Architectural Frameworks of Massive Multiplayer Online Games Similarities And Differences

Table of Contents

[Abstract 2](#_Toc121912568)

[INTRODUCTION 3](#_Toc121912569)

[Architectures 3](#_Toc121912570)

[Client-server architecture 3](#_Toc121912571)

[Multi-server architecture 6](#_Toc121912572)

[(P2P) architecture 7](#_Toc121912573)

[Other architecture 7](#_Toc121912574)

[Issues with architectural decisions 8](#_Toc121912575)

[Colyseus 8](#_Toc121912576)

[Platform for Distributed-organized Information Terra (doit) 9](#_Toc121912577)

[Kosmos Game 9](#_Toc121912578)

[SOA for MMOG that is loosely coupled 11](#_Toc121912579)

[Client-server architecture 12](#_Toc121912580)

[Multi-server architecture 12](#_Toc121912581)

[Peer-to-Peer architecture 12](#_Toc121912582)

[UNIVARIATED OPINIONS ON GAME STATE 13](#_Toc121912583)

[Colyseus 13](#_Toc121912584)

[Doit Platform 14](#_Toc121912585)

[Kemos Game 14](#_Toc121912586)

[Distribution of the Load 16](#_Toc121912587)

[Network Bandwidth 16](#_Toc121912588)

[Scalability 17](#_Toc121912589)

[PERFORMANCE COMPARISONS 18](#_Toc121912590)

[Analysis and Future Trends 18](#_Toc121912591)

[Infrastructure 20](#_Toc121912592)

[Architecture 21](#_Toc121912593)

[Performance 22](#_Toc121912594)

[Scalability 22](#_Toc121912595)

[Persistence 23](#_Toc121912596)

[Security 23](#_Toc121912597)

[CONCLUSIONS 24](#_Toc121912598)

[Bibliography 25](#_Toc121912599)

# Abstract

Massive Multiplayer Online (Role Playing) Games (mmogs or mmorpgs), which concurrently interact in persistent, virtual, online, multiplayer-only worlds, are growing in popularity with the expansion of the IT sector and the success of the electronic gaming business. Different mmogs use a variety of architectural styles, each one selected for certain reasons. There is a demand for a very big shared state with many actors accessing it simultaneously in a variety of resource-intensive applications, such as Massively Multiplayer Online Games (mmogs) and large-scale simulations, which increases the risk of competing resource use. The assessment and comparisons will provide information on how to pick an appropriate architecture for an MMOG with specific problems.

In this paper, we will show different architectures with their primary concerns and systematic mapping study of the state-of-the-art in software technology. We will also look into how these designs maintain a consistent picture of the game state given a large number of players engaged. In addition, we'll examine the performance characteristics of different systems, including load distribution, network bandwidth, and scalability. We classify the chosen studies according to the deduced criteria to compare their methodologies, expose the difficulties each study encountered, and expose any potential research trends. Finally, we make an effort to pinpoint future research areas that seem likely to make it possible to employ standardized technology for this class of systems.

# INTRODUCTION

Millions of people interact and play in diverse virtual worlds with strangers or friends thanks to the popularity of massively multiplayer online games (mmogs). Everquest from Sony and World of Warcraft from Blizzard are two of the most Ill-known examples. A. Gamers are being forced to become more competitive by the quickly expanding MMOG market, which highlights the significance of entertainment computing technology. Massive Multiplayer Online Games (mmogs) generally employ one of three primary architectures: a) client-server architecture; b) multi-server design; or c) peer-to-peer (P2P) architecture. Generally speaking, I will describe each of them in terms of their constituent parts and common MMOG behaviors.

# Architectures

Massive Multiplayer Online Games often employ one of three major architectures:

1. Client-server architecture
2. Multi-server architecture
3. Peer-to-peer (p2p) architecture is an option

In general, we shall describe each of them in terms of their elements and common MMOG behaviors.

## Client-server architecture

Depicts a typical client-server architecture for an MMOG in (Figure 1). In this design, the server serves as the central controller, connecting to each client and keeping track of the overall game state. It also stores all copies of avatars and items. On the server, all player interactions and updates take place. The clients can obtain from the server essential data about the whole planet. Since the server is the only controller in this design, deployment, maintenance, and modification are relatively straightforward. The client-server architecture is a distributed framework that distributes the burden across several resource requesters and resource providers (referred to as servers) (called clients). Messages in a predefined language are used by machines in a distributed system to communicate. In a normal client-server system, the client can send these messages to the server to request resources, and the server will reply by sending the requested resource back to the client. Client devices are unaware of one another and are unable to communicate with one another unless the servers specifically send messages to them.

