Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

AI-empowered game architecture and application for resource provision and scheduling in multi-clouds

Lei Yu[1] and Yucong Duan[2*]

Abstract

Current deep learning technologies used a large number of parameters to achieve a high accuracy rate, and the number of parameters is commonly more than a hundred million for image-related tasks. To improve both training speed and accuracy in multi-clouds, distributed deep learning is also widely applied. Therefore, reducing the network scale or improving the training speed has become an urgent problem to be solved in multi-clouds. Concerning this issue, we proposed a game architecture in multi-clouds, which can be supported by resource provision and service schedule. Furthermore, we trained a deep learning network, which can ensure high accuracy while reducing the number of network parameters. An adapted game, called flappy bird, is used as an experimental environment to test our neural network. Experimental results showed that the decision logic of the flappy bird, including flight planning, avoidance, and sacrifice, is accurate. In addition, we published the parameters of the neural network, so other scholars can reuse our neural network parameters for further research.

Keywords Deep neural network, Resource provision, Model data, AI-empowered application, Game in multi-clouds

Introduction

IoT users, virtual reality application developers, or game platform operators just need to rent resources from the cloud providers without maintaining the cloud system. However, a single cloud may not have sufficient services for applications in different regions, and the lonely cloud sometimes has performance degradation. Therefore, it is not a feasible way to rely solely on a single cloud to provide all resources and services for some users to avoid the cloud provider lock-in.

The resource capacity of the cloud is different and the service request arrivals are dynamic. Cloud users will have bad quality of experience (QoE) if the resources are not enough. To provide good QoE cost-effectively in multi-clouds, an effective resource provision and service scheduling method is critical, which will reduce network congestion or system crashes. However, AI applications that are deployed on it should change their architecture to better adapt to multi-cloud scenarios.

Deep learning techniques are good at dealing with complex and dynamic systems. It is common for deep learning to have hundreds of millions of parameters and hundreds of megabytes of storage space. Therefore, it is an urgent problem to reduce the network size and improve the training speed in multi-cloud. To solve this problem:

*Correspondence:
Yucong Duan
duanyucong@hotmail.com
[1] Department of Computer Science, Inner Mongolia University, Hohhot, China
[2] Department of Data Science and Big Data Technology, Hainan University, Haikou, China

(1) We trained a deep learning network, which can ensure high accuracy while reducing the scale of network parameters.
(2) We adapted a flappy bird game, which is originally played by human beings, to build an evaluation environment.

Through the mouse or the keyboard, players in the game have two action options. One action is to make the bird flap its wings, and the other action is to do nothing. If the bird flaps its wings, it will experience an upward force, which will cause an upward acceleration, and the bird will fly upward. If the player does nothing, the bird will fall to the ground due to gravity. The flappy bird may touch both the upper and lower pipes during flight. When it hits them, the game is over and the flappy bird gets a negative payoff. The purpose of the bird is to pass most pipes to get the highest payoff. We trained a deep neural network to enable computers freely choose between two actions previously chosen by humans: the bird flaps its wings or not at a given point in time. To increase the randomness, we added random factors to the environment, such as the height of the pipes, the spacing between the upper pipes and the lower pipes, and the location of the pipes. However, the distance between the two pipes in the horizontal direction is fixed, and this fixed form does not affect the learning ability of the neural network and the generalization ability of the neural network.

Related works

Cloud computing has significantly promoted the alteration of many industries [1]. Myers et al. [2] used cloud resources by attaching the storage capacity and web server. Service scheduling [3] in the multi-cloud environment is a multi-constraint, multi-objective, and multi-type optimization problem [4], where traditional basic scheduling algorithms do not consider the real load and linking status of the work node. The scheduling problem is to find the optimal group of resources satisfying multiple constraint objectives, which are combinatorial optimization problems [5].

Panwar et al. [6] divided task scheduling into two stages to reduce task time and improve cloud resource utilization. Taking both the execution period and cost into account, Chen et al. [4] modeled cloud scheduling [7] as an optimization problem and proposed a multi-objective ant colony system. George et al. [8] used the Cuckoo Search algorithm for minimizing the computation time of tasks, while Ghasemi et al. [9] proposed a scheduling method for minimizing the processing time and transmission cost. Compared with traditional scheduling algorithms, heuristic algorithms have a robust ability for optimization, but still have slow convergence, and easily fall into local optimal solutions.

