VMAgent

Release 0.0.1

Jarvis

Apr 19, 2022

INSTALLATION

VMAgent is a platform for exploiting Reinforcement Learning (RL) on Virtual Machine (VM) scheduling tasks. It is developed by the Multi-Agent Artificial Intelligence Lab (MAIL) in East China Normal University and Algorithm Innovation Lab in Huawei Cloud. VMAgent is constructed based on one month real VM scheduling dataset called Huawei-East-1 from HUAWEI Cloud and it contains multiple practicle VM scheduling scenarios (such as Fading, Rcovering, etc). These scenarios also correspond to the challanges in the RL. Exploiting the design of RL methods in these secenarios help both the RL and VM scheduling communities.

Key Components of VMAgent:

- SchedGym (Simulator): it provides many practical scenarios and flexible configurations to define custom scenarios.
- SchedAgent (Algorithms): it provides many popular RL methods as the baselines.
- SchedVis (Visulization): it provides the visualization of schedlueing dynamics on many metrics.

CHAPTER

ONE

INSTALLATION

1.1 Install from Source

First clone our git repo:

```
git clone https://github.com/mail-ecnu/VMAgent.git
cd VMAgent
```

Then create the virtual environment to satisfy dependency with conda:

```
conda env create -f conda_env.yml
conda activate VMAgent-dev
```

Finally Install our simulator:

