Use of Server Mesh in the Context of Games

*Miguel Ferreira (1230199)*

*Instituto Superior de Engenharia do Porto, Portugal*

***Abstract*—.The current architecture of online multiplayer games faces significant limitations regarding seamless data transfer, often forcing players to endure loading screens between transitions and server lag. Additionally, issues such as limited server capacity and poor scalability in distributed systems hinder the overall gaming experience. Addressing these challenges has become a focal point of technological innovation in the gaming industry, with several companies competing to develop solutions. One of the most promising approaches is server meshing, which enables dynamic load distribution and horizontal scalability across multiple servers. Cloud Imperium Games (CIG) is at the forefront of this innovation, demonstrating the potential of server meshing to revolutionize gaming. By enabling near-limitless scalability and reducing server load bottlenecks, this technology represents a significant leap toward creating vast, persistent game worlds with seamless player experiences. This paper explores the technological foundations of server meshing, its potential impact on game design, and its implications for the future of distributed systems in the gaming industry.**

***Index Terms*—Distributed systems, Dynamic server loading, Server mesh, Static server mesh, Dynamic Server Mesh, Static Server mesh, Start Citizen, Ashes of Creation, Usefulness.**

# I. INTRODUCTION

As the demand for large-scale online multiplayer games grows, like MMOs, so do the technical challenges associated with supporting vast game worlds and thousands of concurrent players. Traditional server architectures often impose limitations in terms of scalability and performance, leading to lag, the necessity for loading screens, needing to slow down the system, and overall poor user experiences. Addressing these issues requires innovation in distributed systems architectures in games.

One of the leading innovations in this landscape is server meshing, which is being pioneered by companies like Cloud Imperium Games (CIG) and Intrepid Studios and is expected to change the industry by enabling dynamic load balancing and near-limitless scalability. Server meshing allows multiple servers to work together, distributing player loads dynamically and creating vast, seamless game worlds without the performance bottlenecks associated with traditional server models.

## A. Abbreviations and Acronyms

CIG: Cloud Imperium Games

MMO: Massively Multiplayer Online

TiDi: Time Dilation

NPC: Non-Player Character

AWS: Amazon Web Services

# II. Current Distributed software architecture

The current scenery of distributed systems offers solutions that are highly scalable and reliable, allowing for mass exchange of data, tools like Kubernetes allow for easy horizontal scalability and as distributed architectures become more prevalent it is a must-use tool[1], there are already corroborated solutions for making systems fault tolerant like using redundancy between servers[2].

## Differences

Even if modern software architecture has been improving, even being able to work with millions of data, our games server architecture still has problems in proving these capabilities. The main problem is that games are full of state, and state management is still one of the biggest hurdles to solve, this problem is accentuated on game servers due to its innate need for high state management [3]. As such, we have seen these kinds of limitations in display, for example, with Eve Online when the biggest online battle happened [4], it was needed to reinforce the servers and use a special technology created by the formers, called TiDi[5] which slows down game time so servers do not crash.

# III. Server Meshing: A Technological Overview

Server meshing is the process of dynamically splitting the load of a game world across multiple servers, effectively treating them as a unified entity. This contrasts with the more traditional sharded servers’ model, like the mega servers of Guild Wars 2[6], where game worlds are divided into static instances. In server meshing, a game world can be hosted across multiple servers that coordinate dynamically based on real-time load and player activity.

Two primary types of server meshing have been proposed:

1. **Static Server Mesh**: In this configuration, the game world is partitioned into predefined areas, with each area assigned to a specific server. Players seamlessly transition between these areas without needing to experience loading screens. However, the static mesh does not allow for dynamic reallocation of server resources [7].
2. **Dynamic Server Mesh**: This more advanced configuration allows for real-time load distribution. The game world is not divided into static zones but rather allocated based on player density and server capacity. Servers can dynamically adjust to handle more players in high-traffic areas or offload areas with fewer players, optimizing performance and scalability [7].

## Server Meshing in Depth

It is important to note that discussing server meshing in detail is challenging, as only a few companies have publicly disclosed information about this technology, and none of them have fully implemented it for public scrutiny. Consequently, this section will hypothesize how the system might work based on available information, followed by a comparison with traditional software implementations to assess the plausibility of these claims.

Server meshing involves the complex orchestration of multiple servers that handle different parts of a virtual game world. The innovation here is how it allocates resources dynamically, meaning the system adjusts the workload in real time based on player activity and server conditions. In traditional server architectures, game worlds are divided into distinct, isolated instances or “shards”, with each server hosting a specific area [6], in contrast, dynamic server meshing treats the game as a continuous whole, meaning that the world is divided into smaller zones or regions, which are then allocated to different servers. In static server meshes, these zones are predetermined and fixed. However, in dynamic server meshes, the system continuously monitors the load in each zone and dynamically reassigns them to different servers based on real-time needs. This could involve splitting a heavily populated zone across multiple servers if the player count increases, this is the partitioning part.