In their work, Assiotis and Tzanov cover MMOG designs. The authors suggest a client-server design, often known as a centralized distributed architecture, to accommodate several concurrent users without compromising performance or security. The area of interest serves as the focal point of this inquiry. With the aid of this idea, game designers may divide expansive worlds into smaller sections that can be hosted on various nodes. This design enables the distribution of bandwidth and computing power requirements among several nodes. This architecture, nevertheless, brings forth fresh difficulties:

* Particularly if these locations are remote, players may not always be interested in learning about changes for specific parts of the map.
* Even when two players are hosting on different nodes, they still need to be able to view and communicate with one other when they are close to the boundary between two portions.
* For events that take place close to borders and have an impact on players on both sides, it's possible that the game state will be invalid regardless of the synchronization mechanism used.

The authors offer a notion called Area of Interest to address the first issue (aoi). According to this idea, each player has their own Area of Influence (aoi), which extends a given distance from their position and is where they can get event notifications. Naturally, players are not interested in hearing updates on events that are too far away for them to see or hear. Depending on the player type, an aoi's size can change. Because of the range of his weaponry, a player wielding a sniper rifle, for instance, needs a greater aoi than a player using a handgun.

The authors have identified four main situations that need to be handled when players are close to border regions for the second and third problems:

* Event updates from both servers must be accessible to a player standing on the dividing line between two zones run by distinct servers.
* A player's unexpected movement to a location controlled by a separate server is known as "teleporting" in video games.
* A server's coverage area may be crossed by an event that started on that server. A common illustration of this is launching a rocket that moves from one location to another before detonating.
* A border-area event may have an impact on many areas, each of which is hosted on a distinct server. A bomb going off at a border and impacting players in neighboring territories is an illustration of this.

The authors offer a unique approach for distributing large-scale worlds on a client-server architecture by proposing additional solutions to these issues, such as subscribing a player to both servers when they are close to their border. According to the study's findings, the authors were able to increase the client-server architecture's effectiveness and, as a result, the performance of mmogs.

In particular, the game server receives and processes orders from the clients to mimic a world through computational and data activities. The server computes a global state of the game world based on these commands, which depicts the locations of entities and the interactions that are occurring between them. Finally, the client devices that display this information to the player get answers from the server that include the updated state. Operating an effective architecture is crucial for this reason. The authors outline three parallelization methods typically used with client-server systems to enable thousands of concurrent players:

* Zoning: Divides the gaming environment into regions that are "managed independently by distinct computers." This method is very helpful in leisurely games like mmorpgs.
* Replication: Parallelizes game sessions where several players congregate in key areas. Each server synchronizes the state of other "shadow entities" that are based on other machines while computing the state of a number of "active entities" that are based on it. The majority of fast-paced games, including first-person shooters, adopt this strategy.
* Instancing: "Starts many concurrent instances of heavily crowded zones to distribute the session burden." These areas are apart from one another.

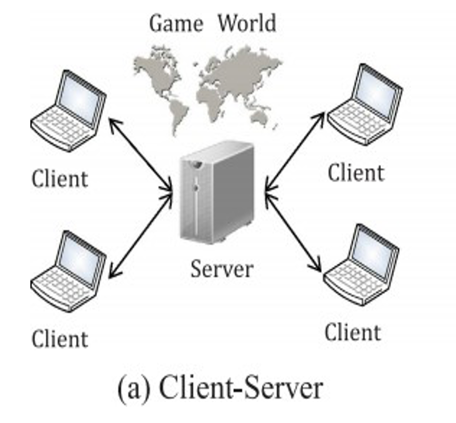


Figure 1: Client-server architecture

## Multi-server architecture

The client-server architecture is expanded by multi-server architecture (shown in Figure 2(b)), frequently as a result of a high user base. With this design, the global game world is typically split up into numerous areas, each of which has its own client-server setup. It implies that each server has its own clientele and a separate, larger universe for which it is accountable. Through server-to-server communication, each area is connected to the others, allowing the servers to exchange information about their respective global worlds and maintain a stable overall game state. But in the majority of mmogs, gamers in one region are cut off from those in other zones. In order to facilitate this communication across several areas, there is a handoff mechanism between the servers.

