Latency Issues and Solution on Cloud Gaming

MSc Research Project
Cloud Computing Group B

AKILESH PALANIVELU
Student ID: 23109831

School of Computing
National College of Ireland

Supervisor: SHREYAS SETLUR ARUN

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Akilesh Palanivelu
Student ID:	23109831
Programme:	MSc in Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Shreyas Setlur Arun
Submission Due Date:	03/01/2025
Project Title:	Latency Issues and Solution on Cloud Gaming
Word Count:	9728
Page Count:	23

Signature:	AKILESH PALANIVELU
Date:	27th January 2025

Latency Issues and Solution on Cloud Gaming

Akilesh Palanivelu

23109831

Abstract

Cloud gaming has transformed the dynamics of the gaming world and has al- lowed users to avail of high-end games on machines that would otherwise not be cap- able. However, latency still appears to be a problem at hand. This paper involves the study of latency in cloud-based gaming environments and its remediation. Since there was no VGPU server and also no edge server, one had to substitute these; thus, the VGPU server was replaced with the AWS EC2 instances, and the edge server was replaced with a local setup. A simple block HTML/JavaScript game was designed to produce metrics including the frame rate, input latency, and response time. The game executes on three different configurations, including a local server, an AWS EC2 with Flask's built in server, and an AWS EC2 instance running with a Gunicorn server. Metrics were obtained via Flask backends and rendered with Prometheus. From the test results, network latency is minimised because the AWS region selected has a geographical location close. On this view, the Gunicorn server outperforms Flask's build server with suitability towards production environments. With that in mind, recommendations shall include optimization of the servers proximity to users. Also, recommendations include employing production grade servers like Gunicorn for effective latency control. This research provides a good detail aspects and dives into reducing latency in cloud gaming, which can be practically applied to improve the user experience in similar architectures.

Keywords: Latency, Cloud Gaming, VGPU, EC2, Guniorn

1 Introduction

Cloud computing has brought with it the transformative age for gaming by changing the fundamental approaches on how games are created, published, and consumed. In such a revolution, there are cloud games that offer great computational resources on distant cloud servers to enable very realistic gaming experiences for its customers on a variety of gadgets, even those without specific hardware. Cloud gaming allows for the seamless access to resource pooling games without the need for powerful local hardware by offloading intensive processing tasks to remote servers. This model will not only focuses access to high quality gaming but also in the way for innovative business models and gaming experiences.

However, despite its promising potential, technical challenges are significant in cloud gaming, with latency being one of the most critical issue. Latency, the delay between a user's input and the corresponding response in the game, can severely degrade the gaming experience, leading to user frustration and less engagement. High latency can cause noticeable lag, affecting the responsiveness of controls, the fluidity of animations,

and the overall immersion of the player. Even slight delays in competitive and fast paced games can make all the difference between victory and defeat, hence the need for effective latency mitigation strategies that work on the go.
This paper addresses the latency issues in cloud gaming environments and presents possible solutions for improvement of performance and user experience. The study is motivated by the necessity to optimize cloud gaming infrastructures with respect to latency minimization for smooth and responsive gameplay. This work creatively replaces these specialized components, the VGPU servers and the dedicated edge servers, with an AWS EC2 instance and a local setup, respectively. In so doing, the research addresses immediate technical limitations and presents a practical approach for measuring latency in a controlled environment but also at a scale.
A key part of this research is the development and deployment of a simple HTML/JavaScript block game, designed to generate and measure such key performance metrics as frame rate, input latency, and response time. The experiment was conducted using three different server configurations—local server (playing the role of an edge server), AWS EC2 instance with Flask's built-in server as the VGPU server, and AWS EC2 instance with Gunicorn server, simulating the integration of the edge and VGPU servers. Additionally, use Flask for backend processing and Prometheus to get further depths into how analysis in visualization in these kinds of insights brings along the performance dynamics of both of these setups.

1.1 Motivation

This research is motivated by the rapidly increasing popularity of cloud gaming and the corresponding need for high performance, low latency gaming experiences. With cloud gaming services like Google Stadia, NVIDIA GeForce Now, and Microsoft xCloud increasingly becoming popular, gamers expectations for seamless, lag free gameplay become even more challenging to meet. In this highly competitive landscape, latency can be the deciding factor for a cloud gaming platform to either make it or break it. Latency has impacts on both short-term usability in the responsiveness of user inputs and long term usability in the satisfaction and retention of users. Gamers can very quickly leave a service that fails to deliver, and latency optimization is the utmost concern for service providers. It is also technically complex with the management of multiple layers involving network infrastructure, server performance, and application responsiveness. Addressing these multiple challenges requires a comprehensive understanding of the underlying factors contributing to latency and the implementation of effective mitigation strategies.
Another compelling motivation for this research is the practical limitation encountered in accessing specialized VGPU servers and edge servers within the constraints of the available AWS account. Virtual GPUs and edge computing play an important role in reducing latency, since these bring the computational resources closer to the end-users and enhance graphical processing capabilities. However, since the end users cannot use these resources, there was a need for a creative way to emulate these functionalities using available AWS EC2 instances and a local setup. This replacement did not only provide a reasonable work-around but also offered the ability to explore the impact of various alternative server configurations on latency.
In addition, designing a custom HTML/JavaScript block game as a controlled testing environment is a critical tool to generate and measure performance metrics. This custom game allows the frame rates, input latency, and response times to be monitored in precise detail

so that how the different server configurations affect these parameters can be analyzed in depth. Flask for backend processing and Prometheus for metrics visualization further complement the research with real time data collection and a comprehensive performance evaluation.

1.2 Research Question

1. Can variations in server configurations in the cloud along with integration of Gunicorn server affect the latency metrics in cloud gaming environments?

2 Related Work

2.1 Cloud Gaming Architectures

Cloud gaming has been identified as one of the emerging transformative paradigms in the gaming industry through cloud computing to deliver high fidelity gaming experiences for users across a myriad of devices (6). The processes involved in rendering and physics calculations are not processed locally on the user's device as it is done in traditional gaming. This architecture allows for access to resource-intensive games without needing powerful local hardware, democratizing high-quality gaming experiences accessibly (11). The two key parts include the game server which oversees all the game logic and rendering, as well as synchronizing the state of the game; and a client device that acts to be the primary interface in an input and output operation (**?**). The video output of the game frame is transmitted from the server to the client, who sends user inputs back in real time. Such gaming applications require a robust interaction between the client and servers with high bandwidth and minimum latency, which are also inherent in such applications by nature (7).

There are various architectural designs that have been proposed lately to optimize the performance along with scalability of cloud game services. Among the notable ones is the use of Virtual GPU (VGPU) servers, which facilitates dedicated graphical processing capabilities efficiently for rendering tasks (16). VGPU servers allow for the parallel processing of multiple game instances, thereby improving the scalability of cloud gaming platforms by supporting a large number of concurrent users without major performance degradation (17).

