Towards Fog-Assisted Virtual Reality MMOG with Ultra-Low Latency*

YOSHIHARA Tsuyoshi and FUJITA Satoshi

Graduate School of Engineering, Hiroshima University,
Kagamiyama 1-4-1, Higashi-Hiroshima, 739-8527, Japan

Abstract. In this paper, we propose a method to realize a virtual reality MMOG (Massively Multiplayer Online Video Game) with ultra-low latency. The basic idea of the proposed method is to introduce a layer consisting of several *fog nodes* between clients and cloud server to offload a part of the rendering task which is conducted by the cloud server in conventional cloud games. We examine three techniques to reduce the latency in such a *fog-assisted* cloud game: 1) To maintain the consistency of the virtual game space, collision detection of virtual objects is conducted by the cloud server in a centralized manner; 2) To reflect subtle changes of the line of sight to the 3D game view, each client is assigned to a fog node and the head motion of the player acquired through HMD (Head-Mounted Display) is directly sent to the corresponding fog node; and 3) To offload a part of the rendering task, we separate the rendering of the background view from that of the foreground view, and migrate the former to other nodes including the cloud server. The performance of the proposed method is evaluated by experiments with an AWS-based prototype system. It is confirmed that the proposed techniques achieve the latency of 32.3 ms, which is 66 % faster than the conventional systems.

Keywords: Cloud game, fog computing, positional tracking, rendering of 3D game view.

1 Introduction

In recent years, video games have attracted considerable attention as a cloud-friendly network application. Attempts to create online video games have a history of more than 20 years. Ultima Online[1], the forerunner of the massively multiplayer online role-playing game (MMORPG), was released in 1997. It has a game field provided by a game server called *shard*, and users connecting to the same shard can communicate in real-time through chats and actions, while it does not support the collaboration across shards. After the success of Ultima Online, many attractive titles were released successively which include Ragnarok Online[2] and Final Fantasy XI[3], while those early MMORPGs are based on dedicated servers rather than the cloud. Cloud services such as Google App Engine and Amazon EC2 have become

* A preliminary version was presented as "Fog-Assisted Virtual Reality MMOG with Ultra Low Latency" by the same authors in CANDAR 2019.

1 `https://uo.com/`

2 `https://ragnarokonline.gungho.jp/`

3 `https://www.finalfantasyxi.com/jp/`

popular around 2006 with the emergence of Gaming as a Service (GaaS) such as PlayStation Now and GeForce Now. More recently, a cloud game service named *Stadia* was announced by Google in March 2019 [1], which is going to deliver game streams to the players in an extremely high bit rate (i.e., 60 fps 4K resolution, and in the future, 120 fps 8 K resolution) using YouTube infrastructure.

Most of conventional cloud games have a drawback so that *it incurs a non-negligible communication delay to the cloud server*. As of 2019, data centers of public cloud are located around the world, and the RTT (Round Trip Time) to the nearest data center is generally around 200 ms while it strongly depends on the network environment of the client. Although such a moderate RTT is enough for many cloud applications such as office tools and schedule management, it is not satisfactory for *real-time* applications including video games. It is known that human vision can perceive the delay exceeding 100 ms, although the delay shorter than 20 ms could not be recognized. In addition, in a specific type of video games called virtual reality (VR) games, the delay exceeding 40 ms causes a serious symptom called VR sickness to the players. This motivates us to develop a method to bound the latency of reflecting the users' action to the game view on the screen by 40 ms, for example, while keeping a high quality of the rendering, e.g., full HD of 90 fps.

In this paper, we propose a method to reduce the latency in cloud games. The basic idea is to introduce a new layer called **fog layer** consisting of several fog nodes between the client layer and the cloud server layer, and to offload the rendering of 3D game view from the cloud server. Although a similar idea has already been proposed in the literature [9, 13–15, 20, 22, 26], our proposal has the following advantages and originalities:

- To keep the overall game space consistently, the collision detection of virtual objects is conducted by the cloud server in a centralized manner.
- The rendering of the 3D game view is conducted by fog nodes with motion data received from the cloud server. To reflect subtle changes in the line of sight to the rendered game view, the motion acquired by the HMD (Head Mounted Display) worn by the player is *directly* sent to the corresponding fog node to alleviate the VR sickness as much as possible.
- Since each fog node conducts the rendering of game view for all players assigned to it, it becomes easily overloaded as the number of players and the frequency of updates increase. To overcome this issue, we separate the rendering of the background view from that of the foreground view, and offload the former to the cloud server.

To evaluate the performance of the proposed method, we conduct preliminary experiments on a prototype system consisting of AWS, a fog node and clients. The results of evaluations indicate that the proposed method reduces the latency of positional tracking in the conventional cloud-based scheme from 48.3 ms to 32.3

ms, and significantly reduces the computation time required for the rendering of the 3D game view.