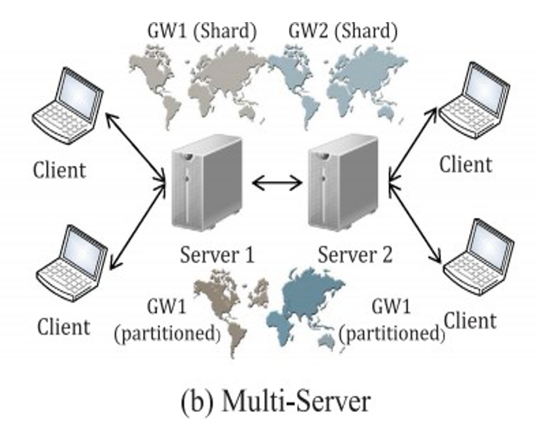


Figure 2: Multi-server architecture

## (P2P) architecture

In order to increase scalability, a peer-to-peer (P2P) architecture is suggested (see Figure 3(c)). The copies of avatars and items are handled by each peer, which also manages some aspects of the overall game state. Each peer functions as both a client and a server. Due to the fact that the increased load will be dispersed across more nodes and that additional resources will be added by adding new nodes, this design can scale up quite well.

They contend P2P:

* Reduces communication delays and gets rid of localized congestion.
* Enables users to start their own games with little financial outlay.
* Enables games to get around server-only calculation bottlenecks.
* Has no single point of failure, making it more accessible and durable.

In addition, the authors describe how mmogs can utilize peer-to-peer storage. Mutual exclusion must be used to ensure data consistency while using peer-to-peer storage. Unstructured and structured networks are the two sub-types of P2P architectures that are discussed. Clients in unstructured P2P can send files to one another directly, however in structured P2P, resource names are translated into network addresses through a distributed hash table.

They contend that this strategy has drawbacks that put scalability restrictions on it. On the other hand, the peer-to-peer strategy may be able to provide the following benefits:

* Inherent scalability, as the number of users, increases the number of resources available.
* Robustness—systems built on this architecture have the ability to fix themselves when a peer fails.
* As network traffic is dispersed among the users, bottlenecks are avoided.

## Other architecture

Other architectural types exist; one of them is a hybrid architecture in which client-server and peer-to-peer architectures can be combined. Mmogs can benefit from using designs like Service-Oriented Architecture (SOA), an example of which is provided in Section 1, because it is also feasible to learn from other distributed systems. The remaining portions of the essay are structured as follows: The primary considerations for selecting a certain type of architecture are listed in Section 2. In Section 3, we investigate how different designs preserve consistent views in the game state with a particular emphasis on consistent views. Finally, section V wraps up this study and examines the key takeaways. Section IV compares various designs in light of a few performance metrics.

# Issues with architectural decisions

Five particular architectures—each of which corresponds to a typical design described in section I—will be shown in this section. We outline the primary problems for each design, the factors that led to the choice of that particular architecture, as well as the specific methods that were employed in the projects to address those concerns. A comparison of all these architectures might provide some guidance on how to select architectures for mmogs.

## Colyseus

In Colyseus, the client-server architecture is expanded. In order to identify and duplicate items before they are accessed at a node, it makes use of the game's weak consistency and predictive workload and offers a query interface and pre-fetching subsystem. These factors led to the selection of this architecture:

1. Consistency

When activities are carried out simultaneously and potentially at odds with one another in a distributed system, inconsistent states might occur. For instance, if several players operate on an object in the game simultaneously, only the first successful action should be reported to all players, and they should all view the changes in sequence.

2. Simplicity

The architecture of mmogs needs to be as straightforward as feasible. Programmers like to utilize a straightforward design since it makes their jobs simpler. When employing a more straightforward design, system engineers may simply manage and control the whole game state.

## Platform for Distributed-organized Information Terra (doit)

Doit platform is a middleware for mmogs that uses a standard 4-tier design for implementation. The client layer, proxy/gateway layer, cell servers layer and database layer are conceptually different operations in this client-server architecture. These factors led to the selection of this architecture:

1) Developmentally Simple

An MMOG platform should offer simpler and quicker ways to create content because it normally takes two to three times as long to create and release a single-player game as compared to an MMOG.

2. Simplicity of Deployment

To offer a more enticing service, mmogs are liable to ongoing modifications to their content. All content must be updated remotely, which makes content deployment simple, especially in collocation scenarios.

