Cost-Efficient and Quality-of-Experience-Aware Player Request Scheduling and Rendering Server Allocation for Edge-Computing-Assisted Multiplayer Cloud Gaming

Yongqiang Gao, Chaoyu Zhang, Zhulong Xie, Zhengwei Qi, and Jiantao Zhou, *Member, IEEE*

***Abstract*—Prompted by the remarkable progress in both cloud computing and GPU virtualization, cloud gaming has been attracting more and more attention in the gaming industry. With the cloud gaming model, players do not need to download or install the game on local devices, and constantly upgrade their devices. Despite these advantages, cloud gaming faces several challenges for its success, including long response delay, poor game fairness, and high operational cost. To this end, this article proposes an edge computing-assisted multiplayer cloud gaming system named ECACG to improve multiplayer cloud gaming experiences and operating costs by offloading the game rendering task to the nearby edge server. Based on the ECACG, two decision processes are completed. One is player request scheduling and the other is rendering server allocation. The decision problem is formulated into a constrained multiobjective optimization model. A novel hybrid algorithm based on deep reinforcement learning and heuristic strategy is developed to solve the optimization problem. The effectiveness of the proposed ECACG is evaluated by simulation experiments based on the real-world parameters. The simulation results show that compared with the existing schemes, the proposed ECACG can achieve lower rental costs and better fairness, while providing the good-enough response delay for players.**

***Index Terms*—Cloud gaming, deep reinforcement learning, edge computing, request scheduling, server allocation.**

I. Introduction

PROMPTED by the noteworthy progress in cloud computing, recent years have witnessed the explosive growth of cloud gaming services. According to the latest research report [1] published by the consultancy Newzoo in 2020, the global cloud gaming market accounts for U.S. $585 million in 2020 and is expected to reach U.S. $4.80 billion by 2023. This arresting profit has attracted many companies to develop their own cloud gaming platforms, e.g., Google Stadia [2], Microsoft XCloud [3], and Nvidia GeForce Now [4]. In cloud gaming, games are deployed on cloud servers, and gamers can access it via thin clients with Internet connections. With the cloud gaming platform, gamers can easily play 3-D video games on any device, such as desktop PCs, tablets, and smartphones, whenever and wherever possible.

This article focuses on multiplayer cloud gaming, which is a new type of multiplayer online gaming in the cloud gaming era. Multiplayer cloud gaming service typically employs a remote rendering architecture that is composed of two major components: 1) game server and 2) rendering server. Game servers are identical to that in traditional multiplayer online gaming, and in charge of executing the game logics, such as maintaining consistent game states among players, processing players' login/logout event, and storing players' information. Rendering servers play two roles. One is to appear as the "clients" that are connected to the game servers to exchange game states. The other is to act as the "servers," which are utilized to render game scenes, and then stream it to players operating on thin-client devices. Thin-client devices in this architecture are only responsible for sending user interaction inputs and displaying received game scenes.

With a multiplayer cloud gaming model, players do not need to download or install the game on local devices, and constantly upgrade their devices. Despite these advantages, multiplayer cloud gaming currently faces several challenges that hinder it from becoming a leading gaming model. First, multiplayer cloud gaming may undergo long response delay because most computing tasks are offloaded to remote cloud servers. Second, multiplayer cloud gaming may suffer from unfairness among players because the player diversity (e.g., geographical locations and network conditions) brings out response delay differences. Third, multiplayer cloud gaming may have high operating costs because its service providers usually need to build and maintain a large number of cloud datacenters for increasing player coverage.

Recently, edge computing has been proposed as a promising technique to support multiplayer cloud gaming [5]. Edge

Manuscript received June 26, 2021; revised November 15, 2021; accepted November 29, 2021. Date of publication December 6, 2021; date of current version July 7, 2022. This work was supported in part by the National Natural Science Foundation of China under Grant 61662052; in part by the Natural Science Foundation of Inner Mongolia Autonomous Region under Grant 2021MS06002; in part by the Science and Technology Planning Project of Inner Mongolia Autonomous Region under Grant 2021GG0155; and in part by the Major Research Plan of Inner Mongolia Natural Science Foundation under Grant 2019ZD15. *(Corresponding author: Yongqiang Gao.)*

Yongqiang Gao, Chaoyu Zhang, and Jiantao Zhou are with the College of Computer Science, Inner Mongolia University, Hohhot 010021, China (e-mail: gaoyongqiang@imu.edu.cn; 31909073@mail.imu.edu.cn; cszjtao@imu.edu.cn).

Zhulong Xie is with the Research and Development Department, Perfect World Company Ltd., Beijing 100101, China (e-mail: xiezhulong@pwrd.com).

Zhengwei Qi is with the School of Software, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: qizhwei@sjtu.edu.cn).

Digital Object Identifier 10.1109/JIOT.2021.3132849

computing has several advantages over traditional cloud computing. By shifting the rendering tasks from the cloud to the edge, edge computing can offer cloud gaming service closer to players to achieve lower delay. In addition, large coverage range makes it suitable for serving a large scale of players from all over the world in a cost-efficient fashion. Finally, edge computing can provide the flexible end-to-end communication in the edge of the network, which makes it easier to achieve fairness among multiple players. However, compared with cloud servers, edge servers have limited resources, which makes it difficult to run resource-intensive video games on edge servers. Therefore, the combination of edge computing and cloud computing provides an excellent opportunity to achieve satisfactory gaming experience at low cost.

To this end, this article proposes an edge computing-assisted multiplayer cloud gaming system that enables cloud gaming service with acceptable operating cost and gaming experience. In the proposed system, the game logics are still performed in cloud servers, but the rendering tasks are executed on edge servers, which are adjacent to the player and connected to the cloud. With the advantages of edge computing in terms of the reduction in delay, the improvement in flexibility, and the reduction in cost, the proposed system can shorten response delay, ensure relatively fairness among players, and reduce operation cost. Specifically, the main contributions of this article are summarized as follows.

1) An edge computing-assisted multiplayer cloud gaming system, called ECACG, is proposed to improve the cloud gaming experiences and operating costs by efficiently scheduling the player gaming request and allocating the virtual rendering server to the suitable edge node.
2) A constrained multiobjective optimization model is developed for player request scheduling and rendering server allocation in the EDCAG. A hybrid algorithm based on the deep reinforcement learning and heuristic strategy is designed to address the multiobjective optimization problem.
3) Extensive simulations are carried out to prove the effectiveness of the proposed system. The simulation results show that compared with other alternatives, the proposed system can achieve lower rental costs and better fairness, while providing the good-enough response delay for players.

The remainder of this article is organized as follows. The related works are summarized in Section II. Section III describes the system design and problem formulation. The proposed hybrid algorithm is presented in Section IV. In Section V, the experimental evaluations and results are discussed. Finally, concluding remarks are given in Section VI.

II. Related Work

Many efforts have been devoted to reducing the latency in cloud gaming platform. Han *et al.* [6] developed a distributed algorithm based on the potential game theory to optimize virtual machine placement in mobile cloud gaming through resource competition with the goal of minimizing player gaming experience loss. Sabet *et al.* [7] identified five game characteristics that can influence the delay sensitivity of cloud gaming, and presented a latency compensation technique that can improve cloud gaming experiences by utilizing these characteristics to adapt the game. Li *et al.* [8] proposed a novel cloud gaming system named *Themis* that applies reinforcement learning to partition the resource with the objective of minimizing the response delay in the cloud gaming system. Zhang *et al.* [9] presented a novel framework named *EdgeGame* to adaptively adjust the video bitrates in order to reduce network delay and bandwidth consumption. Amiri *et al.* [10] proposed a hierarchical software-defined network (SDN) controller architecture that utilizes the online convex optimization to allocate player requests to suitable data centers with the goal of minimizing network delay and maximizing bandwidth utilization. Chen *et al.* [11] addressed the rendering server allocation problem that aims at minimizing the interplayer delay, while preserving good-enough response delay experienced by players. These studies are different from that in this article because they focus solely on improving the cloud gaming experiences and thus, cannot provide cost savings.