Along with central cloud servers, there have been studies toward integrating edge computing, such that computation resources are nearer to end-users, making it closer to latency-reduced responsiveness (10). Geographically deployed geographically near-by user clusters-based edge servers reduce round-trip time for data transfer. This is crucial in maintaining the real-time interactivity that is necessary in the case of gaming (15). Hybrid architectures which distribute computational tasks between centralized servers and edge servers, balance scalability in performance with low latency thus ensuring smooth gaming even when network conditions degrade (12).

Moreover, the recent breakthroughs of the networking technologies, which include Software-Defined Networking (SDN), are also implemented within the architecture of cloud gaming for better control of the network and smooth data flow management (2). SDN provides the dynamic real-time resource allocation of the network to increase the efficiency and reliability of data transfer from servers to clients and vice versa (5) (14).

Server type infrastructure plays the major role in deciding cloud gaming service performance. Even traditional cloud servers such as AWS EC2 instances flexible, scalable, and therefore probable to incur latency greater in nature because they happen to be centralized (5). On the other hand, edge servers reduce latency because they minimize the physical distance between the server and the user but require advanced resource management for proper handling of distributed workloads (2). Hybrid architectures of central cloud servers and the edge servers are trying to gain the advantages of these models, such as providing a low latency while offering scalability (12).

CDNs in cloud architectures for gaming support game content distribution through data caching near the user (18). It reduces loads on centralized servers and shortens data transmission times. Thus, the latency associated with the same is lowered, and users are assured of a good experience. CDNs and edge computing, along with VGPU servers, comprise a solid architecture that promises high-quality, low-latency gaming experience for all global users (16).

2.2 Latency in Cloud Gaming

Latency is considered a critical parameter for any cloud-based platform. As defined, latency represents a delay between the time at which a user inputs and a corresponding action reflected in the game (7). Generally, high latency creates perceivable lag, that deteriorates the fluidity of gameplay and results in annoying and decreased satisfaction among the end-users (9). Latency in cloud gaming is that latency can have a lot of effect on the playability and fairness of gaming, so reducing latency remains a primary concern for most cloud gaming providers (3).

Multiple studies have measured and evaluated the causes of latency over cloud gaming environments. Choy et al. (9) had carried out a measurement study of comprehensive latency contributions due to network transmission delay, server processing time, and delays introduced at the client's rendering. The overall goal for all these was a general reduction in latency, while each component is kept minimal. Chen et al. (7) proposed methods that accurately measured the latency in the cloud gaming system; valuable in- sights into bottlenecks in performance, hence needs to be addressed, came out of such researches.

Due to the reliance of cloud gaming on the real-time data transmission that takes place between the client and the server, it is significantly affected by latency (5). Other resources that are bandwidth limitation, network congestion, or even physical distance between the user and server will degrade the latency problem (4). For these challenges, latency hiding techniques include predictive input algorithms and frame buffering to bridge network-induced

delays (3). In addition, optimization of routing of data packets along with efficient compression algorithms will also decrease the time required to send over the network (20).
Another critical performance metric in cloud gaming is frame rate, impacted by latency. Generally, a high frame rate implies smoother animation and more responsive controls, which means the user will have a better experience in general (1). However, across the entire pipeline in cloud gaming, from input processing to rendering and transmission, minimizing latency leads to reaching a higher frame rate (18). Hence, in order for this frame rate to be smooth and realistic in the context of the cloud gaming experience, addressing latency becomes necessary not just for frame rate but seamless and immersive (14).
In addition, latency affects the synchronization of several players in multiplayer gaming environments. Deng et al. (12) studied the server allocation problem for multiplayer cloud gaming, focusing on the necessity of efficient resource provisioning to maintain low latency during peak usage times. Their study showed that dynamic server allocation based on real-time demand could reduce latency by a large margin, thus improving the multiplayer gaming experience.
Apart from network and server side, client-side optimizations take a part in the game of latency management. The study by K¨ama¨ra¨inen et al. (17) focused on measurement on imperceptible latency in mobile cloud gaming and the optimization of the client-side rendering and input processing for the minimum perception delay. The conclusion made on this study was that latency in cloud gaming environment has to be reduced through both a holistic approach in both sides of the server-side as well as client-side optimizations. In addition, the server's geographical distribution affects latency since the physical distance among users and the server may differ. Baldovino (4) had presented an overview of network issues in cloud gaming focusing on the importance of selection of geographically optimal location for servers to reduce latency. By placing servers in proximity to the user base, cloud gaming providers can minimize latency significantly, thereby improving the frame rate and responsiveness during gameplay (10).

2.3 Server Configurations and Their Impact on Latency

The configuration of the servers in a cloud gaming setup has a good sized effect on many latency metrics, consisting of the frame rate, input latency, and reaction time. Traditional server configurations, including those using built-in web servers like Flask's, have been well researched as far as their performance for handling concurrent connections and large traffic loads are concerned (7). Flask's development server is good for development and testing but not for production because it introduces latency under heavy workloads
(8).
To overcome these limitations, production-grade web servers such as Gunicorn have been used to improve the performance and reliability of cloud gaming backends (10). Gunicorn is designed to allow for multiple simultaneous connections as it uses a pre-fork worker model. Such a model allows Gunicorn to handle higher volumes with reduced latency (15). Comparative studies have shown Gunicorn to be faster compared to Flask's built-in server in terms of response times and scalability, thus well-suited for cloud applications like gaming (17).
In addition to server software, server infrastructure—whether local, edge, or cloud based—plays a crucial role in determining latency outcomes (2). Local servers, as edge servers, have

an advantage of proximity to the users, thus reducing the network transmis- sion delay (10). Still, they are limited by limited computational resources compared to those in cloud-based servers. On the other side, cloud EC2 instances, especially VGPU equipped ones, are known for robust computational power but induce higher latency because of being centralized (5). Hybrid configurations, with both edge and VGPU servers, attempt to strike a balance between both sets of configurations and reduce latency by balancing local processing against centralized computational resources (12). For instance, having edge servers for doing local processing of immediate input by the user while making rendering-intensive jobs go off to VGPU-equipped servers at the cloud reduces total latency (14). The divi- sion of labor should therefore ensure that tasks needing minimal latency are performed in edge servers, and by making resource-intensive tasks performed in powerful cloud servers it helps in increasing both response times and performance.

Empirical studies have always pointed out the effect of server configurations on latency metrics. Alam et al. (1) showed that optimizing server placement and using advanced server software can result in significant improvements in frame rate and response time. Their study revealed that placing servers closer to user clusters and using scalable server software such as Gunicorn can significantly reduce latency, thus improving the gaming experience.

Franco et al. (15) claimed that edge computing is beneficial for cloud gaming in the sense that it provides reliability, timeliness, and load reduction at the edge. In their study, they concluded that edge servers not only reduce latency as they are closer to users geographically but also divide the computation load so no single server becomes a bottleneck (15). This would mean that even during the peak usage period, the latency is kept low to ensure constant performance across the gaming platform.