3. Ease of Upkeep

In order to keep players at a reasonable cost, MMOG providers seek to offer assistance. Programs should be able to automatically identify players' states that are harmful in order to assure fairness in mmogs. This would help vendors retain users for the lowest possible cost.

## Kosmos Game

Based on the distributed architecture suggested in the , the Kosmos game is created. The project offers a method for distributing the game among numerous game servers that makes use of the location of interest. It employs a system that divides the vast virtual globe into smaller parts, with separate servers managing each sector. Depending on the player's position in the virtual world, clients can group together and view a single large virtual world. Unless the server tells the client, the client keeps sending actions to the same server. The following are the justifications for choosing this type of architecture:

1. Scalability

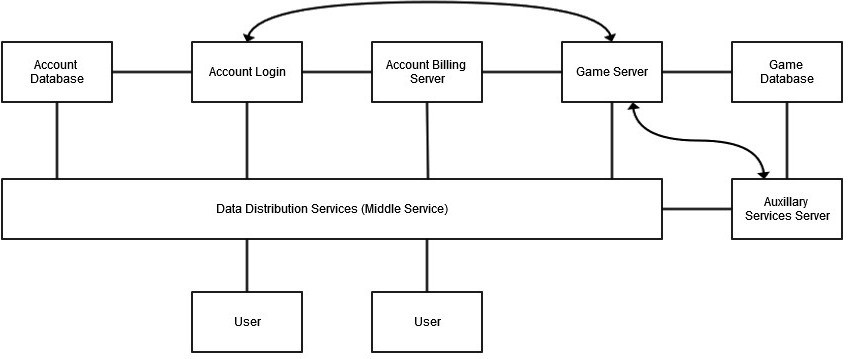
Another key issue with mmogs is scalability; the system's design needs to be able to grow as more people use it. With about 33,000 people, Eve Online asserts to be the game with the most active players at once. The game's architecture should be able to handle peak traffic while also allowing for future growth.

1. Tolerance for Error

Users of mmogs anticipate little to no downtime from the game servers, which must be operational at all times. The World of Warcraft publisher, Blizzard, has recently suffered losses due to network failures that are expected to cost $26,200 per hour.

1. Consistency

The basic concept of mmogs is the use of users' avatars to interact with other players and the environment. The players all contribute to the game's ongoing evolution.



## SOA for MMOG that is loosely coupled

A special middleware based on free source DSS is presented for Massive Multiplayer Online Games (mmogs) to satisfy real-time requirements. This middleware is a loosely connected Service Oriented Architecture (SOA). Additionally, as a real-time SOA's fundamental requirement is to guarantee service quality, the following are this architecture's primary concerns:

1. High Bandwidth

High-quality visuals or videos are sought after by MMOG providers since they will draw in more players and increase revenue. High bandwidth is typically required for better graphics, and this may be attained by selecting an architecture that can meet these needs.

1. Tight Timing Restrictions

Because it is crucial to preserve the same game state on each player's side and produce the impact for each player in mmogs, strict scheduling constraints for game state management are also a serious problem. Therefore, an appropriate middleware should be selected for the MMOG.

|  |  |  |
| --- | --- | --- |
| Architecture/ Project name | Corresponding typical architecture | Main Concerns |
| Cloyseus | Client-server | +Consistency control  ++Simplicity  ++Easy Management |
| Doit Platform | Multi-server | +Easy Management and Operations |
| Kosmos Game | Multi-server | +Scalability  +Fault Tolerance  ++Consistency |
| The peer@play Project | Peer-to-Peer | ++Cost  ++Scalability |
| A loosely coupled SOA for MMOG | Service-Oriented | +High Bandwidth  +Strict Timing Constraint |

Table 1: Comparisons among the architectures

# Client-server architecture

We can utilize client-server architecture when consistency control and/or simplicity are the primary goals. Due to its simplicity, client-server architecture is the foundation of the majority of mmogs. Compared to peer-to-peer architecture, this design makes it easier for programmers to develop code. The highest degree of control over the game environment is provided by the server role.

# Multi-server architecture

We can utilize client-server architecture when consistency control and/or simplicity are the primary goals. Due to its simplicity, client-server architecture is the foundation of the majority of mmogs. Compared to peer-to-peer architecture, this design makes it easier for programmers to develop code. The highest degree of control over the game environment is provided by the server role.

