10-2021

CloudNPlay: Resource optimization for a cloud-native gaming system

Angelus WIBOWO
Nanyang Technological University

Nguyen Binh Duong TA
Singapore Management University, donta@smu.edu.sg

CloudNPlay: Resource Optimization for A Cloud-Native Gaming System

Angelus Wibowo
School of Computer Science and Engineering
Nanyang Technological University

Ta Nguyen Binh Duong
School of Computing and Information Systems
Singapore Management University

***Abstract*—Cloud gaming enables people playing graphically intensive games from their less powerful, or even outdated computing devices. It is challenging to realize cloud gaming as it requires minimal latency in server-side processing, rendering and streaming, which are expensive in terms of resource requirements, e.g., powerful GPU servers. Commercial gaming providers, e.g., Google Stadia, Amazon Luna, etc., hardly disclose any information on how they optimize gaming performance and cloud cost. In this work, we aim to investigate resource cost optimization for such cloud gaming systems. In contrast to previous work which have been focusing more on theoretical approaches, we deliver a fully functional, cost-optimized cloud-native gaming system, called CloudNPlay, implemented entirely on AWS Lambda, EC2 and other AWS services. We have conducted extensive evaluations on the gaming performance including latency and frame rates; as well as cloud resource cost reduction. The results indicate that games hosted on CloudNPlay are highly playable. More importantly, CloudNPlay reduces the resource cost by around 24% compared to a non-optimized deployment. We plan to open-source CloudNPlay to facilitate further research in cloud gaming.**

***Index Terms*—cloud gaming, video streaming, latency, cost optimization**

I. Introduction

Video games these days can be found on almost every computing platform, from PC to consoles and mobile phones. As technology advances, the graphics quality of video games is also getting better. This has resulted in much bigger file sizes, e.g., 100GB or more, for certain games. Not only that, games with high graphics quality demand good hardware to run smoothly; normally a decent GPU is needed. The needs for more storage and better GPUs force people to upgrade their hardware, so that they can play the most up-to-date high-end games with better experience. However, upgrading hardware does not make much sense especially when you only play occasionally.

With the development of cloud computing, it is now possible to serve various applications and pay for computing resources on a pay-as-you-go basis. In cloud gaming, all the processing and rendering tasks will be handled on cloud servers; then the rendered video will be streamed to the client over the Internet. The client device displays the streamed content, captures user inputs, and sends them back to cloud servers for further processing [1]. In this way, the cloud server handles most of the computationally demanding tasks, so that we are able to play games with high-end graphics without the need of constantly upgrading our hardware. In addition, game developers can support more platforms easily with less cost and effort.

However, there are a few critical issues that any cloud gaming solutions have to consider. First, since the input data needs to be sent to the server and the game rendering needs to be streamed back to the client via the Internet, game playability is highly dependent on the network conditions and server-side processing. Latency is often a bottleneck, causing noticeable lags during gameplay and may degrade user experience significantly [2]. Second, the cost of using GPU servers for game rendering can be expensive, e.g., on Amazon EC2, a Windows GPU instance can be charged an hourly rate of several dollars; and this is just one server. A cloud gaming service provider handling many concurrent users should manage such cost properly to stay in business.

Various commercial cloud gaming solutions have been introduced in the past few years. Notably, OnLive and Gaikai were launched in 2010 and 2012, respectively, but later both were acquired and closed down by Sony. Recently, Google, NVIDIA and Sony announced their own proprietary cloud gaming platforms named Stadia, GeForce Now and PS Now, respectively. At the time of writing, Microsoft xCloud and Amazon Luna are available in early access mode. It is clear that cloud gaming is a highly competitive area right now; and it is very challenging for new entrants to the market. This is partly because these big cloud gaming providers typically do not reveal much information on how their services are designed, implemented and optimized [3]. GamingAnywhere [4] has traditionally been used in cloud gaming performance research, but it has not been updated for many years (latest release was in 2014).

In this work, we design and implement a cloud-native gaming architecture, named CloudNPlay, which might serve as a reference and an extensible solution for new cloud gaming services. In particular, we have made the following contributions:

- We design and implement a fully functional cloud gaming solution using AWS EC2 and Lambda, among other AWS services. The system is cloud-native; it has been designed as a set of loosely coupled cloud services which can be extended, deployed and scaled quickly with minimal initial capital expenditure.