Considering the dynamic nature of computing resources and the heterogeneity of cloud platforms, Deep Reinforcement Learning (DRL) shows continuous decision-making ability [10] for resource allocation and service scheduling policies in cloud environments [11]. Cheng et al. [12] designed a scheduler combining resource and scheduling based on Deep Q-Learning, which reduced the energy consumption and the task rejection rate in the cloud. Based on the Deep Q-Learning algorithm, Wei et al. [13] proposed a QoS-aware scheduling framework that can reduce the average response time of jobs under variable loads. Meng et al. [14] designed an online server-side task scheduling algorithm by combining reinforcement learning with DNN. Ran et al. [15] used the Deep Determining Policy Gradient (DDPG) algorithm to find the optimal task assignment structure. Zhang et al. [16] proposed a multi-task scheduling algorithm based on deep reinforcement learning to reduce the completion time of the job. Dong et al. [17] proposed an algorithm to vigorously schedule tasks that have priority associations in the cloud manufacturing environment. For industrial IoT, an AoI-aware energy control and computation offloading method [18] is proposed to enhance intelligence.

Without discussing the related services and their diversity, references [6, 11, 19] discussed a single service type, while references [7, 20] just gave the resource weight coefficients. References [16, 17] did not consider the data transmission cost in the scheduling process of composite services. References [4, 10, 20, 21] did not take into account the service scheduling parallelism. Although artificial intelligence and game theory are well applied to this topic, they are used in specific cloud environments [22]. In addition, we compared parts of the algorithms in Table 1.

Table 1 Summary

References	Solved problem	Process	Advantage
[23]	Action is of low importance in some states	The advantages and disadvantages of state and action, are respectively analyzed	Lessened the range of Q-value
[24]	Overestimation	Decomposing the max operation	More stable training results
[25]	Changed samples in experience replay	Improved the experience buffer training policy	Improved the performance of DDQN
[4]	Optimized execution time and cost	A pheromone update rule is designed	Better global search ability
[6]	Improved resource utilization, processing cost, and transmission time	The task scheduling is performed in two phases	Reduced makespan for tasks
[9]	Optimized load balancing	Each firefly flies towards a firefly that looks brighter than itself	Reduced transmission cost of workflow
[13]	Met the QoS requirements of users	Learned from its experiences without prior knowledge	Improved user satisfaction
[14]	Delay-sensitive task scheduling	Designed a reward function to reduce the average timeout period of tasks	Improved the scheduling efficiency of server-side tasks
[15]	Model-free policy for continuous action	Combines DPG and DQN	continuous action space
[16]	Scalable parallel tasks	A fully connected layer and an output layer	Improved task scheduling performance
[20]	Reduced energy consumption	Used the task priority to calculate the critical resources of the task graph	Reduced energy consumption in the data center
[26]	Optimize multiple objectives	Trained two DRL-based agents as scheduling agents	Reduced the average job duration
[27]	The efficiency of resource management	Proposed a blacklist mechanism	Converged quickly

AI-empowered game architecture and application

The software components in one cloud application can be wrapped by cloud services, and these cloud services can be deployed in multi-clouds when a service registration component is ready. In addition, offloading and resource allocation should be concerned in certain areas. Figure 1 shows a deployment graph for multi-cloud applications, where the gateways represent service registrations (Zookeeper, Eureka, Nacos, or Consul). In addition, load balancing, fault handling, routing, etc. can be added by Eureka, Ribbon, Feign, and Hystrix services. The component of resource provision and service schedule can be deployed in other network nodes.

Cloud applications including games can be divided into multiple software components, and these software components can be deployed in different places, such as different servers in one cloud, or even in different clouds. Therefore, a distributed game theoretical and credibility-guaranteed approach [28] is suitable for this purpose. Figure 2 shows an architecture of a cloud game that is deployed in different clouds, where each software component interacts with the other. This architecture is suitable for more broad applications in multi-clouds.

The player in our game controlled a flappy bird and made it fly over obstacles made of pipes of different lengths, avoid humans, and attack wolves. The game ends when the flappy bird hits the pipe or falls to the bottom of the screen. The flappy bird will obtain a positive payoff for each pipe it passes, and the player needs to score as many points as possible.

