Scalability and availability for massively multiplayer online games

Guillaume Turchini, Sebastien Monnet, Olivier Marin

Scalability and availability for massively multiplayer online games

Guillaume Turchini
LIP6
Université Pierre et Marie Curie
Paris, France

Sebastien Monnet
LIP6
Université Pierre et Marie Curie
Paris, France

Olivier Marin
LIP6
Université Pierre et Marie Curie
Paris, France

***Abstract*—MMOGs (Massively Multiplayer Online Games) are getting ever more popular, but current game server architectures do not scale with the number of players. Instead of addressing the issue, the most common workaround in the industry is to use multiple distinct and non communicating game servers. After a brief overview of existing game server architectures and methods to distribute server load, this position paper outlines another kind of architecture that should scale and discusses the difficulty of evaluating game platforms on a large scale.**

I. Introduction

Designing a scalable game server architecture that guarantees availability for every player is no easy task. An MMOG server must accomodate a large number of connected players, offering each of them a consistent view of the game world and the best possible quality of service for a great gaming experience. In particular, short response times for player actions are often essential [1].

Adapting the capacity of the game server architecture to achieve satisfactory availability is very complex because the number of players connected to a game varies significantly during the day. Figure 1 shows the wide variation of the average number of players connected to a single *World of Warcraft Realm* during the day between January and September 2006 [2]: it becomes very crowded around 11pm and almost deserted around 7am. Such a variation can lead to a game server crash if the server is not powerful enough. In fact, game server architectures have a long tradition of crashing every time a new highly anticipated game is released. Usually the main solution is to highly oversize the game server to absorb player peaks. However this leads to a higher cost of the server architecture, with mostly underused machines.

Adding to this complexity, the distribution of player avatars in the game world is very skewed. Players often congregate around "hotspots" like towns. Different methods of splitting the game world help reducing the load of a single machine hosting the game server.

To support many players, some MMOG solve the problem by game design. For instance, Dragon Quest X reduces the possible interactions between players : groups are composed of 4 players and only able to cheer other groups, thus reducing the time constraint. This paper only considers MMOG with high number of players and high interaction between them.

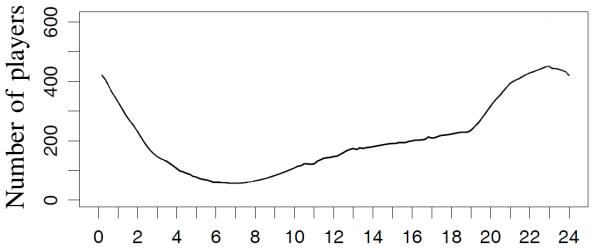

Fig. 1: Average daily distribution of the number of connected players in a single World of Warcraft realm

This paper compares the scalability of exisiting architectures, proposes hints towards improved scalability with respect to the number of connected players and paves the way for the simulation of scalable game server architectures.

II. Virtual World Partitioning

Several partitioning methods that are common to different architectures allow the management of very large amounts of players.

Zoning divides the world in different zones managed by different machines. Game designers often separate these zones with fixed and arbitrary geographical limits. Players may incur a significant waiting time (*loading screen*) between these zones while they are transferred between servers. Connecting players to two servers in an "overlap" zone at the border can acheive seamless transfers. Zones borders can also be dynamic to avoid player congestions, but dynamic zoning is more complex and costly, especially when player mobility is high.

Instead of cutting the world into zones, *Sharding* consists in duplicating the world and separating players into small groups: each group plays on a different copy of the world, called a "Realm" or a "World" in most games. Analogous to poker tables, shards can be added according to the demand, but they remain fully separate and there can be no communication or interaction among them.

Instancing corresponds to having multiple copy of a zone, possibly on different servers, each copy having a smaller amount of player. This can lead to strange situations, like having a meeting at some place, but players cannot see each

other. This is how "dungeons" in role playing games are often managed to have the same content experienced by different groups without interference.

Cloning consists in multiple machines that manage the same zone. This method allows concurrent computations associated with the same zone, such as pathfinding for different units in real time strategy games.

The problem with Sharding and Instancing is that players cannot really play together. The cloning method is not always applicable because it can involve multiple locking between servers that modify the same values, which can lead to a very high latency. The efficiency of zoning is closely tied to player locations that can vary drastically during the game.