Moreover, Gharsallaoui et al. (16) compared various cloud gaming systems, and the authors pointed out the differences in latency performance between different server set- tings. The outcomes proved that hybrid architectures were better in minimizing latency through the use of both edge and cloud servers instead of a purely centralized or purely distributed configuration (16). This fits in well with the overall generalization in the literature that hybrid configurations are best where latency reduction meets scalability with computations.

Server configurations even matter in multiplayer cloud games. Deng et al. (12) studied a few strategies in server allocation regarding multiplayer sessions. In terms of such an experiment, their analysis indicates that appropriate resource provisioning along with the dynamic server allocation is the necessity to have latency low, and frames should be displayed with higher frame rates. Their results also suggested that demand-based allocation of dedicated servers to multiplayer sessions in real time could prevent server overload, thus reducing latency spikes as well (12).

In addition, integration of SDN into server configuration has shown the potential for low latency. Amiri et al. (2) suggested a game-aware network management based on SDN, which adjusts network resources in real-time according to the patterns of gaming traffic. Thus, latency-sensitive gaming data gets priority over less important traffic, which keeps the latency low and frame rates high (2).

2.4 Strategies for Latency Reduction and Performance Optimization

Reducing latency and optimizing back-end performance is one of the important needs for achieving high quality of user experience for cloud gaming environments. Given that latency is a multilateral factor in cloud gaming, what is needed would be network optimization strategies that also focus on enhancing the server infrastructure and application levels (6). This chapter discusses strategies planned and implemented to counteract latency for improving the performance of cloud gaming.

2.4.1 Network Optimization

One of the techniques to reduce latency over the network is optimization of where to place the servers geographically. By placing the servers within the geographic regions where most of the users exist, the physical distance that data has to traverse is minimized hence reducing delay associated with transmission (10). For instance, cloud regions like Mumbai have been touted to reduce latency by a considerable margin over a farther region of operation while serving users in Asia (4). Furthermore, through Content Delivery Networks, delivering content becomes more efficient, and data is served from the nearest possible location (18).

2.4.2 Server Infrastructure Optimizations

Server infrastructure optimization is required for reducing latency and for the perform- ance of the backend. Moving away from development-focused servers, such as Flask's in- built server, to production-grade servers like Gunicorn can significantly improve backend performance and reduce response times (7). Because of Gunicorn's model of worker, mul- tiple concurrent connections are supported, thus traffic loads are not allowed to increase latency (17). Additionally, the optimal load balancing method would balance the work- load efficiently among several servers in order not to allow one server alone to become a bottleneck, especially during heavy traffic loads (2).
Hybrid server configurations combining edge and cloud servers are very well suited for balancing latency with computational scalability. Immediate user inputs can be dealt with by deploying edge servers, while resource-intensive rendering tasks can be delegated to VGPU-equipped cloud servers. This decreases overall latency by a considerable mar- gin while maintaining frame rates (14). This makes sure that latency-sensitive operations are locally managed, and resource-intensive tasks are managed by powerful centralized servers (12).

2.4.3 Application-Level Optimizations

Other than these network and server improvements, it is actually application-level op- timizations that do most of the work to minimize latency. Other techniques, including predictive input algorithms and frame buffering, can eventually neutralize any network delay that may be inevitable, providing smoother gameplay at high latencies (3). Pre- dictive input algorithms operate based on the principle of predicting user action from historical input data and allow the client to pre-render those anticipated frames (19). This approach reduces the perceived latency by handling the user inputs preemptively, thus making the game more responsive (3).

Table 1: Critical Analysis of Cloud Gaming Literature

Study	Key Findings	Limitations
Cai et al. (6)	Comprehensive survey on cloud gaming architectures and future directions, emphasizing VGPU and edge computing benefits.	Limited empirical evaluation of hybrid architectures.
Choy et al. (10)	Proposed hybrid edge-cloud architecture to minimize latency, showing significant performance improvements.	Lacked a detailed analysis of cost im- plications.

Chen et al. (7)	Developed methodologies for accurately measuring latency in cloud gaming systems.	Focused primarily on centralized architectures without considering hybrid solutions.
Anand and Wenren (3)	Proposed latency-hiding techniques, including predictive input algorithms, for thin-client cloud gaming.	Implementation challenges in dynamic gaming scenarios were not addressed.
Amiri et al. (2)	Introduced SDN-based game-aware network management to optimize net- work resource allocation.	Did not address latency variability in large-scale multiplayer gaming.
Baldovino (4)	Highlighted networking issues in cloud gaming and the role of geographic server placement.	Did not explore advanced server software optimizations.
Deng et al. (14)	Explored deep reinforcement learning for dynamic resource allocation, demonstrating reduced latency.	Applicability to real-time multiplayer gaming requires further validation.
Franco et al. (15)	Studied reliability and load reduction at the edge, showing lower latency with edge servers.	Limited scalability analysis for global gaming platforms.

3 Methodology

This paper seeks to evaluate and address latency issues which are inherent to cloud-based gaming environments. Latency refers to the delay of time between the input given by a user and its corresponding reaction on the screen, which can affect the overall user experience of a cloud-based gaming environment. An integrated experimental approach has been applied in analyzing the latency across various configurations of servers, and consequently, it also proposes a viable solution for improving the performance. The methodology encompasses design and development of a simple block-based game, various server environments setup, implementing mechanisms for metrics collection, and analysis of the collected data to derive meaningful insights.

3.1 Experimental Design and Setup

For scientifically testing latency in cloud games, a controlled experimental structure is developed. The backbone of the experiment is formed as a block-based relatively simple game made using HTML and JavaScript. This thus presents a constant and reproducible testing ground for a configuration of any server whereas does not get bogged by the complexity of its mechanics while trying to evaluate the latency.

The game is meant to respond to the arrow keys entered by the user and get the red block to traverse the screen. The responsiveness between the block movement and a user's keystroke, known as input latency, is measured here. Further, it logs the frame rate in terms of frames per second, or FPS, and the response time, both crucial determinants of game performance and quality of the playing experience. The index.html file, representing the game's HTML, is deployed in three different environments. Each environment simulates different server configurations: a local set-up acting as an edge server, a cloud-based EC2 instance running Flask's built-in server, and another cloud-based EC2 instance using the Gunicorn server.

3.2 Server Configuration and Deployment

Due to technical constraints in support of AWS accounts, the initially proposed VGPU server and edge server are replaced by more accessible alternatives. Instead of a VGPU server, a standard cloud EC2 instance is used; edge server functionality is instead replicated using a local setup. This pragmatic approach allows the research to remain feasible while still providing valuable insights into latency issues in cloud gaming.

