Comparison between client-server, peer-to-peer and hybrid architectures for MMOGs

Vítor de Albuquerque Torreão
School of Electronics and Computer Science
University of Southampton
Southampton, United Kingdom
vdat1g13@soton.ac.uk

ABSTRACT
Massively Multiplayer Online Games have become very popular among the player community, the game industry and the researchers. As the traditional client-server architecture started to show its limitations, researchers started to develop new architectures that could potentially substitute the client-server model. The peer-to-peer architecture was proposed, but it also had its drawbacks. Researches, then, proposed hybrid approaches that combined the best of both other architectures. However, there are still many issues to be overcome before the industry can implement any of these proposed architectures. Meanwhile, the research in MMOGs is in constant growth.

Categories and Subject Descriptors
C.2.4 [**Computer-communication networks**]: Distributed Systems—*client/server, distributed applications*

General Terms
Theory

Keywords
Software architecture, Computer game

1. INTRODUCTION
Massively Multiplayer Online Games (MMOGs) are large distributed applications where players share a virtual world with other thousands of people from all parts of the World. MMOGs have attracted the attention of both the game industry and the researchers.

When designing a MMOG, the architecture is perhaps the most important decision. A recent research done in [19] by M. Zhu et al. with the objective of categorise games' architectures yielded an interesting side result: they have discovered that most of the published research done in computer games architectures is, in fact, about network game architectures, especially MMOGs.

In this paper three different architectures are described and compared. These architectures are: *client-server* (e.g. [1, 14, 15]), *peer-to-peer* (e.g. [12, 5, 6]) and *hybrid* (e.g. [10, 3, 4, 2]) architectures.

In [4], Kim et al. point out that client-server architectures are by far the most commonly used in current commercial MMOGs. Ricci and Carlini [16] mention some reasons as to why this is true, among them the fact that one central server gives the game operator a high degree of administrative control over the game. This includes authentication, copy protection, accounting and billing, and easy update of the client-side code.

However, Jones et al. [11] highlighted some business issues in the client-server architecture: the need to buy a large amount of hardware and employ many staff to keep it running. It is also mentioned large costs on energy and floor space. Similarly, Merabti and El Rhalibi [13] point out that data storage, server maintenance and bandwidth form the largest proportion of the cost in MMOGs and are necessary because of the client-server model: they suggest that a different architecture could reduce these costs.

Therefore, researches have investigated the possibility of implementing a MMOG using a peer-to-peer or a hybrid architecture. However, these also have their merits and drawbacks which are going to be discussed in the remaining sections of this paper.

This paper is organised as follows: on section 2, the client-server architecture is presented and explored; in section 3, the peer-to-peer architecture is discussed; section 4 debates the hybrid architecture and, finally, section 5 compares the three architectures according to five different factors.

2. CLIENT-SERVER ARCHITECTURE
The client-server is an architecture model on which every node in the network plays one of two roles: the role of server or client.

On this model, one server machine is the point of centralization and it performs the function of managing the global state of the game. On the other hand, the client machines share this global environment via direct connection to the server machine [16]. The server then responds with the requested information individually to each client [13].

While the global state of the game is being simulated in the server machine, to which the players (clients) do not have access, there is a subset of it running on each client machine, called local state. In a typical MMOG, every local state is remotely connected to the global state, i.e. they are on physically distant machines. Furthermore, in a typical MMOG, the client machine's local computational resources are used to render the game scene and by doing so it lessens the server's rendering capability requirements [19].

This model with a single central server should handle multiplayer games with a small quantity of players playing concurrently, but it should not be enough to support the load of thousands, or even millions, of players which is the common scenario in current MMOGs [13]. Besides, this model is very susceptible to the single-point of failure problem: if the server machine is disconnected from the network for any reason, the game becomes offline and all players are prevented from playing [2].

Because of that, through out this paper the term *client-server* will not refer to this limited model, but to what the literature calls *distributed client-server architecture* which uses some techniques to workaround these issues. There are many proposed solutions in literature to solve or minimize these problems. *Zoning*, *mirroring*, *instancing*, *functional partitioning* and creating *shards* are example of techniques that can be used for this purpose.

2.1 Zoning

This technique is based on *spatial partitioning* [18], which proposes the partitioning of the game world into distinct areas. These areas can then be processed independently and in parallel. Thus, players in the same partition can interact through the same server.