The remainder of this paper is organized as follows. Section 2 gives an overview of virtual reality MMOGs. Section 3 overviews related work concerned with the infrastructure for cloud games. Section 4 describes the details of the proposed method. Section 5 summarizes the results of the evaluations. Finally, Section 6 concludes the paper with future work. This paper is an extended version of a paper [24] presented at CANDAR 2019. The difference to the conference version is summarized as follows: 1) we add recent papers concerned with cloud games including CloudFog as related work; 2) we add the way of assigning a fog node to each client; 3) in the evaluation, we add results of experiments concerned with the effect of offloading when the number of clients assigned to a fog node increases; and 4) to clarify that the parallel collision detection causes an inconsistency in the game field, we add results of supplemental experiments.

2 Preliminaries

2.1 VRMMOG

The genre of video games can be designated by the combination of viewpoint and game theme. Concerned with *viewpoint*, it is either first-person, second-person, third-person or omnipresent, where omnipresent video game gives a full control of the player's view (i.e., area of interest) from various angles and distances. On the other hand, *game theme* defines how the player interacts with the game content, e.g., shooting, sports, turn-based role-playing (RPG), action role-playing (ARPG), real-time strategies (RTS), and management simulation. Game genre is designated with those words, such as first-person shooter, third-person ARPG, and omnidirectional RTS. Among those genres, the first-person shooter is the most difficult to be implemented as a cloud game, while the third person RPG is relatively easy to be implemented.

In this paper, we focus on the first-person VRMMOG (Virtual Reality Massively Multiplayer Online Game) as the target game genre. In this genre, dozens of players simultaneously dive into the game world to clear missions such as quests and battles in a cooperative manner. In VR games, the game view is displayed on the HMD worn by each player. Although there are two types of HMDs called high-end type and mobile type in the market, we will focus on the mobile type because of the ease of deployment. Mobile HMD displays the screen of a smart phone by setting it on a dedicated goggle such as Google Cardboard [3] and Google Daydream. On the other hand, high-end HMD is connected to a high-end device such as PlayStation 4 and PlayStation VR (PSVR) [2] through HDMI cables, and displays the output of the device on the screen. The resolution of the display is full HD or higher in both types. In addition to those quantitative metrics, HMD provides a realistic view to the players with a binocular parallax stereoscopic vision. In cloud games, such a game view is rendered at the server-side and is delivered to the client-side through the network. A typical bit rate of the game stream is 15 Mbps or less which is reasonable for many game players since the maximum transmission rate of IEEE802.11g is 54 Mbps.

In addition to such a display function, HMD is equipped with a sensor function for the positional tracking. **Positional tracking** is the task of detecting the precise position of the HMD, which is mandatory to fill the gap between the rendered view and the view expected from the position of the HMD. The system of positional tracking is either outside-in or inside-out, where the former uses externally installed sensor devices or cameras and the latter uses sensors mounted on the HMD (or on smartphone). Mobile HMDs generally use inside-out, which implies that in our setting, sensor data acquired by the smartphone should be uploaded to the server-side, and after rendering the game view reflecting the player's head motion, it must be delivered to the client-side as quickly as possible.

In summary, the role of the client in a cloud game is:

- to receive a stream of rendered game view from the server-side and to display it on the screen of HMD, and
- to acquire sensor data and control signals from HMD and controller, respectively, and to forward them to the server-side after converting them into an appropriate format.

2.2 Expected Performance

VR sickness is a common phenomenon experienced by VR game players. Although the precise reason is not identified yet, according to the sensory discrepancy theory, this phenomenon is caused by the contradiction between the sensation perceived through the screen and the sensation predicted from past experiences. For example, when a player moves the line of sight to the right by moving the head wearing HMD, it would cause a temporal gap (i.e., mismatch of sensation) between the change on the screen and the change expected from the action. Although human visual system has a mechanism to compensate for such a subtle gap, in a specific situation in which a high concentration on the visual input should be kept for a certain time, it gradually affects the sense of balance and causes symptoms such as nausea and headache.

To alleviate such a VR sickness, the designers of Eagle Flight [4] attempted: 1) to reduce the gap between visual information and balance, 2) to reduce the shortage of frame rate and the frame drop, 3) to avoid strenuous movement of objects in a close distance, 4) to avoid the acceleration movement in the view, and 5) to avoid an action passing through of a VR wall, where the last three items are issues of game design and the first two items are issues of game infrastructure.

Based on the above observations, we set the target performance of the proposed system as follows: First, the expected frame rate is set to 90 fps which coincides with the least frame rate supported by PSVR[4]. Next, the latency (i.e., reaction time against the player's action) of the system is set to be 40 ms, which is consistent with an observation such that the latency exceeding 40 ms causes VR sickness and humans cannot perceive the delay in the visual field of less than 20 ms [5].