There is also some work focusing on minimizing the cloud gaming provider's cost while satisfying the quality of experience requirement of gamers. Basiri and Rasoolzadegan [12] presented a resource allocation framework in cloud gaming to minimize the operating costs while satisfying the quality of experience requirement of users. Deng *et al.* [13] investigated the server allocation problem for multiplayer cloud gaming with the goal of minimizing the total server rental and bandwidth cost under real-time latency constraint. Wu *et al.* [14] presented an online control algorithm to perform intelligent request dispatching and server provisioning with the goal of reducing the provisioning cost of cloud gaming service providers while still ensuring the quality of experience requirements. Tian *et al.* [15] applied the Lyapunov optimization theory to adaptively adjust virtual machine allocation, and video streaming bit rate settings in a distributed cloud gaming system with the objective of minimizing the overall service cost while providing good-enough quality of experience for players. Jaya *et al.* [16] proposed a cloud gaming framework for MMORPG, which consists of player, rendering server, map server, and central game server. Several online heuristics were developed to allocate rendering servers in order to minimize the rental cost of rendering servers under latency constraints. Different from these studies, this article proposes an edge computing-assisted cloud gaming framework, and employs a hybrid of deep reinforcement learning and heuristic strategy to schedule player requests and allocate rendering servers with the goal of minimizing both the maximal delay difference among interacting players and the rental costs of rendering servers, while providing the good-enough response delay for players.

Recently, edge computing has been widely applied in resource management owing to its high bandwidth and ultralow latency. Mao *et al.* [17] investigated the multidimensional resource allocation scheme in a wireless-powered mobile edge computing system with the goal of minimizing the energy consumption of the system by using

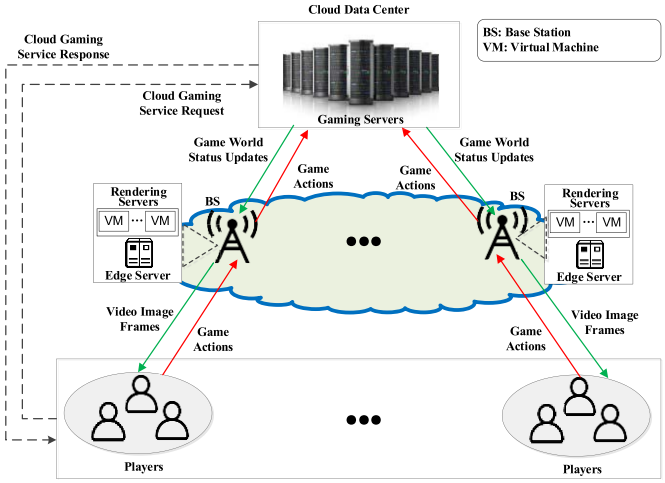


Fig. 1. System architecture.

computation and communication cooperation among users. Feng *et al.* [18] developed an online algorithm based on Lyapunov optimization to minimize the total system cost while ensuring the overall system performance in mobile-edge computing. A resource scheduling algorithm based on block coordinate descent and successive convex approximation [19] was proposed to minimize the maximum task execution delay among Internet of Things devices by the efficient deployment of multiple unmanned aerial vehicles and the fair allocation of computing resources in space-aerial-assisted cloud-edge computing systems. Zeng *et al.* [20] proposed a volunteer-assisted collaborative offloading scheme in vehicular edge computing to maximize the utility of the edge servers by efficiently utilizing the idle resources in volunteer vehicles. Different from these studies, this article investigates how to effectively and economically utilize edge server to support multiplayer cloud gaming.

III. System Design and Problem Modeling

This section first presents the architecture of the proposed ECACG, and then describes the proposed optimization model.

A. System Overview

Fig. 1 illustrates the system architecture of the ECACG platform, which includes game servers residing in cloud data center, rendering servers residing in the edge network, and players dispersed at different regions. Game servers are deployed on virtual machines running at gaming service provider's cloud servers, and rendering servers are deployed on virtual machines running at edge servers. A gaming provider can rent several virtual rendering servers from edge computing service providers. Each player may reside within the overlapping coverage area of multiple edge servers. Each edge server may cover multiple players in a specific geographical area. Players are connected to virtual rendering servers running on nearby edge servers via a local area network (LAN). Virtual rendering servers are connected to game servers via wide area network (WAN). Each rendering server can serve multiple players, but each player can only be connected to a rendering server. Previous work [21] shows that the response delay is mainly dominated by the game video streaming rate in cloud gaming. The main idea behind the ECACG is to stream game videos from nearby edge servers to players, instead of from clouds, which can reduce the traffic relayed from clouds and thus, improve response delay and flexibility.

When a player logs in the game, a connection between player and game server is first established, and then the game server dispatches the player to the appropriate rendering server, and deploy the rendering server on nearby edge servers. Once the connection between player and rendering server is established, the connection between player and game server is terminated. Each rendering server can be active or inactive. A rendering server is seen as active if it currently provides service for one or more players while inactive rendering servers is not providing service for any players. In addition, there is a rental cost associated with each active rendering server.

Specifically, the game process in the ECACG platform may be described as follows. First, a registered player logs into the platform and selects a preferred game from a list of available games, and then sends a cloud gaming service request to the game server in the cloud data center. Upon the receipt of the request, the game server searches for an appropriate virtual rendering server based a gaming request scheduling policy, and places the virtual rendering server at an eligible edge server based on a rendering server allocation policy. Finally, the address of the virtual rendering server is returned to the player in response to the cloud gaming service request. Receiving the address, the player begins to play the game. During the gaming session, the game actions issued by the player are sent to the virtual rendering server at the edges and then forwarded to the game server at cloud data center. Upon the receipt of the game actions, the game server computes the game world status updates and sends it back to the virtual rendering server. Based on the information included in the updated game world, the virtual rendering server generates new video image frames and streams the rendered frames back to the player. The above procedure is repeated until the gaming session ends.

By dividing the jobs of multiplayer online gaming into cloud servers and edge servers, cloud servers are in charge of game logics and interplayer interactions while edge servers are responsible for rendering game scenes for players. This architecture design provides the flexibility in player request scheduling so that players in any geographical area can be scheduled to any available rendering servers. In addition, the flexibility enables the reduction in response delay, the improvement in interplayer unfairness, and the reduction in cost.

B. Problem Formulation

In this article, time is separated into many slots with unit length. Given a set of players P, a set of virtual rendering servers for rent R, and a set of edge servers E at each time slot, the goal of the game service provider is to find optimal player request scheduling and rendering server allocation such that

response delay does not exceed a predefined threshold limit, and the difference of response delay among interacting players and the rental cost of virtual rendering servers are minimized.

Without loss of generality, a heterogeneous computing environment is considered in which each edge server e and each rendering server r can handle the rendering and streaming workloads of up to C_e and k_r players, respectively. Multiple virtual rendering servers can run simultaneously on the same edge server. A player is only connected to a virtual rendering server during a game session period. The connection between a player p and a virtual rendering server r is represented by a binary variable $x_{p,r}$ that takes value 1 if p is connected to r, and 0 otherwise. Each activated rendering server r will incur a rental cost V_r, and a binary variable y_r is defined to describe whether r is used. For each used rendering server r and each of its eligible edge servers $e \in E$, a binary variable $z_{r,e}$ is defined to indicate whether r is allocated to e.

