduling scenarios (*fading*, *recovering*, and *expansion*) based on Huawei Cloud's scheduling data, which is the core of our platform. Flexible configurations are further provided to make the *simulator* compatible with practical cloud computing architecture (i.e., Multi Non-Uniform Memory Access) and scenarios. Researchers then need to instantiate the *scheduler* to interact with the *simulator*, which is also pre-built in various types (e.g., heuristic, machine learning, and operations research) of scheduling algorithms to speed up the algorithm design. The *visualizer*, as an auxiliary component of the *simulator* and *scheduler*, facilitates researchers to conduct an in-depth analysis of the scheduling procedure and comprehensively compare different scheduling algorithms. We believe that `VMAgent` would shed light on the AI for the VM scheduling community and the demo video is presented in https://bit.ly/vmagent-demo-video.

## 1 Introduction

The main challenge for leading cloud computing providers, e.g., AWS, Azure, Alibaba Cloud, and Huawei Cloud, is to fulfill customers' dynamic resource requirements (CPU/GPU, memory, etc.) seamlessly by real-time scheduling the incoming requests to the physical machines (PMs) of the particular cluster. This status quo makes designing efficient VM scheduling algorithms one of the critical tasks in cloud computing. Many classical combinatorial optimization methods were used to solve offline VM scheduling problem [Wolke *et al.*, 2015]. However, practical scheduling scenarios mainly rely on heuristic methods [Bays, 1977; Hadary *et al.*, 2020] with low time complexity due to the urgent online requirement. Unfortunately, these methods heavily depend on expert knowledge and may get stuck in suboptimals.

Recently, deep reinforcement learning (DRL) has shown promising results on several combinatorial optimization problems [Nair *et al.*, 2020]. [Bello *et al.*, 2016; Nazari *et al.*, 2018; Deudon *et al.*, 2018] adapt DRL to solve routing problems or travelling salesman problems (TSP); [Duan *et al.*, 2019; Hu *et al.*, 2017] apply the DRL to solve the bin packing problem; And more relevantly, [Sheng *et al.*, 2022] firstly propose a DRL algorithm for the VM scheduling, which shows that DRL can obtain better performance than heuristic methods in simple VM scheduling tasks. These all indicate that DRL has great potential for solving VM scheduling problems. However, different from the aforementioned practical combinatorial optimization problems, the lack of realistic benchmarks makes VM scheduling algorithms only evaluable in the laboratory environment (e.g., [Sheng *et al.*, 2022]) and cannot substantially benefit industrial applications.

In this paper, we propose a practical and efficient VM scheduling platform called `VMAgent` to assist in developing machine learning, especially DRL methods for VM scheduling. It consists of three components: *simulator*, *scheduler* and *visualizer*, as shown in Figure 1. Specifically, the *simulator*, which is the core of the entire platform, needs to simulate the practical VM scheduling scenarios and provide efficient interaction interfaces for the *scheduler*. In order to construct a *simulator* which substantially benefit industrial applications, we collect realistic VM requests data from Huawei Cloud's actual operation scenarios and abstract three general realistic scheduling scenarios (i.e., *fading*, *recovering*, and *expanding*) from these real-world VM scheduling processes. Flexible configurations are further provided to make the *simulator* compatible with more physical cloud computing architectures and scenarios. For researchers in the DRL community to get started quickly, we introduce the gym-like [Brockman *et al.*, 2016] interface style into the *simulator* implementation.

*Corresponding author

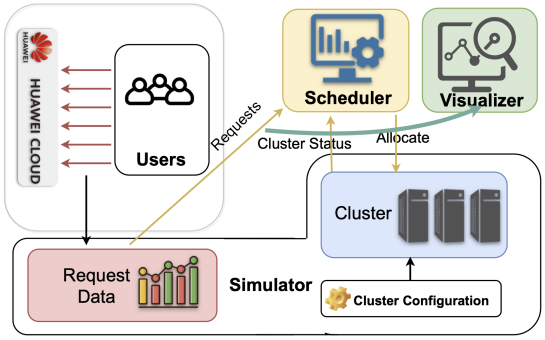



Figure 1: `VMAgent` platform overview. `VMAgent` consists of three components: *simulator*, *scheduler* and *visualizer*. The *simulator* abstracts three general realistic scheduling scenarios based on Huawei Cloud realistic data. The *scheduler* interacts with the *simulator* and the *visualizer* can take the interaction data to visualize the scheduling procedure and the comparison of various algorithms.