Each server environment is meticulously configured to host the game and manage metric collection. The local setup involves running a Python HTTP server using the command python -m http.server 8000, which serves the index.html file. Concurrently, the Flask backend is deployed by setting the FLASK APP environment variable to app.py and executing flask run. This configuration allows for the collection and handling of metrics via Flask's endpoints.

For the cloud-based setup configurations, there are two that are made. The first employs Flask's built-in server on an Amazon EC2 instance using Amazon Linux 2023. The mirror of a local setup operating in the cloud serves to give grounds for the comparison of locality and the latency of running in a cloud environment. The second makes use of Gunicorn: a WSGI HTTP server implemented for production and replaces the built-in Flask server. This replacement tries to see whether a more powerful server would be able to decrease latency and make better performance metrics as compared to Flask's default server.

3.3 Game Development and Metrics Collection

The game is very simple in its development, using HTML and JavaScript to create an interactive environment where a red block can be moved using arrow keys. The game's simplicity is deliberate, focusing on essential interactions that are crucial for latency measurement. The JavaScript code is embedded within the index.html file and is con- sistent across all three environments, ensuring that any differences in performance are attributable to the server configurations rather than the game code itself.

The study takes the metrics collection, frame rate, input latency, and response time. The number of frames rendered per second is the frame rate that determines how smooth and well-performing the game will be. The input latency is determined by recording the time between the user's action of pressing an arrow key and the movement of the block on the screen. It is the time elapsed between when the server has received the input and its processing and then its appearance in the game.

All these metrics are recorded with the help of the browser's performance.now() function, which offers high-resolution timestamps for measuring. They are then forwarded to the backend server with HTTP POST requests. This Flask application processes and submits these metrics to Prometheus-an open-source monitoring and alerting toolkit. It then allows it to scrape them at a predetermined interval and continuously monitor the game performance.

3.4 Backend Implementation with Flask and Prometheus

The backend infrastructure is implemented with Flask, a lightweight web framework for Python, in combination with Prometheus for metrics collection and monitoring. App.py

defines endpoints for receiving and exposing metrics. It has a POST endpoint /metrics that receives JSON-formatted metrics from the game, updating Prometheus gauges for frame rate, input latency, and response time. It has a GET endpoint at /metrics. That endpoint exposes metrics in the form that's consumable by Prometheus to do live monitoring and analysis.
For each server setup, Prometheus has a prometheus.yml configuration file with scrape in- tervals and targets. For the local setup, Prometheus is configured to scrape metrics from localhost:5000, the location of the Flask application. For the cloud setups, Prometheus is configured to scrape metrics from the appropriate public or private IP address of the EC2 instance. This way, one can be sure that one has metrics collected consistently in all the server setups so that a robust dataset would be used for comparison purposes. With more installation and configuration steps on the cloud setup using Gunicorn, install and configure Prometheus on Amazon Linux 2023. The stated version of Prometheus compatible with the operating system is downloaded and extracted for configuration with a predefined configuration. With its deployment on the EC2 instance designed to handle loads and built for performance superior to the built-in Flask server, this will also help potentially reduce latency as well as improve the speed of the response time.

3.5 Data Collection and Analysis

The data is gathered by playing the game in all the three environments: the local setup, the cloud EC2 instance with Flask's built-in server, and the cloud EC2 instance with Gunicorn. Prometheus periodically scrapes the metrics. In this case, Prometheus gathered the frame rate, the input latency, and response time for every setup, which are plotted via the dashboard of Prometheus in such a way that it facilitates the user to intuitively compare their performance across the setups.
Data collection undergoes a detailed analysis to identify pattern and discrepancy of latency as well as frame rates. Statistical methods are used comparing the performance metrics between setups of local and cloud setup as well as between servers of Flask and Gunicorn in the cloud environment so that the degree of server configuration and deployment environment, which influence latency and game performance, can be identified.

4 Design Specification

4.1 Introduction

This chapter describes the design and specification of the system implemented to overcome latency in cloud gaming. Because of the unavailability of VGPU servers and edge servers on AWS, the proposed architecture has been modified using alternate resources: the EC2 instance is used as a substitute for the VGPU server, and a local setup is used to simulate the edge server. This chapter describes the design principles, system com- ponents, and the implemented architecture.

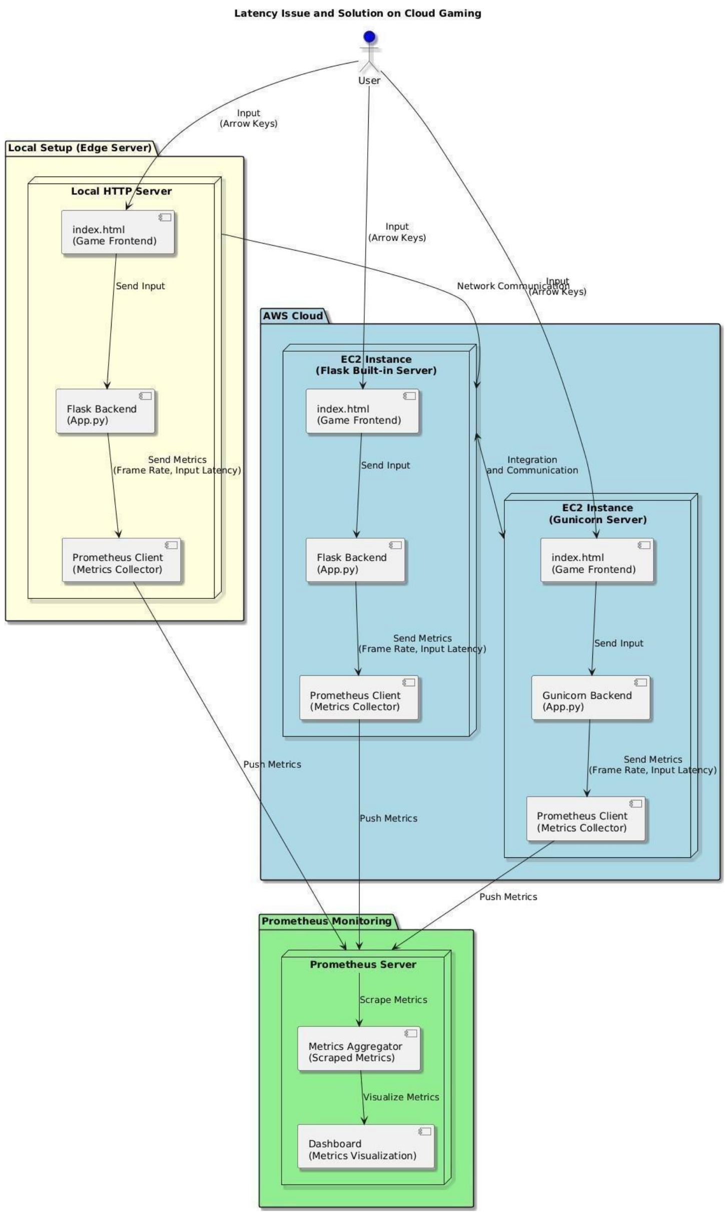
4.2 System Architecture

The architecture consists of three environments:

1. **Local Setup:** Acts as the edge server.