3 Related Work

The design of sophisticated Cloud Gaming Platform (CGP) has been a main concern in realizing an efficient handling of requests issued by a huge number of game players in real-time. It is known that the performance of CGP is significantly affected by the resource allocation scheme and the architecture. In this section, we overview related works concerned with those issues.

3.1 Resource Assignment

Many existing works on CGP explore an effective way of assigning tasks to virtual machines (VMs) and assigning resources to each VM. Wang *et al.* [23] show that a proper scheduling of VMs could reduce the cost of CGP while preserving a high performance. Similarly, [10] proposes a QoE-aware VM assignment strategy, and [16] proposes a heuristic method to assign minimum number of VMs to fulfill the requirement given by the MMOG. Resource allocation reflecting the attribute of tasks is also being considered; e.g., [8] improves the performance of CGP by considering the popularity of games served by OnLive, and [12] proposes a resource allocation strategy based on the expected completion time of each game session, which regards the corresponding problem as a bin-packing problem with minimum number of bins, while it is also pointed out that classical First Fit and Best Fit algorithms do not work effectively since the input distribution for CGP is highly unbalanced [12].

Effective utilization of GPUs is another issue towards sophisticated CGPs. Zhao *et al.* [27] analyze the performance of CPU/GPU server and conclude that a local processing at the client-side is a key to attain high QoE. Kim *et al.* [11] proposed an architecture consisting of several servers and a single GPU to enable the *sharing* of game view rendered by the GPU. Resource allocation schemes proposed in [17, 25] maximize the GPU utilization shared by several servers. Shea and Liu [18] found that GPU pass-through significantly degrades the data transfer rate between main memory and GPU, while it is being resolved [19].

[4] In fact, game contents for PSVR are either created at 120 fps, 90 fps, or re-projected to 120 fps.

3.2 P2P-Assisted Architecture for MMOG

Since the geographical location of players joining an MMOG is globally distributed, CGP should be designed to cover those players with as short latency as possible. The impact of geographical distance to the latency was empirically studied by Choy *et al.* [6]. They showed that the location of data centers in US leaves an unacceptable level of RTT to a large portion of the country. More specifically, the coverage of clients whose latency from Amazon EC2 is less than 80 ms[5] is only 70% despite many clients exist in densely populated area. The same authors proposed a hybrid cloud architecture [7] to expand the coverage of low latency area with the aid of Smart Edges, where Smart Edge is an extension of edge server used in Content Delivery Networks (CDN).

Süselbeck *et al.* [20] proposed a method to reduce the latency of cloud MMOG. Their idea is to assign the MMOG client corresponding to a game player to the same machine with the MMOG server corresponding to the game field, so that the communication between game client and game server can always be quickly done. On the other hand, Lin and Shen proposed CloudFog [13, 14] to overcome the following issues concerned with cloud games:

1. It requires a broad downlink capacity (e.g., OnLive recommends 5 Mbps or higher), and requires to cover many users with a guaranteed latency [6].
2. It must reduce the communication cost. In fact, with an average traffic of 27 TB per 12 hours, Amazon EC2 costs about $ 130,000 per month ($ 0.085 per GB) [8].

The basic idea of CloudFog is to recruit super nodes called *fog* near the clients to conduct the rendering of the game view, where the rendering can be performed on the cloud server if no suitable super node can be identified. In addition, CloudFog uses the following techniques to improve the performance:

1. Fog nodes are selected according to the reputation score as well as the upload capacity and the latency.
2. It uses an adaptive adjustment of the coding rate. A similar technique has been deployed by Tian *et al.* [21], who showed that it reduces the server cost by 25% while preserving QoE.
3. It takes into account the friend relationship in SNS during the assignment of players to fog nodes, which could be regarded as an extension of [20].
4. It dynamically adjusts the number of candidates for fog nodes based on the number of participants which is predicted by analyzing the access log.

[5] It is known that human perceives the latency exceeding 100 ms. Thus if the processing time at the server-side is 20 ms, acceptable communication delay is calculated as 80 ms.

4 Proposed Method

The proposed system consists of cloud server, fog nodes, and clients. In this system, a cloud game proceeds as follows:

1. Each client continuously generates: 1) sensor data acquired by HMD and 2) output signals of the controller, where the former is directly sent to the corresponding fog node (for the positional tracking), and the latter is sent to the cloud server (for the interaction with virtual objects). The way of assigning clients to fog nodes will be described in Section 4.2.
2. The cloud server conducts the collision detection of virtual objects reflecting all signals received from players to generate a data stream representing the consistent movement of objects in the virtual space (see Section 4.1 for the details).
3. The resulting data stream is sent to fog nodes, with additional information on the state of objects, which is used to generate a stream of game view at fog nodes. The basic rendering scheme is described in Section 4.3, and an extended way to offload the rendering task is discussed in Section 4.4.
4. After receiving game view from the corresponding fog node, the client displays it on the HMD.