Zoning [1] is based on partitioning the game's global world into geographical areas. Each of these areas are simulated by different server machines and are not necessarily of the same size. In MMOGs, the game semantics usually support this natural partitioning of the virtual world: each zone may be a country or city. The player may move between zones and the time needed to make the transition between different servers may be perceived by the player or not, it will depend on the game semantics [16].

Thus, each zone server machine is responsible for handling a subset of the players connected to the game and, therefore, they share the computational load gaining scalability.

2.2 Mirroring

As the name suggests, *mirroring* [14] is a technique based on having multiple servers holding a copy of the global state of the game. The responsibility of simulating the game world is equally shared among the servers.

A subset of the game entities is assigned to each of the servers. An entity which is in the subset of a particular server is called *active entity* for that server, whereas for all other servers that entity is a *shadow entity*. When computing a new game state, each server calculates only the states of its active entities. It then sends the new state of its active entities to the other servers, which will update their corresponding shadow entities. This technique relies on the fact that updating the state of a shadow entity using the message received from another server is much faster than calculating it, as was verified by Müller and Gorlatch in [14].

2.3 Instancing

Some researchers (e.g. [16, 15]) consider *instancing* to be a simplification of *mirroring*. This technique is based on distributing the load over a number of servers, each of which runs a parallel instance of high populated zones in the game. These instances are completely independent from each other, that is, players on different instances cannot see each other even if their coordinates are close. This technique is widely used in current MMOGs due to its easy implementation on MMOGs that use *zoning* [15].

2.4 Functional partitioning

In the *functional partitioning* [16] technique, different servers handle distinct functional properties of the game. Again, the objective is to share the load among the machines that constitute the server entity.

As an example, the game operators may allocate one machine just to handle the login requests, thus acting as a login server. They may also separate the functionality further by adding machines to act as patch servers, chat servers and VoIP servers. The machines allocated to handle the updates of the game state would then be called game server.

2.5 Shards

Some MMOGs, specially the most famous ones, have many clusters of servers, which allow many copies of the game to run simultaneously. These clusters are also called *shards*. Usually, each *shard* is located in different places, or even continents, of the real world. This allows players to choose to connect to the closest shard, thus reducing the network latency [13].

Note that with the presented techniques the server entity's responsibility is distributed among many server machines. It is called distributed client-server architecture, because these different machines all collaborate to play one, and only one, role: the server. These machines are usually geographically close to each other and connected through Local Area Networks [19].

In figure 1, it is possible to visualise an example of this architecture model. The functional partitioning is represented by the chat, login and patch servers. Inside the game server group, each sub server may be a zone, a shard, an instance or a mirror server.

3. PEER-TO-PEER ARCHITECTURE

In contrast with the client-server architecture, in the peer-to-peer (P2P) architecture every node has the same responsibility, that is, play the same role. The game will keep running even if one of the nodes is disconnected [13].

In this model, the game's global state is neither updated by nor kept in a centralised entity. Every player's machine must use its resources to render the game for the player and also to compute a part of its global state. The model can be

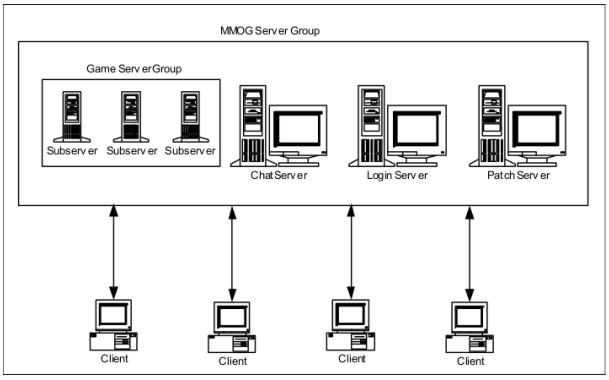

Figure 1: An example of a distributed client-server architecture. Figure taken from [13].

seen as a giant grid computer, where each part contributes with some calculations.

This architecture has received much attention by researchers due to the following advantages. Since the game state is simulated on the player's machines, this model provides better scalability, that is, with more players joining the game, there are more resources available. Additionally, since the game state is not kept in one single location, the P2P architecture is potentially immune to the single point of failure problem. Besides, since there is no central server having to handle all the information been transmitted, the risk of forming bottlenecks in the network is reduced [19].