2. **Cloud EC2 Instance with Flask Server:** Acts as the VGPU server.
3. **Cloud EC2 Instance with Gunicorn Server:** Integrates the edge server and VGPU server for optimized performance.

4.2.1 Architecture Diagram



Latency Issue and Solution on Cloud Gaming
User
Input (Arrow Keys)
Local Setup (Edge Server)
Local HTTP Server
index.html (Game Frontend)
Send Input
Flask Backend (App.py)
Send Metrics (Frame Rate, Input Latency)
Prometheus Client (Metrics Collector)
Input (Arrow Keys)
Network Communication
Input (Arrow Keys)
AWS Cloud
EC2 Instance (Flask Built-in Server)
index.html (Game Frontend)
Send Input
Flask Backend (App.py)
Send Metrics (Frame Rate, Input Latency)
Prometheus Client (Metrics Collector)
Integration and Communication
EC2 Instance (Gunicorn Server)
index.html (Game Frontend)
Send Input
Gunicorn Backend (App.py)
Send Metrics (Frame Rate, Input Latency)
Prometheus Client (Metrics Collector)
Push Metrics
Push Metrics
Push Metrics
Prometheus Monitoring
Prometheus Server
Scrape Metrics
Metrics Aggregator (Scraped Metrics)
Visualize Metrics
Dashboard (Metrics Visualization)

Figure 1: System Architecture

4.2.2 Purpose of Multiple Environments

The design aims to compare performance metrics across these setups:

- Local setup represents the edge server.
- EC2 with Flask Server simulates a standalone VGPU server.

- EC2 with Gunicorn Server demonstrates integration and performance improvement possibilities.

4.3 System Design

4.3.1 Game Development

A simple block game was created using the following technologies:

- HTML
- JavaScript

The game features a movable block controlled by arrow keys. It measures performance metrics such as frame rates, input latency, and response time.

4.3.2 Backend Design

The backend utilizes Flask for metric collection and Prometheus for visualization. Flask receives and exposes game metrics, while Prometheus queries and visualizes these metrics for analysis.

4.4 Specification of Components

4.4.1 Frontend Design

The game interface consists of a block moving within a canvas. The following features are implemented:

- Canvas for rendering the game.
- Keyboard inputs for block movement.
- Metrics calculation (frame rate, input latency, etc.).

4.4.2 Backend Implementation

The backend collects metrics such as:

- **Frame Rate (FPS):** Frames rendered per second.
- **Input Latency:** Time between input and visual response.
- **Response Time:** Time taken for backend responses.

4.4.3 Prometheus Setup

Prometheus is configured to scrape metrics from:

- **Flask Server:** Local and cloud setups.
- **Prometheus Instance:** Hosted locally or on the cloud EC2 instance.

4.4.4 Cloud EC2 Configurations

- **Local Setup:** http://127.0.0.1:8000 for the game, http://127.0.0.1:5000/ metrics for Flask metrics.
- **Flask Server on EC2:** http://3.110.119.8 for the game, http://3.110.119. 8:5000/metrics for metrics.
- **Gunicorn Server on EC2:** http://65.1.130.2:8000 for the game, http://65. 1.130.2:5000/metrics for metrics.

4.5 Key Design Considerations

4.5.1 Challenges Addressed

- Network latency due to geographic distances.
- Backend optimization for better response time.

5 Implementation

It had implementation towards researching into latency issues in cloud gaming using in-frastructure with metrics capture, analysis, and performance optimization. The process included setting up a simple yet functioning gaming application, deploying this application across three different environments, and then integrating it with a backend system and Prometheus in the capture and visualization of its key metrics. The three environments were designed to represent an edge server, a VGPU server, and their integrated configuration, respectively. Three different setups were made-one using a local setup, one using a cloud EC2 instance with Flask's built-in server, and the other using another EC2 instance using Gunicorn. The primary goal of these setups was to analyze the effects of the infrastructure choices on metrics like frame rate, input latency, and response time. This meant a straightforward block-moves game served as a key game type built out through HTML and JavaScript that any Web user, regardless of technology, would be capable of running on their equipment. The graphics were simply output through the HTML5 canvas feature, giving a crisp seamless effect due to the smooth motions made in it. In this game, a red block could be dragged across the canvas using arrow keys, with each dragging an opportunity to measure and record key performance indicators. Several JavaScript functions were implemented here to calculate the frame rate by measuring the time that has elapsed between frames and find input latency by measuring the delay of the keypress event relative to the corresponding movement on the screen. Response time metrics were generated using a mock delay to simulate server responses.

A Flask-based backend was developed to collect and process these metrics. The developed backend was necessary to capture the performance data that was being generated by the game and expose it to Prometheus for monitoring. The Flask application defined two main endpoints: one for receiving metrics via POST requests and another for exposing them in a format compatible with Prometheus via GET requests. The POST endpoint processed JSON-formatted data from the game, which included frame rate, input latency, and response time values. Upon receiving this data, the backend updated corresponding Prometheus gauges—

data structures designed for real-time tracking of numerical metrics. These metrics would get scraped by Prometheus at regular intervals to enable visualization and analysis. Being lightweight and easy to use, Flask was an appropriate choice for developing this backend system.

Prometheus also had a significant role in implementation in providing a platform for monitoring and visualizing the performance of the system. With integration, real-time analysis of all collected metrics was made available in terms of insight provided to the behavior of a system under different conditions. The local setup of it involves installation on a Windows machine. The configuration file that should be updated to point towards including the Flask backend is named prometheus.yml. This enabled Prometheus to peri- odically update data from the Flask application, allowing it to use that data for querying and graphing. The Prometheus web interface had graphing utilities for frame rate, input latency, and response time, which were used to score each environment on its ability to impact latency.

Deploying across the three environments made the differences in performance between the local setup and the two EC2 instances-the one with Flask's built-in server and the one with Gunicorn-clear. The local setup, that is, the edge server showed the least latency because it was deployed closer to the client, while the EC2 instance with Flask's built-in server showed more latency since the network overhead associated with cloud-based com- munication was its reason. The EC2 instance running Gunicorn, however had significant improvements in response times and frame rates compared to the Flask-built-in server setup, which proved correct the decision to use a production-grade web server. The deployment process involved running the game application on an HTTP server and setting up the Flask backend to handle metric submissions. On the local setup, Python http.server module was used to server the game on port 8000, while launching the Flask application using command flask run. For EC2 instances, the same code for the game application and backend are deployed, with Gunicorn replacing Flask's built in server in the third environment to improve performance. In all setups, Prometheus was configured to scrape metrics from the corresponding backend endpoints, thus collecting data in a uniform manner across all environments.