Regarding the quality of experience and playability, two metrics are considered, namely, responsiveness and fairness. The responsiveness is measured by the response delay, which corresponds to the time lag between the player sending a command and the corresponding game frame being displayed on the screen. The fairness is measured by the difference of response delay among interacting players. Response delay consists of network delay, processing delay, rendering delay, and playout delay. Network delay is essentially the network round-trip time, which can be measured by tools, such as *Ping* and *Traceroute*. Rendering delay is the time consumed by virtual rendering server to handle a player's command and generate the corresponding game frame. Processing delay is the time consumed by the game server to update the game state. Playout delay is the time consumed by the player to receive and display the game frame. Finally, if a player p is connected to a virtual rendering server i running on the edge server j, the corresponding response delay L_p is defined by

$$L_p = NL_{p,j} + RL_i + GL_j + PL + BL_p \tag{1}$$

where $NL_{p,j}$ denotes the network delay between player p and edge server j and GL_j denotes the network delay between edge server j and game server. RL_i represents the rendering delay of virtual rendering server i. PL represents the processing delay of the game server. BL_p represents the playout delay of the player's game client. Similarly, the difference $D_{p,q}$ of response delay between two players is defined by

$$D_{p,q} = \begin{cases} |L_p - L_q|, & \text{if two players interact with each other} \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Suppose R^+ is the set of active rendering servers, the optimization problem can be formulated as

$$\textbf{Minimize} \quad \max_{p,q \in P} \{D_{p,q}\} \tag{3}$$

$$\textbf{Minimize} \quad \sum_{s \in R} (V_s \bullet y_s) \tag{4}$$

$$\text{s.t.} \quad \sum_{r \in R} (z_{r,e} \bullet k_r) \le C_e \quad \forall e \in E \tag{5}$$

$$\sum_{p \in P} (x_{p,r}) \le k_r \quad \forall r \in R^+ \tag{6}$$

$$\sum_{r \in R} x_{p,r} = 1 \quad \forall p \in P \tag{7}$$

$$\sum_{r \in R} z_{r,e} = 1 \quad \forall r \in R^+ \tag{8}$$

$$L_p \le L^* \quad \forall p \in P \tag{9}$$

$$x_{p,r}, v_r, z_{r,e} \in \{0, 1\} \quad \forall p \in P \quad \forall r \in R^+ \quad \forall e \in E. \tag{10}$$

Two objective functions defined by (3) and (4) aim to find the player request scheduling and the rendering server allocation that minimize both maximal delay difference and rental cost simultaneously. Constraints (5) and (6) prevent the total workload exceeding the capability of the edge server and virtual rendering server, respectively. Constraint (7) ensures that each player is served by one virtual rendering server. Constraint (8) ensures that each active rendering server is only deployed on an edge server. Constraint (9) ensures that the response delay perceived by all players does not exceed the predefined threshold L^*. Constraint (10) defines the domain of the variables of the problem.

IV. Algorithm Design

The optimization problem formulated above is obviously NP-hard. Recent study indicates that deep reinforcement learning is a promising tool for solving multiobjective optimization problem [22]. This section will depict how to use a hybrid algorithm (DRL-H) based on deep reinforcement learning and heuristic strategy to search for near-optimal solutions. In the DRL-H, the deep reinforcement learning is applied to find a sequence to schedule player requests and a sequence to allocate virtual rendering servers while the heuristic algorithm takes two sequences as input, and then generates the scheduling of player requests to virtual rendering servers and the allocating of virtual rendering servers to edge servers.

There are two types of multiobjective reinforcement learning algorithms: 1) single-policy methods and 2) multiple-policy methods. Single-policy methods have better computational efficiency than multiple-policy methods. Since the optimization problem described above needs real-time decision making, this article adopts the single-policy method based on the weighted linear scalarization to learn the optimal policy. Therefore, the quality of a candidate solution ρ is assessed by the following equation:

$$QS(\rho) = \alpha f_1(\rho) + \beta f_2(\rho) \tag{11}$$

where α and β are the weights corresponding to objective functions f_1 and f_2 defined by (3) and (4), respectively, and their sum is equal to 1.

A. Neural Network Design

Inspired by the studies of [23] and [24], this article utilizes a neural network architecture named pointer network to generate the sequence to schedule player requests and the sequence to allocate virtual rendering servers. Fig. 2 shows the architecture of the proposed pointer network. It can be seen from Fig. 2 that the proposed pointer network is composed of two components: one is an encoder of players and virtual rendering servers and

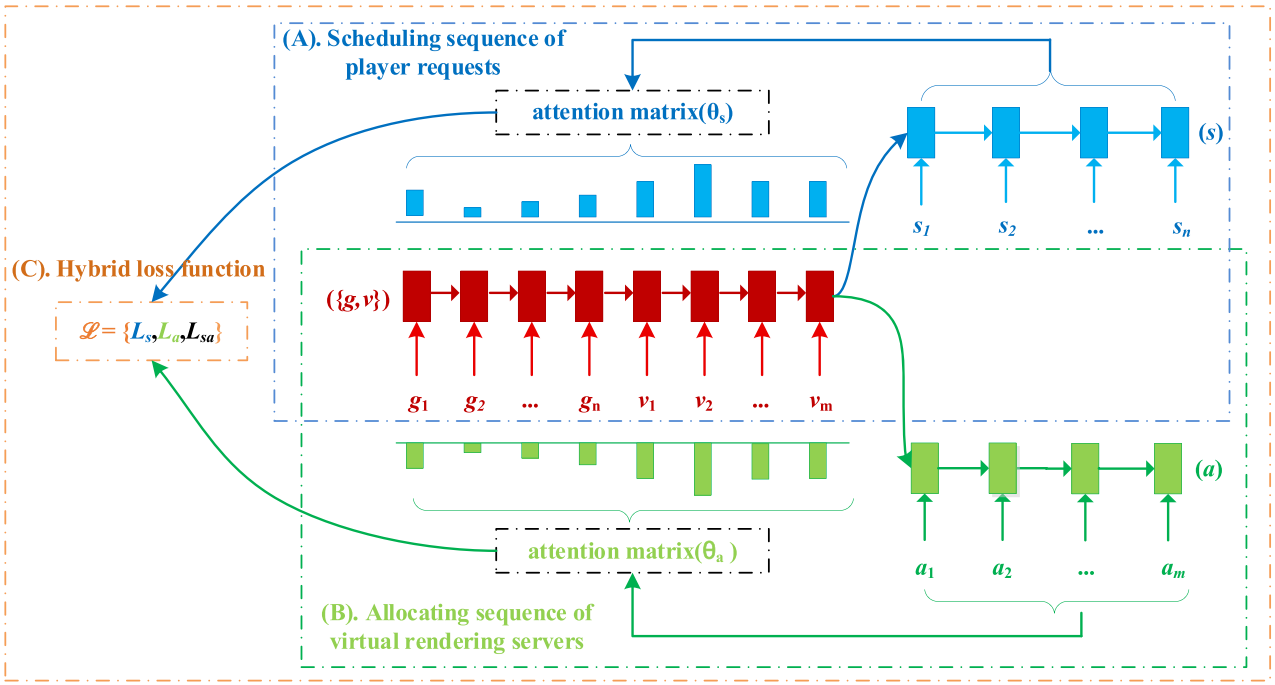


Fig. 2. Dual pointer network architecture.

the other is a dual pointer network decoder. Both encoder and decoder are long short-term memory recurrent neural networks (LSTM-RNN). The encoder receives two input sequences, $g = \{g_1, g_2, \ldots, g_n\}$ and $v = \{v_1, v_2, \ldots, v_m\}$, where g_i is a vector that consists of the location and playout delay of the ith player, and v_i is a vector that consists of the capacity and processing delay of the ith virtual rendering server. In the proposed dual pointer network decoder, the first decoder learns a scheduling sequence (s) of players requests while the second decoder learns an allocating sequence (a) of virtual rendering servers.

The encoder handles the input sequences g and v into a sequence of hidden states $h^e = h_1^e, h_2^e, \ldots, h_{m+n}^e$ where h_i^e represents the hidden state of the ith input in the encoder. h^e is taken as input to the dual pointer network decoder, and the decoding happens sequentially. The input of the decoder cell at decoding time step t consists of two parts described as $d_t = (s_{t-1}, a_{t-1})$. The outputs of the dual pointer network decoder are a permutation of the input player requests $s = \{s_1, s_2, \ldots, s_n\}$ and a permutation of the input rendering servers $a = \{a_1, a_2, \ldots, a_m\}$, respectively.

Because of the particularity of scheduling and allocating operation, it is not allowed to schedule the same player request and allocate the same virtual rendering server repeatedly. Therefore, the intraattention mechanism [25] is applied to prevent the output sequence from including the same player request twice or the same virtual rendering server twice.