There are also some drawbacks in this approach. All nodes are connected with each other and are sending messages in order to update the game state and maintain a consistent, shared sense of virtual world. This generates a high network traffic as the number of players grows and needs optimization [19]. A player only needs to know what is happening in his avatar's surroundings. This player's resources could be used to simulate only a certain area of the game world, and the player could only be updated about the events that happened in this area, the Area of Interest (AOI) [7].

Additionally, since the game state is administrated by the client's machines and player can directly access and update the global game state, this model is vulnerable to cheating [9].

The game world in a MMOG can be updated even when some of its players are not connected. Thus, it is necessary to have state persistence. The game must be able to store information about the player such as his possessions. The players expect the game to retrieve the information between login sessions [7]. A recent survey done by Gilmore and Engelbrecht in [8], however, have shown that no current storage approach is well suited for a P2P MMOG.

Yet another issue resides in the fact that Non Playable Characters (NPCs) are an important part of a MMOG and they need to be provided to the players. Whereas in the traditional client-server architecture they are hosted in the game servers and consume a significant amount of resources, in the P2P model they must be hosted in the client's machines [7].

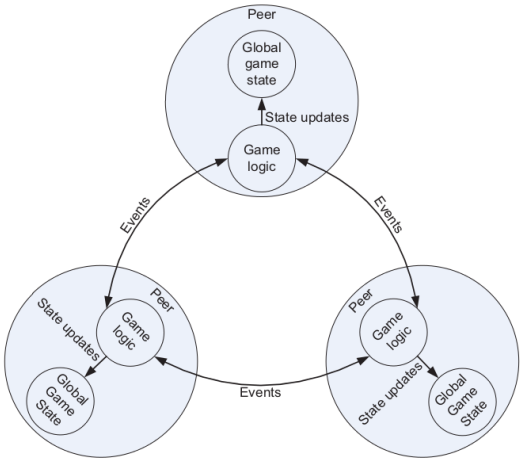

Figure 2: An example of a peer-to-peer architecture. Figure taken from [11].

The problem of selecting which players will have the task of hosting each NPC is an important subject for research [11].

Moreover, there are some business issues with this approach: since the players never connect to a server operated by the game publisher, it is difficult to charge the customer for fees related to the playing service. In addition to that, since this model relies on the players contributing with their computational resources, there must be an effective way to encourage them to do so, such as a rewarding mechanism [19].

Figure 2 presents an example of a Peer-to-peer architecture. It is possible to see there is no central entity ruling the global game state, instead, every player is responsible for keeping and updating a part of it. It is also possible to see the high network demand as each player has to communicate with many players to maintain consistency.

4. HYBRID ARCHITECTURES

The hybrid architectures try to combine the advantages of both client-server and peer-to-peer architectures. The objective is to provide better scalability than the client-server model, at lower costs to the game publisher, while retaining desired characteristics such as the control over the game state and easy ways to charge the player for the playing service [3].

In this model, a centralised server maintains the game state (keeps it consistent and persists it) and makes operations such as authentication, authorisation and content-distribution. These operations are far less resource consuming than the operations done by the server in the client-server architecture. Therefore, the hardware cost for these servers would be significantly reduced [2]. Additionally, a sufficient quantity of clients are selected, according to their computational and network resources, and each is left in charge of mediating the communication between a designated subset of the players and the central server. This also decreases the cost of the game publishers on network bandwidth for the central server [4]. The task relied to these selected players is to handle non state-changing moves, also known as positional

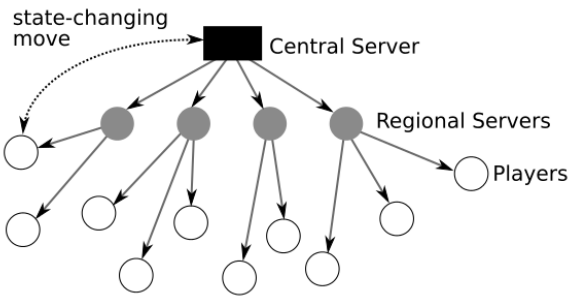

Figure 3: An example of a hybrid architecture. Figure taken from [10].

moves [10], which are the majority of actions taken by players in a typical MMOG [12].