- We consider the problem of resource optimization in such a cloud-native gaming service taking into considerations of various realistic cost and billing constraints. We then implement a simple but practical cost optimization algorithm into CloudNPlay. This is in contrast with previous research, e.g., [5], [6], [7], [8], which mostly consider theoretical approaches and do not implement their proposal in an actual cloud gaming system.
- We evaluate the performance of cloud gaming in realistic conditions, and found that our system delivers acceptable latency and frame rate for a good gaming experience. On top of that, we demonstrate that our resource optimization produces a total cost saving of up to 24%.

II. CloudNPlay: System Design

A. Architecture

Our design considers a typical cloud gaming service [1], which includes the following components:

- **User authentication**: to enable user logins to the system which then shows their account information and a list of available games. User can then select a game to play.
- **Game server management**: the cloud gaming system needs to provision the selected games on appropriate servers (normally with GPUs). User is then redirected to a game server and starts playing. The game server processes user inputs based on a game-specific logic, and renders the game screen accordingly. The game server then captures and encodes the game screen using a video/audio encoder, and streams it to the client device. We have to perform resource optimization to reduce cloud cost in the long run.
- **Client rendering and input handling**: the client software captures user inputs and sends them to the game server via the network. The client software also decodes the received stream using a video decoder, and displays it on user device.
- **Front-end web interface**: for user login, registration, account details, billing, etc.

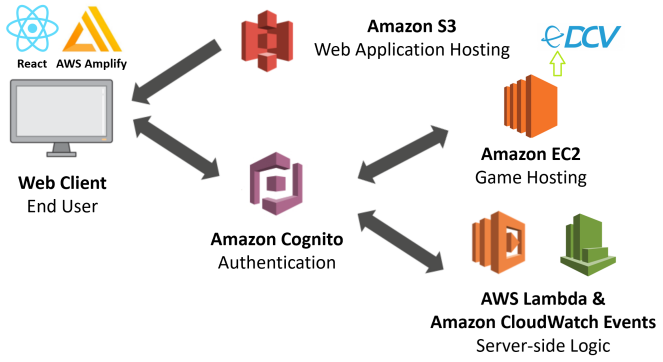

Fig. 1. CloudNPlay: cloud-native gaming architecture

CloudNPlay is cloud-native; it consists of a set of loosely coupled cloud services implemented using serverless (AWS Lambda), CloudWatch, EC2, etc. Adopting serverless reduces back-end infrastructure maintenance and management tasks, thus providing more time for developers to focus on the application logic. It might also be more cost effective, since we do not need to pay for idle computing time.

Various AWS and other services are used in CloudNPlay as shown in Figure 1. In particular: 1) game server management which includes provisioning and optimization algorithms are handled with AWS Lambda and Amazon CloudWatch Events; 2) user authentication is implemented via Amazon Cognito; 3) front-end React web interface is hosted on Amazon S3; 4) AWS Amplify and AWS SDK for Javascript are used to integrate the web app with cloud services; 5) Amazon EC2 is used for on-demand provisioning of game server instances, and 6) NICE DCV (https://www.nice-dcv.com) and Parsec (https://parsec.app) are used for game streaming.

B. Resource optimization

We identify two main server management tasks that require resource optimization in CloudNPlay, which are "get instance" and "release instance". These two tasks need to be run each time a player requests to play a game, and when he/she decides to stop playing, respectively. Each AWS EC2 instance is a game server used for rendering, streaming, and processing game-related logic. In this work, we mainly use Windows instances on EC2; as there are many popular game titles released for Windows. In order to optimize instance cost, we note that AWS EC2 charges Windows instance usage on a hourly basis. For each partial instance-hour consumed, AWS will charge it as one full hour even though the instance has only been used for a few minutes[1]. Considering this fact, CloudNPlay implements a flexible instance sharing mechanism between different users, using the following key ideas:

1) Instance requests will be handled by the system; users are not allowed to select a particular instance or instance type.
2) When user requests for an instance, the system will first find a running instance that is available instead of a stopped instance. If there are multiple running instances available, we select the instance having the longest *residual* duration. We define the residual duration as the amount of time an instance can continue to run until its current 1-hour billing cycle is up.
3) When user releases an instance, i.e., he has stopped playing, the instance will be kept running until its residual time becomes negligible, e.g., 1-2 minutes left. When this instance is still on, the system will try to assign the running instance to another user request.
4) We create an Amazon machine image (AMI) containing the most popular games offered in the system. In this way, any running EC2 instance created from the image will have most of the popular games, so it would be able to handle most user requests for different games. In this paper, we focus on instance cost optimization, as game

[1]https://aws.amazon.com/ec2/pricing/on-demand

storage optimization is a separate problem that has been studied previously [8].