Researchers then need to instantiate the *scheduler* to interact with the *simulator*, which is also pre-built various types of scheduling baselines to speed up the algorithm design. These baselines range from widely-used heuristic methods to well-performed DRL methods, and we benchmark the baselines in Table 1. Meanwhile, considering the data-hungry of the DRL algorithm, the *scheduler* is designed in a multi-processing scheme and can interact with multiple *simulator* instances in parallel. The *visualizer*, as an auxiliary component of the *simulator* and *scheduler*, facilitates researchers to conduct in-depth analysis of the scheduling procedure (in *uni-policy mode*) and comprehensively compare different scheduling algorithms (in *multi-policies mode*).

Besides benefiting the scheduling community, `VMAgent` is also a flexible playground for DRL. For example, more allocatable physical machines correspond to a larger state and action space; unpredictable requests for resources release (i.e., the *recovering* scenario) will make the DRL algorithm face a highly non-stationary environment; and an extensible physical machine pool (i.e., the *expanding* scenario) will guide the algorithm design to the field of life-long DRL. Coincidentally, these challenges are also three key issues of applying DRL to real-world problems. To sum up, `VMAgent` can provide a powerful platform to explore the law and scope of DRL for VM scheduling applications and investigates the challenges for applying RL to real-world problems.

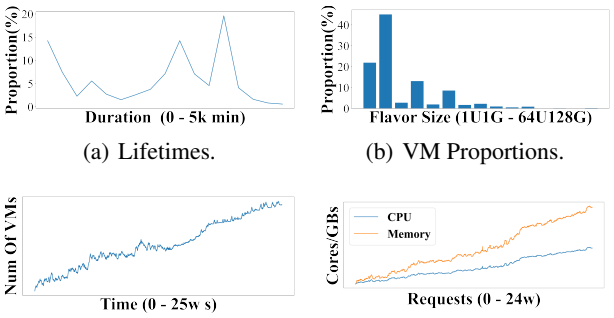
## 2 The `VMAgent` Platform

The `VMAgent` project[1] is a VM scheduling platform for designing DRL-based VM scheduling algorithms. In this section, we will give a detailed introduction to the three components of the `VMAgent` platform, the *simulator*, the *scheduler*, and the *visualizer*. Before that, we will first illustrate how the `VMAgent` platform models the VM scheduling procedure. Although modeling this procedure belongs to the *simulator* and *scheduler* design, it is singled out for explanation considering that it is the cornerstone of the entire platform.

[1]https://github.com/mail-ecnu/VMAgent.

### 2.1 Scheduling Fundamentals

`VMAgent` models the practical VM scheduling based on physical system architectures and the open-source real-world VM scheduling dataset, `Huawei-East-1`[2], from Huawei Cloud. The `Huawei-East-1` dataset was collected in the east china region of Huawei Cloud for one month, which includes 241743 requests and 15 types requested virtual machines, and relevant statistical analyses are shown in Figure 2.




(a) Lifetimes. (b) VM Proportions.

(c) Alive VMs. (d) Scheduled Resources.

Figure 2: Statistical analyses of `Huawei-East-1` dataset. a) the VM alive times varies while b) most VMs are relatively small. The c) shows the number of VMs increases with different speeds while d) shows the trend of scheduled CPU and memory.

We model practical VM scheduling as a finite number of repetitions of the following four steps under DRL context, which is shown in Figure 3. In each loop, for the current request in the request sequence (constructed from the `Huawei-East-1` dataset), (1) the *simulator* preprocesses request information to obtain **observations**; (2) the *scheduler* performs VM scheduling **action** accordingly; after the current request is processed, (3) the *simulator* turns to process the next request in the sequence; (4) the *simulator* calculates the **reward** according to the scheduling result and the next request's information. The above loop will continue until the **stop condition** is satisfied. Below we define some of the terms in the above procedure in detail.

The **observations** is the combination of sorted cluster status and the current request information. The cluster status includes the number of PMs within the cluster, the scheduled resources, and architectures (e.g., double NUMA [Lameter, 2013]) of each PM. We sort the cluster status based on the scheduled resources (CPU and memory) to make an efficient representation. The current request information can be categorized into two types: allocation, which includes the feature (e.g., the amount of CPU and memory) of resources, and release, which only includes the request ID in the sequence.

The **action** is the scheduled ID of PM (and NUMA in some tasks) and the **reward** is the cumulative number of successful scheduling. When an allocation can not be satisfied (i.e., no PMs can satisfy required resources), the cluster is denoted as not available, and we take it as the **stop condition**.