Our method is based on reinforcement learning, and Fig. 3 shows the method framework. The test begins in the process of 'Finish Train and Validation'. To reduce the number of neural network parameters, we adjusted the input layer and preprocessed the input data. The neural network parameters are trained on batches by replay buffers and a current optimal strategy.

Algorithm 1 shows the process in each game step: the first pipe will be removed if it is out of the screen before the next blocker is shown. Then we check if the bird, the human, the wolf, the upper pipes, and the lower pipes crash by the function checkCrash (player, upperPipes, lowerPipes, blocker). The function returns 'hard' if the player collided with the ground or pipes and obtains different rewards.

```
step()     # one game step
    if upperPipes[0]['x'] < -PIPE_WIDTH      # 'x' of the first upper pipe
        upperPipes and lowerPipes pop(0)    # remove the first upper pipe and the lower pipe
        if blocker['x'] < - BLOCKER_WIDTH
            if random.random() > P_BLOCKER
                blockerType = 'human' if random.choice([0,1],p=[0.2,0.8]) == 0 else 'wolf'
    crash_type = checkCrash(player, upperPipes, lowerPipes, blocker)
    if player.y == 0
        terminal = True
        reward = -1
    if crash_type == 'hard' or crash_type == 'blocker'
        if crash_type == 'hard'      # GROUND or pipes
            terminal = True
            reward = -1
        else:
            blocker['x'] = -500
            if blocker['blockerType'] == 'human'
                terminal = True
                reward = -3
            if blocker['blockerType'] == 'wolf'
                reward = 2
    return state, reward, terminal
checkCrash(player, upperPipes, lowerPipes, blocker)
    if player.y >= GROUND
        return 'hard'
    else
        for each uPipe, lPipe
            uCollide = Collision(bird, uPipe)
            lCollide = Collision(bird, lPipe)
            if uCollide or lCollide        # the bird collide a pipe
                return 'hard'
        bCollide = Collision(bird, blocker)    # the bird collide a blocker
        if bCollide
            return 'blocker'
    return 'none'
```

Algorithm 1.

