CMPE 492

Performance Analysis of Multiplayer Online Games

Muhammet Şen

Advisor:

Atay Özgövde

TABLE OF CONTENTS

1. INTRODUCTION

With the development more advanced softwares and hardwares, game industry produces more complex and demanding games. Multiplayer games can be shown as an example since they allow players from all over world to connect with each other. With each generation of multiplayer games; maps are getting bigger and interaction capabilities between players are advancing. One aspect with a margin for further development is network architecture that is responsible for communication between clients and servers. This project aims to simulate and analyze different server architectures that a multiplayer game can utilize.

1.1. Broad Impact

By analyzing and optimizing the network structure, this project can enhance the QoS and QoE of multiplayer games. The main problem of multiplayer games that heavily affect QoE for gamers is latency, the delay between transmitting data between two or more points and actually processing that data. Network structure a game uses directly impacts that delay.

Another enhancement of this project is improving the efficiency and performance of game servers that can be a huge step to consume less energy and materials to run multiplayer games.

1.2. Ethical Considerations

This project is a proof of concept that aims to simulate and analyze different network structures and algorithms for multiplayer games. We do not use and store any personal data.

2. PROJECT DEFINITION AND PLANNING

2.1. Project Definition

This project consists of two parts: creating an experimental platform that can be used to implement and analyze different algorithms and methods for load distribution and communication in multiplayer games. Second part is actually testing out different algorithms on the platform.

2.2. Project Planning

2.2.1. Project Time and Resource Estimation

In the first two week of the project, I worked on understanding the transport package of Unity[1] and creating a basic two server - one client application to estimate different ways I would use to utilize multiple servers in a single game world. First demo I had was focused on assigning objects to each server and showing both objects in the game world.

The third week was about designing messaging schemas and different message types. Iterating over scenarios and edge cases was an important process since I had discovered many edge cases and technically unfavorable situations.

Following week was focused on passing object ownerships between servers. We decided to divide servers according to coordinates, so checking an object's location was enough for passign over the ownership of that object.

Similar to the last week, I worked on passing over the client connection between different servers. An important difference between passing over client and object own-

[1]https://docs-multiplayer.unity3d.com/

erships is that object ownerships are transferred via server-to-server communications while client ownerships are transferred via server-to-client communications.

An important part of this project is providing a dynamic and customizable working environment. The following week I worked on making the game world as parameterized as possible. There are many configurations that affect the overall performance, like number of servers, clients, game objects. I added configuration schemas and command line arguments to make testing easier.

As a part of parametrically modifying the environment, I worked on creating tiles and walls around the game world. Placing the tiles was trivial, however, achieving the precision of walls' locations took a bit more time than I anticipated.

In order to tidy up the unity object tree, I worked on assigning parent objects to server, player, and cube objects. Servers are the parents of their objects. I created a GameManager object to be the parent of servers, just to make the UI less complicated.

I noticed that some objets are not fully removed from player's screen, espcially in server buffering approach. I have improved the broadcasting algorithm as well as the garbage collector system. Every object which has not been updated for the last x seconds is removed from the screen.

Following week I started on area of interest concept. Although implementing the actual mechanism was trivial, I tried to make it more clear to demonstrate. I added different radius circles around the player to make sure everything is working as expected.

To try out different parcel shapes, I added a hexagon grid system. It required a few changes around the codebase to fit everything under the same Grid structure. Both square and hexagon grids share the same base class, so using this sub classes, as well as adding a new one, is easier. Specifically, adding walls around the hex map took a lot of time and effort.

Starting with the deployment and testing, I needed a more powerful computer than I have to be able to work with 9 or more server instances. I requested an EC2 server from the company I am currently working at and they happily approved that request. I created an EC2 instance with 16 cpus and 40 GB of memory, which is more powerful than I need to run 9 or 16 server tests.

I created scripts to easily copy the current build to the remote server. Also, I had dockerized the application to run server containers. This made it easier to implement the metric collection system, thanks to Docker's stats API.