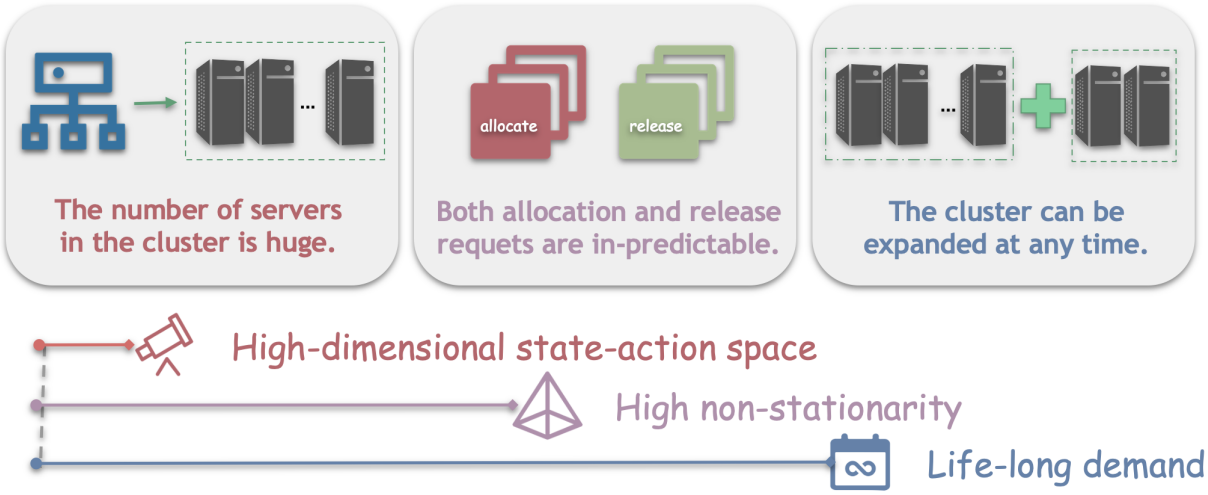
```
python3 setup.py develop
```

CHAPTER

TWO

SCENARIOS

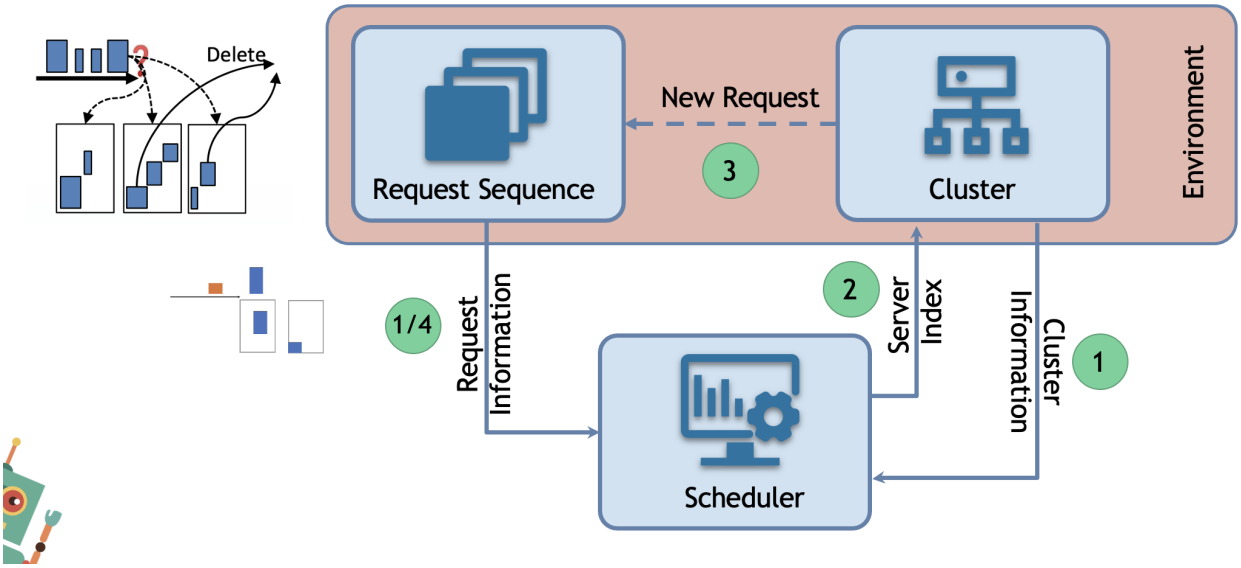
Our VMAgent provides multiple virtual machine scheduling scenarios in the practical cloud computing. These scenarios differ from each other on the cluster's feature and the request's feature. Moreover, these different scenarios also pose different perspective difficluties on the reinforcement learning methods. We summarize the cluster features, request features and their corresponding difficuluties on RL below.



2.1 Scheduling

The virtual machine scheduling problem can be divided into three main components: Request Sequence, Cluster and Scheduler.

Breifly speaking, a number of users request virtual machine resources and proposes requests sequentially. Each time the scheduler observes a request, it will check the cluster and find a server in the cluster to handle the request. The server then allocate corresponding resources for the request. When there are no server in the cluster can handle the request, the scheduler will be terminated. The scheduler is designed to avoid termination.



2.2 Cluster

For a cluster, it includes N servers. The server often has its attribute (c, m, numas) where the numas is the number of numa it has and the c and m are the number of the cpu and memory each numa has. In our VMAgent, the c=40, m=90 and the numas=2. For a RL scheduler, if the N increases, then the state and action dimension exponentially increase.

A cluster also featured by whether it can expand the servers. In practical scenarios, the cloud service provider needs to ensure the cluster never terminated because of the short of server resources. Thus they often buy a bunch of servers when the availiable resources in the cluster are lower than the threshold. With more servers added, the RL scheduler will face a different problem and this makes it a life-long learning challanges.

2.3 Request

For the request, it contains (rc, rm) where the rc and rm are the number of cpu and memory it needs. The requests can be divided into two categories: creation request and deletion request. For the creation request, the cluster need to assign corresponding resources to it. For the deletion request, the clueter need to find where the resources that the request occupied and remove it. In dedicated cloud, users often buy the servers for a long time and the deletion requests are few. While for public cloud, the deletion requests happen more frequently. This brings a high non-stationary challange for the RL scheduler.

2.4 Clustomize Your Scenario

Our VMAgent provides flexible configurations to customized your own scenarios. In the `vmagent/config/envs`, we provide several examples. Take the `expand.yml` as an example:

```
N: 20
cpu: 40
mem: 90
allow_release: True
double_thr: 10
```

It has a cluster with 20 servers and each server has 2 numas. Each numa hash 40 cores cpu and 90 GB memory. It handles the deletion (allow_release) requests. For a request that requires more than 10 cores cpu, it will be distributed on a server's two numas. Users are allowed to change the configuration to their own scenarios.

CHAPTER

THREE

DATASET

VMAgent is constructed based on one month real VM scheduling dataset called Huawei-East-1 from HUAWEI Cloud. The Huawei-East-1 is placed in our repository.

3.1 Data Format

The data format is concluded below

Field	Type	Description
`vmid`	`int`	The virtual machine ID
`cpu`	`int`	Number of CPU cores
`memory`	`int`	Number of Memory GBs
`time`	`int`	Relative time in seconds
`type`	`int`	0 denotes creation while 1 denotes deleteion

3.2 Statistical Analysis

The statsical information of the dataset is listed below.