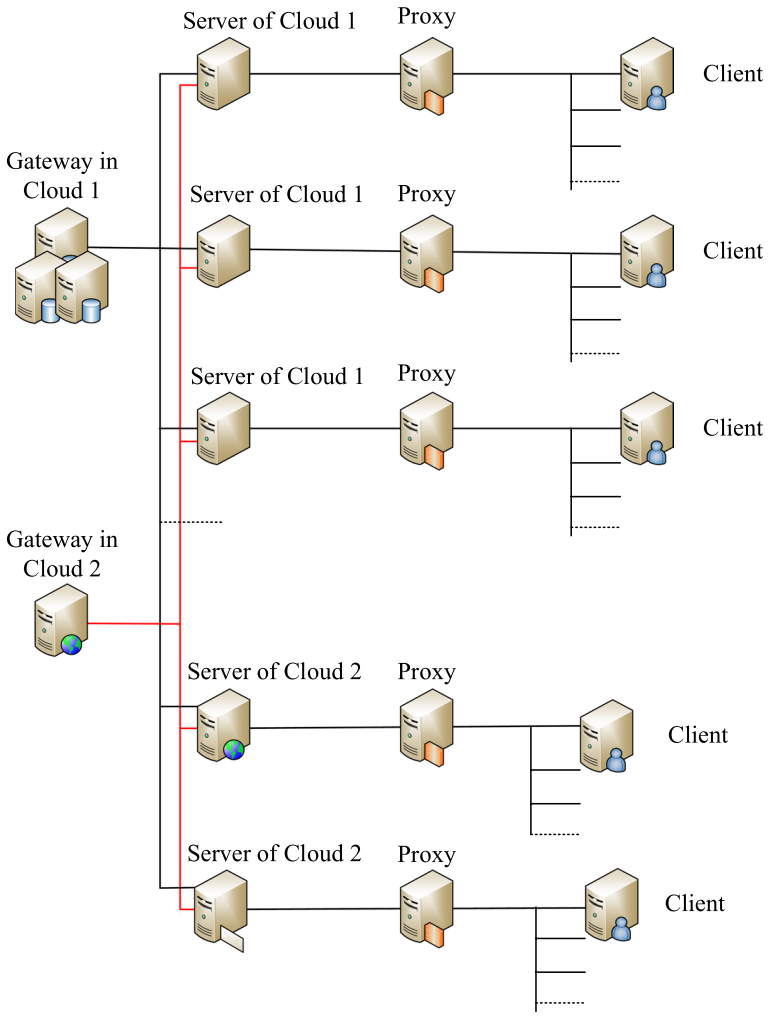

Fig. 1 Deployment graph for multi-cloud applications

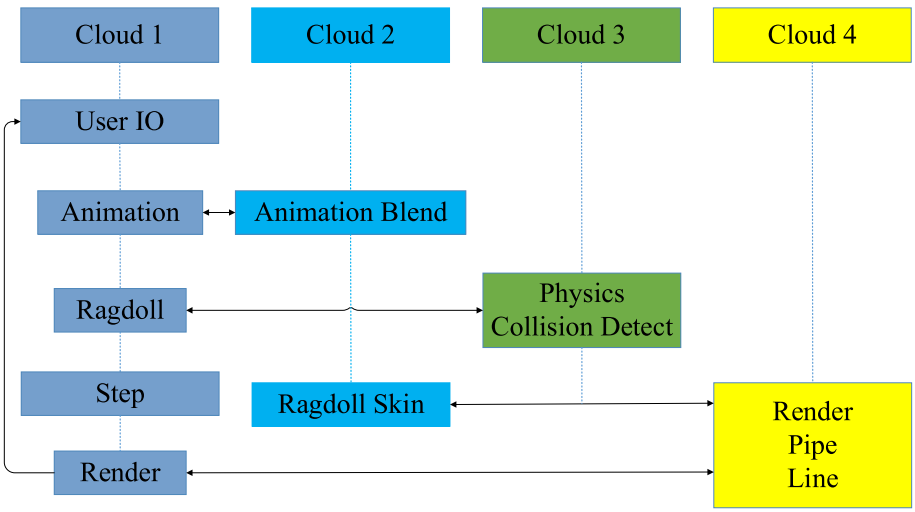

Fig. 2 Architecture of cloud games

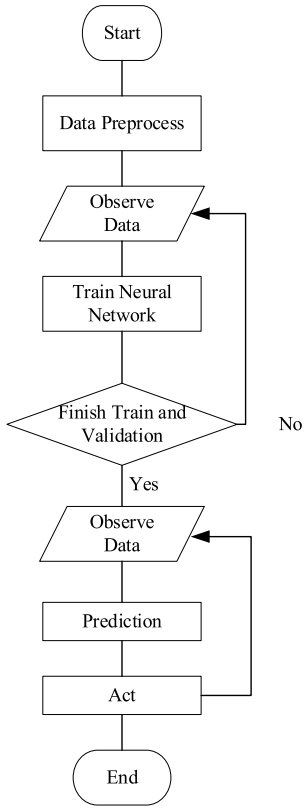

Fig. 3 Framework

Axis Aligned Bounding Boxes or Oriented Bounding Boxes and Bounding Spheres are often used for collision detections:

$$\begin{cases} Radius = \frac{\sqrt{(x_{max}-x_{min})^2+(y_{max}-y_{min})^2+(z_{max}-z_{min})^2}}{2} \\ Dist = ||Center_a - Center_b|| \\ Dist < Radius_a + Radius_b \rightarrow intersect(a,b) \end{cases} \quad (1)$$

Experiments and data analysis

We trained the neural network by having Al and the player play the adapted flappy bird game, and recorded their actions during the game. Table 2 shows the parameter settings for recording the data. In addition, the neural network training process and data collection process are shown in two videos.