4.1 Consistent Collision Detection

In VRMMOGs, there are various objects in the virtual space such as buildings, items, avatars and monsters which *interact* with each other. Each object has physical attributes such as hardness, viscosity, and coefficient of restitution, and if two objects collide at a certain speed, it causes a unique physical reaction such as repulsion and sinking (e.g., the collision of tennis balls must have a different appearance from the collision of eggs). Such a reaction relies not only on the initial state of the virtual space, but also on interruptions conducted by the players through controllers (e.g., imagine the game screen of breakout). In addition, the outcome of such calculations must be *unique*; namely it must be independent of the observer and the calculator.

The proposed method assumes the use of Unity as the game engine, and assigns the task of such a collision detection to the cloud server with a physics engine of Unity called PhysX. In Unity, each object can have physical attributes such as mass, moving direction, and the moving speed by attaching Rigidbody component. In addition, the object shape used in the rendering of a collision is defined by Collider component, and collision-related attributes such as friction coefficient and elasticity are defined by the physical property material. The output of such calculations is a data stream representing the motion of objects in the virtual space. As will be described later, this data stream is used by fog nodes to acquire information necessary to render the game view, e.g., how much impact the blow of a monster has given to the avatar (namely it is critical hit or not).

4.2 Distributed Assignment of Clients to Fog Nodes

The data stream generated by the cloud server is sent to fog nodes to generate a stream of game view for each player. In the proposed method, we *densely* prepare fog nodes over the network so that each client corresponding to a player has at least one fog node within a designated RTT, e.g., 20 ms. Base station of cellular network is a good candidate for such a fog node [26]. Each client is assigned to exactly one fog node, and each fog node conducts the rendering of game view for all clients assigned to it, where clients assigned to a fog node might *not* share the game view; namely, the rendering should be independent for each client in the worst case.

The assignment of clients to fog nodes is conducted in a similar way to the assignment of requests to edge servers in CDN. More specifically, by using a variant of DNS server as the routing mechanism, a given request is redirected to the IP address of an appropriate fog node which is selected by the routing algorithm by considering the network proximity, traffic, and the load of nodes. The reader should note that all commercial CDNs such as Akamai and CloudFlare, and many P2P-assisted CDNs such as NetSession, LiveSky, Tudou, and KanKan adopt such a DNS redirect mechanism.

4.3 Rendering with Unity

The game view in VRMMOG is created by taking snapshots of 3D model by a virtual camera, where the position and the orientation of the camera is refined by the data received from HMD with a Unity library. Snapshots are created with the graphic engine of Unity, where to obtain a stereoscopic vision, we should install two cameras at positions separated by the distance between the left and right eyes of the avatar. Finally, created snapshots are converted to a texture, encoded in PNG format, and delivered to each client.

The reader should note that although general 3D modeling merely uses the shape, position, and the velocity of objects as the basic information, VRMMOGs use supplemental information such as the color and the brilliance of an item (since it might change due to the effect of a magic), the physical strength of avatar (to be restored by a recovery medicine), and a routine motion invoked by specific commands such as spell cast.

4.4 Offload of the Rendering

In the proposed system, fog nodes assigned many clients could easily be overloaded which increases the rendering time; i.e., the latency. In general, 3D game view

Table 1. Spec of machines used in the testbed.

Cloud	Amazon EC2; Instance type: g2.8xlarge
OS: HW	Windows Server 2016 Datacenter CPU: Intel Xeon E5-2670 0 @2.60GHz; Memory: 60GB
Fog node	Windows PC
OS HW	Windows 10 Education CPU: Intel Core i5-7500 3.40GHz; Memory: 8GB; GPU: NVIDIA GeForce GTX1060 3GB
Client	Google Pixel3 XL
OS HW	Android 9.0 SoC: Snapdragon 845; CPU: 64-Bit Octa-Core (Kryo 385 Gold x4 & Kryo 385 Silver x4 2.80GHz); Memory: 4GB; GPU: Adreno 630

can be divided into foreground (FG) view and the background (BG) view, where FG view is updated more frequently than BG view and is more sensitive to the positional tracking than BG view. Thus as an option of the rendering, we propose to divide it into two sub-tasks for FG and BG views, respectively, and offload the latter to other machines including the cloud server. Those two sub-tasks can be synthesized in the following manner: 1) BG view is rendered by simply omitting objects included in FG view; 2) the resulting BG view is encoded in PNG format and is sent to the corresponding fog node; and 3) the received BG view is regarded as a *background texture* for the rendering of FG view.

5 Evaluation

5.1 Setup

To evaluate the performance of the proposed method, we implement a testbed using Amazon EC2 as the cloud server, high-end PC as the fog node, and Google Pixel3 XL as the client (detailed specification of those components is summarized in Table 1). The resolution of the game stream is fixed to 1920×1080 (Full HD). We use Aterm WG2200HP (IEEE 802.11ac compliant and with maximum transmission rate of 866Mbps) for the wireless connection with the client. We also use Oculus Rift with resolution 1080×1200 (one eye) as the high-end HMD. All programs are written in C#.