Number of VM types	Number of creation requests	Number of deletion requests	Time duration	Server location
15	125430	116313	30 Days	East China

To gain better understanding of the cpu and memory distribution, we plot the histograms of the cpu and memory.

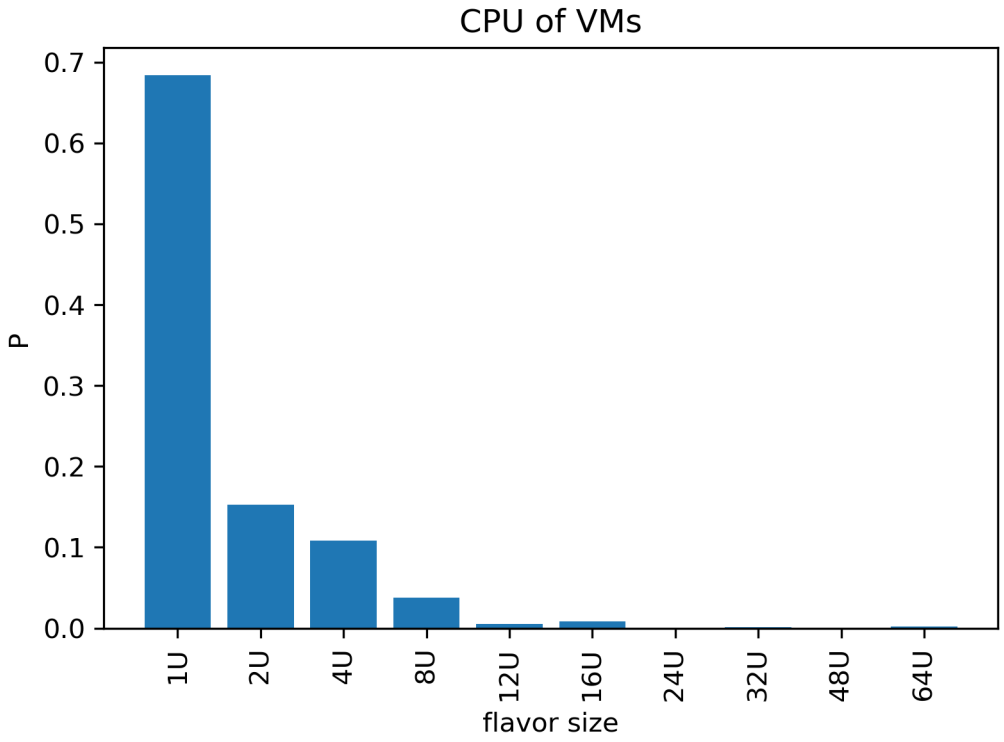
More than 2/3 requests only consumes 1U and less than 2G. We also plot the statiscs of the (cpu, mem) request:

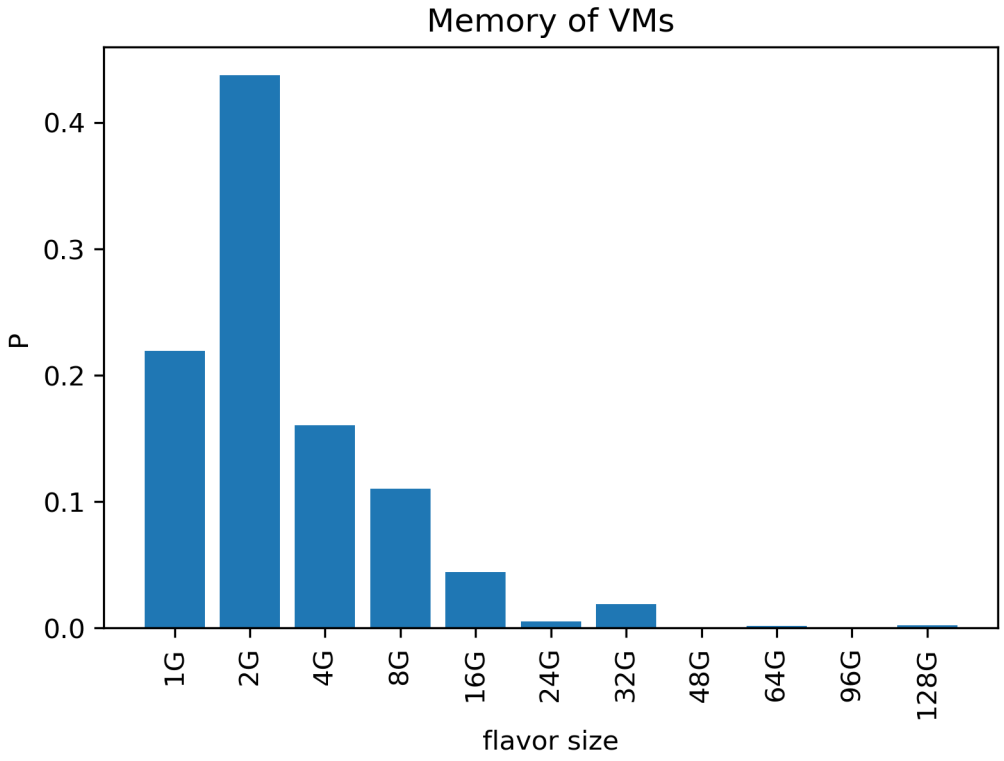
The 1U1G,1U2G, 2U4G and 4U8G constitues the main body of the requests.