The goal of VM scheduling or the *scheduler* is to increase the cluster's availability and gain more income from the re-

[2]https://github.com/huaweicloud/VM-placement-dataset.

quests. The availability is measured by the number of scheduled requests, while the income is calculated based on scheduled VMs' prices and alive duration.

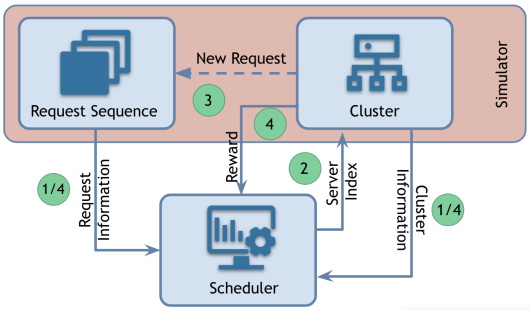



Figure 3: The modeled practical scheduling procedure.

## 2.2 Simulator

To reflect pratical VM scheduling procedure, we abstract three scenarios, i.e., *recovering*, *fading* and *expansion*, from the `Huawei-East-1` dataset in the simulator design.

**Recovering.** The *recovering* scenario considers both allocation and releasing requests, which is common in the public cloud when the PM pool will not be expanded. The scheduled resources are released from the cluster when a release request comes. Due to the dynamic, unpredictable release requests, the *scheduler* faces a high non-stationarity environment.

**Fading.** The *fading* scenario only allows allocation requests, which is common in the dedicated cloud. When the number of PMs is large, the curse of dimensionality raises.

**Expansion.** The *expansion* scenario considers that the PM pool will be expanded if the remaining resources are lower than a certain threshold, common in the public cloud. The cluster will replenish PMs before scheduling is terminated. This scenario leads to a life-long VM scheduling problem.

We utilize YAML to provide flexible configurations to make the *simulator* compatible with physical architecture. The configurations can be categorized into two types: cluster and scene. The following example shows the configuration of the *expansion* scenario.

```
cluster_args:
  N: 100
  CPU: 40
  MEM: 90
  double_numa: True
env_args:
  allow_release: True
  growing_threshold: 0.8
  growing_nums: 20
```

Besides the YAML configuration, the *simulator* follows the gym-like interface and provides an efficient multi-process sampling interface. This makes it an RL-friendly simulator.

## 2.3 Scheduler

Researchers need to instantiate the *scheduler* to interact with the *simulator*, which is also pre-built various types of VM scheduling baselines to speed up the algorithm design. It includes the popular heuristic online VM scheduling methods (i.e., first-fit and best-fit [Bays, 1977]) and many DRL methods (e.g., DQN [Mnih *et al.*, 2015], A2C [Mnih *et al.*, 2016], SchedQ [Sheng *et al.*, 2022]). We benchmark some of the baselines in different scenarios with 5 PMs, and the results are shown in Table 1, which illustrates the advantage of DRL.

| | Metric/Algos | SchedQ | DQN | A2C | FF | BF |
|---|---|---|---|---|---|---|
| Fading | Availability | **276.1** ± 1.18 | 275.1 ± 0.19 | 274.0 ± 1.40 | 270.6 | 270.6 |
| | Incomes | **103.11** ± 0.83 | 102.43 ± 0.11 | 101.69 ± 0.98 | 99.2 | 99.2 |
| Recovering | Availability | **393.4** ± 2.63 | 392.5 ± 1.59 | 392.1 ± 0.96 | 371.0 | 384.4 |
| | Incomes | **211.87** ± 3.88 | 210.28 ± 1.87 | 209.88 ± 0.97 | 185.1 | 200.3 |

Table 1: Benchmark for baseline schedulers. DRL methods obtain better performances on both availability and incomes.

## 2.4 Visualizer

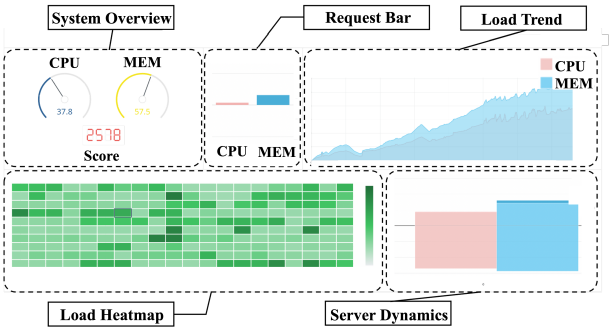



Figure 4: Visualization of the cluster status on the system overview, load heatmap, load trend and the PMs' dynamics.