# Peer-to-Peer architecture

With regard to cost and scalability for one MMOG, the peer@play Project expands P2P architecture. P2P designs often offer the most potential for scaling of any architecture since each peer adds fresh resources to the system. Another benefit of P2P is that the game provider incurs no additional costs for the addition of these resources because peers handle the load and expensive servers are not required.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Properties | Client-server | Multi-server | Peer-to-Peer | SOA |
| Consistency | Good | Great | Average | Average |
| Scalability | Bad | Good | Great | Average |
| Cost | Bad | Bad | Great | Bad |
| Easy management | Great | Average | Average | Bad |
| High bandwidth | Good | Bad | Good | Good |

TABLE 2. Evaluation of MMOG architectures

# UNIVARIATED OPINIONS ON GAME STATE

For various players to engage with one another in the fictional world created by the game logic, mmogs need consistent state. If a player takes an action, for instance, in a game of chess, the content should be updated and shared in accordance with that action so that the other players may view the consistent state of the game. Consistency requirements are computed and maintained by the servers in client-server and multi-server systems. In a peer-to-peer architecture, each peer computes its own local image of the game logic based on the data it gets from the other peers, replacing the need for a server. To account for network latency for different peers, several synchronization protocols are employed.

## Colyseus

The consistency criteria are ensured by using the single copy consistency model. The consistency model of a conventional client-server architecture forms the foundation of this strategy. Following is an explanation of the consistency mechanism:

Any node that is weakly consistent to the primary copy of an object's replica in Colyseus is permitted to make read-only duplicates of any object. With this strategy, read access latency is reduced and a timely, consistent game state is provided. It calculates the number of copies required in advance to carry out the game's logic using a distributed hash table (DHT) and a dynamically load-balanced range-based DHT.The TACT is used by Colyseus to demonstrate that it is adaptable to accommodate games with various perspective consistency. To examine the effects of discovery latency on view consistency, view inconsistency is also taken into consideration. The findings show that opinion inconsistency has very little of an effect.

As a consequence, Colyseus offers the aforementioned techniques to keep a constant perspective on the game state and produces results to support the design.

## Doit Platform

Unfortunately, the design for the doit platform that is suggested in [4] does not give any context or information on the issues with consistency. However, because it is a standard 4-tiered MMOG design, the consistency issues are handled by the cell servers layer. The core of this architecture, which manages and runs the game logic's virtual environment, is the cell server layer. Through the players' clients, this layer receives the instructions from the players, validates them, and then executes them. It then provides updates to all players who are impacted by the commands. In the virtual world created by the game logic, this layer keeps track of everyone's status.

The general 4-tiered MMOG design successfully handles the consistency concerns, despite the fact that this study does not discuss them.

## Kemos Game

The following two consistency requirements must be met in the Kemos game:

* The sequence of events that impact the state of any given object must be the same for all clients.
* Events that have an impact on any specific item increase monotonically.

In order to meet the aforementioned consistency standards and provide its gamers with higher-quality service, Kemos Game employs ID tracking technology. Human player inputs make up the majority of the game's inputs, and these inputs are what generates most of the events. For instance, when a player fires a rocket, the player produces an action and transmits it to the server to which it is connected.

E. A SOA for MMOG that is loosely coupled

Sadly, the design suggested in [9] prevented different players from maintaining a consistent picture of the game state. At the very least, the report makes no mention of consistent viewpoints.

Table III provides comparisons of the consistent viewpoints in each unique architecture. We may infer from the table that all conventional architectures provide consistent views in order to satisfy consistency requirements, whereas SOA-based design does not. Different procedures are used for each architecture to satisfy its consistency requirements.

|  |  |  |
| --- | --- | --- |
| Architecture/ Project name | Offer consistent views? | Mechanisms to maintain consistent views |
| Cloyseus | Yes | Range-based DHT and dynamic load balancing  View consistency using the TACT model |
| Doit  Platform | Yes | Generic 4-tiered MMOG architecture's cell server layer |
| Kosmos Game | Yes | ID tracking mechanism |
| The peer@play Project | Yes | Graph of a k-connected network using a string of happenings model of update-linearizability and consistency |
| A loosely coupled SOA for MMOG | No | N/A |

TABLE 3. Comparisons of consistent views

# 

# Distribution of the Load

Colyseus

The load-balancing method it employs, known as leave-join, dynamically switches lightly loaded nodes to heavily loaded nodes.

Doit Platform

Shards are used to create a generic 4-tier architecture, which effectively manages dynamic collaboration between cell servers and distributes the processing of game interactions among a number of servers throughout the network.