With the above testbed, we conduct several experiments described as follows:

1. At first, we evaluate the *reaction time* of positional tracking in Section 5.2. The proposed method is compared with cases in which: 1) the positional tracking is conducted at the cloud server instead of fog node; or 2) high-end HMD is used

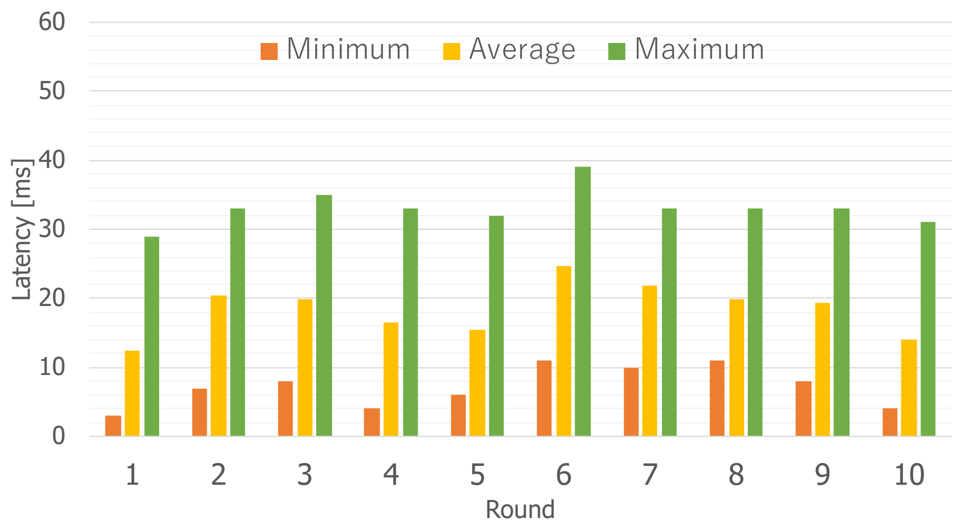

Fig. 1. Tracking time of the fog-assisted method (excluding the decoding time conducted at the client).

instead of mobile HMD. It is confirmed that the proposed fog-assisted method achieves a reaction time of less than 40 ms even with mobile HMD.

2. Next, we evaluate the effect of off-loading of the rendering task in terms of the reaction time in Section 5.3. It is confirmed that the reaction time increases as the number of clients assigned to the fog node increases, but the rate of increase reduces by off-loading part of rendering task to the cloud server.
3. Finally, we evaluate the load of collision detection conducted on the cloud server in Section 5.4. In this experiment, we assume that the virtual space is a room closed by walls and the ceiling and the room is filled with N moving balls, where N is used as a parameter. It is certified that the calculation time quadratically increases as N increases, and in the current implementation shown in Table 1, it takes about 500 ms when $N = 10000$.

5.2 Latency of Positional Tracking

In this paper, we define the latency of positional tracking to be the time period from which HDM starts to move with an intention of eye movement to which it is reflected to the screen of the HMD, where the former can be identified by monitoring the output of sensors mounted on the HMD. To identify the timing of the latter event, we consider the following *ball catching game*:

1. Consider a 3D space partially filled with balls.
2. The system randomly selects a ball, marks it with red color, and requests the player to move his head so that the marked ball is at the center of the view.

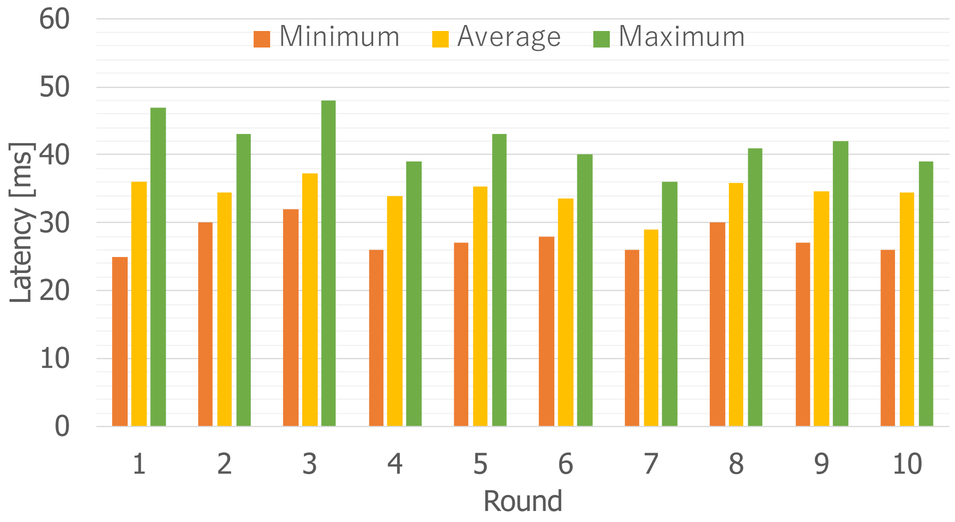

Fig. 2. Tracking time of cloud-based method (excluding the decoding time conducted at the client).