Following week I worked on WebGL and ways to deploy the player instance. Unity's transport package got had been updated just one month ago and started supporting websocket connection. I upgraded the unity version of the project and add conditional checks for UDP and websocket connection options.

Rest of the project was trying out different configurations, creating scripts for making plots and modifying them. I fixed some bugs as I encounter during the tests.

2.2.2. Success Criteria

We aim to create an underlying infrastructure that will be used to try out different network designs, server architectures, approaches and algorithms. We try to make everything parameterized, like number of servers, players, objects, so that someone who just wants to try out a new algorithm to distribute the game world between different servers does not have to deal with the infrastructure, communication protocols, servers etc. After creating such a platform, I worked on and tested different ways to distribute computational demands of a game among game servers.

2.2.3. Risk Analysis

During the development process, we are working on our local machines. However, to fully understand the load distribution we will deploy many server instances to cloud providers, like AWS or Digital Ocean. Since we want to test everything thoroughly, we may be limited with the cloud providers' free tier allowances. Another risk is getting a huge bill from that cloud provider. We should be very careful about how much computation time we have in our free tier. The company I am currently working at provided me a powerful EC2 machine.

2.2.4. Team Work (if applicable)

Not applicable.

3. RELATED WORK

- Ploss et al. (2008) [1] focus on porting Quake 3 game engine to a multi-server infrastructure with the help of the RTF (Real Time Framework) middleware they had developed. Their ported version of Quake engine performs similar to the stock one and allow scalability, which is major criteria in MMOGs, massively multiplayer online games.

- Kohana et al. (2015) [2] work on server selecting mechanisms not only in games but in general WEB. They develop an algorithm and network structure that can predict the load on one server and try to choose one with free resources. Their performance analyses are focused on measuring the network latency as well as expected load on different servers.

- Burger et al. (2016) [3] focus on online multiplayer battle arena games which highly depend on latency due to fast nature of the games. They try to distribute the users geographicly to minimize the delay between users and servers. They analyzed the previous matches' histories and statistics to distribute user more efficienty.

- Moll et al. (2019) [4] develop a server to server system to synchronize game state by utilizing NDN, Named Data Networking, model. Since interserver communication creates an overhead, they test on Minecraft due to game world's sizes. Minecraft's infinite world map creates room for benefiting from scalability.

- Boulanger et al. (2006) [5] investigate different investigate management techniques and ways to notify players about other players' state changes. They also discuss about simulating a real player's actions randomly. This approach makes it easier to test and explore different solutions easily.

- Prasetya et al. (2008) [6] propose a method to focus on partitioning the game world in an efficient way to decrease the network usage of the device. Although their research is primarily focused on mobile games, their method would be applicable to other platforms as well.

4. METHODOLOGY

This project is based on the Unity Game Engine[2] and Unity Multiplayer Networking library[3] . We started the project by evaluating different aspects of networking in game engines and ways to improve potential flaws they have. The naive approach that we had decided to use as a benchmark was sending every update to every running server, simply relaying every message to every other server. Even though this approach is not efficient, implementing was easy and we had a tool to evaluate more advanced approaches.

In our experimental setup, we run n number of servers with each server having m number of objects, with a predefined velocity, in the beginning. Those n*m objects travel across the map and servers transfer objects to neighbor servers when an object passes through a border. The world is surrounded by walls with a physical material to bounce colliding objects perfectly, without losing kinetic energy just like a DVD logo.

Each server instances in our experimantal setup was running as a headless unity applications that listens on three different ports, one for UDP connections from clients, one for WebSocket connections from clients, and one for connections from other servers. This clear distinction between different message types allowed us to isolate use cases and message flows. There are certain message types for each connection channel.

Client to server messages are triggered by the client to request a functionality from a server, like getting the latest status of the map or registering a new player. Server to server messages are usually used to transfer objects between different servers.

Player activities are not the main goal of this project. We utilize a player object just to create a load on the game servers and try out how we can improve the distribution of clients to different servers. As a part of that distribution process, we assign

[2]https://unity.com/
[3]https://docs-multiplayer.unity3d.com/

only one server to a player. Player gets required updates from that server only and the server is responsible for communicating with the rest of the game world. When a player leaves a server's borders, another server is assigned for that player, according to current coordinates, and a new connection is establishes between the player and new server.