Kemos Game

While the game is running, it rearranges the game environment to dynamically balance the server and network demand. When using the technique described in the article, the region managed by one server is divided into two or more sections, and the congested parts can be.

The peers@play Initiative

The system tries to move game chores to a new peer when one enters the game in order to evenly distribute the workload across all peers.

# Network Bandwidth

1) Colyseus

The three parts of the Colyseus communication cost are sending and routing publications, routing, and DHT maintenance.

2) doit Platform

In a standard 4-tier design, Gateways are the performance bottleneck.

3 Kemos Game

The architecture allows for smooth player switching between servers without the need for massive data bursts, and data flow between servers and from client to server is always confined to O(1).

4) The peers@play Initiative

In this project, the coordinator gathers all updates and distributes them to all peers inside its zone in a solitary, integrated update message. Update messages are delayed longer as a result, but the peer's communication capacity is used more effectively.

# Scalability

1) Colyseus

As the number of players in the games rises, the required scaling of the routing capacity for both DHT and range-DHT is not successful.

2) Scalability

Itis provided by the doit Platform Generic 4-tier design, which enables manufacturers to increase the number of servers in a cluster.

3. Kemos Game

It offers a straightforward method for scaling the system by adding more nodes and managing peak user numbers.

4) The peers@play Initiative

As a result of this project's P2P-based architecture, each peer contributes fresh resources to the system, allowing for good system scalability.

|  |  |  |  |
| --- | --- | --- | --- |
| Architecture | Load Distribution | Network Bandwidth | Scalability |
| Client-server | Good | Good | Bad |
| Multi-server | Good | Bad | Good |
| Peer-to-Peer | Great | Good | Great |
| SOA | N/A | Good | Good |

TABLE 4. Evaluation of architectures in terms of performances

# PERFORMANCE COMPARISONS

Some performances are of vital importance in mmogs, these parameters include load distribution, network bandwidth, and scalability. Load distribution is used to investigate how the load is balanced among the servers or nodes. Network bandwidth is important in MMOG because the real-time feature of online games replies on high network bandwidth. Scalability means that architecture should accommodate the growth of the users as the popularity of the system increases. In this section, we will present these three performances in each specific architecture, and list which mechanism they are using to achieve such a performance.

# Analysis and Future Trends

The creation and distribution of mmogs is a difficult yet promising field. The essential functionality of MMOG backends still strongly depends on custom-tailored backends, despite the fact that the rise of cloud computing has helped reduce operational expenses for big distributed systems. Table 4 provides a summary of our findings and details how the criteria listed in Section 4 are handled by the state-of-the-art at this time. Research by the same authors or studies carried out in the same field have been compiled into a single entry for analytical reasons.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Approach | Infrastructure | Architecture | Scalability | Persistence | Performance | Security |
| [1] C. GauthierDickey, D. Zappala and V. Lo, | D | P2P | NS | U | U | L |
| [2] Assiotis and Tzanov | D | CS | NS | U | M & P | L |
| [3] A. Shaikh, S. Sahu, M. Rosu, M. Shea and D. Saha, | Prc | U | E | R | M, S & P | U |
| [4] Lu et al. | D | CS | MS | U | S & M | U |
| [5]Jardine and Zappala | U | H | MS | U | S & P | T |
| [6] Chu | D | CS | MS | R | U | T |
| [7] Blackman and Waldo | D | U | MS | R | U | U |
| [8] V. Nae, R. Prodan and losup, | Prc | CS | E | U | S, M & P | U |
| [9] J. Baker, C. Bond, J. Corbett, J. Furman, | Prc | U | E | N | S & O | U |
| [10] C. Weng and K. Wang, | Prc | CS | E | U | M & P | U |
| [11] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. | Prc | U | E | N | S & P, O | U |
| [12] W. Chuang, B. Sang, S. Yoo, R. Gu, M. Kulkarni and C. E. Killian, | U | H | MS | U | S | U |
| [13]J. Plumb and R. Stutsman, | Prc | P2P | MS | U | S | L |
| [14] D. Meiländer and S. Gorlatch | Prc | U | E | U | S & M,P | U |
| [15] Plumb, S. Kasera and R. Stutsman, | U | H | MS | U | S | U |
| [16] S. Farlow and J. Trahan | U | CS | MS | U | S | U |
| [17] M. Ghobaei-Arani, R. Khorsand and M. Ramezanpour | Puc-iaas | CS | E | U | M & P | U |

Table 4. Using the defined criteria, comparing the examined methodologies (Infrastructure, Architecture, Scalability, Persistence, Performance and Security).