III. Game Server Architectures

Different architectures support these partitioning strategies, the most common being the *Client-Server* architecture. The server handles all update requests from every player, updates the current world state accordingly and sends back player views of the world. This architecture may involve multiple machines with load-balancing mechanisms. However, as a server can only handle a finite number of players, the number of players a static number of servers can handle is also finite. Hence this architecture does not really scale up to an unlimited number of players and is really vulnerable to machine crashes.

Peer-to-Peer [3], [4] architectures incorporate the server logic into the client program: the architecture uses CPU power and network capacity of the players to contribute to the server power. However, game editors are reticent to this kind of architecture. One reason is that this prevents monthly subscription schemes: since all the server logic is already embedded into the client, players can freely set up their own virtual world. It also involves more development as churn and cheating are harder to handle. As cheating is a real problem on a persistent world (the cheat becomes a permanent advantage), it is carefully checked. This implies a bigger latency due to the need of agreement on legal player actions.

Hybrid solutions [5] retain the weaknesses of both previous architectures. For instance, *Super Peers* hosted either on player nodes or on trusted servers represent single points of failure.

A proposed alternative is an architecture that provisions VMs on a Cloud service. The game server launches new VMs during load peaks, and shuts them down when the load decreases.

IV. Our Case for Simulation

To the best of our knowledge, the literature on *Cloud Gaming* does not yet comprise any algorithms that handle insertions and deletions of VMs on demand. In an academic context, Cloud computing is hard to experiment on because of its cost. Besides, simulation facilitates automation of the experiments and allows their reproducilbility: same latencies, same player mobility, ...

Another issue is that provisioning algorithms depend mostly on player distribution inside the game world because their aim is to adapt the sharing of the world between servers.

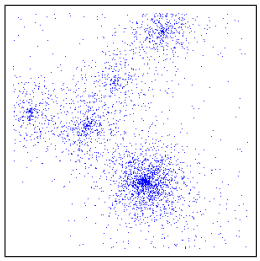

Fig. 2: Example of an avatar distribution on a map

Unfortunately there are few player movement traces available and many of them lack precision. For instance, available World of Warcraft traces indicate only the zone of the player and not the exact position. We will therefore have to generate realistic traces to test our algorithms.

Our mobility model is a refinement of *Blue Banana* [6]. Players gather around hotspots in a power law density. They move on short distances inside a hotspot and travel in straight lines between hotspots similarly to *Lévy Flights*. This mimics movements of people around real-life cities, where it's harder to move in the center of the city and highways concentrate traffic between cities. Most game worlds include forbidden zones such as oceans or high mountains on the map: our model integrates such zones that prevent player movement. Our model also accounts for player connections/disconnections along daily player variations.

V. Conclusion

This paper describes a seamless scalable game server architecture that takes advantage of VMs in Cloud services to increase processing power in crowded zones. Since the architecture depends mostly on the player distribution and since available traces lack precision, we propose a mobility model for the generation of traces. Our future work aims to generate even more realistic traces with group formations and different player activities and profiles. We will then develop load balancing and provisioning algorithms for our Cloud server architecture and compare them to existing architectures using the generated traces.

References

[1] M. Claypool and K. Claypool, "Latency can kill: Precision and deadline in online games," in *ACM SIGMM Conference on Multimedia Systems*, 2010, pp. 215–222.

[2] Y.-T. Lee and K.-T. Chen, "Is server consolidation beneficial to mmorpg? a case study of world of warcraft," in *Cloud Computing*, July 2010, pp. 435–442.

[3] J. Keller and G. Simon, "Solipsis: A massively multi-participant virtual world." in *Parallel and Distributed Processing Techniques and Applications*, 2003, pp. 262–268.

[4] O. Beaumont, A.-M. Kermarrec, L. Marchal, and E. Riviere, "Voronet: A scalable object network based on voronoi tessellations," in *Parallel and Distributed Processing Symposium*, 2007, pp. 1–10.

[5] A. P. Yu and S. T. Vuong, "Mopar: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games," in *Online Games, Network and Operating System Support for Digital Audio and Video*, 2005, pp. 99–104.

[6] S. Legtchenko, S. Monnet, and G. Thomas, "Blue Banana: resilience to avatar mobility in distributed MMOGs," in *Dependable Systems and Networks (DSN)*, 2010, pp. 171–180.