With the proposed instance sharing mechanism, one full instance-hour can actually be shared among several users who only needs to use the instance for a fraction of the hour. In this way, the total instance cost can be reduced. Below we describe the detailed algorithm for each server management task in our system:

1) Get instance: as shown in the pseudo-code below, the system prioritizes getting a running instance to host a user's game instead of a stopped one. This is done to maximize the usage of an already running instance, due to the per-hour billing. After a running instance is assigned to a user, the system will remove the scheduled stop event to prevent it from stopping automatically when the residual time ends.

```
1. User requests to play a game
2. Check if any running instances
   2.1. Yes: return a running instance
   with the longest residual duration
     2.1.1. Remove the scheduled stop on
     the selected instance
   2.2. No: select a stopped instance
     2.2.1. Start the stopped instance
     2.2.2. No stopped instance: start
     a new instance
3. Return instance detail to user
4. Start game streaming
```

2) Release instance: when the user is done playing, we need to release the EC2 instance to save cost. The system will need to calculate the residual time of that instance. If the residual time is less than a given threshold t, e.g., 2 minutes, then the instance will be stopped immediately. Otherwise, a schedule is set using CloudWatch Events to stop the instance when the residual time ends. This is done to prevent AWS from charging another hour of instance usage when the instance is actually not being used. The instance is then tagged as available so it can be used to run another user's game. If the instance is not utilized by other users, and the residual time ends, CloudWatch Events will trigger a Lambda function to stop the instance.

```
1. User chooses to stop playing
2. Get residual time R of the instance
   2.1. If R < t: stop the instance
   2.2. Else:
     2.2.1. Add a scheduled stop for the
     instance
     2.2.2. Tag the instance as available
```

III. Implementation Details

This section provides specific implementation details for our system, including game hosting, game streaming, algorithm implementation, and the front-end web interface.

A. Game hosting

Microsoft Windows Server 2016 Base AMI is used for game instances. Windows AMI is chosen since most popular games are available on Windows. EC2 instance type of g3.4xlarge is selected; it is equipped with NVIDIA Tesla M60 GPU and optimized for graphics-intensive applications. Another reason for choosing this instance type is that its price is relatively cheaper compared to the other GPU instances. Since the instance does not have any audio device by default, a virtual sound card called "VB Audio Virtual Cable" is installed. In this way, the instance can output sound which can be captured by the game streaming software.

B. Game streaming

To stream games from EC2 instances to clients, Parsec is installed on both game servers and clients. Parsec can handle multiplayer sessions and has good supports for a variety of game controllers. To optimize performance and gaming experience, we adjust the following Parsec settings.

The following are applied for hosting settings:

- Bandwidth limit is set to "100Mbps": adding "encoder_bitrate = 100" into the Parsec configuration file.
- Resolution is set to "1920x1080".
- H.265 is set to "Off" since not all devices are equipped with H.265 codec.
- Display adapter is set to "NVIDIA Tesla M60".

The following are applied for client settings:

- V-sync is set to "On" to avoid screen tearing due to the difference between the frame rate of the game and the display refresh rate.
- Windowed mode is set to "Fullscreen" to improve gaming experience.
- Renderer is set to "DirectX" which can perform better on Windows clients.

Since game servers do not have display devices, and Parsec works by capturing screen output, NICE DCV is used as a remote desktop software. NICE DCV is capable of streaming graphics-intensive applications; and free to be used on AWS EC2 instances. In our system, NICE DCV Server (64-bit) is installed on the game server instance. NICE DCV also provides a web browser client for users to establish the connection directly via HTTPS.