The core feature of the dynamic server meshing which is its ability to adjust server allocations based on real-time player density, load, and activity could be defined in the following steps:

* **Monitoring**: The system consistently monitors metrics like player count, NPC activity, bandwidth usage and server performance.
* **Load Distribution**: When a region becomes overloaded due to a high number of players, the server meshing system splits the workload, and this can be done in two ways:
  1. **Dynamic Zone Creation**: The system divides the game world dynamically by creating smaller zones within the crowded area.
  2. **Parallel Server allocation**: Multiple servers can work together to handle the same region, parallelizing tasks like physics calculation, NPC movements, and player interactions.

Managing complex interactions implies the need to have a specialized orchestration middleware capable of:

* Deciding which servers handle which areas.
* Implementing mechanics for fault tolerance in case a server fails
* Handling latency-sensitive tasks like collision detection or other game components between servers

Finally, this system must be capable of elastic scalability, allowing it to scale up or down smoothly depending on player demand.

## Analogies with Traditional Distributed Systems

To validate the previous assumptions regarding server meshing, we can draw analogies to conventional distributed systems used in other domains, which share similar principles of dynamic load distribution and resource management.

Dynamic server meshing is somewhat like how Kubernetes orchestrates microservices. In Kubernetes, different services (like database management, user authentication, etc.) are split into containers (or pods) that can scale independently based on demand, so if there is a need for more availability for access to a database, Kubernetes can deploy more instances of the database container [9]. This compares to dynamic server meshing where zones or areas of the game world are treated like “microservices”, with the ability to spawn new instances of server processes dynamically based on player density.

The use of dynamic load balancers in web services (like those offered by AWS) can also be compared to dynamic server meshing. In a web application, traffic is distributed across multiple servers based on real-time load to ensure no single server is overwhelmed [10][11]. Similarly, in server meshing, when too many players congregate in one part of the game world, the system automatically distributes them across multiple servers to balance the load.

Finally, we have replication and eventual consistency in distributed databases (like Cassandra or DynamoDb) that parallels the state management in dynamic server meshing. These databases store copies of data across multiple nodes (servers) and ensure that changes are propagated across nodes in a way that the system eventually [12][13], which is like how dynamic server meshing synchronizes player and world data across multiple servers

Even though it’s not a one-on-one comparison, we can see absolutely the same implementations, the biggest difference is what type of data is being treated, in game development we aren’t just working with numbers or strings or any other simpler types of data, we are dealing with data that has state, components attached to the same data and so on.

## Benefits of Server Meshing

1. **Horizontal Scalability**: Server meshing enables scaling horizontally by adding more servers to the network. This approach contrasts with traditional vertical scaling, where upgrading server hardware becomes cost-prohibitive as player numbers grow.[8]
2. **Seamless Player Experience**: By reducing or eliminating loading screens between server transitions, server meshing allows for a more immersive experience. Players can traverse vast game worlds without interruptions, making the game more persistent and alive [14].
3. **Reduced Bottlenecks**: Dynamic server meshing addresses the issue of overcrowding in high-traffic areas by allocating more servers to those areas, reducing lag and enhancing performance [8][14].

# IV. Where Is It Being Used

There are two major games where this technology is being developed, the two being:

1. **Ashes Of Creation**
2. **Star Citizen**

Though their implementations differ slightly, both aim to optimize load distribution and create seamless player experiences across vast game worlds

## A. Ashes Of Creation

In Ashes of Creation, server meshing ensures seamless transitions between servers without loading screens. Key features include:

* **Authority Representation**: Each player is managed by an authoritative server. When moving between servers, the player’s state is transferred without creating new instances, reducing resource costs.[14]
* **Proxy System**: Players are represented by proxy actors on neighbouring servers, which are promoted to authoritative actors when needed. This system minimizes the need to recreate players during transitions.[14]
* **Data Reuse and Event Communication**: Data is efficiently reused across servers, and event information is decentralized, ensuring smooth transitions and collaboration between servers.[14]
* **Fault Tolerance**: If a server fails, adjacent servers maintain gameplay stability, ensuring the system continues without interruption.[14][15]

## B. Star Citizen

Star Citizen employs a similar approach but with an added replication layer that stores and synchronizes game state data between servers and clients. Key components include:

* **Replication Layer**: This layer manages the game state and ensures consistent synchronization between server nodes, facilitating smooth player transitions.[18]
* **Streaming Bubbles**: As players move, the system dynamically loads and hands off control of entities between servers, allowing seamless transitions without interruptions.[19]

In both games, server meshing allows for dynamic load distribution and near-limitless scalability, enhancing the player experience and reducing performance bottlenecks.[19]

## C. What does this allow in the two games?

In both Ashes of Creation and Star Citizen, server meshing transforms how players interact with the game world:

1. **Seamless Exploration**: Players can move freely across the game world without being interrupted by loading screens.
2. **Massive Player Interactions**: Allows thousands of players to share the same space, meaning huge battles, and large-scale events.
3. **Dynamic World Events**: The game world adapts in real time. Players can participate in events that bring together large numbers of people, like kingdom wars in Ashes of Creation.
4. **Persistent Universe**: Players will experience a world that feels alive and always active. You can leave a mark on the world, like affecting the economy, territories, or factions.
5. **Large, Connected Worlds**: Instead of being restricted to one server or isolated area, players can seamlessly interact with others across vast distances.
6. **Uninterrupted Gameplay**: Players won’t experience sudden disconnects or crashes. The world continues without interruptions

# V. Conclusion

A conclusion section is not required. Although a conclusion may review the article's main points, do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions.

