Distributed Servers for Multiplayer Minecraft

Samuel J. Huddart

2024

Work in Progress
May 2024 Update

Contents

1 Introduction

Minecraft is the best-selling video game in history, amassing 300 million sales. Released in 2011 the popular 3D sandbox game offers players an infinite canvas for creativity with diverse playstyles and game modes. Thousands of servers act as virtual communities, fostering connections and shared experiences across the globe.

Minecraft's multi-player server architecture faces limitations when it comes to scaling for large player bases. The core issue lies in its single-process design, restricting the number of players a single server can handle effectively.

Larger servers often resort to proxies to connect multiple server instances. While this works for mini-game lobbies with limited player sizes, it falls short for the quintessential Minecraft experience – single-world survival. This approach forces large communities to fragment their player base across separate worlds/servers, creating a disjointed and less immersive experience.

Furthermore, traditional server hosting often relies on a single location, leading to high latency for players geographically distant. This translates to a frustrating experience for players connecting across continents.

This paper investigates potential solutions to address these limitations. We will explore various approaches, analyzing their advantages and disadvantages to create a more scalable and enjoyable gameplay experience for large Minecraft communities.

2 Solutions

2.1 Distributed

One promising approach to address these limitations lies in distributed systems. These systems distribute the Minecraft server workload across multiple interconnected machines. This offers a three-fold advantage:

- **Scalability:** By distributing the processing tasks, a distributed system can handle a significantly larger number of players compared to a single server instance.
- **Reduced Latency:** Distributed systems enable "edge" hosting, where server instances can be located geographically closer to player populations. This significantly reduces latency, leading to a smoother and more responsive gameplay experience for players worldwide.
- **Reliability:** Distributed systems inherently offer improved uptime and reliability. If one machine encounters an issue, the others can continue handling player traffic, minimizing downtime and ensuring a more consistent experience for players.

However, implementing a distributed system for Minecraft comes with its own set of challenges.

- **Complexity:** Implementing a distributed Minecraft server requires a more complex system architecture compared to traditional single-server setups. Minecraft servers present a unique technical challenge due to the game's single-server architecture.
- **Bandwidth:** Distributing the workload across multiple servers significantly increases network traffic between them. This requires careful planning and sufficient bandwidth allocation to ensure smooth communication and avoid bottlenecks.

- **Redundant Processing:** One potential drawback of distributed systems is the possibility of redundant processing. In a single-server setup, calculations are performed only once. However, in a distributed system, there's a chance that the same calculations might be performed on multiple machines for different players, potentially leading to wasted processing power and increased costs.
- **Increase Latency:** While distributed systems can reduce latency by placing servers closer to players, poorly optimized network configuration can introduce additional "hops" between servers. These extra hops, especially if geographically distant, can lead to increased latency and a degraded player experience.

Despite these drawbacks, distributed systems offer a compelling path forward for Minecraft servers. Their potential for scalability significantly outweighs many of the cons. With careful design and implementation, a distributed system can be a highly effective solution for large and geographically diverse player communities.

2.2 World Partitioning

Dividing the Minecraft world into smaller sections across servers can improve performance by focusing processing power on specific player locations. This approach, however, introduces additional complexity in managing player movement and interactions between partitions.

The traditional method of dividing the world utilizes "fixed regions." This approach splits the Minecraft world into pre-defined sections. When a player approaches a boundary, they are switched to a different server handling that specific region.

Switching servers can be jarring for players, often involving an abrupt transition and a loading screen. This disrupts the immersion and overall experience. Furthermore, fixed regions are static and cannot dynamically adapt to player distribution. When large enough, this can lead to uneven server load, with some regions experiencing high traffic while others remain underutilized. These drawbacks highlight the need for more sophisticated world partitioning techniques.

Our proposed solution moves away from fixed regions and introduces the concept of "sub-spaces." Sub-spaces are dynamic sections of the world that adapt to player movement. Imagine a bubble that expands and contracts around each player, encompassing the area they interact with.

- Sub-spaces prioritize the player's experience. The world sections loaded on the server dynamically adjust based on the player's location and activity.
- Sub-spaces can grow or shrink as players explore. This ensures efficient resource allocation, focusing processing power on areas with active players.
- When players approach each other, their sub-spaces can merge seamlessly, eliminating the need for abrupt server switches and loading screens. Conversely, as players move apart, their sub-spaces can split, maintaining efficient resource utilization.

2.3 Separation of Responsibility

The traditional Minecraft server architecture presents a significant hurdle for building a truly scalable distributed system. Our approach proposes decoupling the core functionalities of the Minecraft

server. This essentially involves breaking down the monolithic server software into smaller, independent modules; much like commonly used micro-services pattern.

By separating functionalities like world management, player connections and simulation into distinct services, we can distribute them across different servers. This flexibility allows for independent scaling of each module based on its specific resource needs.

For Example: Separating player movement data from the overall world simulation;