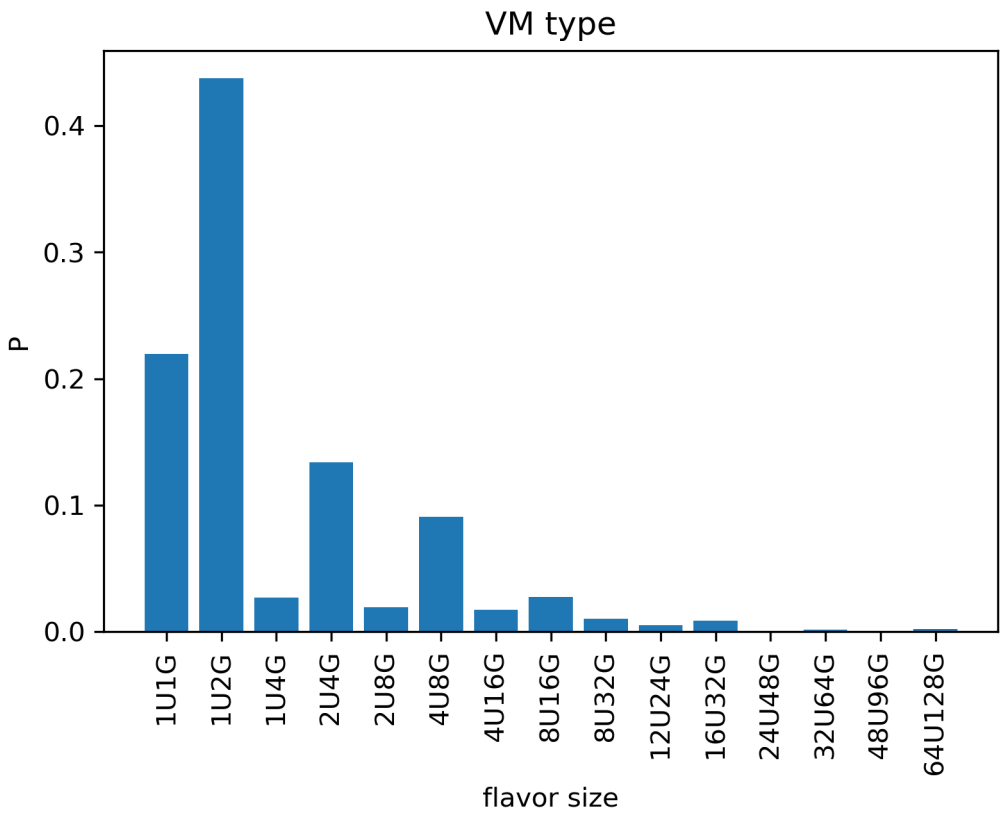
We also visualize the dynamic of virtual machine during the month:

Although there exists deletion request, the number of alive virtual machines increses from 0 to more than 8000. It should be noted that, even in the one month, the VM's dynamic is highly related to the time. `Increase, Flux, Increase, Flux` happens through the one month.

We also visualize the allocated cpu and memory dynamic above. They can be helpful in constructing domain knowledge.