## Infrastructure

We can see that the majority of the articles that we looked at (55%) talked about using private cloud infrastructure to deploy MMOG backends. The public cloud iaas layer infrastructure is ranked second at 14%, after the dedicated method, which is still popular (21%). It is important to note that the proportion obtained by adding up all different kinds of cloud infrastructures is 79%, underscoring the current trend of using cloud infrastructures to power MMOG backends.

Figure 4, which depicts the utilization of various types of infrastructure across time, further supports this. Prior to 2010, while cloud computing was still relatively new, the dedicated method predominated; however, advancements after 2010 seem to have sparked a move toward several sorts of cloud environments, particularly private clouds. A minor portion of these methods have also made use of public cloud services. Iaas is most frequently employed in these situations (66%), and paas is utilized in the remaining 33%. Although there are good reasons for this, such as the operators' requirement for complete performance control, it can also be claimed that this prevents the solutions from utilizing the benefits of public clouds, such as their elasticity, cheap cost, high availability, global coverage, and so on.

Figure 4. Choice of infrastructure over time—as derived from the studied works.

## Architecture

Peer-to-peer architectures were used less frequently (31%) than the well-known client-server paradigm (52%), according to the papers' architectural choices. At least among scholars, there also seems to be a tendency toward hybrid structures. Figure 5 demonstrates how over the years, the examined articles have made an effort to build on various MMOG backend architectures. Although the client-server approach is a little more common, no one architecture seems to be clearly dominating or to be on the rise.

Figure 5. Choice of software architecture over time—as derived from the studied works.

## Performance

The performance of a game, albeit not in every case, is an important aspect since it has a direct impact on how well it is received by the players. The bulk of the publications we have examined used simulations for case studies and simulations of various kinds or to existing mmogs to test the effectiveness of their methodologies. The majority (73%) of the research used simulations to evaluate their recommendations. Only 24% of them used another kind of application, while 76% of them made use of mmogs, either one that already existed or ones that the creators created. The effectiveness of this technique for MMOG performance evaluation is highlighted by the use of simulations rather than mathematical models. It is also important to note that while only a tiny portion of simulations employed their own composite measures, the majority of them used straightforward metrics like latency, bandwidth, and overall resource usage to assess performance.

## Scalability

Scalability is a crucial need for MMOG backends. We may anticipate considerable swings in user demand, particularly in the early stages of launching a new game, so having an elastic strategy is crucial. The scalability of the methods outlined in our chosen studies is demonstrated over time in Figure 6. Overall, 94% of these methods can be scaled manually or automatically, depending on the situation. This demonstrates how important scalability is for MMOG backends. In addition, we see that non-scalable methods were only applied in the early days of mmogs and that they were gradually phased out in favor of more scalable and elastic methods.

Figure 6. Scalability over time—as determined in the studied works.

## Persistence

The choice of data persistence mechanism has an impact on the scaling effectiveness and performance of a game. Most games need data persistence to retain data like the game state and user information (particularly those with persistent environments). Persistent storage is made possible by database systems, however rdbmss and nosql systems differ greatly from one another. According to Table 4, 5 of the 8 studies (63%) utilised nosql systems, while the other 37% made use of rdbmss. More significantly, we can see that whereas nosql was utilized in every study published after 2010, RDBMS was used in every study published before 2010. There is a strong trend towards employing nosql databases in mmogs, even though the overall conclusions of this study cannot be confirmed or extrapolated to include all known investigations.

## Security

Finally, it seems that most MMOG developers disregard security. Only 6 of the 34 items in our table even mention this criterion, which is indicative of the situation. But it's important to recognize the value of security. Security is a crucial component of any MMOG since a growing number of contemporary games provide microtransactions involving real money and keep user data on their servers. Regardless, P2P architectures, where security must be handled with greater care, are frequently brought up when talking about these security regulations. Most research (67%) prefer to employ "loose security" in these situations, which is security dependent on algorithms and encryption rather than architectural design. In order to protect sensitive data, tight security—which is based on architectural design—relies on hybrid architectures that leverage the client-server model at the system's core.

It should be emphasized that the methodologies we have looked at do not always completely correspond with the chosen criteria, which places limitations on our study. This is inevitable since many of the publications in the survey are research-focused and, as a result, do not comprehensively address every facet of the underlying system. For instance, few have openly examined the consequences of their recommendations for security.