These selected clients are called regional servers [10]. The central server selects them by measuring their computational resources, bandwidth capacity and latency. The low latency is important because the updates from the central server will arrive to the players through their regional server. Since measuring the latency between regional servers and players is difficult, because the set of players changes over time, the central server will, in most hybrid architectures, select the nodes which are closest to them [10].

The hybrid architecture also has some challenges to overcome. A system that implements a hybrid architecture must be capable of adding new peers, that is, allocating them to a group under one of the regional servers. The system will need to efficiently distribute the work load among the regional servers as well as being able to redistribute it if one of them becomes unavailable and avoid data loss when it so happens. They must also resist to attempts of cheating, especially when made by players with more access to game functionality (the regional servers). Also, hybrid MMOGs, just like the peer-to-peer, need to provide some sort of incentive to users, so they can make their resources available. It is necessary that this be provided keeping the costs as low as possible [3].

Figure 3 presents an example of a Hybrid architecture. The figure shows that for each player there is a regional server which will take care of non-state-changing moves. Whereas the state-changing moves will he handled directly by a central server.

5. COMPARISON

In this section, the selected criteria for the systematic comparison are presented and then each architecture is discussed according to the criteria.

5.1 Scalability

Scalability is perhaps the biggest challenge in MMOGs [5]. Being scalable means the architecture can support more and more players as they join the game without running out of resources. These resources can be hardware, to simulate the game world, network bandwidth, to communicate to players about updates, and network latency, to keep the game interactive.

Scalability is the most common issue in client-server architectures [16]. The solution is to add more hardware, increase cluster sizes, in order to support the growth in player numbers. The distributed server capacity must be balanced in accordance to resource necessity during peak loads, that is, the periods when the number of connected players is the highest. This causes another problem: in other times, apart from the peak times, when the resource necessity is lower, much of the already acquired (and paid) resources will be underutilized.

In peer-to-peer architecture, as players join the game, they also bring with them their computational and network resources, which are added to the total available processing power and network bandwidth [5]. Still, when developing a P2P architecture for a MMOG, one should not take this scalability for granted: without a efficient way to share work load between the peers, it is easy to overwhelm any peer's resources and cause a bottleneck in the network [3]. In peer-to-peer schemes, the players' machines' upload bandwidth are highly more used them in a client-server machine. This is a potential bottleneck for the network since upload speeds in household machines are significantly lower. This can also increase in case the players use other applications that consume network resources, such as VOIP applications, which are commonly used in conjunction with MMOGs [11].

In hybrid architectures, the central server shares the processing and network loads with some of the players. While the server handles state-changing moves, the selected players' machines handle the non state-changing moves, such as positional moves. This division of resources greatly reduces the bandwidth requirements for the server and allows a better scalability than the client-server architecture, but is less scalable than the peer-to-peer, since many players can join without contributing with their resources. It is important that the server chooses wisely which players are assigned as regional servers, because they will handle some of the load for a group of other players. If a player with a big latency is chosen, all players assigned to that regional server are going to suffer from high latency [10].

5.2 Persistency

In a MMOG, the game world always evolve over time, even when players are not connected to the game. In other words, the game must store all information about players, such as their profile and possessions between each login session. When players log back into the game, they expect to continue playing from where they left off. Therefore, a MMOG is a persistent world [7].

The centralised server solution present in the client-server architecture grants a high level of control over the system, which makes persistence possible. In a client-server architecture, the information of the player's characters are kept in the data centres. Every time the user logs back, the server just retrieves the information from its database [3].

In a peer-to-peer environment, however, this is much more difficult to obtain. In a classical P2P application, data may disappear when the last peer holding that information disconnects [17]. In [7], Fan et al. concluded their session on state persistency claiming that the persistency area of P2P

Table 1: Comparison Summary

	client-server	peer-to-peer	hybrid
Scalability	Costly	High and free	Lower costs
Persistency	Easily implemented	Still immature	The central server deals with persistency
Consistency	Easily implemented	Trade-off: consistency or interactivity	easier than P2P
Cost	High costs	Little or even non-existent	Lower costs than client-server
Security	Easiest among the three	Harder to secure	Easier when compared to P2P

MMOGs is still immature and many problems are still waiting to be investigated. In their survey on state persistency in P2P architectures [8], Gilmore and Engelbrecht came to the conclusion that none of the current P2P architectures for MMOGs have a suitable state persistency scheme.