Regarding the first pointer network decoder, at decoding time step t, the probability distribution to copy the input token can be described by

$$\tau_j^t = \varphi_1^T \tanh\left(\omega_1 h_j^e + \omega_2 h_t^d + \omega_3 h_t^{\text{in}}\right) \tag{12}$$

$$p(s_t) = \text{softmax}\left(\tau^t\right) j \in \{1, 2, \ldots, m+n\} \tag{13}$$

where φ_1^T, ω_1, ω_2, ω_3 are learnable parameters, h_t^d denotes the decoder hidden state computed from d_t, and h_t^{in} represents the intra-attention context vector, which is defined by

$$h_t^{\text{in}} = \sum_{k=1}^{t-1} \left(\alpha_{t,k}^d h_k^d\right) \tag{14}$$

where $\alpha_{t,k}^d$ denotes the intra-attention weights, and can be computed by

$$\alpha_{t,k}^d = \text{softmax}\left(\varphi_2^T \tanh\left(\omega_4 h_k^d + \omega_5 h_t^d\right)\right) \tag{15}$$

where φ_2^T, ω_4, and ω_5 are learnable parameters. The policy gradient method called proximal policy optimization (PPO) [26] is applied to train the first pointer network. Suppose θ_s denotes all parameters of the first pointer network, the policy loss can be formulated as

$$L_s(\theta_s) = \mathbb{E}_t\left[\min\left(r_t(t)\hat{A}, \text{clip}(r_t(t), 1-\varepsilon, 1+\varepsilon)\hat{A}\right)\right] \tag{16}$$

where $r_t(t) = (\pi_{\theta_s}(s_t|g, v)/\pi_{\theta_s^{\text{old}}}(s_t|g, v))$ is the probability ratio of the old policy $\pi_{\theta_s^{\text{old}}}$ and the new policy π_{θ_s}, ε is the clipping bound, and $\hat{A}$ is the advantage function.

Similarly, at decoding time step t, the second pointer network decoder computes the probability distribution to copy the input token based on a decoder context vector c_t, which is described by

$$c_t = \left[l_t^e \parallel h_t^d \parallel h_t^{\text{in}}\right] \tag{17}$$

where $\parallel$ denotes vector concatenation operator, and l_t^e represents the input context vector, which is computed by

$$l_t^e = \sum_{k=1}^{m+n} \left(\alpha_{t,k}^e h_k^e\right) \tag{18}$$

$$\alpha_{t,k}^{e} = \text{softmax}\Big(\varphi_3^T \tanh\Big(\omega_6 h_j^e + \omega_7\Big[h_t^d \parallel h_t^{\text{in}}\Big]\Big)\Big) \quad (19)$$

where φ_3^T, ω_6, and ω_7 are learnable parameters. Finally, the probability distribution for the allocating sequence of virtual rendering servers can be defined by

$$p(a_t) = \text{softmax}(\omega_8 c_t + \omega_9) \quad (20)$$

where ω_8 and ω_9 are learnable parameters. The cross-entropy loss is employed to train the second pointer network. Suppose θ_a denotes all parameters of the second pointer network, the policy loss can be computed as

$$\begin{aligned} L_a(\theta_a) &= -\log p\big(a^*|\theta_a, g, v\big) \\ &= -\sum_{j=1}^{m} \log p\Big(a_j^*|a_{1:j-1}^*, \theta_a, g, v\Big) \end{aligned} \quad (21)$$

where $a^* = \{a_1^*, a_2^*, \ldots, a_m^*\}$ is the target ground-truth sequence for the given inputs g and v. Inspired by the hill climbing method, the allocating sequence corresponding to best-so-far solution is selected as the target ground-truth sequence.

B. Neural Network Training

The proposed pointer network shares an identical encoder, but has two different decoders. In order to tie two pointer networks, a new loss function L_{sa} is defined by

$$L_{sa} = \delta\ L_a(\theta_a) + (1-\delta)L_s(\theta_s) \quad (22)$$

where δ is a hyperparameter. For the purpose of maximizing the learning ability of two pointer networks, a hybrid of L_{sa}, L_a, and L_s is applied to train the proposed neural network. During training stage, the proposed neural network will select one from three loss functions based on a probability distribution. Algorithm 1 shows the training procedure of the proposed neural network.

Algorithm 1 Training Algorithm

Input: training sample set T, number of training steps M, batch size B
Output: $\theta = \{\theta_s, \theta_a\}$
1. Initialize pointer network parameters θ_s and θ_a
2. Do
3. Select a batch of sample t_i for $i \in \{1, 2, \ldots, B\}$
4. Sample the scheduling sequence s^i based on $p_{\theta_s}(\cdot|t_i)$ for $i \in \{1, 2, \ldots, B\}$
5. Sample the allocating sequence a^i based on $p_{\theta_a}(\cdot|t_i)$ for $i \in \{1, 2, \ldots, B\}$
6. Utilize the heuristic algorithm to generate the solution u_i corresponding to each pair of (s^i, a^i) for $i \in \{1, 2, \ldots, B\}$
7. Evaluate the quality of each solution u_i based on Eq. (11) for $i \in \{1, 2, \ldots, B\}$
8. Compare all the solutions in current iteration to obtain the iteration-best solution.
9. Compare the best-so-far solution with the iteration-best solution to obtain a new best-so-far solution
10. Compute the gradients of three loss functions L_{sa}, L_a and L_s based on the best-so-far solution
11. Select a gradient g_θ according to the probability distribution
12. Update $\theta = ADAM(\theta, g_\theta)$
13. Until the number of iterations is reached to M
14. Return θ

C. Heuristic Algorithm Design

For the scheduling sequence s and the allocating sequence a generated by the proposed pointer network, a heuristic algorithm is developed to search for a feasible solution $O =< PV, VS >$, where PV denotes a mapping of player requests to virtual rendering servers and VS denotes a mapping of virtual rendering servers to edge servers. For each player, all the virtual rendering servers that satisfy constraint (6) are called the eligible virtual rendering servers for the player. Similarly, for each virtual rendering server, all the edge servers that satisfy constraint (5) are called the eligible edge servers for the virtual rendering server. Algorithm 2 shows the implementation step of the heuristic algorithm. Initially, PV and VS are empty. In each iteration, the heuristic algorithm extracts one player request from s at a time, and try to dispatch it to an active eligible virtual rendering server deployed on an edge server covering the player. If all active virtual rendering servers do not enough capacity to accommodate the player request, the heuristic algorithm will activate the first virtual rendering server in a and dispatch the player request to the activated virtual rendering server, and then the activated virtual rendering server is allocated to an eligible edge server covering the player. Once the heuristic algorithm finishes processing each player request in s, PV and VS will include the entire mapping of player requests to virtual rendering servers and the entire mapping of virtual rendering servers to edge servers. Finally, the algorithm returns the solution related to the given sequences.

D. Algorithm Complexity Analysis

The implementation of the DRL-H can be divided into two phases: 1) an offline training phase and 2) an online application phase. In the offline training phase, the proposed pointer network is trained to obtain the optimal parameters based on historic data. In the online application phase, the trained pointer network is first employed to generate a sequence to schedule player requests and a sequence to allocate virtual rendering servers, and then the proposed heuristic algorithm takes two sequences as input and returns the scheduling of player request to virtual rendering servers and the allocating of virtual rendering servers to edge servers. According to [23], the proposed pointer network has $O((n+m)^2)$ complexity, where n is the number of players issuing the gaming requests and m is the number of virtual rendering servers. The proposed heuristic algorithm has $O(m*n)$ complexity. Therefore, the computational complexity of the DRL-H is $O((m+n)^2 + m*n)$ during the online application, which can be simplified to $O((m+n)^2)$. The computational complexity in the training phase is related to the number of training

Algorithm 2 Heuristic Algorithm

Input: scheduling sequence s, allocating sequence a Output: a solution $O =< PV, VS >$

1: PV=∅;
2: VS=∅;
3: TP=∅
4: For i=1 to *s.size()* do
5: Find all active virtual rendering server residing in edge serves covering player $s[i]$ and add them to TP
6: If all virtual rendering server in TP can't accommodate player $s[i]$ then
7: Remove the first virtual rendering server from a
8: *S*chedule the player $s[i]$ to the removed virtual rendering server
9: Add the dispatch to PV
10: Allocate the removed virtual rendering server to a nearby eligible edge server with the least load
11: Add the allocation to VS
12: Else
13: Schedule the player $s[i]$ to an eligible virtual rendering server with the earliest removal time in TP
14: Add the dispatch to PV
15: EndIf
16: Remove all virtual rendering servers in TP
17: EndFor
18: Return a solution O

steps and the batch size of training samples in Algorithm 1. A batch of samples is used at each training step for gradient computation and parameter update, which has a complexity of $O(B*((m+n)^2+m*n)+U)$ where B and U are the batch size and the gradients' dimension, respectively, and $O(B*((m+n)^2+m*n)$ and $O(U)$ are the computational complexity of lines 3–9 and 10–12 in Algorithm 1, respectively. Since there are M training steps, the computational complexity in the training phase is $O(M*(B*((m+n)^2+m*n)+U))$, which can be simplified to $O(M*(B*(m+n)^2+U))$.