3. The player tries to clear the mission.
4. When the request is fulfilled, the system removes the mark in the rendered view to notify the player that the mission completes.
5. The client records the timing at which the unmarked view is received from the system.

Note that in the above setting, the measured time period does not include the reaction time of the player.

For fair evaluations, we ask two students to play the above ball catching game to eliminate the effect of individual differences. More concretely, after one minute of practice, each player repeats a round consisting of 10 trials of mark-and-catch, and measure the latency of each trial. Such a round is repeated 10 times to eliminate the effects of habituation and the fatigue. Figure 1 summarizes the results, where it is confirmed that the measured latency is 18.5 ms on average. In actual game environment, game view is displayed on the screen after being decoded, which takes 13.78 ms at the client (i.e., Google Pixel3 XL). Hence the overall latency perceived by the player is calculated to be 32.28 ms on average which is certainly less than 40 ms.

For comparison, we conduct similar experiment in different environment in which: 1) the positional tracking is conducted on the cloud server instead of fog node (Case 1) or 2) high-end HMD is used instead of mobile HMD (Case 2). Figure 2 summarizes the results for Case 1, where the average latency (excluding the decoding time) increases to 34.5 ms, which increases the overall tracking time perceived by the player to 48.28 ms. Concerned with Case 2, with the aid of high-speed HDMI connection, the measured latency of the proposed method reduces to 7.8 ms

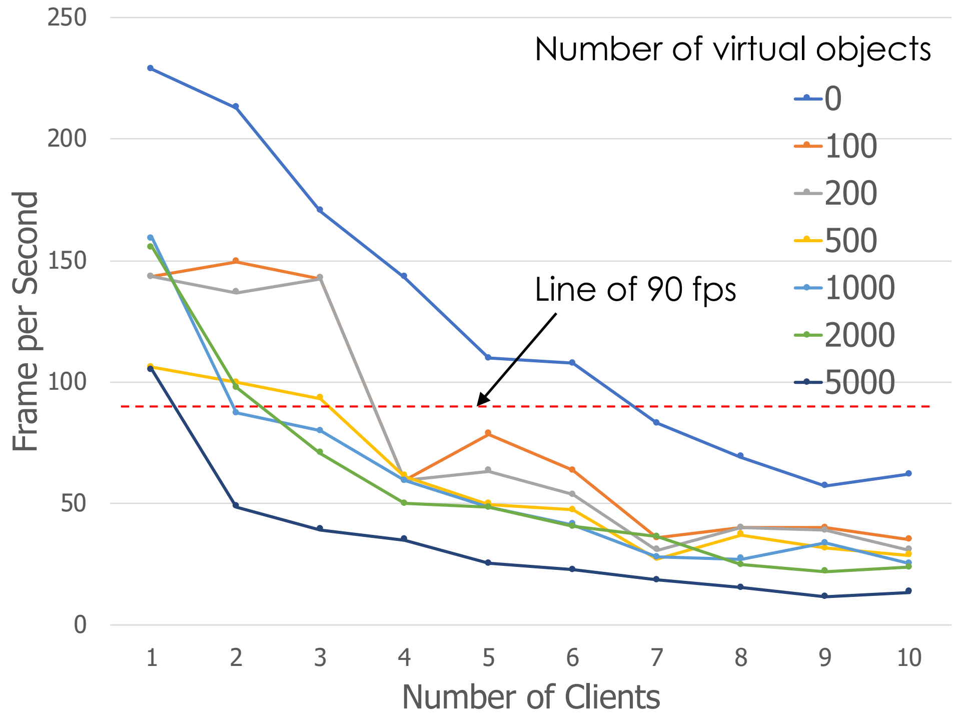

Fig. 3. Impact of the number of clients to the frame rate.

and that of cloud-based method reduces to 25 ms (excluding the decoding time at the client). This implies that we will have a chance to realize a positional tracking of less than 20 ms with the progress of wireless communication technology.

5.3 Time for the Rendering

Next, to clarify the load of fig nodes concerned with the rendering task, we evaluate the performance of fog node in terms of the frame rate by increasing the number of clients assigned to it. Recall that our target is to deliver a stream of game view with a sufficiently high resolution such as Full HD to the clients with a sufficiently high frame rate such as 90 fps.