6 Evaluation

6.1 Introduction

The findings from this experiment implementation give a breakdown of how the system behaves in various configurations. Frame rate, input latency, and response time are among the metrics that were collected using Prometheus, which will give the real-time idea of how the system had behaved on a local machine. These were the metrics from which ideas were drawn into understanding whether the architecture had been efficient in eliminating or reducing latencies. The results shown here start with the performance of the local environment as seen in the graphs provided and lay a basis for comparison with other configurations such as the use of Flask and Gunicorn servers with EC2 instances.

6.2 Results of Local Setup

6.2.1 Frame Rate

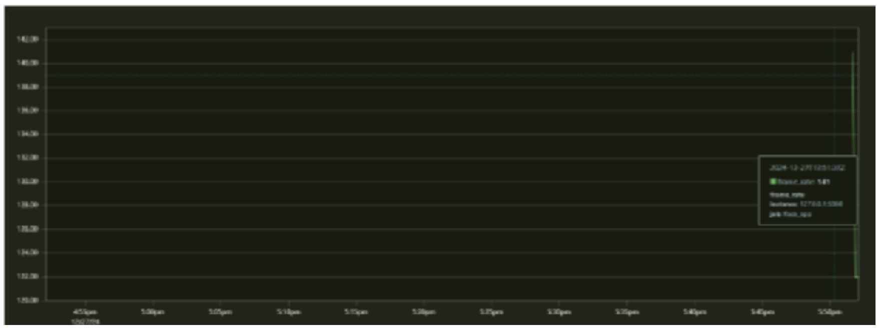


Figure 2: Frame rate observed in the local setup.

The frame rate in the local setup was highly consistent, as depicted by the graph below. The system achieved an average frame rate of 141 FPS, which shows that the local envir- onment is capable of a smooth gaming experience. Throughout the observed period, no significant frame drops were encountered, which indicates the excellent performance of the local setup in handling graphical rendering and game mechanics. This high perform- ance can be credited to the proximity of the client and server, where network-induced delays are also eliminated and interaction is completely seamless.

6.2.2 Input Latency

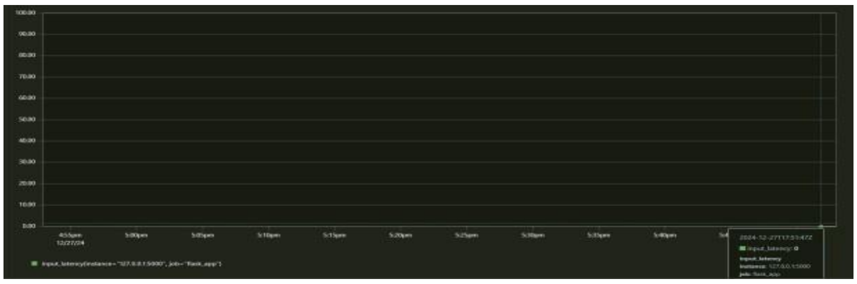


Figure 3: Input latency observed in the local setup.

The graph on input latency is exceptional as the input latency is recorded at 0 milliseconds during the whole monitoring period. This output verifies that the local configuration gives instant feedback to user inputs, which is fundamental in gaming environments. Lack of input delay also confirms the appropriateness of using the local environment as a benchmark to assess performance in more complex configurations, for example, cloud- based configurations.

6.2.3 Response Time

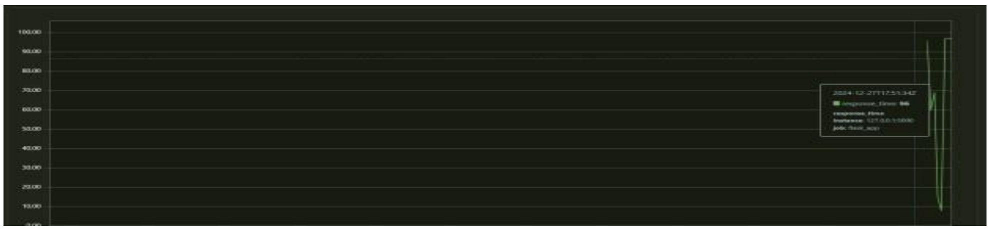


Figure 4: Response time observed in the local setup.

The response time recorded in the local setup, as shown in the graph, hovered around 96 milliseconds. This value is indicative of the time taken for the system to process requests and respond, including the time required for backend computations and interactions between the game and the Flask application. While slightly higher than expected for a local setup, this response time still reflects a high level of performance and responsiveness, suitable for a latency-sensitive application like gaming.

6.3 EC2 Instance with Flask Built-in Server Results

6.3.1 Frame Rate

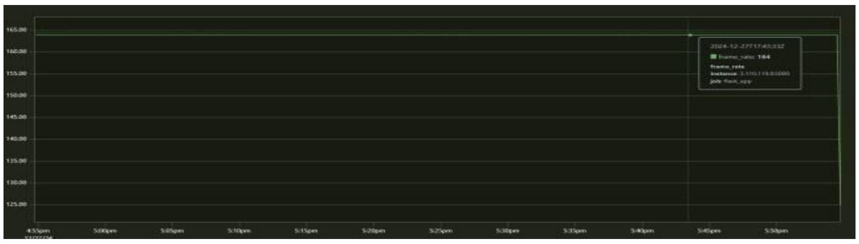


Figure 5: Frame rate observed in the EC2 Flask built-in server setup.

In the EC2 instance with Flask's built-in server, the frame rate was recorded at an average of 164 frames per second (FPS), as depicted in the graph. This performance is slightly higher than the frame rate achieved in the local setup, which recorded 141 FPS. This increased frame rate would imply that the cloud infrastructure supported by the server's processing power was the cause for a smooth gaming experience. However, this improvement may also be due to reduced graphical load or other optimization factors inherent to the EC2 environment.

6.3.2 Input Latency

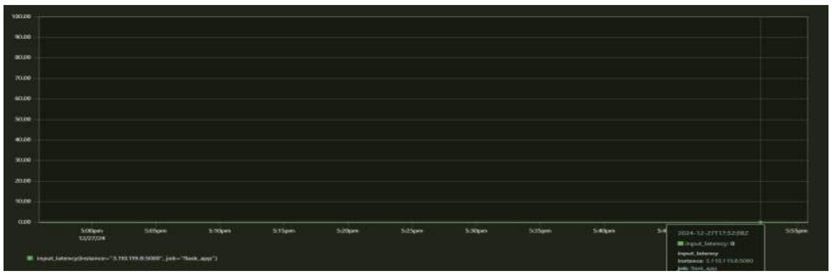


Figure 6: Input latency observed in the EC2 Flask built-in server setup.

The input latency for the EC2 Flask setup was consistently low, and during the mon- itoring period, it was recorded at 0 milliseconds. This indicates that user inputs were processed almost instantaneously, much like the results observed in the local setup. The efficiency with which the Flask server could handle input events without delay proves its capability to offer responsive interactions in a cloud-hosted environment.