Game data and analysis

The game speed, data collected speed, and neural network training speed can be accelerated in many ways. Parts of the game parameters are used to simulate a true physical world. Notice that X and Y are screen coordinates, and there is a relative movement between the bird and the pipes:

Table 2 Parameters

Parameters	Meanings
Gamma = 0.99	The decay rate of past observations
Observation = 320	Timesteps to observe before training
Batch = 320	Size of minibatch
Explore = 20,000	Frames over which to anneal epsilon
Initial_epsilon = 0.1	Starting value of epsilon
Final_epsilon = 0.0001	The final value of epsilon
Replay_memory = 50,000	Size of experiences
Frame_per_action = 1	Actions can be used per frame
Learning_rate = 1e-4	Learning rate
Actions = 2	Number of valid actions

1. velocity along X of pipe, human, and wolves: -4
2. bird's starting velocity along Y: 0
3. bird max velocity along Y or max descend speed: 10
4. bird min velocity along Y or max ascend speed: -8
5. bird downward acceleration: 1
6. bird acceleration on flapping: -9

The actions of Al were recorded. The neural network training process and data collection process are shown in two videos. Video 1 is divided into two parts. The first part shows the results of our deep neural network at the initial state of training. The results can be directly watched through the game images. At this time, the flappy bird flaps its wings at random, does not learn some strategies, does not avoid people, and occasionally attacks the wolf. The second part of the video shows the results of the model of the trained neural network. This model makes the bird avoid all pipes, the ground, and the sky while ensuring that it only attacks wolves, not humans. Under this situation, the flappy bird with the current model is suicidal.

Model data of deep neural network

Table 3 shows that our neural network only needs to train 69,506 parameters to implement all the decision logic of the flappy bird, including flight planning, avoidance, and

Table 3 Model data summary

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 512)	3584
dense_2 (Dense)	(None, 128)	65,664
dense_3 (Dense)	(None, 2)	258

Total params: 69,506

Trainable params: 69,506

Non-trainable params: 0

sacrifice. This is a small neural network than neural networks that often require millions or hundreds of millions of training parameters because we optimized the algorithm and do not use Convolutional Neural Networks (CNN). Comparing CNNs that take all information as input, we found that human knowledge and elaborate designing can significantly reduce the input size of the neural network, and then reduce the whole size of the neural network in the experiments. That is why our neural network is small and the training is extremely fast.

In addition, the following methods are used in the neural network: Activation: relu, Loss: mse, Optimizer: adam. The human and wolf in the lower pipe can be regarded as two organisms blocking the progress of the flappy bird. The neural network shown in video 2 trained a cruel flappy bird. To achieve its goal of passing through most pipes, the bird attacks any creature. The goal of training the flappy bird is just to keep it from falling or hitting the pipe, so we can see that it has already passed through 410 pipes.

Interpreting the two systems

We added some game environment parameters. For example, the spacing of pipes is random, and the other randomness is reflected in the timing of the appearance of humans and wolves. Therefore, the entire game environment is uncertain, requiring a lot of time to debug the game and deep learning modules.

The whole software system consists of two modules. The first module is responsible for the game's operation and display, including the collision effect and animation effect. And the other module is responsible for letting Al play the game, which uses deep learning. Software bugs generated by these two modules will affect each other, causing a bug cycle, so debugging takes a certain amount of time. However, there are many skills in the software development process and the process of deep learning and training. Using these skills can speed up software development, system debugging, and training, which can avoid endless bug cycles. Sometimes, game bugs slow down deep learning, and even fail deep learning. Sometimes, deep learning bugs affect the game, so it is a mutually influencing system. Therefore, when readers reuse our dataset, they need to use it in conjunction with the game.

Self-control and explainable AI

Some scholars now believe that neural networks have a rudimentary self-consciousness. Our neural network also has some self-consciousness because it is composed of three psychological components: self-awareness, self-experience, and self-control, which are interrelated and mutually restricted. Self-awareness includes self-perception, self-analysis, and self-criticism. Our bird is written by program code. Without self-analysis and self-perception, it cannot live longer in this virtual environment. The second point of self-consciousness is self-experience. Individuals can feel self-love, self-esteem, a sense of responsibility, a sense of obligation, and a sense of superiority. The third point of self-consciousness is self-control. The main elements of self-control are self-restraint and self-discipline. The strongest point of our bird is self-control. Therefore, from the definition point of view, our neural network already has a certain degree of self-awareness. According to the design of the parameters of our neural network, an artificial agent can be trained into a cold agent, a sentimental agent, or an agent with morality, obligation, and self-love.

In video 1, we adjusted and optimized the neural network parameters to create a new agent. The agent in the face of different social groups has a certain sense of self-consciousness and can make different judgments. In other words, it can make different choices depending on whether the blocker is a person or a wolf. It can choose to commit suicide to save human life or sacrifice the wolf so that the bird can live longer. In the latter case, we can imagine that the bird carries important information and the cost of passing on the information is the loss of a wolf or the loss of all the wolves in the path.