CPU of VMs
P
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
1U
2U
4U
8U
12U
16U
24U
32U
48U
64U
flavor size

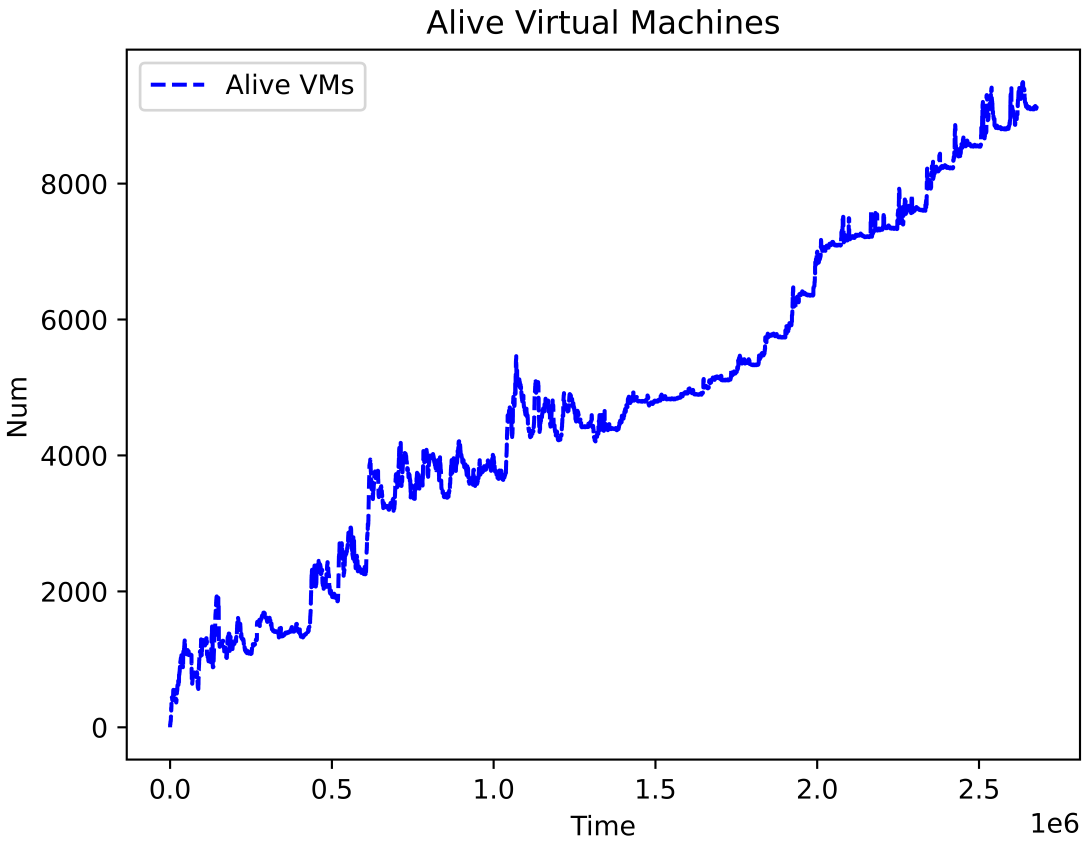

Memory of VMs
P
0.0
0.1
0.2
0.3
0.4
1G
2G
4G
8G
16G
24G
32G
48G
64G
96G
128G
flavor size