There may be testcases where presence of multiple clients are required. The platform supports building the player object with WebGL support and running in the browser. One can easily launch the player by visiting a URL. We have a Selenium[4] automater that can run as many client instances as wanted.

To being able to analyze different approaches, we are using a metric collection system. There are three metrics we are mostly interested in: cpu, memory, and bandwidth usage. To analyze cpu and memory consumption, we utilize top[5] tool. It provides an automatization friendly process we can use to evaluate cpu and memory usage of our server and client instances.

To gather client side network usage, we will use a well known tool called Wireshark[6] . It let's us listen for every request send from one application to another. We can easily filter and analyze every request each server or client makes.

Since server instances are running inside Docker containers, we can easily collect mentioned metrics from the Docker Stats API. The platform has required python scripts to connect to and fetch by using docker sdk.

[4]https://www.selenium.dev/
[5]https://man7.org/linux/man-pages/man1/top.1.html
[6]https://www.wireshark.org/

5. REQUIREMENTS SPECIFICATION

This projects aims to create an experiment platform that can be used to try out different load distribution and server management strategies. Moreover, it aims providing required tools and setups to analyze those strategies, usually by exporting different kinds of metrics mentioned in Methodology section of the report.

Analyzing outputs of a setup is just as important as being able to implement that setup. In order to deliver precise and consistent results, the platform will export CPU, memory, and network metrics. Network metrics, packet counts and statuses of those packets of the client will be collected by Wireshark tool. Docker's stats API will be used to collect metrics of server containers. This export process produces a structured and easy to analyze format.

Being able to test different scenarios is another requirement of this project. Since using only one computer to test out multiple instances may result in a performance loss and incorrect analysis, developers shall be able to deploy the server instances to other remote machines, preferably to cloud providers, to better investigate their implementation and capabilities. Since server instances are already containerized, deploying to one or more servers is trivial. Similarly, player instance can be built via WebGL and can be run in the browser.

6. DESIGN

6.1. Information Structure

Communication between servers and clients are established on "SerializableObject" class that have certain fields to transfer position and velocity information as well as methods to easily serialize and deserialize objects.

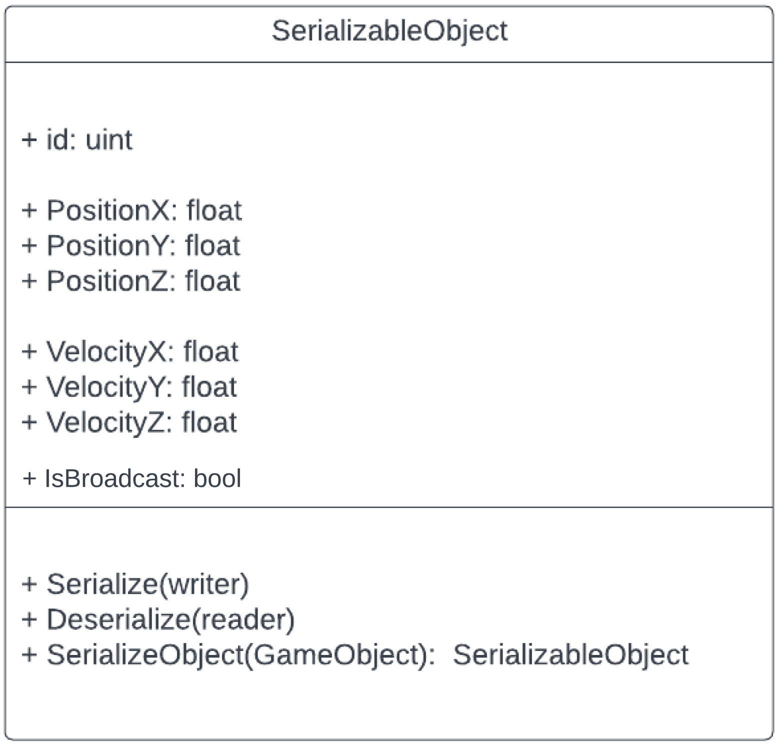


Figure 6.1. SerializableObject Class Diagram

6.2. Information Flow

The communication between components of the platform can be categorized into two groups, server-server and client-server communications.