Conclusion

Cloud applications including games can be divided into multiple software components, and these software components can be deployed in different clouds. We proposed an architecture that is suitable for more broad applications in multi-clouds, which can be supported by resource provision and service schedule. To reduce the number of neural network parameters in multi-clouds, we adjusted the input layer and preprocessed the input data. The neural network parameters are trained on batches by replay buffers and a current optimal strategy. Our method is based on deep reinforcement learning, while current deep reinforcement learning technologies use a large number of parameters and the number of parameters is commonly huge. To improve both training speed and accuracy in multi-clouds, distributed deep learning is also widely applied. An adapted game, called flappy bird, is used as an experimental environment to test our neural network. Experimental results showed that the decision logic is accurate. In addition, the results show that our neural network only needs to train 69,506 parameters to implement all the decision logic of the flappy bird, including flight planning, avoidance, and sacrifice because we optimized the algorithm and do not use Convolutional Neural Networks.

Authors' contributions
Y.L. and YC.D. wrote the main manuscript text and all authors reviewed the manuscript.

Funding
This work was supported by Natural Science Foundation of Inner Mongolia Autonomous Region (No. 2022MS06024), NSFC (No. 61962040), Hainan Province Key R&D Program(ZDYF2022GXJS007,ZDYF2022GXJS010), Hainan Natural Science Foundation(620RC561), Hainan Province Higher Education and Teaching Reform Research Project(Hnjg2021ZD-3).

Availability of data and materials
The datasets generated for this study can be found here: sinacloud.net/dump123/model.h5.
The two videos generated for this study can be found here:
video 1: https://www.bilibili.com/video/BV1eZ4y1a7MJ/.
video 2: https://www.bilibili.com/video/BV1u5411Q7rs/.

Declarations

Ethics approval and consent to participate
The research has consent for Ethical Approval and Consent to participate.

Consent for publication
Consent has been granted by all authors and there is no conflict.

Competing interests
The authors declare no competing interests.

Received: 5 December 2022 Accepted: 14 August 2023
Published online: 30 August 2023