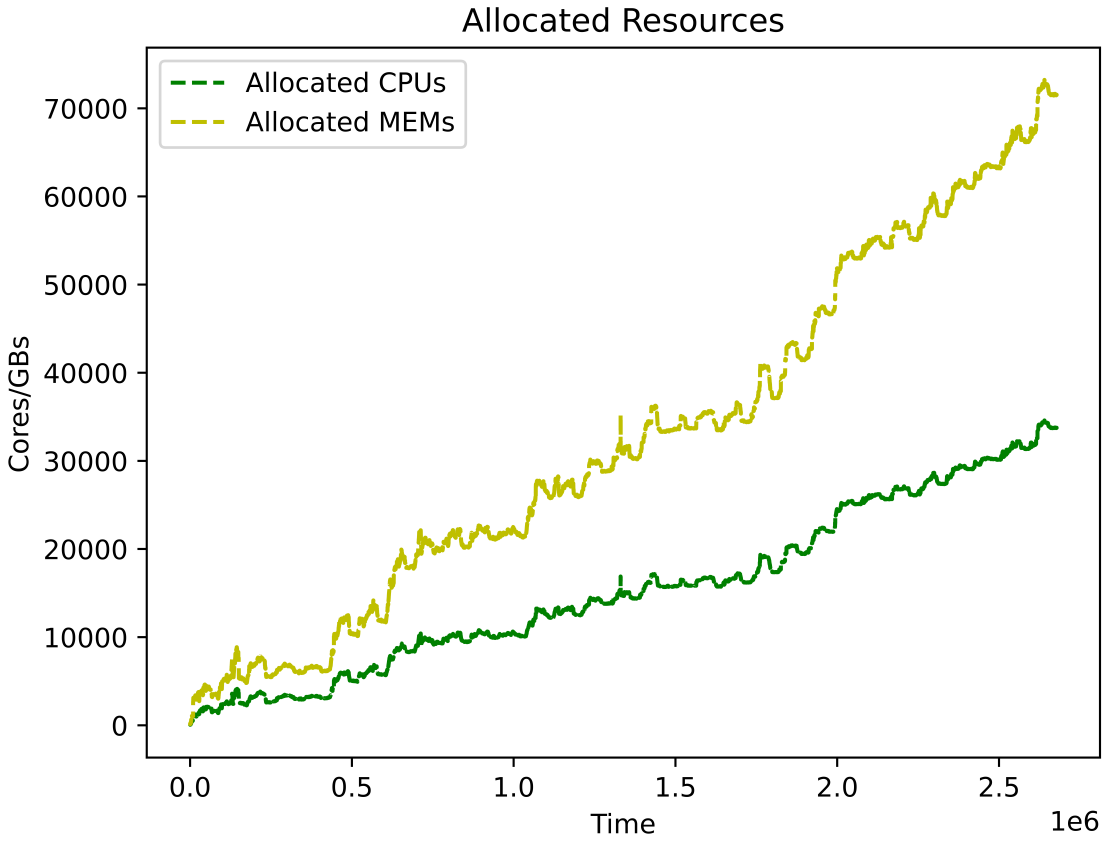


3.3 Naive Baselines performance

Another way to describe the dataset is measuring performance of naive baselines in the dataset. We adopt First-Fit and Best-Fit as the naive baselines and conduct experiments on different settings.

We conduct fading and recovering experiments with 5, 20, 50 servers and each server has 40 cpu and 90 memeory.


Alive Virtual Machines
Alive VMs
8000
6000
4000
2000
0
Num
0.0
0.5
1.0
1.5
2.0
2.5
Time
1e6


Allocated Resources
Allocated CPUs
Allocated MEMs
Cores/GBs
70000
60000
50000
40000
30000
20000
10000
0
0.0
0.5
1.0
1.5
2.0
2.5
Time
1e6

Scenario	Number of servers	Method	Number of Allocations	Terminated CPU Rate	Terminated MEM Rate
Fading	5	BestFit	211.7 ± 30	$91.6\% \pm 9.4\%$	$83.6\% \pm 9.2\%$
		First-Fit	224.5 ± 28	$98.3\% \pm 1.9\%$	$90.0\% \pm 1.9\%$
	20	BestFit	735.1 ± 83	$63.5\% \pm 29.2\%$	$35.7\% \pm 21.9\%$
		First-Fit	888.0 ± 65	$91.6\% \pm 8.5\%$	$64.7 \pm 5.6\%$
	50	BestFit	1674.5 ± 28	$91.6\% \pm 1.1\%$	$84.3 \pm 1.0\%$
		First-Fit	2298.3 ± 19	$95.5\% \pm 0.7\%$	$91.5\% \pm 0.5\%$
Recovering	5	BestFit	221.1 ± 29	$96.3\% \pm 5.6\%$	$88.1\% \pm 5.7\%$
		First-Fit	222.7 ± 27	$97.2\% \pm 3.4\%$	$89.0\% \pm 3.4\%$
	20	BestFit	850.0 ± 13	$99.1\% \pm 0$	$95.8\% \pm 0$
		First-Fit	926.1 ± 10	$98.7\% \pm 0.5\%$	$96.5\% \pm 0.3\%$
	50	BestFit	1829.6 ± 37	$92.8\% \pm 1.4\%$	$88.8\% \pm 0.2\%$
		First-Fit	2301.7 ± 19	$95.0\% \pm 0.5\%$	$91.1\% \pm 0.4\%$

CHAPTER

FOUR

GYM

Here we define the state, action, reward function and transition function in the VM scheduling.