Client - server communications are designed to update the client's state regularly in order to keep the game world synchronized. Another functionality is making sure client is connected to the right server, according to their location.

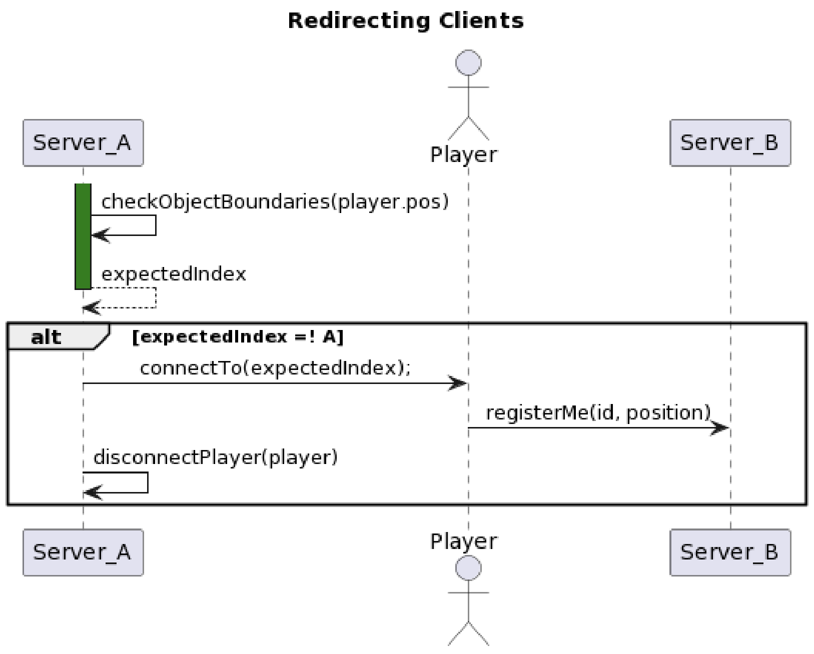


Figure 6.2. Client to Server Sequence Diagram

Server - server communications are used to equally distribute the load among different game servers. Currently, we are utilizing pass over mechanism to move objects between servers. This distribution mechanism is the point we are trying to improve. There are many ways we can apportion the load and carry state data.

The naive approach is sending an object's state to every other server, like flooding algorithms in networks. This method does not consider a server's relation to another, like being geographically close to each other. This method results in a lot of bandwidth use and overhead between servers. However, a server may choose not to send an object's state to one of it's clients, the object may be too far away or cannot be seen by the player.

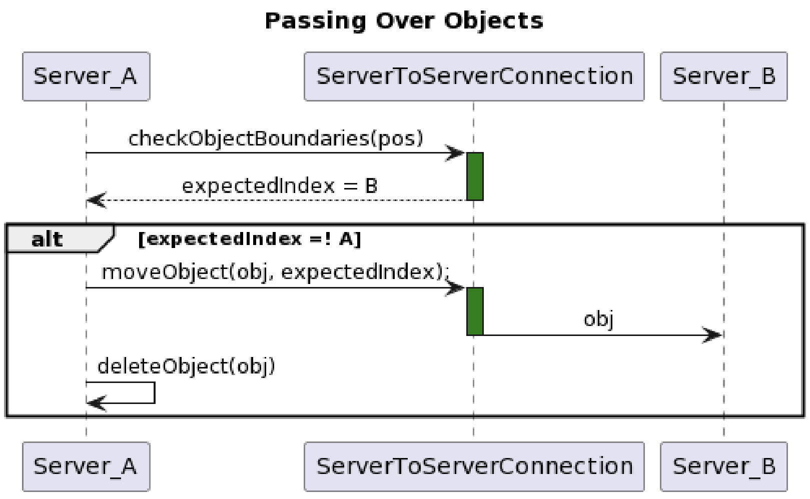


Figure 6.3. Server to Server Sequence Diagram

6.3. System Design

Platform consists of servers and clients. One of the main goals of the project is allowing parametrization of different configurations, like number of servers and clients. Each server and client can run isolated and separately.