References

1. Chen Y, Hu J, Zhao J, Min G (2023) Qos-aware computation offloading in Leo satellite edge computing for IoT: A game-theoretical approach. Chin J Electron 33:1–12
2. Myers TA, Chanock SJ, Machiela MJ (2020) Ldlinkr: An r package for rapidly calculating linkage disequilibrium statistics in diverse populations. Front Genet 11:157
3. Chen Y, Zhao J, Wu Y, Huang J, Shen XS (2022) Qoe-aware decentralized task offloading and resource allocation for end-edge-cloud systems: A game-theoretical approach. IEEE Trans Mob Comput 21:1–17
4. Chen Z-G, Zhan Z-H, Lin Y, Gong Y-J, Gu T-L, Zhao F et al (2019) Multiobjective cloud workflow scheduling: a multiple populations ant colony system approach. IEEE Trans Cybern 49:2912–2926. https://doi.org/10.1109/TCYB.2018.2832640
5. Chen Y, Zhao J, Hu J, Wan S, Huang J (2023) Distributed task offloading and resource purchasing in NOMA-enabled mobile edge computing: hierarchical game theoretical approaches. ACM Trans Embed Comput Syst
6. Panwar N, Negi S, Rauthan MMS, Vaisla KS (2019) Topsis–pso inspired non-preemptive tasks scheduling algorithm in cloud environment. Clust Comput 22:1379–1396. https://doi.org/10.1007/s10586-019-02915-3
7. Fangzheng Liu, Jiwei Huang, Xianbin Wang (2023) Joint Task Offloading and Resource Allocation for Device-Edge-Cloud Collaboration with Subtask Dependencies. IEEE Trans Cloud Comput. https://doi.org/10.1109/TCC.2023.3251561
8. George N, Chandrasekaran K, Binu A (2016) Optimization-aware scheduling in cloud computing. In: Proceedings of the International Conference on Informatics and Analytics. Pondicherry India, pp 1–5
9. Ghasemi S, Kheyrolahi A, Shaltooki AA (2019) Workflow scheduling in cloud environment using firefly optimization algorithm. JOIV Int J Inf Vis 3:237–242. https://doi.org/10.30630/joiv.3.3.266
10. Orhean AI, Pop F, Raicu I (2018) New scheduling approach using reinforcement learning for heterogeneous distributed systems. J Parallel Distributed Comput 117:292–302. https://doi.org/10.1016/j.jpdc.2017.05.001
11. Liang J, Ma B, Feng Z, Huang J (2023) Reliability-aware task processing and offloading for data-intensive applications in edge computing. IEEE Trans Netw Serv Manage. https://doi.org/10.1109/TNSM.2023.3258191
12. Cheng M, Li J, Nazarian S (2018) Drl-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers. In: IEEE 23rd Asia and South Pacific design automation conference, Jeju, Korea (South). pp 129–134
13. Wei Y, Pan L, Liu S, Wu L, Meng X (2018) Drl-scheduling: An intelligent QoS-aware job scheduling framework for applications in clouds. IEEE Access 6:55112–55125. https://doi.org/10.1109/access.2018.2872674
14. Meng H, Chao D, Huo R, Guo Q, Li X, Huang T (2019) Deep reinforcement learning based delay-sensitive task scheduling and resource management algorithm for multi-user mobile-edge computing systems. In: 4th International Conference on Mathematics and Artificial Intelligence, Chengdu China. pp 66–70
15. Ran L, Shi X, Shang M (2019) Slas-aware online task scheduling based on deep reinforcement learning method in cloud environment. In: IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Zhangjiajie, China. pp 1518–1525
16. Zhang L, Qi Q, Wang J, Sun H, Liao J (2019) Multi-task deep reinforcement learning for scalable parallel task scheduling. In: IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA. pp 2992–3001
17. Dong T, Xue F, Xiao C, Li J (2020) Task scheduling based on deep reinforcement learning in a cloud manufacturing environment. Concurr Comput Pract Exper 32:e5654. https://doi.org/10.1002/cpe.5654
18. Huang J, Gao H, Wan S, Chen Y (2023) Aoi-aware energy control and computation offloading for industrial IoT. Futur Gener Comput Syst 139:29–37
19. Almezeini N, Hafez A (2017) Task scheduling in cloud computing using lion optimization algorithm. Int J Adv Comput Sci Appl 8. https://doi.org/10.14569/ijacsa.2017.081110
20. Xiaoqing Z, Yajie H, Chunlin A (2018) Data-dependent tasks rescheduling energy efficient algorithm. In: IEEE 4th International Conference on Computer and Communications, Chengdu, p 2542–2546
21. Chen Y, Gu W, Xu J, Zhang Y, Min G (2023) Dynamic task offloading for digital twin-empowered mobile edge computing via deep reinforcement learning. In: China Communications. pp 1–12
22. Huang J, Wan J, Lv B, Ye Q, Chen Y (2023) Joint computation offloading and resource allocation for edge-cloud collaboration in internet of vehicles via deep reinforcement learning. IEEE Syst J 17:2500–2511
23. Wang Z, Schaul T, Hessel M, Hasselt H, Lanctot M, Freitas N (2016) Dueling network architectures for deep reinforcement learning. In: International conference on machine learning, New York NY USA. pp 1995–2003
24. Van Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double q-learning", Phoenix, Arizona USA. Proc. AAAI Conf. Artif. Intell.30, pp 2094–2100
25. Schaul T, Quan J, Antonoglou I, Silver D (2016) Prioritized experience replay. In: International Conference on Learning Representations, San Juan, Puerto Rico. https://doi.org/10.48550/arXiv.1511.05952
26. Islam MT, Karunasekera S, Buyya R (2021) Performance and cost-efficient spark job scheduling based on deep reinforcement learning in cloud computing environments. IEEE Trans Parallel Distrib Syst 33:1695–1710. https://doi.org/10.1109/tpds.2021.3124670
27. Liang S, Yang Z, Jin F, Chen Y (2020) Data centers job scheduling with deep reinforcement learning. Advances in knowledge discovery and data mining. Springer International Publishing, New York, pp 906–917
28. Chen Y, Zhao J, Zhou X (2023) A distributed game theoretical approach for credibility-guaranteed multimedia data offloading in MEC. In: Information Sciences

Publisher's Note