C. Resource optimizing

The resource optimization algorithm described above is implemented with Amazon CloudWatch Events and AWS Lambda. CloudWatch Events is used to set a schedule for when an instance needs to be stopped. On the other hand, a Lambda function is implemented in Python to carry out the actual shutdown of unused instances. This Lambda function is triggered by CloudWatch Events schedules. To ensure proper execution of the function, we need an IAM (Identity and Access Management) role configured with AmazonEC2FullAccess, AWSLambdaFullAccess, and CloudWatchEventsFullAccess policies. We take note of the function

execution cost, measured in GB-seconds, as it might affect the total operating cost. Empirical data indicate around 0.1536 GB-seconds for each function invocation.

D. Web front-end

Using the web front-end, users are able to start/stop gaming sessions with ease. The user interface is built using React, providing authentication, account creation, available game listing, and game playing options. Amazon Cognito is used to handle user authentication. The web front-end is hosted in Amazon S3, and it is publicly accessible. It is integrated with the cloud backend via: 1) AWS Amplify for connecting to Amazon Cognito and deployment to Amazon S3; and 2) AWS SDK for JavaScript for accessing AWS EC2 (game hosting), as well as CloudWatch Events and AWS Lambda.

IV. System Evaluation

We evaluate CloudNPlay in two different aspects, namely the gaming performance and the cost optimization mechanism.

A. Gaming performance

In this section, we present our evaluation of important performance considerations in a cloud gaming system; including instance startup time, graphics performance, and game streaming performance.

1) Instance startup time: The startup time is defined as the amount of time needed for the system to start a new game instance. We found that it takes around 12 seconds to start a stopped g3.4xlarge instance, which is quite normal for EC2 Windows instances. A long startup time may negatively affect the user experience. If there are more users, it is good to keep more running instances [9] to reduce game startup time.

2) CPU and GPU performance: We measure the performance of the selected EC2 instance type using 3DMark to ensure that it is suitable for gaming purposes. In particular, 3DMark Time Spy, a DirectX 12 benchmark for gaming PCs, is used. 3DMark Time Spy renders scenes with 2560 x 1440 resolution; and includes both graphics and CPU tests. We observe that g3.4xlarge scored 3575 points in the test, which is better than a typical gaming PC with Intel Core i5-4590 and NVIDIA GeForce GTX 970 (scored 3362).

3) Cloud gaming performance: We measure actual cloud gaming performance by streaming "Life is Strange", created by Dontnod Entertainment, on our system for roughly 30 minutes. The game was running in borderless mode with V-sync turned on. Screen resolution was set to the popular setting of 1920 x 1080. We then measure the bitrate, frame rate, and response delay during the game streaming process.

- **Bitrate** is defined as the amount of data being transmitted to the client per second. With higher bitrates, higher stream quality can be achieved. Bitrate values can be obtained from the Parsec console during game streaming. We measured values of 2Mbps, 40.4Mpbs, and 23.9Mpbs as the minimum, maximum and average bitrates, respectively. From the result, it can be observed that the bitrate is good enough for game streaming. As a comparison, game streamers on Twitch use bitrate between 3 to 6 Mbps to stream their 1080p videos at 60fps [10].
- **Frame rate** is defined as the amount of frames rendered per second (fps). Our system was able to maintain a frame rate of 60fps most of the time, which is considered the ideal target for most games [11]. More specifically, we measured a minimum fps of 56.6, a maximum fps of 60.1, and an average fps of 59.92 from CloudNPlay.
- **Response delay** can be defined as the total time it takes from the time a user input is received until the corresponding change is seen by the user. Such response delay can be segmented further into three components: 1) network delay, i.e., the round trip time between client and server; processing delay - the time needed for a server to process game logic and do rendering; and playout delay - the time required to decode and display the game scenes on the client [12]. We do not consider the last component as it is not something a cloud gaming system can control. In our experiments, we measured the network latency and the processing delay. The average total delay was around 11.6ms, which is considered great for interactive gameplay [13], [14].

B. Cost optimization

CloudNPlay can perform instance sharing between different users to reduce resource cost. As it is not economically feasible to evaluate CloudNPlay with thousands of actual users, in this paper we describe a large-scale simulation based analysis of the cost optimization algorithm implemented in CloudNPlay. The Python-based simulation makes use of an actual game trace obtained from the Game Trace Archive[2]. In particular, we use the WoWSession dataset containing records of "World of Warcraft" game sessions of more than one hundred players from May to July 2009. The dataset provides timestamps of when a gaming session starts and ends. The maximum number of users playing at the same time is 35 in the trace.