4.1 State

SchedAgent make scheduling based on the cluster status and the current request information. For a cluster has N servers and each server has two numas. Each numa has two type of resources: cpu and memory. The cluster status is represented as a vector with Nx2x2 shape. For the request, each request contains information about cpu and memory, which makes us represent it with a vector with shape 2. Thus the whole state represented as [cluster, request] which has (Nx2x2+2) shape.

4.2 Action

In our VM scheduling, the scheduler is to select which server to handle the current request. Thus the action space of the scheduler is N. It should be noted that, if the selected server is unable to handle the request, it will be treated as invalid one. Due to the double numa architecture, for the small requet (requested cpu is smaller than a threshold) it needs to be allocated on a specific numa of a server. This makes the action space to N. For simplicity, our SchedGym makes the action space 2N and the large request is handled by action%2.

4.3 Reward

In our VMScheduling, the scheduler is to avoid termination. We denote a scheduler perform better than others if it can handle more request with the same number of resources. Thus one of the simplest way is designing reward as '+1' after the scheduler handle a request.

The '+1' reward makes a difficulty on understand the impact of large request. Thus we propose another reward `request['cpu]`. The scheduler gain more reward if it handle a larger request.

4.4 Transition Function

When a server handle a creation request (c_0, m_0), it will allocate (c_0, m_0) resource for the request. Specifically, if the server is $[[c_1, m_1], [c_2, m_2]]$ and (c_0, m_0) is a large creation request. The server will be $[[\frac{c_1-c_0}{2}, \frac{m_1-m_0}{2}], [\frac{c_2-c_0}{2}, \frac{m_2-m_0}{2}]]$. If (c_0, m_0) is a small request and server's first numa is to handle it, then the server will be $[[c_1 - c_0, m_1 - m_0], [c_2, m_2]]$. For the deletion request, the minus above will turn to add.

4.5 Interaction Example

```
import numpy as np
from schedgym.sched_env import SchedEnv

DATA_PATH = 'vmagent/data/Huawei-East-1.csv'
env = SchedEnv(5, 40, 90, DATA_PATH, render_path='../test.p',
                    allow_release=False, double_thr=32)
MAX_STEP = 1e4
env.reset(np.random.randint(0, MAX_STEP))
done = env.termination()
while not done:
    feat = env.get_attr('req')
    obs = env.get_attr('obs')
    # sample by first fit
    avail = env.get_attr('avail')
    action = np.random.choice(np.where(avail == 1)[0])
    action, next_obs, reward, done = env.step(action)
```

CHAPTER

FIVE

FRAMEWORK

Our VMAgent's framework is mainly based on the pymarl. It consists `controller`, `learner`, `components`, `modules` and `utils`.

5.1 Controller

The controller plays the role on output actions for sampling.

5.2 Learner

The learner plays is to update the agent's policy

5.3 Components

It provides key components for learning and sampling. It includes the implementations of replay memory and action selector (i.e., epsilon-greedy action selection).

5.4 Modules

It provides different networks of the agent, including critic network, policy network etc.

5.5 Utils

It provides several utils for reinforcement learning.

CHAPTER

SIX

DQN

DQN[1] is a popular off-ploicy reinforcement learning algorithm. In our VMAgent, we implement the DQN with Double Q[2] and Dueling Q[3]. The DQN agent out Q values for each server (NUMA) and we take epsilon-greedy to select action based on the Q values.

6.1 Example

Train DQN in fading environment with 5 servers, and parameters gamma=0.99 learning_rate=0.003:

```
python vmagent/train.py --env fading --alg dqn --N 5 --gamma 0.99 --lr 0.003
```

[1] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." nature 518.7540 (2015): 529-533.

[2] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." Proceedings of the AAAI conference on artificial intelligence. Vol. 30. No. 1. 2016.

[3] Sewak, Mohit. "Deep q network (dqn), double dqn, and dueling dqn." Deep Reinforcement Learning. Springer, Singapore, 2019. 95-108.

CHAPTER

SEVEN

A2C

A2C[1] is a popular Actor-Critic reinforcement learning algorithm which uses the advantage function instead of the original return in the critical network. In our VMAgent, we implement the A2C with advantage function. The criitic out the q value for each server (NUMA) and the actor out probility of actions.