Since state-changing moves are dealt by the central server in hybrid architectures [10], these architectures deal with state persistency in a way very similar to the client-server architecture. Once the central-server receives a state-changing move from one of the players, it updates the state of the game, which can be stored in the data centre, and then sends the update to the regional server, which in turn, will propagate the change to the other players in the region.

5.3 Consistency

As players interact with the game world in real time, the player's machines must have an updated copy of the game state, or at least a copy of a subarea of the world that interests the player. Thus, the MMOG must provide a consistent view of the game world and be able to workaround inconsistencies in order to give the players a good gameplay experience [5].

In the client-server architecture, the presence of a centralised entity makes game consistency easier, since the server is the only one to make updates in the game state, that is, there is only one version of the game state: the one in possession of the server. However, network latency may cause inconsistencies on the client-side [16].

In general, peer-to-peer architecture have poor support for deleting and updating content, therefore, the task of keeping consistency among copies of the game world is a challenge, especially because MMOGs enforce chronological ordering of the events, i.e. the state-changing events must be dealt with in order, according to the physical time at which they occurred [17]. Although many consistency models for P2P MMOGs have been proposed, there is always a trade-off between consistency and interactivity [16].

On hybrid architecture, the task of keeping the game world consistent is much more easy, when compared to the P2P model, because every state-changing action, that is, actions that can possibly cause inconsistency, made by players are handled directly by the central server [10].

5.4 Cost

MMOGs are regarded as one of the most profitable segments of the entire online game market. In order to get an MMOG up an running, the game operators must spend money in order to build the necessary hardware and network structure on which the game will run. Therefore, the choice of architecture for the MMOG has a direct impact on the initial and maintenance costs [13]. Thus, the cost will no doubt influence companies' decision on which architecture they should build their games.

Companies that choose the client-server architecture have to spend a lot buying hardware and employing staff to maintain it [11]. However, it is argued in [4] that the cost to contract the necessary network service is the real issue. Jones et al. [11] also mention great costs on energy and floor space.

On the other hand, in peer-to-peer architectures the computational and network resources needed are brought by the new player who joins the game, and the game operator does not need to buy it. Also, there is no central hardware which needs to be maintained and there is no need for staff to do it. Floor and energy are also unnecessary. Therefore, the cost to keep the game running is little or even none [3].

In hybrid architectures, since there are powerful servers which still deal with crucial activities, such as persisting the game, keeping it consistent and handling user login and authentication, there is a cost related to acquiring the necessary hardware and maintaining it. But, since some players will contribute with computational and network resources and share the load, the hardware does not need to be as powerful and costly as the client-server model. Thus, a hybrid architecture can deal with large populations at a much lower cost than the client-server architecture [3].

5.5 Security

One can define cheating in a MMOG context as a player exploiting the game design and implementation in order to gain any sort of advantage over other players [5]. Since in most MMOGs, some level of competition between players is expected, the architecture must have some way of detecting an attempt to cheat and be able to reject it. Because the MMOG business model is based in the number of users, it is extremely important to give the player a fair game experience [16].

In a client-server architecture, the centralised entity is capable of analysing and validating every action request made by a player before propagating it to other players in the game. This gives the client-server model an easier way to detect and prevent cheating [7].

Peer-to-peer architectures, on the other hand, let the client nodes manage the game world of the MMOG [16]. Since the players have the responsibility to manage the game data and handle the events that happen in shared environments, this model is vulnerable to unscrupulous players [9]. In [7], Fan et al. arrive to the conclusion that P2P architectures are

harder to secure when compared to the client-server architecture. Although they point out that research into cheat mitigation techniques are starting to produce interesting results.

Since hybrid architectures maintain a central control over the game state changing actions done by players, it is easier for these architectures to detect and mitigate cheats attempts [10]. However, the regional servers, are player nodes with some access to game functionality, therefore, resisting cheat attempts made by these nodes is extremely important in this architecture [3].

A summary of the comparison can be found in Table 1.

6. CONCLUSIONS

Although very costly to build and maintain, the client-server architecture is still the architecture of choice in the industry due to the higher control the game operators have over the game.