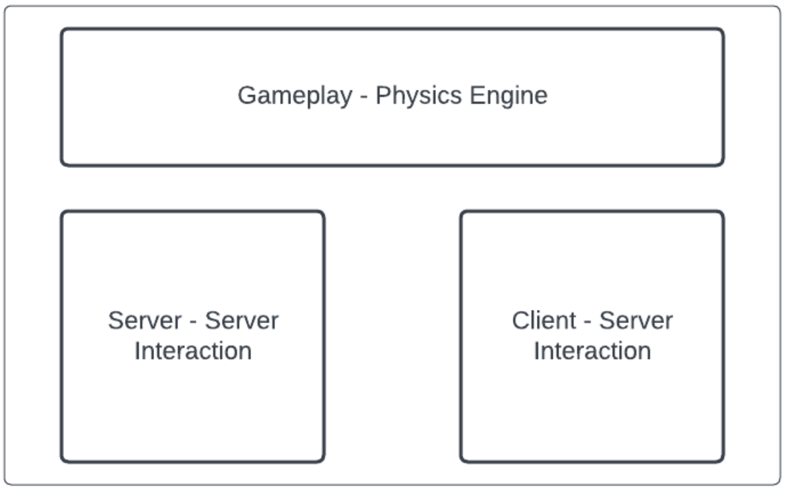

Gameplay - Physics Engine
Server - Server Interaction
Client - Server Interaction

Figure 6.4. High Level Software Architecture of Servers

6.4. User Interface Design (if applicable)

Not applicable.

7. IMPLEMENTATION AND TESTING

7.1. Implementation

The core infrastructure that provides communication between different elements is ready and can be used to try out different methods and algorithms. Clients are able to connect to servers and get the periodic state information from them. Since we aim to focus on distribution and communication part of multiplayer games, we are not planning to work more on UI and UX of the platform. We have implemented the naive approach to test our infrastructure. Although there are some minor bugs, usually about disconnecting previous connections, the platform is running and ready to try out new things. All of the mentioned work is pushed to our GitHub repository[7] .

We are trying out different distribution algorithms to improve network usage and computational demand of servers. A further work is implementing some kind of a game mechanic to make it easier to visualize and demonstrate the project. That mechanic should contain a problem caused by the underlying network solution.

There are many scripts for running automated tests, parsing outputs of those tests, and creating plots from parsed results.

7.2. Testing

Although the platform is designed to be deployed to many servers, one can easily run it on a single machine by setting the configuration.

[7]https://github.com/muhammetssen/SpatialPartitioning

7.3. Deployment

Server instances are deployed to an EC2 machine and there is an easy to use deployment script that can be used to build and update the deployment by

- Copying the build folder to the EC2 via rsync[8]
- Build the docker image
- Run x number of containers where x is the number of servers desired