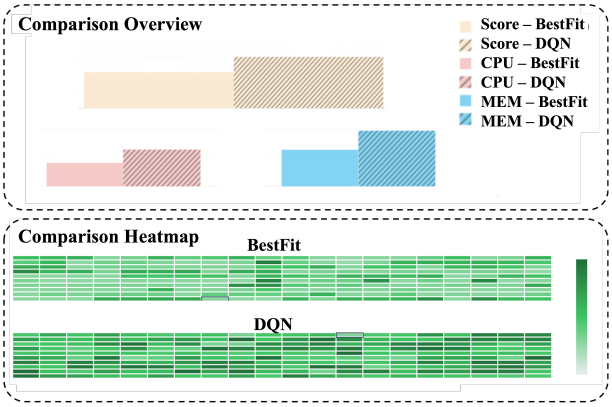



Figure 5: Visualization for comparison between best-fit and DQN, which includes scheduling resulted heatmap and cluster overview at each timestep.

The *visualizer*, as an auxiliary component of the *simulator* and *scheduler*, facilitates researchers to conduct an in-depth analysis of the scheduling procedure and comprehensively compare different scheduling algorithms. *Visualizer* provides two modes: *uni-policy mode* (Figures 4) and *multi-policies mode* (Figure 5). *Uni-policy mode* visualizes the dynamic of cluster status and the *scheduler*'s decisions when continually handling requests. The *multi-policies mode* makes visualization on comparisons among different schedulers.

We finally conclude that our paper proposes a practical and efficient VM scheduling platform, `VMAgent`. It well supports RL methods design in real-world VM scheduling scenarios.

## Acknowledgements

This work was supported in part by the National Key Research and Development Program of China (No. 2020AAA0107400), STCSM (No. 20DZ1100304 and 19ZR141420) and Shenzhen Science and Technology Program (JCYJ20210324120011032).

## References

[Bays, 1977] Carter Bays. A comparison of next-fit, first-fit, and best-fit. *Communications of the ACM*, 20(3):191–192, 1977.

[Bello *et al.*, 2016] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *ICLR*, 2016.

[Brockman *et al.*, 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[Deudon *et al.*, 2018] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. Learning heuristics for the TSP by policy gradient. In *CPAIOR*, pages 170–181, 2018.

[Duan *et al.*, 2019] Lu Duan, Haoyuan Hu, Yu Qian, Yu Gong, Xiaodong Zhang, Yinghui Xu, and Jiangwen Wei. A multi-task selected learning approach for solving 3D flexible bin packing problem. In *AAMAS*, pages 1386–1394, 2019.

[Hadary *et al.*, 2020] Ori Hadary, Luke Marshall, Ishai Menache, Abhisek Pan, Esaias E Greeff, David Dion, Star Dorminey, Shailesh Joshi, Yang Chen, Mark Russinovich, et al. Protean: VM allocation service at scale. In *OSDI*, pages 845–861, 2020.

[Hu *et al.*, 2017] Haoyuan Hu, Xiaodong Zhang, Xiaowei Yan, Longfei Wang, and Yinghui Xu. Solving a new 3D bin packing problem with deep reinforcement learning method. *arXiv preprint arXiv:1708.05930*, 2017.

[Lameter, 2013] Christoph Lameter. NUMA (Non-Uniform Memory Access): An overview. *Queue*, 11(7):40–51, 2013.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[Mnih *et al.*, 2016] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, pages 1928–1937, 2016.

[Nair *et al.*, 2020] Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, Ravichandra Addanki, Tharindi Hapuarachchi, Thomas Keck, James Keeling, Pushmeet Kohli, Ira Ktena, Yujia Li, Oriol Vinyals, and Yori Zwols. Solving mixed integer programs using neural networks. *arXiv:2012.13349*, 2020.

[Nazari *et al.*, 2018] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence V Snyder, and Martin Takac. Reinforcement learning for solving the vehicle routing problem. In *NeurIPS*, pages 9861–9871, 2018.

[Sheng *et al.*, 2022] Junjie Sheng, Yiqiu Hu, Wenli Zhou, Lei Zhu, Bo Jin, Jun Wang, and Xiangfeng Wang. Learning to schedule multi-NUMA virtual machines via reinforcement learning. *Pattern Recognition*, 121:108254, 2022.

[Wolke *et al.*, 2015] Andreas Wolke, Boldbaatar Tsend-Ayush, Carl Pfeiffer, and Martin Bichler. More than bin packing: Dynamic resource allocation strategies in cloud data centers. *Information Systems*, 52:83–95, 2015.