V. Performance Evaluation

This section evaluates the effectiveness of the proposed ECACG platform by conducting extensive simulations. The experimental setups and results are elaborated as follows.

A. Experimental Setup

This article adopts the EdgeCloudsim [27] to simulate an edge computing system, which is constructed within a 2-D coordinate, and covers a 2×2 km^2 square area. Twenty five base stations are deployed on this area, and each base station has a coverage radius of 200 m. The location of base station is determined by its horizontal and vertical coordinates that are randomly chosen from the interval of [0, 2] km. Each base station is equipped with an edge server which can handle the workload of up to 50 players. There are ten types of Amazon EC2's GPU instances available for virtual rendering server deployment, and their processing delay, rental prices, and capacities are shown in Table I. Each type of virtual rendering server is selected with the same probability. During each time slot, the number of players' requests arrived in system is set to λ, and each player has its own location, which is also randomly dispersed in the square area. The playing duration for each player is randomly generated from the interval of [1, 3] hr. The interaction pattern among players is obtained from the avatar history data set of World of Warcraft (WoW) collected by Lee *et al.* [28]. The network delay between player and its nearby edge servers varies from 20 to 60 ms, while the network delay between edge servers and gaming servers varies from 30 to 80 ms. The processing delay of gaming server is set to 3 ms according to [29]. Considering the diversity in the player device, the playout delay is randomly generated from the interval of [20, 40] ms based on [29]. The threshold for the response delay is set to 200 ms according to the empirical data provided by [30]. In all experiments, the runtime is set to 5 h and the length of time slot is set to 1 h. For each experiment, the average results of 20 independent runs are reported.

TABLE I
Type of Virtual Rendering Servers

Type	Processing delay (ms)	Capacity	Price ($/hour)
g4dn.2xlarge	60	2	0.752
g4dn.4xlarge	53	4	1.204
g4dn.8xlarge	39	5	2.176
g4dn.12xlarge	30	6	3.912
g4dn.16xlarge	24	7	4.352
g4dn.metal	10	16	7.824
g3.4xlarge	50	3	1.14
g3.8xlarge	37	5	2.28
g3.16xlarge	26	7	4.56
g3s.xlargeSan	60	2	0.75

The training samples for the proposed pointer network are generated as follows. First, 200 000 solutions are randomly generated. Next, the ECACG platform is configured according to each generated solution. Finally, each sample is generated by operating the ECACG platform for 1 h and computing the corresponding rental cost and maximal delay difference. 64% of the generated samples is used for training and the remaining 36% is used for testing. In addition, the batch size is set to 128, and each LSTM cell contains 256 hidden units. The hyperparameter δ in loss function L_{sa} is set to 0.5. The proposed pointer network is trained by the Adam optimizer [31] with an initial learning rate of 0.001 and a decay factor of 0.94 every 4000 steps. PPO uses the clipped surrogate objective and the corresponding clipping bound is set to 0.2. The weights α and β are set to 0.5 and 0.5, respectively. The training steps is set to 100 000. Three loss functions are selected with probabilities (0.4, 0.4, and 0.2), respectively.

B. Simulation Results

The first set of simulation experiments is conducted to evaluate the training performance under various learning rates and batch sizes. Fig. 3 shows the influence of the learning rate on the training performance. As can be observed, a small learning rate leads to slow convergence rate, and a large learning rate

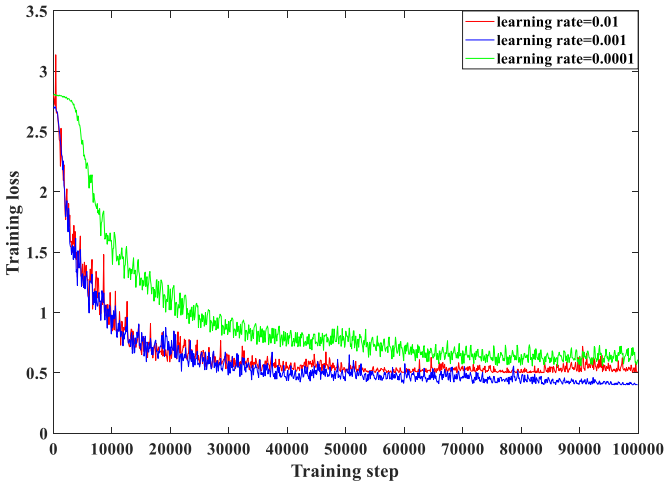


Fig. 3. Training performance under different learning rates.

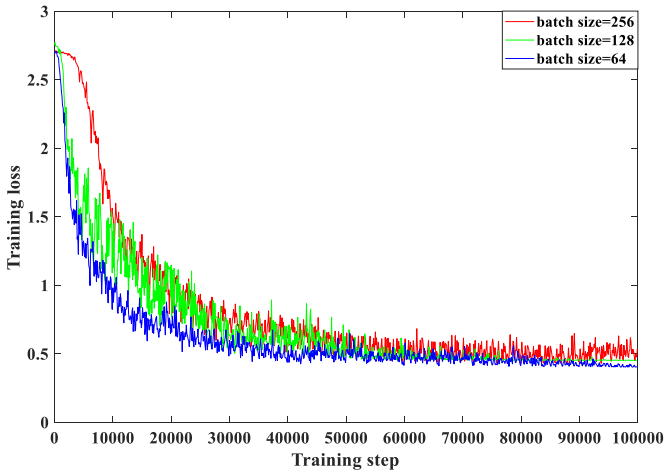


Fig. 4. Training performance under different batch sizes.

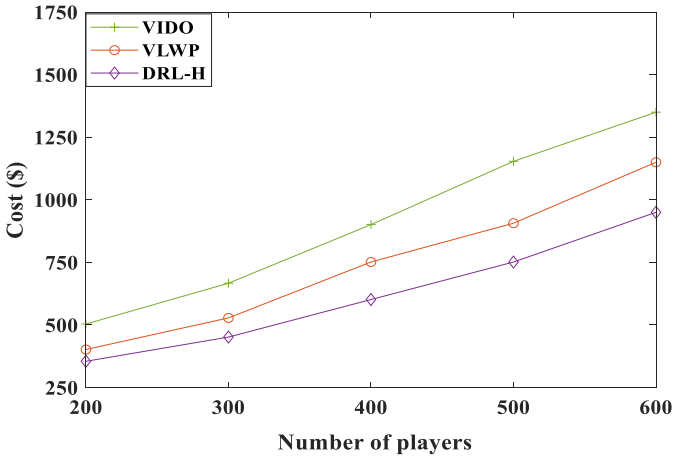


Fig. 5. Cost of three algorithms under $\mu = 200$.