Figure 2 shows the number of instance-hours consumed with, and without the cost optimization algorithm based on instance sharing. We note that there are a total of 35 instances that have been started in both cases. When cost optimization is applied, different users can actually reuse the same instances that have been running. In this figure, we can see that the first few instances have been more utilized to serve many different users - the system did not have to shut them down with cost optimization. It is observed that CloudNPlay can save around 3571 instance-hours, i.e., 30.6% reduction in instance cost.

We further calculate the total resource cost which includes not just the instance-hours, but also data storage fee, data transfer for game streaming, Lambda request cost, etc. This is to better assess the overall cost saving from implementing the resource optimization algorithm. We use the following parameters from running the game trace simulation, and from measurements of CloudNPlay:

[2]http://gta.st.ewi.tudelft.nl/

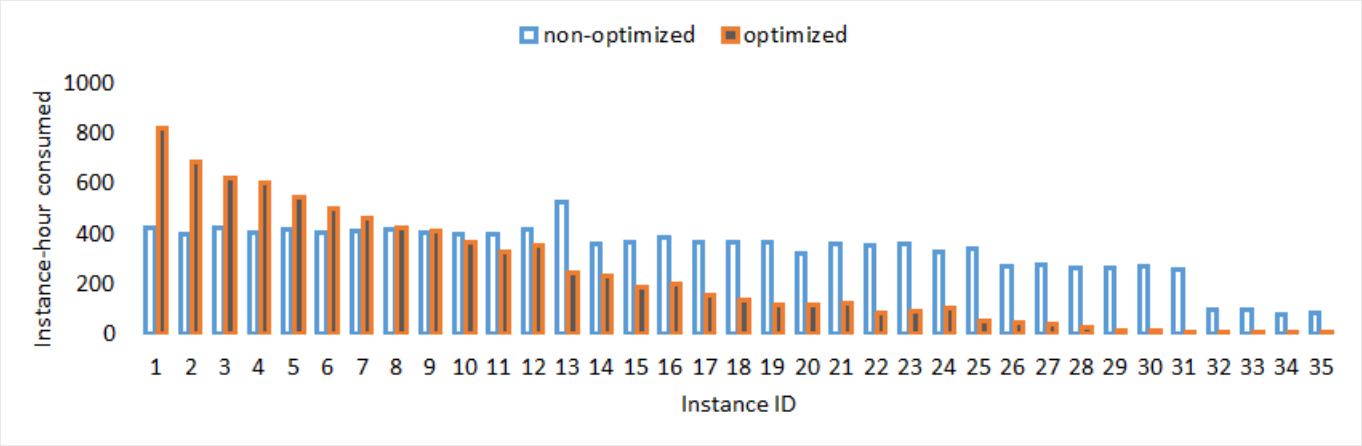

Fig. 2. Number of instance-hours consumed: with and without cost optimization

- The dataset records game activities from around a hundred users for 2 months, in which all users play the same game (World of Warcraft).
- There are roughly 7000 game sessions in the data set - we assume that on average we have 3500 game sessions for a month.
- Average duration of the game sessions is 3628.31 seconds, i.e., roughly 1 hour.
- The system consumes 11671 instance-hours without cost optimization; and 8100 instance-hours with cost optimization. Therefore we assume that there are 5836 instance-hours and 4050 instance-hours consumed for a month, respectively.
- The Lambda function is triggered 1723 times to run the cost optimization algorithm, i.e., roughly 862 times for a month. The Lambda cost is 0.1536GB-seconds each time.
- The game size is assumed to be 70GB, which is based on World of Warcraft system requirements.
- Average data transfer rate is 23.90Mbps, i.e., 10.755GB per hour.

Table I shows the rate of all services used by CloudNPlay as well as the total cost calculations without and with cost optimization. We observe a total cost saving of 24% approximately. We envisage that if there are more consecutive players, there can be more instance-hour savings and vice versa.