Even though the peer-to-peer architecture is very promising, there are still many issues to overcome, and much research that needs to be done, before peer-to-peer MMOGs can become a reality in game industry. Persistency and consistency are the main issues to address, while security is only beginning to produce meaningful results.

In the meantime, the hybrid architecture appears to be an interesting alternative to game publishers with limited budgets. Since it requires less powerful and expensive hardware, while still keeping some of the advantages of the client-server architecture, it could help them develop and release their games.

7. REFERENCES

[1] W. Cai, P. Xavier, S. J. Turner, and B.-S. Lee. A scalable architecture for supporting interactive games on the internet. In *Proceedings of the sixteenth workshop on Parallel and distributed simulation*, PADS '02, pages 60–67. IEEE Computer Society, 2002.

[2] C. Carter, A. El Rhalibi, and M. Merabti. A novel scalable hybrid architecture for mmog. In *Multimedia and Expo Workshops (ICMEW)*, pages 1–6. IEEE Computer Society, July 2013.

[3] A. Chen and R. R. Muntz. Peer clustering: a hybrid approach to distributed virtual environments. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, NetGames '06, New York, NY, USA, 2006. ACM.

[4] K. chul Kim, I. Yeom, and J. Lee. Hyms: A hybrid mmog server architecture. *IEICE Transactions on Information and Systems*, E87-D(12):2706–2713, December 2004.

[5] A. Denault and J. Kienzie. Journey: A massively multiplayer online game middleware. *IEEE Software*, 28(5):38–44, September 2011.

[6] S. Douglas, E. Tanin, A. Harwood, and S. Karunasekera. Enabling massively multi-player online gaming applications on a p2p architecture. In *In Proceedings of the IEEE International Conference on Information and Automation*, pages 7–12. IEEE, 2005.

[7] L. Fan, P. Trinder, and H. Taylor. Design issues for peer-to-peer massively multiplayer online games. *Int. J. Adv. Media Commun.*, 4(2):108–125, Mar. 2010.

[8] J. Gilmore and H. Engelbrecht. A survey of state persistency in peer-to-peer massively multiplayer online games. *Parallel and Distributed Systems, IEEE Transactions on*, 23(5):818–834, 2012.

[9] T. Izaiku, S. Yamamoto, Y. Murata, N. Shibata, K. Yasumoto, and M. Ito. Cheat detection for mmorpg on p2p environments. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, NetGames '06, New York, NY, USA, 2006. ACM.

[10] J. Jardine and D. Zappala. A hybrid architecture for massively multiplayer online games. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '08, pages 60–65, New York, NY, USA, 2008. ACM.

[11] A. Jones, L. Liu, N. Antonopoulos, and W. Liu. An analysis of peer to peer protocols for massively multiplayer online games. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, pages 1994–1999, 2012.

[12] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 96–107, 2004.

[13] M. Merabti and A. El Rhalibi. Peer-to-peer architecture and protocol for a massively multiplayer online game. In *Global Telecommunications Conference Workshops*, pages 519–528. IEEE Computer Society, November 2004.

[14] J. Müller and S. Gorlatch. Rokkatan: scaling an rts game design to the massively multiplayer realm. *Comput. Entertain.*, 4(3), July 2006.

[15] R. Prodan and V. Nae. Prediction-based real-time resource provisioning for massively multiplayer online games. *Future Gener. Comput. Syst.*, 25(7):785–793, July 2009.

[16] L. Ricci and E. Carlini. Distributed virtual environments: From client server to cloud and p2p architectures. In *High Performance Computing and Simulation (HPCS)*, pages 8–17. IEEE Computer Society, July 2012.

[17] G. Schiele, R. Suselbeck, A. Wacker, J. HaĬĹhner, C. Becker, and T. Weis. Requirements of peer-to-peer-based massively multiplayer online gaming. In *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium*, pages 773–782, 2007.

[18] S. Singhal and M. Zyda. *Networked Virtual Environments: Design and Implementation.* Addison-Wesley Professional, New York, 1999.

[19] M. Zhu, A. I. Wang, and H. Guo. From 101 to nnn: a review and a classification of computer game architectures. *Multimedia Systems*, 19(3):183–197, June 2013.