7.1 Example

Train A2C in fading environment with 5 servers, and parameters gamma=0.99 learning_rate=0.003:

```
python vmagent/train.py --env fading --alg a2c --N 5 --gamma 0.99 --lr 0.003
```

[1] Schulman J, Moritz P, Levine S, et al. High-dimensional continuous control using generalized advantage estimation[J]. arXiv preprint arXiv:1506.02438, 2015.

CHAPTER

EIGHT

PPO

PPO[1] is a popular Actor-Critic reinforcement learning algorithm which forces the updata of the policy not to be large. In our VMAgent, we implement the PPO2. The criitic out the q value for each server (NUMA) and the actor out probility of actions.

8.1 Example

Train PPO in fading environment with 5 servers, and parameters gamma=0.99 learning_rate=0.003:

```
python vmagent/train.py --env fading --alg ppo --N 5 --gamma 0.99 --lr 0.003
```

[1] Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms[J]. arXiv preprint arXiv:1707.06347, 2017.

CHAPTER

NINE

SAC

SAC[1] is a popular Actor-Critic reinforcement learning algorithm whith maximum entropy RL. In our VMAgent, we implement the SAC with automatic entropy adjustment. The criitic out the q value for each server (NUMA) and the actor out probility of actions.

9.1 Example

Train SAC in fading environment with 5 servers, and parameters gamma=0.99 learning_rate=0.003:

```
python vmagent/train.py --env fading --alg sac --N 5 --gamma 0.99 --lr 0.003
```

[1] Haarnoja T, Zhou A, Abbeel P, et al. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor[C]//International conference on machine learning. PMLR, 2018: 1861-1870.

CHAPTER

TEN

VISUALIZATION USAGE

10.1 Install requirements

```
pip install -r requirements.txt
```

10.2 Start server

You can either run a development server or deploy it to a production environment.

10.2.1 Run a development server

```
python run.py
```

10.2.2 Deploy to production via gunicorn

Install `gunicorn`, and run the following command.

```
gunicorn dashboard.wsgi
```

10.3 Upload your data

After starting the service, visit and upload the data (pickle file) you want to visualize through the web page.

For the format of the file, please refer to the data format.

CHAPTER

ELEVEN

DATA FORMATS

11.1 Supported Data Formats

We support data files in two formats: `Data` or `Raw Data` in a single pickle file.

11.2 Objects

The structure and necessary fileds of each object are described below.

11.2.1 Data

Field	Type	Description
`name`	`string`	Scheduling algorithm name
`data`	`List[Frame]` (a.k.a `Raw Data`)	

11.2.2 Raw Data

```
[
    Frame1,
    Frame2,
    Frame3,
    ...
]
```

11.2.3 Frame

Field	Type	Description
`server`	`List[Server]`	The status of each server at the current time.
`request`	`Request`	The info of the request at the current time.
`action`	`int`	The resource id to which the current request is assigned, which is calculated by `server id * 2 + numa id`.

11.2.4 Server

```
[
    [CPU1, MEM1], # CPU and MEM usage of NUMA1
    [CPU2, MEM2]  # CPU and MEM usage of NUMA2
]
```

11.2.5 Request

Field	Type	Description
`cpu`	`int`	Required CPU
`mem`	`int`	Required memory
`type`	`int`	`0` for `allocation` and 1 for `release`
`is_double`	`bool` or `int`	Whether the request is a double-numa request or not

CHAPTER

TWELVE

SCHEDGYM

12.1 schedgym package

12.1.1 Submodules

12.1.2 schedgym.mySubproc_vec_env module

12.1.3 schedgym.sched_env module

12.1.4 Module contents

CHAPTER

THIRTEEN

VMAGENT

13.1 components package

13.1.1 Submodules

13.1.2 components.action_selectors module

13.1.3 components.replay_memory module

13.1.4 Module contents

13.2 controllers package

13.2.1 Submodules

13.2.2 controllers.basic_controller module

13.2.3 controllers.ppo_controller module

13.2.4 controllers.sac_controller module

13.2.5 Module contents

13.3 learners package

13.3.1 Submodules

13.3.2 learners.a2c_learner module

13.3.3 learners.ppo_learner module

13.3.4 learners.q_learner module

13.3.5 learners.sac_learner module

13.3.6 Module contents

13.4 utils package

13.4.1 Submodules

PYTHON MODULE INDEX

INDEX