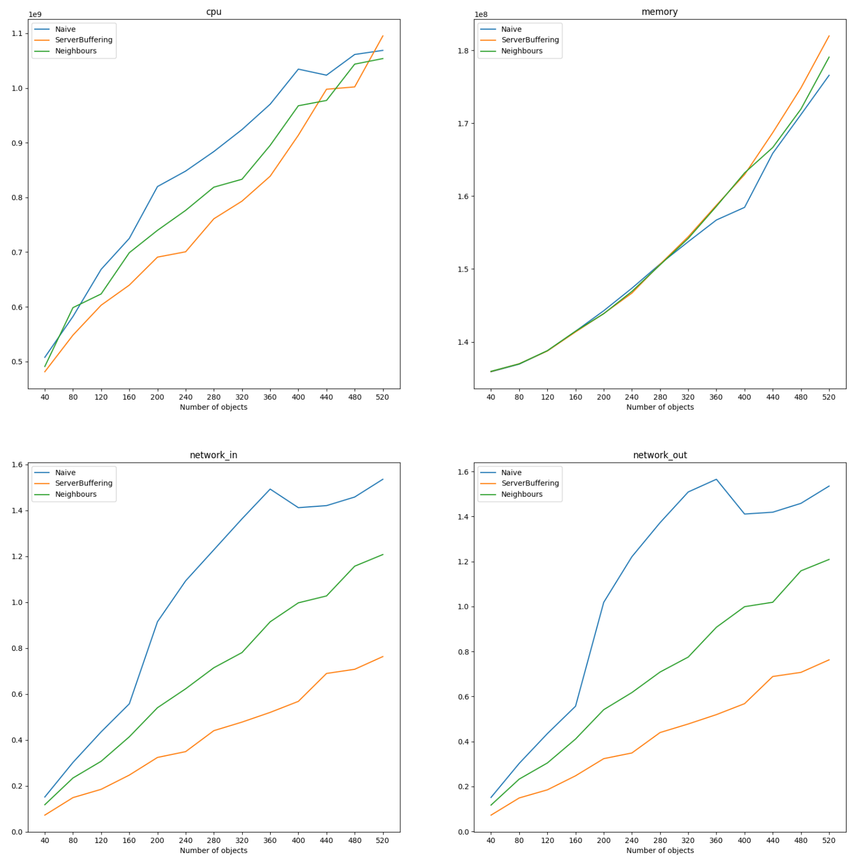
Similarly, WebGL built can be deployed behind a proxy like NGINX[9] .

[8]https://linux.die.net/man/1/rsync
[9]https://www.nginx.com/

8. RESULTS

With the first part of the project, we have a platform that is ready to work on. Currently, we can run the platform with n number of servers where n is a square number and as many clients as we want to. Second part of the project is focused on different implementations and solutions to handle server-server and server-client communication.

Following figure shows 3 different solution types' metrics. As expected naive approach uses the most CPU and network while server buffering decreases CPU usage slightly and network usage heavily.



9. CONCLUSION

First part of the project was building the platform that will be used as a test environment for new algorithms and approaches. The platform is deployed and ready to use. We had run many use cases and configurations to test different aspects of the system.

For further improvement, metric collection system of the client depends on different tools and methods, like Wireshark for networking and top command for cpu metrics. Having a similar system to servers' docker stats api would be easiser to work with.

REFERENCES

1. Ploss, A., S. Wichmann, F. Glinka and S. Gorlatch, “From a single- to multi-server online game: a Quake 3 case study using RTF”, pp. 83–90, 12 2008.

2. Kohana, M. and S. Okamoto, “A Server Selection Method for Web-Based Multi-server Systems”, *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, pp. 112–117, 2015.

3. Burger, V., J. F. Pajo, O. R. Sanchez, M. Seufert, C. Schwartz, F. Wamser, F. Davoli and P. Tran-Gia, “Load Dynamics of a Multiplayer Online Battle Arena and Simulative Assessment of Edge Server Placements”, *Proceedings of the 7th International Conference on Multimedia Systems*, MMSys ’16, Association for Computing Machinery, New York, NY, USA, 2016, `https://doi.org/10.1145/2910017.2910601`.

4. Moll, P., S. Theuermann, N. Rauscher, H. Hellwagner and J. Burke, “Inter-Server Game State Synchronization Using Named Data Networking”, *Proceedings of the 6th ACM Conference on Information-Centric Networking*, ICN ’19, p. 12–18, Association for Computing Machinery, New York, NY, USA, 2019, `https://doi.org/10.1145/3357150.3357399`.

5. Boulanger, J.-S., J. Kienzle and C. Verbrugge, “Comparing interest management algorithms for massively multiplayer games”, p. 6, 01 2006.

6. Prasetya, K. and Z. Wu, “Performance analysis of game world partitioning methods for multiplayer mobile gaming”, pp. 72–77, 10 2008.