Figure 3 summarizes the results, where the vertical axis is the frame rate observed by the clients which is averaged over all clients and 10 trials. It is confirmed that although we could keep a high frame rate of 90 fps up to six clients when the number of objects in the game view is zero, the maximum number of clients covered by the fog node reduces to three when the number of objects increases to 100. In addition, it could cover only one client when the number of objects in the game view becomes 1000. The non-monotonicity of the frame rate with respect to the number of objects and the number of clients is due to the effect of compression scheme used

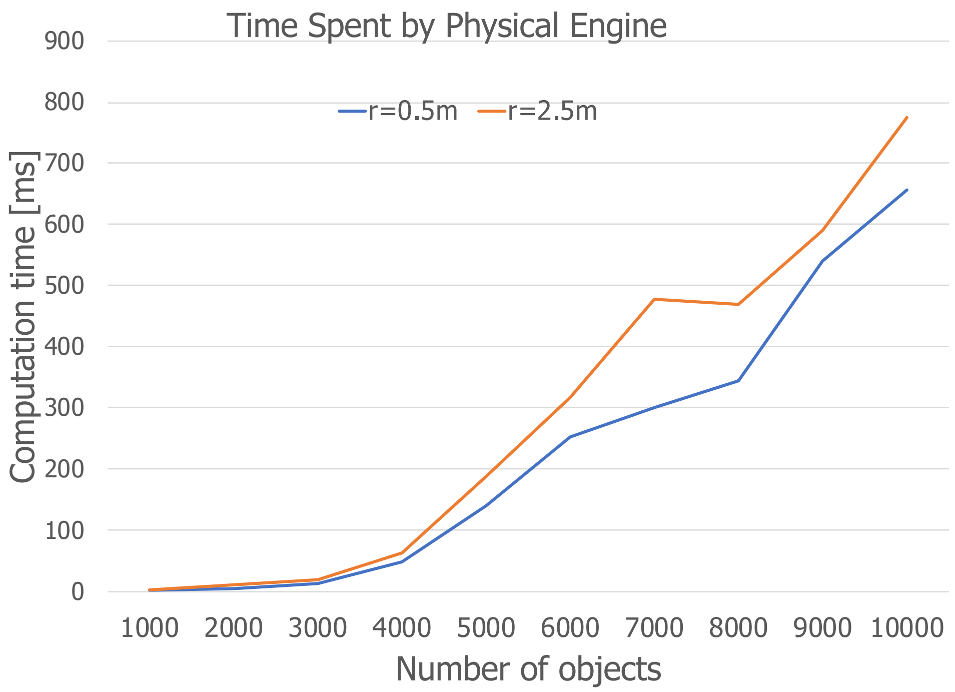

Fig. 4. Computation time required for the collision detection conducted at the cloud.

in modern video codec. Although there exists such a non-monotonicity, the above results indicate that we could cover more clients with a sufficiently high frame rate by migrating the rendering of background view to the cloud server and by sharing the resulting view (i.e., texture) among several fog nodes. Detailed evaluation of the effect of offloading is left as a future work since it strongly depends on the game design.

5.4 Performance Analyses of Collision Detection

Finally, we measure the computation time required for the collision detection by varying the number of moving balls given in a closed 3D space of 100 meters cube[6]. Figure 4 summarizes the results. It is confirmed that the time required for the collision detection increases as the number of moving balls increases and it depends on the times of collisions. In fact, as is shown in the figure, it takes slightly longer time when the radius of balls increases from 0.5 meters to 2.5 meters.

[6] The computation time is obtained by referring to FixedUpdate.PhysicsFixedUpdate label of the Unity Profiler. Note that it does not include the rendering time.

6 Concluding Remarks

This paper considers a fog-assisted cloud game platform to realize a virtual-reality MMOG. Several techniques are examined to achieve ultra-low latency. The result of evaluations indicates that the proposed method realizes the latency for the positional tracking in only 32.3 ms which is significantly shorter than conventional cloud-based platform, and suggests that the latency of less than 20 ms could be achieved with the development of wireless communication technology compatible with HDMI connection.

We leave the following issues as future work: 1) to evaluate the overall performance of the proposed method in actual VRMMOG; and 2) to propose a method for the dynamic offloading of the rendering task assigned to fog nodes.

Conflicts of Interest

The authors declare no conflict of interest.

Acknowledgements

This work was partially supported by KAKENHI, the Grant-in-Aid for Scientific Research (B), Grant Number 16H02807.