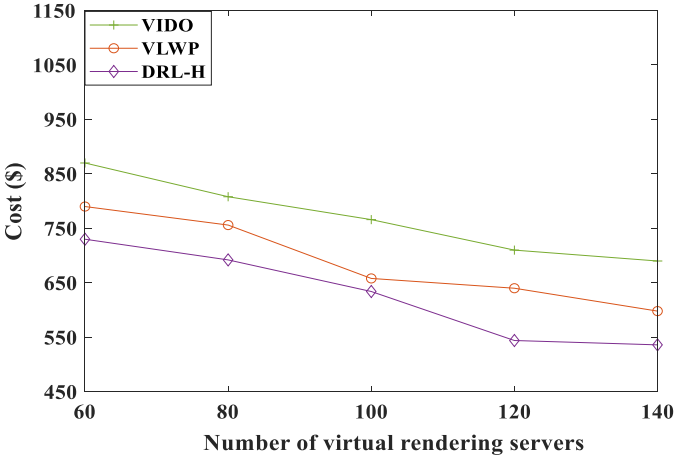


Fig. 6. Cost of three algorithms under $\lambda = 300$.

results in high loss values. Finally, the learning rate is set to 0.001 in this article because it can lead to relatively high convergence speed and low loss value. Fig. 4 shows the training performance under different batch sizes. It can be seen from Fig. 4 that the convergence speed decreases as the batch sizes get larger, but the increase in batch size does not significantly affect the training performance. In this article, the batch size is set to 128 since it can reduce the time of one round of training with a slight sacrifice for convergence speed.

In the second set of simulation experiments, the performance of the DRL-H in the ECACG system is compared with that of variants of two state-of-the-art algorithm in the traditional gaming system. One is a variant of the IDO algorithm proposed by [11], named VIDO. The other is a variant of LWP algorithm proposed by [16], named VLWP. In contrast to IDO, VIDO first generates the circular convex set of each edge server, a subset of all possible covering patterns of virtual rendering servers by each edge servers, and then dispatch each player to an edge server with the least delay difference increase. Different from LWP, VLWP first selects an active virtual rendering server with the lowest waste price for each player; if an eligible active virtual rendering server cannot be found, a new virtual rendering server will be activated and then allocated to a nearby edge server with the lease load. The performance of three algorithms is evaluated under different problem sizes. The problem size is defined as a two-tuple (λ, μ), where μ is the number of virtual rendering servers.

Figs. 5–9 show the maximal delay difference, the rental cost, and the cumulative distribution function (CDF) of response delay provided by three algorithms for various problem sizes. The following results are obtained from these figures.

1) With increasing λ, the overall rental cost increases. The reason is that three algorithms need to activate more virtual rendering servers to serve the increased players. In addition, the overall rental cost decreases with increasing μ. This is because three algorithms have more cost-efficient options to allocate virtual rendering servers, and schedule player request.
2) With increasing λ, the maximal delay difference increase. This is because three algorithms have to sacrifice some delay to meet other requirements when μ becomes more and more insufficient due to the increase of λ. Moreover, the maximal delay difference decreases with increasing μ. The reason is that three algorithms

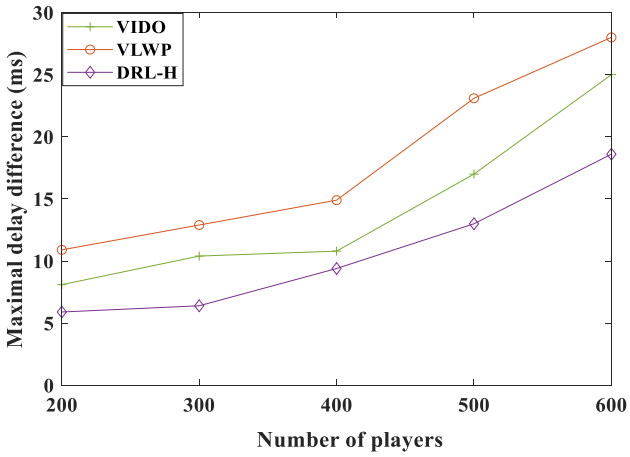


Fig. 7. Maximal delay difference of three algorithms under $\mu = 200$.

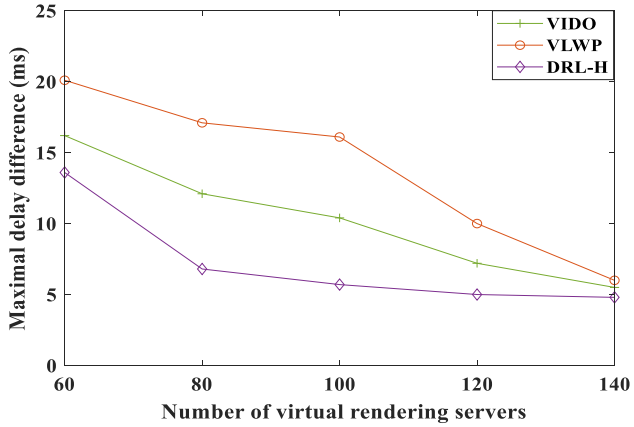


Fig. 8. Maximal delay difference of three algorithms under $\lambda = 300$.

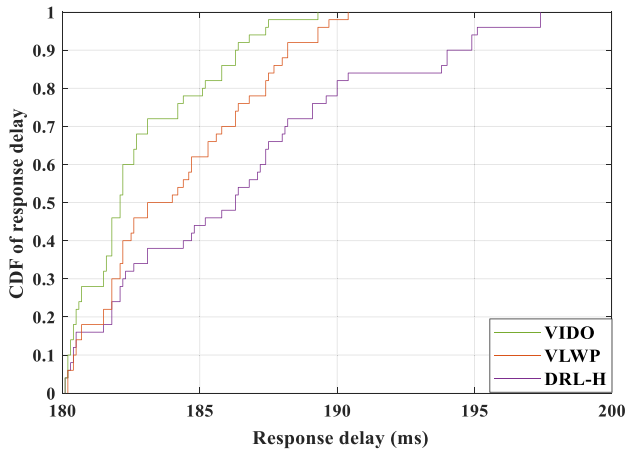


Fig. 9. CDF of response delay for three algorithms under $\lambda = 500$ and $\mu = 200$.

have more choice to meet all requirements when μ becomes more and more sufficient.

3) In contrast to VIDO, VLWP performs better in the aspect of the rental cost, but worse in the aspect of the maximal delay difference. This is because VLWP only optimizes the rental cost of virtual rendering servers and does not consider response delay difference among interacting players.
4) In contrast to VLWP, VIDO performs better in the aspect of minimizing the delay difference among interacting players, but worse in the aspect of the rental cost. The reason is that it only optimizes the response delay difference among players, and does not consider the rental cost of virtual rendering servers.
5) The DRL-H performs best in the aspect of the maximal delay difference and the rental cost. This is because that DRL-H simultaneously optimizes the response delay difference among interacting players and the rental costs of virtual rendering servers.
6) Three algorithms can keep the response delay perceived by players below the predefined threshold. But, VIDO has the smallest boundary of the response delay, VLWP has the moderate boundary of the response delay, and DRL-H has the biggest boundary of the response delay. The reason is that DRL-H tends to sacrifice more response time to preserve the fairness among interacting players and reduce the rental costs.

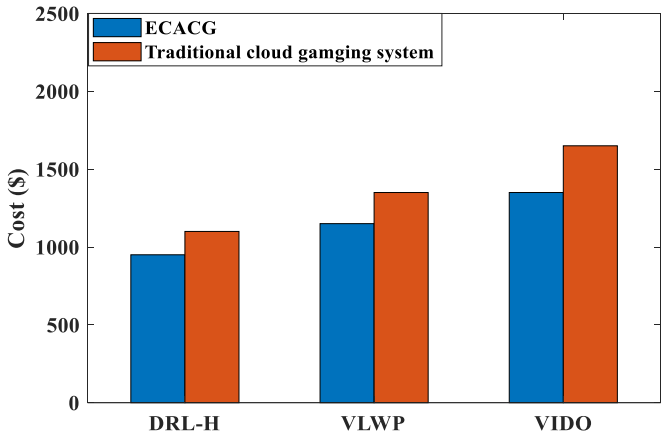


Fig. 10. Cost of three algorithms under two different platforms.