V. Related Work

Research in cloud gaming have been mostly focused on two distinct aspects: 1) improving the interactivity via more efficient game rendering and streaming performance; and 2) optimizing the cloud resource cost. In [15], the authors described a distributed game engine with a loosely coupled graphical renderer, enabling rendering operations to be executed across different cloud instances. [16] presents techniques to improve cloud gaming video encoding via enhancing the perceived video quality with regard to network bandwidth constraints, as well as reducing the computational complexity. In [17], the authors proposed game-agnostic techniques to reduce the network bandwidth requirement for game streaming by exploiting weaknesses of the human visual system. In [7], the authors considered cloud rendering optimization via reusing common information between hundreds of concurrent players to minimize rendering server rental cost. In this paper, we focus more on cost reduction techniques; but at the same time we conduct measurements of the game rendering and streaming process to ensure that CloudNPlay is able to deliver good interactive performance.

Much research has been conducted in the area of resource optimization for cloud gaming, for instance [5], [8], [18], [19], etc. In [8], the authors argued that in cloud gaming systems, storage cost could be substantial compared to instance cost. They then proposed cloud server provisioning approaches that can optimize both instance cost and storage cost. Similarly, [20] considers a multiplayer cloud gaming architecture in which a game server and several rendering servers should be provisioned for a set of players. In this case, it is important to minimize both server rental cost and network bandwidth cost. CloudNPlay only considers single-player games in its current version. [19] considers game theory to optimize virtual machine placement in mobile cloud gaming. In [5], the authors considered consolidating several games on to the same cloud server to improve utilization and cost. Such consolidation requires careful considerations on performance interference between games. Our work differs in the sense that we use a fully functional cloud gaming implementation. In addition, we allow only one running game on a sufficiently sized instance, and implement instance-sharing to minimize resource wastage due to per-hour billing for Windows instances.

Currently, there are several providers offering commercial cloud gaming services, e.g., Google Stadia, GeForce Now, PS Now, Amazon Luna, etc. However, not much details on how they have optimized gaming interactivity and cost have been published. Very recently, there have been some attempts to study these gaming services. For instance, [3] analyzes network performance of Stadia, GeForce Now and PS Now from a user-perspective. In contrast, our work delivers a fully

TABLE I
MONTHLY COST CALCULATIONS

Service	Rate	Not optimized	Optimized
AWS EC2 instance (g3.4xlarge)	$2.406 per instance-hour	5938hrs x $2.406 = **$14041.42**	4050hrs x $2.406 = **$9744.3**
Amazon EBS (Storage)	$0.12 per GB-month	35 instances x 70GB x $0.12 = **$294**	
Data Transfer (In)	Free	**$0**	
Data Transfer (Out)	Free up to 1GB/month $0.12 per GB for next 9.999TB/month $0.085 per GB for next 40TB/month	10.755GB/hr x 3500hrs = 37642.5GB 9999 x $0.12 + 27642.5 x $0.085 = **$3549.49**	
AWS Lambda request	Free up to 1M requests/month $0.20 per 1M requests thereafter	**$0** (Not used)	**$0** (Still within free tier)
AWS Lambda compute time	Free up to 400000GB-seconds/month $0.00001667 per GB-second thereafter	**$0** (Not used)	**$0** (Still within free tier)
Amazon CloudWatch Events	Free	**$0** (Not used)	**$0**
Total		**$17884.91**	**$13587.79**

functional cloud gaming system to encourage more practical research in performance issues and resource cost optimization from the service provider perspective. GamingAnywhere [4] has been traditionally used for this purpose. However, it has not been updated for many years, and it is not known whether latest games and cloud-native services can be supported on the platform.

VI. CONCLUSION

In this paper, we have described CloudNPlay, a cloud-native system designed to demonstrate that it is possible to build a cloud gaming system entirely based on existing cloud services. CloudNPlay implements the full application stack from user management, authentication, game streaming, and game client. On top of that, we have developed a simple but practical resource optimization algorithm to reduce cloud cost, which could be a huge barrier of entry for new cloud gaming providers.

We conducted empirical performance evaluation of CloudNPlay in both gaming performance and cost saving aspects. We demonstrated that it is possible to play graphics-intensive games in CloudNPlay with good resolution, frame rates, and responsiveness. At the same time, CloudNPlay can significantly reduce cloud resource cost with a simple instance-sharing mechanism. Our evaluation showed a total cost saving of around 24% considering all factors such as instance-hour, storage, data transfer and serverless request costs. We plan to release CloudNPlay as an open-source project to facilitate further research in this area.