References

1. Google Stadia, `https://stadia.dev/`
2. PlayStation VR, `https://www.jp.playstation.com/psvr/`
3. Google Cardbord, `https://vr.google.com/intl/ja_jp/cardboard/`
4. Eagle Flight, `https://www.ubisoft.com/en-us/game/eagle-flight/`
5. J. Carmack. "John Carmack' s Delivers Some Home Truths on Latency." `http://oculusrift-blog.com/`
6. S. Choy, B. Wong, G. Simon, and C. Rosenberg. "The brewing storm in cloud gaming: A measurement study on cloud to end-user latency." In *Proc. 11th IEEE Annu. Workshop Netw. Syst. Support Games (NetGames'14)*, 2012, pages 1–6.
7. S. Choy, B. Wong, G. Simon, and C. Rosenberg. "A hybrid edge-cloud architecture for reducing on-demand gaming latency." *Multimedia Syst.*, 20(5): 503–519, 2014.
8. D. Finkel, M. Claypool, S. Jaffe, T. Nguyen, and B. Stephen. "Assignment of games to servers in the OnLive cloud game system." In *Proc. Annu. Workshop Netw. Syst. Support Games (NetGames'14)*, 2014.
9. A. Franco, Em. Fitzgerald, B. Landfeldt, U. Körner. Reliability, timeliness and load reduction at the edge for cloud gaming, in *Proc. International Performance Computing and Communications Conference*, 2020.
10. H.-J. Hong, D.-Y. Chen, C.-Y. Huang, K.-T. Chen, and C.-H. Hsu, "QoEaware virtual machine placement for cloud games," In *Proc. Annu. Workshop Netw. Syst. Support Games (NetGames'13)*, 2013, pp. 1–2.
11. S. S. Kim, K. I. Kim, and J. Won, "Multi-view rendering approach for cloud-based gaming services." In *Proc. Int. Conf. Adv. Future Internet (AFIN)*, 2011, pages 102–107.

12. Y. Li, X. Tang, and W. Cai. "Play request dispatching for efficient virtual machine usage in cloud gaming." *IEEE Trans. Circuits Syst. Video Technol.*, 25,(12): 2052–2063, 2015.
13. Y. Lin and H, Shen. Leveraging Fog to Extend Cloud Gaming for Thin-Client MMOG with High Quality of Experience," In *Proc. 35th ICDCS*, IEEE, 2015, pages 734–735.
14. Y. Lin and H, Shen. "CloudFog: Leveraging Fog to Extend Cloud Gaming for Thin-Client MMOG with High Quality of Service." *IEEE Trans. Parallel Distrib. Syst.*, 28(2): 431–445, 2017.
15. K. Manoj. Enhancing Cloud Gaming User Experience through Docker Containers in Fog Nodes, Masters thesis, Dublin, National College of Ireland, 2019.
16. M. Marzolla, S. Ferretti, and G. D' Angelo. "Dynamic resource provisioning for cloud-based gaming infrastructures." *ACM Comput. Entertainment*, 10(3): 4:1–4:20, 2012.
17. Z. Qi, J. Yao, C. Zhang, M. Yu, Z. Yang, and H. Guan. "VGRIS: Virtualized GPU resource isolation and scheduling in cloud gaming." *ACM Trans. Archit. Code Optim.*, 11(2): 203–214, 2014.
18. R. Shea and J. Liu, "On GPU pass-through performance for cloud gaming: Experiments and analysis." In *Proc. Annu. Workshop Netw. Syst. Support for Games (NetGames' 13)*, 2013, pages 6:1–6:6.
19. R. Shea, D. Fu, and J. Liu. "Cloud gaming: Understanding the support from advanced virtualization and hardware." *IEEE Trans. Circuits Syst. Video Technol.*, 25(12): 2026–2037, 2015.
20. R. Süselbeck, G. Schiele, and C. Becker. "Peer-to-peer support for low latency massively multiplayer online games in the cloud," In *Proc. 8th Annu. Workshop Netw. Syst. Support Games (NetGames' 09)*, 2009, pages 1–2.
21. 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.*, 25(12): 2064–2077, 2015.
22. A. Tsipis, K. Oikonomou, V. Komianos, and I. Stavrakakis. Performance Evaluation in Cloud-Edge Hybrid Gaming Systems, in *Proc. 3rd International Balkan Conference on Communications and Networking (BalkanCom'19)*, 2019.
23. S. Wang, Y. Liu, and S. Dey. "Wireless network aware cloud scheduler for scalable cloud mobile gaming." In *Proc. IEEE Int. Conf. Commun. (ICC)*, 2012, pages 2081–2086.
24. T. Yoshihara and S. Fujita. "Fog-Assisted Virtual Reality MMOG with Ultra Low Latency." In *Proc. CANDAR'19*, 2019, pages 121–129.
25. C. Zhang, Z. Qi, J. Yao, M. Yu, and H. Guan. "vGASA: Adaptive scheduling algorithm of virtualized GPU resource in cloud gaming." *IEEE Trans. Parallel Distrib. Syst.*, 25(11): 3036–3045, 2014.
26. X. Zhang, H. Chen, Y. Zhao, Z. Ma, Y. Xu, H. Huang, H. Yin, and D. O. Wu, Improving Cloud Gaming Experience through Mobile Edge Computing, *IEEE Wireless Communications*, 26(4): 178–183, 2019.
27. Z. Zhao, K. Hwang, and J. Villeta, "Game cloud design with virtualized CPU/GPU servers and initial performance results," In *Proc. Workshop Sci. Cloud Comput. Date (ScienceCloud)*, 2012, pages 23–30.