The third set of simulation experiments is used to compare the performance when playing games in the ECACG system with that in a traditional cloud gaming system. The traditional cloud gaming system is simulated by replacing edge servers with cloud servers. At the same time, the network delay between player and virtual rendering servers varies from 60 to 120 ms, while the network delay between virtual rendering servers and gaming servers varies from 20 to 50 ms. In the experiment, λ and μ are set to 600 and 200, respectively. Figs. 10–13 show the maximal delay difference, the rental cost, and the CDF of response delay provided by three algorithms for two types of gaming platform. As shown in Figs. 10–13, the ECACG system performs better in terms of the maximal delay difference and the rental cost. In addition, the boundary of the response delay is much smaller in the ECACG system than in the traditional cloud gaming system. The reason is that the ECACG system offloads the game rendering to the nearby edge node, which can achieve a flexible tradeoff between quality of experience and operating costs.

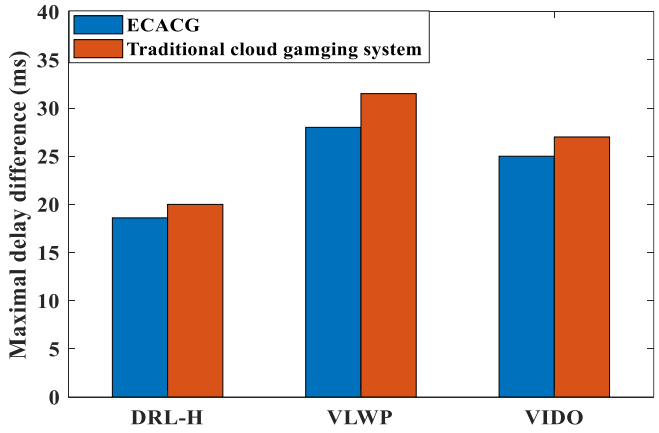


Fig. 11. Maximal delay difference of three algorithms under two different platforms.

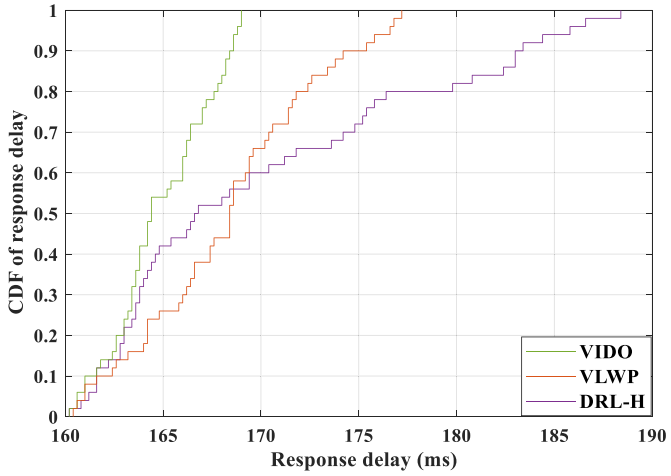


Fig. 12. CDF of response delay for three algorithms in the traditional cloud gaming platform.

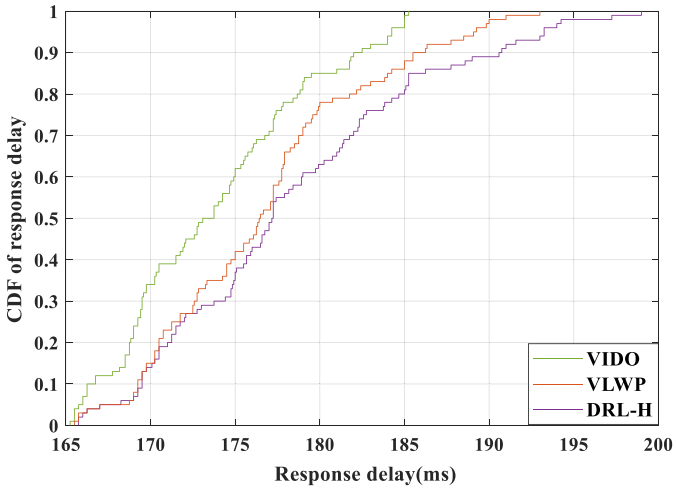


Fig. 13. CDF of response delay for three algorithms in the ECACG platform.

The last set of simulation experiments is conducted to evaluate the impact of three parameters C_e , k_r, and L^* in constraints (5), (6), and (9) on the rental cost and the maximal delay difference. In this set of experiments, the number of virtual rendering servers is fixed at 120 and the number of players' requests arrived in system is fixed at 300. To observe the impact of C_e and L^* on the rental cost and the maximal delay difference, their values are varied from 30 to 50 in a step size of 10 and from 200 to 400 ms in a step size of 100 ms, respectively. Tables II and III show the rental cost and the maximal delay difference yielded by the DRL-H under different values of C_e and L^*, respectively. To observe the impact of k_r on the rental cost and the maximal delay difference, the capacity configuration of virtual rendering servers is divided into three scales, namely, small, medium, and large. In the small capacity configuration, the capacities of ten types of virtual rendering servers are the same as that in Table I. In the medium capacity configuration, the capacities of ten types of virtual rendering servers are 4, 6, 7, 8, 9, 18, 5, 7, 9, and 4, respectively. In the large capacity configuration, the capacities of ten types of virtual rendering servers are 6, 8, 9, 10, 11, 20, 7, 9, 11, and 6, respectively. Table IV shows the rental cost and the maximal delay difference yielded by the DRL-H under different capacity configurations of virtual rendering servers. It can be seen from Tables II–IV that the rental cost and the maximal delay difference decrease gradually with the increase of virtual rendering server capacity, edge server capacity, and response delay threshold. This is because there are more eligible edge servers for placing virtual rendering server or more eligible virtual rendering servers for dispatching player request as virtual rendering server capacity, edge server capacity, and response delay threshold get larger, which makes it easier to achieve lower rental costs and better fairness.

TABLE II
RENTAL COST AND MAXIMAL DELAY DIFFERENCE OF DRL-H UNDER DIFFERENT EDGE SERVER CAPACITIES

Edge server capacity	Rental cost ($)	Maximal delay difference (ms)
30	738.3	5.5
40	634.9	5.4
50	549.7	5.1

TABLE III
RENTAL COST AND MAXIMAL DELAY DIFFERENCE OF DRL-H UNDER DIFFERENT RESPONSE DELAY THRESHOLDS

Response delay threshold	Rental cost ($)	Maximal delay difference (ms)
200	549.7	5.1
300	487.7	4.1
400	363.2	3.0

TABLE IV
RENTAL COST AND MAXIMAL DELAY DIFFERENCE OF DRL-H UNDER DIFFERENT CAPACITY CONFIGURATIONS

Capacity configuration	Rental cost ($)	Maximal delay difference (ms)
small	549.7	5.1
medium	458.9	4.7
large	421.3	4.4

VI. Conclusion and Future Work

This article proposes a novel multiplayer cloud gaming framework named ECACG, which consists of cloud data centers, edge networks, and thin game clients. ECACG applies a hybrid of deep reinforcement learning and heuristic strategy to schedule player request and allocate virtual rendering servers with the goal of minimizing the maximal delay difference and the rental cost, while providing the good-enough response delay for players. The performance of the proposed ECACG is evaluated by extensive simulation based on the real-world parameters. The results show that the proposed ECACG has a better performance compared with other alternatives. The blockchain-based incentive scheme and crowdsourced edge network should be carefully designed to reduce the investment cost of cloud gaming service providers, which will be considered as future work.

References

[1] "Half a Billion Dollars in 2020: The Cloud Gaming Market Evolves as Consumer Engagement & Spending Soar." 2020. [Online]. Available: https://newzoo.com/insights/articles/global-cloud-gaming-market-report-consumer-engagement-spending-revenues-2020-2023

[2] "Push the Envelope of Game Development With Stadia." [Online]. Available: https://www.stadia.dev/ (Accessed: May 1, 2021).

[3] "Xcloud Offical Website." [Online]. Available: https://www.xbox.com/en-US/xbox-gamestreaming/project-xcloud (Accessed: May 8, 2021).