6.3.3 Response Time

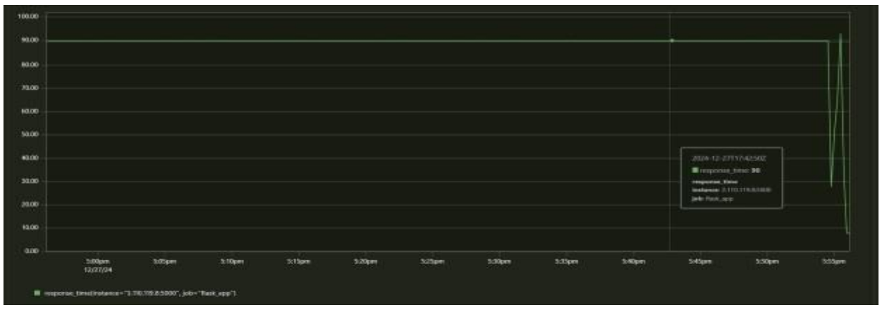


Figure 7: Response time observed in the EC2 Flask built-in server setup.

The response time for the EC2 Flask setup averaged 90 ms, slightly better than 96 ms recorded in the local setup. This should be due to the improved processing of the backend based on the EC2 instance that supports optimized backend processing abilities. Despite being hosted by a cloud server, response times remained competitive, such that latency-sensitive gaming was quite close to seamless.

6.4 EC2 Instance with Gunicorn Results

6.4.1 Frame Rate

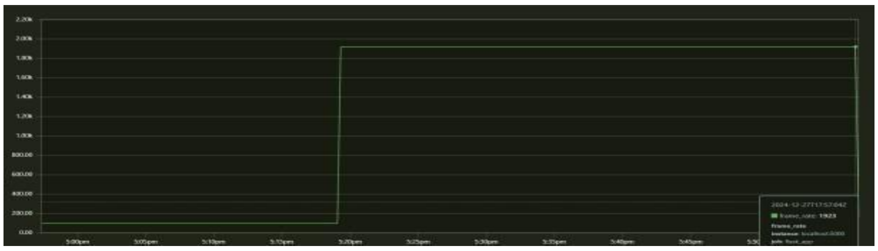


Figure 8: Frame rate observed in the EC2 Gunicorn server setup.

The frame rate recorded in the Gunicorn setup was impressive at an average of 1923 frames per second, as shown in the graph. This is a great improvement from the local setup, which averaged 141 FPS, and the EC2 Flask setup, which averaged 164 FPS. Such a high increase in frame rate shows that Gunicorn is able to handle rendering requests very efficiently and reduces overheads, making the game run smoothly even in a cloud- hosted environment. This further implies that Gunicorn is making an effective use of the computation capabilities of the EC2 instance for performance optimization.

6.4.2 Input Latency

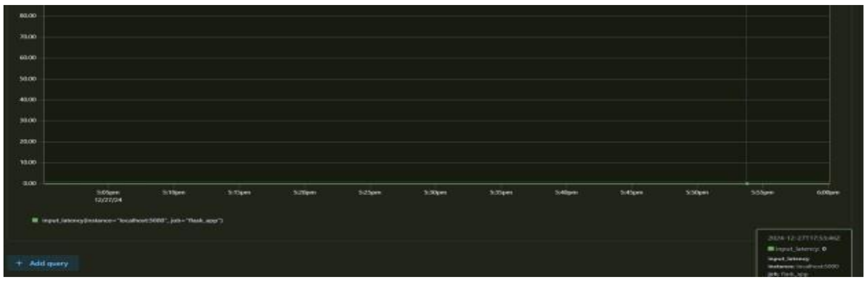


Figure 9: Input latency observed in the EC2 Gunicorn server setup.

The input latency within the Gunicorn setup was recorded to be 0 milliseconds throughout the monitoring period and was consistent with what had been observed in both local and EC2 Flask setups. This shows that the Gunicorn server is fast enough to process user inputs instantaneously, ensuring an extremely responsive gaming experience. The consistency across the setups indicates the robustness of the input-handling mechanism implemented in the gaming infrastructure.

6.4.3 Response Time

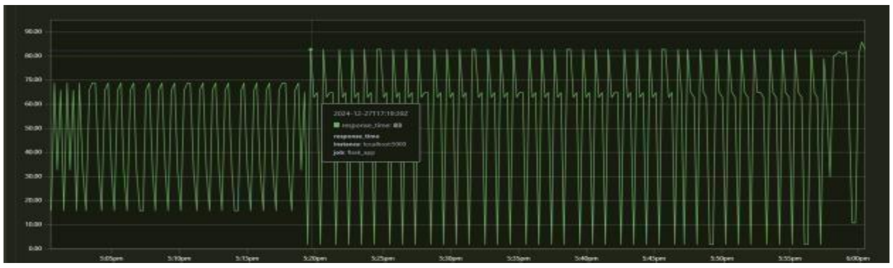


Figure 10: Response time observed in the EC2 Gunicorn server setup.

The average response time of the Gunicorn setup is 83 milliseconds, thus better than the local setup (96 ms) and the EC2 Flask setup (90 ms). It can be seen from the graph that the response time is almost stable with minor fluctuations from time to time, mainly due to network latency fluctuations or server load. So, the reduced response time shows that Gunicorn makes backend computations more efficient while handling them, which gives quicker interactions between the frontend game and the backend services.

6.5 Analysis

Thus, the performance in the local, EC2 Flask and EC2 Gunicorn will show the effect of how different infrastructure and server configures the performance. Under the local setup, then indeed an excellent performance in terms of frame rate, input latency, and response time appeared. The gameplay was smoother with almost no delay there. The other hand, the cloud-based EC2 Flask improved on response time due to the optimization of backend processing power by the cloud instance. Though the frame rate increased in the EC2 Flask setup, the greatest performance boost was actually noticed when switching to Gunicorn.

Gunicorn is a production-grade server, which greatly improved performance by increasing the frame rate and reducing response time. This configuration also outperformed both the local and EC2 Flask setups in terms of its superiority in handling computational loads and backend processing. Moreover, input latency was always at low levels for all setups, which means that the responsiveness of the user experience was not impacted by the server setup.

Although the local baseline is fantastic, cloud configuration especially Gunicorn provides enough performance gains for cloud applications related to gaming. Moreover, it makes Gunicorn an excellent choice for serving environments when frame rate and reaction time matter most in smooth game streaming. The key findings underpin the appropriateness of the need to make the right choice by using a proper server infrastructure supporting specific requirements for cloud game applications.

7 Conclusion and Future Work

This research focuses on latency in cloud gaming. Given the constraint on accessing VGPU and edge servers on AWS, alternative architectural setups are considered. The VGPU server is replaced with a cloud EC2 instance, and the edge server is replaced with a local setup for the purpose of analyzing the performance metrics of frame rate, input latency, and response time. Three configurations are analyzed: local setup, EC2 with Flask's built-in server, and EC2 with a Gunicorn server.