# Appendix

Appendixes, if needed, appear before the acknowledgement.

References

1. *Kubernetes Components | Kubernetes*. (n.d.). Retrieved September 28, 2024, from https://kubernetes.io/docs/concepts/overview/components/
2. *Fault Tolerance in Distributed Systems: Strategies and Case Studies - DEV Community*. (n.d.). Retrieved September 27, 2024, from <https://dev.to/nekto0n/fault-tolerance-in-distributed-systems-strategies-and-case-studies-29d2>
3. Distributed State — Challenges and Options | by Koshy | Medium. (n.d.). Retrieved October 1, 2024, from https://nittikkin.medium.com/distributed-state-management-80c8100bb563
4. *Fury at FWST-8 Battle Report | EVE Online*. (n.d.). Retrieved September 27, 2024, from <https://www.eveonline.com/news/view/fury-at-fwst-8-battle-report>
5. Introducing Time Dilation (TiDi) | EVE Online. (n.d.). Retrieved October 1, 2024, from <https://www.eveonline.com/news/view/introducing-time-dilation-tidi>
6. Introducing the Megaserver System – GuildWars2.com. (n.d.). Retrieved October 1, 2024, from <https://www.guildwars2.com/en/news/introducing-the-megaserver-system/>
7. *The Nosy Gamer: The Server Meshing Wars Have Begun*. (n.d.). Retrieved September 25, 2024, from <https://nosygamer.blogspot.com/2024/07/the-server-meshing-wars-have-begun.html>
8. Horizontal scaling vs vertical scaling: Choosing your strategy | DigitalOcean. (n.d.). Retrieved October 1, 2024, from <https://www.digitalocean.com/resources/articles/horizontal-scaling-vs-vertical-scaling>
9. Overview | Kubernetes. (n.d.). Retrieved October 15, 2024, from <https://kubernetes.io/docs/concepts/overview/>
10. What is Load Balancing? - Load Balancing Algorithm Explained - AWS. (n.d.). Retrieved October 15, 2024, from <https://aws.amazon.com/what-is/load-balancing/>
11. Load balancer in Google Cloud FOTC. (n.d.). Retrieved October 15, 2024, from <https://fotc.com/blog/load-balancer/>
12. Data replication | Apache Cassandra 3.0. (n.d.). Retrieved October 15, 2024, from <https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/architecture/archDataDistributeReplication.html>
13. Amazon DynamoDB auto scaling: Performance and cost optimization at any scale | AWS Database Blog. (n.d.). Retrieved October 15, 2024, from https://aws.amazon.com/blogs/database/amazon-dynamodb-auto-scaling-performance-and-cost-optimization-at-any-scale/
14. *(42) Ashes of Creation Alpha Two Server Meshing Technology Preview - YouTube*. (n.d.). Retrieved September 27, 2024, from https://www.youtube.com/watch?v=pdav0as54mU&t=360s
15. *Server meshing - Ashes of Creation Wiki*. (n.d.). Retrieved September 27, 2024, from https://pt.ashesofcreation.wiki/Server\_meshing
16. Duong, T. N. B., & Zhou, S. (2003). A dynamic load sharing algorithm for massively multiplayer online games. *IEEE International Conference on Networks, ICON*, 131–136. https://doi.org/10.1109/ICON.2003.1266179
17. Aote, S. S., & Kharat, M. U. (2009). A game-theoretic model for dynamic load balancing in distributed systems. *Proceedings of the International Conference on Advances in Computing, Communication and Control, ICAC3’09*, 235–238. https://doi.org/10.1145/1523103.1523153
18. *Server Meshing and Persistent Streaming Q&A - Roberts Space Industries | Follow the development of Star Citizen and Squadron 42*. (n.d.). Retrieved September 25, 2024, from https://robertsspaceindustries.com/comm-link/transmission/18397-Server-Meshing-And-Persistent-Streaming-Q-A
19. *What is Star Citizens Serving Meshing and Persistent Entity Streaming | by Marvas McCladdie | Medium*. (n.d.). Retrieved September 25, 2024, from https://medium.com/@MarvasMcCladdie/star-citizen-is-one-of-the-most-ambitious-and-highly-anticipated-video-games-of-all-time-486abb0a02a9
20. *(25) CitizenCon 2951: Server Meshing & The State Of Persistence - YouTube*. (n.d.). Retrieved September 25, 2024, from https://www.youtube.com/watch?v=TSzUWl4r2rU