# CONCLUSIONS

The distributed nature of mmogs has allowed many of them to become very popular very quickly. At the same time, MMOG developers and researchers have proposed novel solutions to overcome the challenges inherent in the development and deployment of such games and particularly their backends.

After reading twenty publications in this field, this paper provides an overview of the architectures utilized in massively multiplayer online games. We have defined criteria and categories for classifying the different approaches based on a focus analysis of several selected papers. Based on the identified criteria, we studied and analyzed relevant works, placed them into respective categories and then analyzed and summarized our findings.

Mmogs often consider a number of factors while choosing their architectures. As we examine these particular designs in this study, we highlighted the primary factors to consider while selecting each typical architecture. We also discovered that the majority of the MMOG systems we looked into actually offer consistent views through a variety of ways. In this article, we analyzed these specific designs in terms of load distribution, network bandwidth, and scalability. Varied architectures also result in different performance characteristics. The most crucial factor in building and implementing a distributed system, as we have learnt from our study, is the architectural choice. The articles also offered us a quick overview of architectures' concerns and unique performance demands. A constant viewpoint is the most important criterion for an MMOG system, the studies further shown. By developing new models, methods and tools this layer can be utilized and ultimately offer high-level cloud-based solutions for developing and operating MMOG backends more efficiently. These lessons will come in handy when we create our own distributed systems in the future.

# Bibliography

|  |  |
| --- | --- |
| [1] | C. GauthierDickey, D. Zappala and V. Lo, "Distributed Architectures for massively multiplayer online games.," 2004. |
| [2] | M. Assiotis and V. Tzanov, "A distributed architecture for massive multiplayer online role-playing games. NetGames’ 06 Proceedings of the 5th ACM SIGCOMM Workshop on Network and System Support for Games, Hawthorne," October 2005. |
| [3] | A. Shaikh, S. Sahu, M. Rosu, M. Shea and D. Saha, "Implementation of a service platform for online games. In Proceedings of the Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games,". |
| [4] | F. Lu, S. Parkin and G. Morgan, "Load balancing for massively multiplayer online games. In Proceedings of 5th ACM SIGCOMM workshop on Network and System Support for Games," 30–31 October 2006. |
| [5] | 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,* 21–22 October 2008. |
| [6] | H. Chu, "Building a simple yet powerful mmo game architecture.," 23 April 2019. [Online]. Available: https://www.ibm.com/developerworks/library/ar-powerup1. |
| [7] | T. Blackman and J. S. D. S. i. P. D. Waldo, "Technical Report; Sun Microsystems: Menlo," USA, 2009. |
| [8] | V. Nae, R. Prodan and Iosup, "Massively multiplayer online game hosting on cloud resources," 2011. |
| [9] | J. Baker, C. Bond, J. Corbett, J. Furman, A. Khorlin, J. Larson, J. Leon, Y. Li, A. Lloyd and V. Yushprakh, "Providing Scalable, Highly Available Storage for Interactive Services. In Proceedings of the Conference on Innovative Data System Research (CIDR)," USA, 9–12 January 2011. |
| [10] | C. Weng and K. Wang, "Dynamic resource allocation for MMOGs in cloud computing environments.," Limassol, Cyprus, 27–31 August 2012. |
| [11] | F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes and R. B. Gruber, "A distributed storage system for structured data. ACM Trans. Comput. Syst. (TOCS)," 2008. |
| [12] | W. Chuang, B. Sang, S. Yoo, R. Gu, M. Kulkarni and C. E. Killian, "Programming model and runtime support for tightly-coupled elastic cloud applications.," Santa Clara, CA, USA, 1–3 October 2013. |
| [13] | J. Plumb and R. Stutsman, "Exploiting Google’s Edge Network for Massively Multiplayer Online Games.," *IEEE,* 2018. |
| [14] | D. Meiländer and S. Gorlatch, "Modeling the scalability of real-time online interactive applications on clouds," 2018. |
| [15] | J. Plumb, S. Kasera and R. Stutsman, "Hybrid network clusters using common gameplay for massively multiplayer online games.," Malmo, Sweden, 7–10 August 2018. |
| [16] | S. Farlow and J. Trahan, "Periodic load balancing heuristics in massively multiplayer online games.," 7–10 August 2018. |
| [17] | M. Ghobaei-Arani, R. Khorsand and M. Ramezanpour, "An autonomous resource provisioning framework for massively multiplayer online games in cloud environment.," 2019. |