REFERENCES

[1] R. Shea, J. Liu, E. C.-H. Ngai, and Y. Cui, "Cloud gaming: architecture and performance," *IEEE network*, vol. 27, no. 4, pp. 16–21, 2013.

[2] 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 *Proceedings of the 11th ACM Multimedia Systems Conference*, pp. 15–25, 2020.

[3] A. Di Domenico, G. Perna, M. Trevisan, L. Vassio, and D. Giordano, "A network analysis on cloud gaming: Stadia, geforce now and psnow," *arXiv preprint arXiv:2012.06774*, 2020.

[4] C.-Y. Huang, K.-T. Chen, D.-Y. Chen, H.-J. Hsu, and C.-H. Hsu, "Gaminganywhere: The first open source cloud gaming system," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 10, no. 1s, pp. 1–25, 2014.

[5] Y. Li, C. Zhao, X. Tang, W. Cai, X. Liu, G. Wang, and X. Gong, "Towards minimizing resource usage with qos guarantee in cloud gaming," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 426–440, 2020.

[6] Y. Deng, Y. Li, R. Seet, X. Tang, and W. Cai, "The server allocation problem for session-based multiplayer cloud gaming," *IEEE Transactions on Multimedia*, vol. 20, no. 5, pp. 1233–1245, 2017.

[7] I. Jaya, W. Cai, and Y. Li, "Rendering server allocation for mmorpg players in cloud gaming," in *49th International Conference on Parallel Processing-ICPP*, pp. 1–11, 2020.

[8] Y. Li, Y. Deng, X. Tang, W. Cai, X. Liu, and G. Wang, "Cost-efficient server provisioning for cloud gaming," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 14, no. 3s, pp. 1–22, 2018.

[9] T. N. B. Duong, X. Li, R. S. M. Goh, X. Tang, and W. Cai, "Qos-aware revenue-cost optimization for latency-sensitive services in iaas clouds," in *2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications*, pp. 11–18, IEEE, 2012.

[10] "Twitch Streamers - Twitch Video Encoding." https://stream.twitch.tv/encoding/. Accessed: 2021-02-27.

[11] "What Is The Best FPS For Gaming?." https://www.gamingscan.com/best-fps-gaming/. Accessed: 2021-02-27.

[12] K.-T. Chen, Y.-C. Chang, P.-H. Tseng, C.-Y. Huang, and C.-L. Lei, "Measuring the latency of cloud gaming systems," in *Proceedings of the 19th ACM international conference on Multimedia*, pp. 1269–1272, 2011.

[13] K. Raaen, R. Eg, and C. Griwodz, "Can gamers detect cloud delay?," in *2014 13th Annual Workshop on Network and Systems Support for Games*, pp. 1–3, IEEE, 2014.

[14] I. Lee, S. Kim, and B. Lee, "Geometrically compensating effect of end-to-end latency in moving-target selection games," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pp. 1–12, 2019.

[15] J. Bulman and P. Garraghan, "A cloud gaming framework for dynamic graphical rendering towards achieving distributed game engines," in *12th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 20)*, 2020.

[16] Y. Liu, S. Dey, and Y. Lu, "Enhancing video encoding for cloud gaming using rendering information," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 12, pp. 1960–1974, 2015.

[17] G. K. Illahi, T. V. Gemert, M. Siekkinen, E. Masala, A. Oulasvirta, and A. Ylä-Jääski, "Cloud gaming with foveated video encoding," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 16, no. 1, pp. 1–24, 2020.

[18] I. Slivar, L. Skorin-Kapov, and M. Suznjevic, "Qoe-aware resource allocation for multiple cloud gaming users sharing a bottleneck link," in *2019 22nd conference on innovation in clouds, internet and networks and workshops (ICIN)*, pp. 118–123, IEEE, 2019.

[19] Y. Han, D. Guo, W. Cai, X. Wang, and V. Leung, "Virtual machine placement optimization in mobile cloud gaming through qoe-oriented resource competition," *IEEE Transactions on Cloud Computing*, 2020.

[20] Y. Deng, Y. Li, X. Tang, and W. Cai, "Server allocation for multiplayer cloud gaming," in *Proceedings of the 24th ACM international conference on Multimedia*, pp. 918–927, 2016.