[4] "Nvidia GeForce Now Official Website." [Online]. Available: https://www.nvidia.com/en-us/geforce-now/ (Accessed: May 10, 2021).

[5] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.

[6] Y. Han, D. Guo, W. Cai, X. Wang, and V. C. M. Leung, "Virtual machine placement optimization in mobile cloud gaming through QoE-oriented resource competition," *IEEE Trans. Cloud Comput.*, early access, Jun. 12, 2020, doi: 10.1109/TCC.2020.3002023.

[7] S. S. Sabet, S. Schmidt, S. Zadtootaghaj, B. Naderi, C. Griwodz, and S. Möller, "A latency compensation technique based on game characteristics to mitigate the influence of delay on cloud gaming quality of experience," in *Proc. 11th ACM Multimedia Syst. Conf.*, 2020, pp. 15–25.

[8] Y. Li *et al.*, "Themis: Efficient and adaptive resource partitioning for reducing response delay in cloud gaming," in *Proc. 27th ACM Int. Conf. Multimedia (MM)*, 2019, pp. 491–499.

[9] X. Zhang *et al.*, "Improving cloud gaming experience through mobile edge computing," *IEEE Wireless Commun.*, vol. 26, no. 4, pp. 178–183, Aug. 2019.

[10] M. Amiri, H. A. Osman, S. Shirmohammadi, and M. Abdallah, "Toward delay-efficient game-aware data centers for cloud gaming," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 12, no. 5s, pp. 1–19, 2016.

[11] Y. Chen, J. Liu, and Y. Cui, "Inter-player delay optimization in multiplayer cloud gaming," in *Proc. IEEE 9th Int. Conf. Cloud Comput. (CLOUD)*, San Francisco, CA, USA, 2016, pp. 702–709.

[12] M. Basiri and A. Rasoolzadegan, "Delay-aware resource provisioning for cost-efficient cloud gaming," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 4, pp. 972–983, Apr. 2018.

[13] Y. Deng, Y. Li, R. Seet, X. Tang, and W. Cai, "The server allocation problem for session-based multiplayer cloud gaming," *IEEE Trans. Multimedia*, vol. 20, no. 5, pp. 1233–1245, May 2018.

[14] D. Wu, Z. Xue, and J. He, "iCloudAccess: Cost-effective streaming of video games from the cloud with low latency," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 8, pp. 1405–1416, Aug. 2014.

[15] H. Tian, D. Wu, J. He, Y. Xu, and M. Chen, "On achieving cost-effective adaptive cloud gaming in geo-distributed data centers," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 12, pp. 2064–2077, Dec. 2015.

[16] I. Jaya, W. Cai, and Y. Li, "Rendering server allocation for MMORPG players in cloud gaming," in *Proc. 49th Int. Conf. Parallel Process. (ICPP)*, 2020, pp. 1–11.

[17] S. Mao, J. Wu, L. Liu, D. Lan, and A. Taherkordi, "Energy-efficient cooperative communication and computation for wireless powered mobile-edge computing," *IEEE Syst. J.*, early access, Oct. 15, 2020, doi: 10.1109/JSYST.2020.3020474.

[18] J. Feng, L. Liu, Q. Pei, F. Hou, T. Yang, and J. Wu, "Service characteristics-oriented joint optimization of radio and computing resource allocation in mobile-edge computing," *IEEE Internet Things J.*, vol. 8, no. 11, pp. 9407–9421, Jun. 2021.

[19] S. Mao, S. He, and J. Wu, "Joint UAV position optimization and resource scheduling in space-air-ground integrated networks with mixed cloud-edge computing," *IEEE Syst. J.*, vol. 15, no. 3, pp. 3992–4002, Sep. 2021.

[20] F. Zeng, Q. Chen, L. Meng, and J. Wu, "Volunteer assisted collaborative offloading and resource allocation in vehicular edge computing," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 6, pp. 3247–3257, Jun. 2021.

[21] T. Kämäräinen, M. Siekkinen, A. Ylä-Jääski, W. Zhang, and P. Hui, "A measurement study on achieving imperceptible latency in mobile cloud gaming," in *Proc. 8th ACM Multimedia Syst. Conf.*, 2017, pp. 88–99.

[22] J. Wang, C. Jiang, H. Zhang, Y. Ren, K.-C. Chen, and L. Hanzo, "Thirty years of machine learning: The road to Pareto-optimal wireless networks," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 1472–1514, 3rd Quart., 2020.

[23] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 2692–2700.

[24] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, 1992.

[25] P. Romain, C. Xiong, and R. Socher, "A deep reinforced model for abstractive summarization," in *Proc. Int. Conf. Learn. Represent.*, 2018. pp. 1–13.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

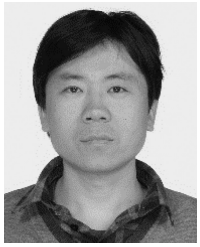
[27] C. Sonmez, A. Ozgovde, and C. Ersoy, "EdgeCloudSim: An environment for performance evaluation of edge computing systems," *Trans. Emerg. Telecommun. Technol.*, vol. 29, no. 11, p. e3493, 2018.

[28] Y.-T. Lee, K.-T. Chen, Y.-M. Cheng, and C.-L. Lei, "World of warcraft avatar history dataset," in *Proc. 2nd Annu. ACM Conf. Multimedia Syst.*, 2011, pp. 123–128.

[29] F. Metzger, A. Rafetseder, and C. Schwartz, "A comprehensive end-to-end lag model for online and cloud video gaming," in *Proc. 5th ISCA/DEGA Workshop Perceptual Qual. Syst. (PQS)*, 2016, pp. 15–19.

[30] M. Dick, O. Wellnitz, and L. Wolf, "Analysis of factors affecting players' performance and perception in multiplayer games," in *Proc. 4th ACM SIGCOMM Workshop Netw. Syst. Support Games*, 2005, pp. 1–7.

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

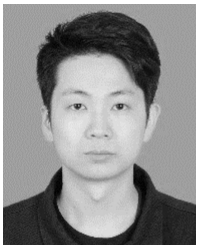


Yongqiang Gao received the Ph.D. degree in computer science from Shanghai Jiao Tong University, Shanghai, China, in 2013.

He is currently an Associate Professor with the College of Computer Science, Inner Mongolia University, Hohhot, China. His research interests include cloud computing, virtualization, and green computing.

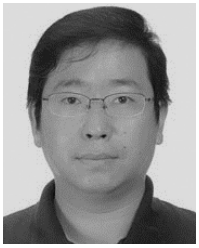
Chaoyu Zhang received the B.S. degree from Tiangong University, Tianjin, China, in 2019. He is currently pursuing the M.S. degree with the College of Software, Inner Mongolia University, Hohhot, China.

His research interests include cloud gaming, virtualization, and artificial intelligence.



Zhulong Xie received the M.S. degree in software engineering from Inner Mongolia University, Hohhot, China, in 2018.

He is currently working as a Software Engineer with Perfect World Company Ltd., Beijing, China. His research interests include distributed computing and cloud gaming.



Zhengwei Qi received the B.S. and M.S. degrees from Northwestern Polytechnical University, Xi'an, China, in 1999 and 2002, respectively, and the Ph.D. degree from Shanghai Jiao Tong University, Shanghai, China, in 2005.

He is a Professor with the School of Software, Shanghai Jiao Tong University. His research interests include program analysis, model checking, virtual machines, and distributed systems.

Jiantao Zhou (Member, IEEE) received the Ph.D. degree in computer science from Tsinghua University, Beijing, China, in 2005.

She was a Visiting Scholar with De Montfort University, Leicester, U.K., from 2009 to 2010. She is currently a Full Professor of Computer Science and Technology with Inner Mongolia University, Hohhot, China. She has published more than 90 papers in international conferences and journals, such as the IEEE TRANSACTIONS ON COMPUTERS, the *Journal of Supercomputing*, ICWS, SCC, and COMPSAC. Her research interests include cloud computing, software engineering, and formal method.