The local configuration served as an edge server establishing a baseline with high frame rates, zero input latency, and 96 ms of response time, setting up the efficiency of proximity based systems. A simple block moving game used metrics with technologies as simple as HTML, JavaScript, and a Flask backend. This setup is effective for reducing latency but not very scalable to scale up to larger deployments.

The EC2 setup with Flask's server had slightly better frame rates and response times. Its response time was decreased to 90ms, while zero input latency was maintained. However, concurrency limitations and scalability in Flask make more potent solutions necessary. This setup demonstrated the trade-offs involved in cloud scalability versus proximity in managing latency. The most important change came in the Gunicorn EC2 setup, where the WSGI server replaced Flask's built-in server. It achieved 1923 FPS, reduced response time to 83ms, and had zero input latency. This proves that the optimized backend solution is of great importance for latency-sensitive applications like cloud gaming. Gunicorn has been able to perform quite well with concurrent requests and is highly efficient; thus, it is the most suitable option for production-level gaming infrastructures. This study points out how architectural decisions affect latency, balancing cloud scalability with local performance to meet gaming demands.

7.1 Future Work

Although this research was able to identify and implement solutions to mitigate latency in cloud gaming, there are still several avenues for future work. First, the study could be expanded to include real-world gaming scenarios with more complex graphics and higher computational demands. The block moving game used in this research, although suitable for testing latency, lacks the intricacy of commercial games that involve high-definition rendering and advanced physics engines. Future research should test proposed architectures with such games for their scalability and robustness.

Further exploration should be made on integrating edge computing with cloud-based servers. The local setup in this study acted as an edge server, but real-world edge servers are closer to the user in geographically distributed regions, which may significantly reduce latency. This hybrid architecture will integrate edge servers with Gunicorn run cloud instances and combine the benefits of both approaches while ensuring minimal latency and maximum scalability.

The study can be expanded by including Virtual GPU capabilities when the required infrastructure becomes available. VGPU is designed specifically for gaming and graphical applications, and inclusion in the current architecture may result in even higher frame rates and response times. Comparing the performance of VGPU-based systems with the results obtained in this study will give a better insight into the optimal configuration for cloud gaming.

References

[1] Alam, N., Raazi, S.M.K.U.R., Mehboob, B. and Ali, S.M., 2024. Improving Cloud Streamed Gaming Quality Using Latency Compensating Gameplay Enhancements. Pakistan Journal of Engineering Technology and Science (PJETS), 12(01), pp.125- 138.

[2] Amiri, M., Al Osman, H., Shirmohammadi, S. and Abdallah, M., 2015, December. SDN-based game-aware network management for cloud gaming. In 2015 Interna- tional Workshop on Network and Systems Support for Games (NetGames) (pp. 1-6). IEEE.

[3] Anand, B. and Wenren, P., 2017, October. CloudHide: Towards latency hiding tech- niques for thin-client cloud gaming. In Proceedings of the on Thematic Workshops of ACM Multimedia 2017 (pp. 144-152).

[4] Baldovino, B., 2022. An Overview of the Networking Issues of Cloud Gaming. Journal of Innovation Information Technology and Application (JINITA), 4(2), pp.120-132.

[5] Basiri, M. and Rasoolzadegan, A., 2016. Delay-aware resource provisioning for cost-efficient cloud gaming. IEEE Transactions on Circuits and Systems for Video Tech- nology, 28(4), pp.972-983.

[6] Cai, W., Shea, R., Huang, C.Y., Chen, K.T., Liu, J., Leung, V.C. and Hsu, C.H., 2016. A survey on cloud gaming: Future of computer games. IEEE Access, 4, pp.7605-7620.

[7] Chen, K.T., Chang, Y.C., Tseng, P.H., Huang, C.Y. and Lei, C.L., 2011, November. Measuring the latency of cloud gaming systems. In Proceedings of the 19th ACM international conference on Multimedia (pp. 1269-1272).

[8] Chen, K.T., Huang, C.Y. and Hsu, C.H., 2014, July. Cloud gaming onward: Research opportunities and outlook. In 2014 IEEE International Conference on Multimedia and Expo Workshops (ICMEW) (pp. 1-4). IEEE.

[9] Choy, S., Wong, B., Simon, G. and Rosenberg, C., 2012, November. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In 2012 11th Annual Workshop on Network and Systems Support for Games (NetGames) (pp. 1-6). IEEE.

[10] Choy, S., Wong, B., Simon, G. and Rosenberg, C., 2014. A hybrid edge-cloud archi- tecture for reducing on-demand gaming latency. Multimedia systems, 20, pp.503-519.

[11] Chuah, S.P., Yuen, C. and Cheung, N.M., 2014. Cloud gaming: A green solution to massive multiplayer online games. IEEE Wireless Communications, 21(4), pp.78-87.

[12] Deng, Y., Li, Y., Tang, X. and Cai, W., 2016, October. Server allocation for mul- tiplayer cloud gaming. In Proceedings of the 24th ACM international conference on Multimedia (pp. 918-927).

[13] Deng, Y., Li, Y., Seet, R., Tang, X. and Cai, W., 2017. The server allocation prob- lem for session-based multiplayer cloud gaming. IEEE Transactions on Multimedia, 20(5), pp.1233-1245.

[14] Deng, X., Zhang, J., Zhang, H. and Jiang, P., 2022. Deep-reinforcement-learning- based resource allocation for cloud gaming via edge computing. IEEE Internet of Things Journal, 10(6), pp.5364-5377.

[15] Franco, A., Fitzgerald, E., Landfeldt, B. and Ko¨mer, U., 2019, October. Reliability, timeliness and load reduction at the edge for cloud gaming. In 2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC) (pp. 1-8). IEEE.

[16] Gharsallaoui, R., Hamdi, M. and Kim, T.H., 2014, December. A Comparative Study on Cloud Gaming Platforms. In 2014 7th International Conference on Control and Automation (pp. 28-32). IEEE.

[17] Ka¨ma¨r¨ainen, T., Siekkinen, M., Yla¨-J¨aa¨ski, A., Zhang, W. and Hui, P., 2017, June. A measurement study on achieving imperceptible latency in mobile cloud gaming. In Proceedings of the 8th ACM on Multimedia Systems Conference (pp. 88-99).

[18] Shea, R., Liu, J., Ngai, E.C.H. and Cui, Y., 2013. Cloud gaming: architecture and performance. IEEE network, 27(4), pp.16-21.

[19] Sun, H., 2019, October. Research on latency problems and solutions in cloud game. In Journal of Physics: Conference Series (Vol. 1314, No. 1, p. 012211). IOP Publishing.

[20] Wu, D., Xue, Z. and He, J., 2014. iCloudAccess: Cost-effective streaming of video games from the cloud with low latency. IEEE Transactions on Circuits and Systems for Video Technology, 24(8), pp.1405-1416.