2IMN10: Architecture of distributed systems

Architecture of massive multiplayer online games

Essay Topic 3

Matej Grobelnik (1281674)
Bram van de Wall (0911838)
Keyshav Suresh Mor (1237978)

October 30, 2018

1 Abstract

Abstract ...

2 Introduction

In this essay we compare different architectural styles that can be found in MMOGs (Massive Multiplayer Online Games), and their conformance to the requirements of a MMOG.

3 Requirements

Before we can reason about the architecture of MMOGs, we must first consider the requirements that originate from such games. Additionally, we also need to acknowledge the differences between different games and genres, which may lead to different prioritization of these requirements. Unless specified otherwise, we will consider the requirements of MMO-RPGs.

3.1 Availability

MMOGs generally require constant availability of a shared state in order to function, as most of the functionality is derived from it – NPCs, other players, etc. The availability is especially important for a game to be successful [1], as downtime can significantly impact a game's growth. This is especially important for P2P-based architectures, as they need to be able to deal with NAT-/Firewall- ed players.[2, 3.5]

3.2 Consistency

While it is obvious that it would not be practical to require a strongly-consistent state of the world to be maintained, individual objects or groups of objects may need various levels of consistency depending on the game mechanics. For most games, interactivity is a much more important than strong consistency, therefore it is reasonable to only require the world to be eventually consistency. [3] For example, when a player destroys an object, it should not be possible for an other player to use the object after the destruction has been processed.

3.3 Interactivity

MMOGs are generally very interactive, while the tolerable delay between executing a command and seeing the results of it vary greatly between games and genres. If this latency is exceeded, players may get frustrated. [2, 3.6]

The acceptable latency also varies between different commands within a game. For example, players will most likely not mind a trade to take a few seconds, while such a latency for position updates may make a game unplayable.

Games may employ client-side prediction techniques to hide latency to the user, such as optimistic updates and interpolation. These techniques may not always be practical however, as prediction may be difficult or inaccurate.

3.4 Scalability

As the first M in MMO suggests, a MMOG needs to be able to handle a lot of concurrent players at once. P2P-Architectures are often mentioned in order to scale easily, but even for those systems proper care needs to be taken to not overload individual peers – which generally have a lot less available bandwidth than a server.[2, 3.7]

Another thing that may or not be important depending on the game is the scalability in the case of greatly localized load. This occurs when many players aggregate in a single spot, which may naturally occur depending on the game. For example: in some MMOs players naturally aggregate in a single location order to trade with each-other. This leads to a great localized load, as even without spatial partitioning of the world, all clients will still expect to receive updates of all nearby players.

Similarly, (localized) load may not always be predictable. For example, player-organized gatherings such as wars or raids may lead to thousands of concurrent players fighting concurrently. [4]

3.5 Security

A lot of items that can be obtained in MMOGs take a lot of time to gather, or are even purchasable using real-life currencies. This gives them real-world economic value and opens up the possibility of virtual crime. [2, 3.8] Where possible, such crimes should be prevented, or at least logged. Especially in P2P-based architectures this is important, as clients can directly mutate the state they are responsible for.

3.6 Persistence

The shared state of an MMOG should be persisted. In P2P-based systems, state should be replicated in order to prevent loss of information when a client goes down.

4 Architectures for sharing state

In this section we will describe two different architectures for MMO-RPGs.

4.1 A Traditional MMOG Architecture

In this section we'll describe a traditional client/server architecture. The client/server in this model mostly refers to the connection between the game client and the game's infrastructure. This infrastructure is made up out of many different components and services with different responsibilities, but are all hidden behind a reverse proxy.

4.1.1 Servers

In this model, the world is partitioned into regions. A dedicated server process is allocated to every region. This server is responsible for everything that occurs within its region such as physics, trading, combat etc. [3]

Players can only ever interact with objects that are in the same region as them. Players and NPCs can move between regions, but this may not be transparent.

In practice there may also be multiple copies of the world, all partitioned over different servers, but we will not consider this here for simplicity.

NPCs that can move between regions in a world are simulated by a dedicated set of servers. Other objects and NPCs are simply simulated by the server they are located in.

Similarly, data that spans regions, such as player data, is stored in a centralized database, where other data is stored in a per-region database.

The reverse proxy is responsible for connecting the client to the server that runs the world the client is in and to additional servers that handle things that are not bound to the client's location in the world, such as chat.

4.1.2 Inter-Region Travel

When a player wishes to travel between regions of a world, their data is first persisted to the central database by the source server and the connection from the proxy to the server is then closed. After this, the proxy connects to the new server which in turn loads the data from the database. While this is happening, the client is presented with a loading screen.

4.1.3 Interest Management

For every object, the region server maintains a list of clients that can see it. When an object becomes visible or invisible to a client, the client is notified. When an object is changed, the change is broadcasted to all clients that can see an object.

4.2 State Management

The region server is always fully responsible for the state of objects within the region. All interactions made by the client are sent in the form of a command to the server associated with the region they are in. There, the command is validated and objects may be updated and a success response is sent as a result. The server may also reject commands, for example if the object that is referenced in the command is no longer there, in that case the client is sent a rejection response.

The client may implement interpolation and optimistic updates to make latency and jitter less noticeable.

4.3 Publish/Subscribe Network Designs for MMOGs

In this section, we describe Publish/Subscribe Network mechanism for MMOGs. We will briefly go through traditional organization of interaction between clients and servers, function of interest managers in these settings and also how the publish/subscribe gaming engines could help transition smoothly and efficiently as multiplayer online games make way for massive multiplayer online games with many more clients and objects in the shared state.

4.3.1 A Brief Overview of Publish/Subscribe Architecture Elements

Traditionally, the Multiplayer game state was managed by a central server which distributed replicas of the object to the clients. Each player (client) receives copies of objects they need to play, which is arbitrated by an interest manager running on the central server.

4.3.2 Interest Management

The task of the interest Manager(IM) is to determine, which objects in the gaming world are interesting for the client, based on the AoI (Area of Interest) of the in-game avatar of the player. The AoI is usually a pre-configured area centered around the game avatar, and represents the world view of the avatar and also of all the surroundings the avatar might find interesting. The node controlled by the player must be aware of these objects in order to display them on the screen.

4.3.3 Replica Management

In order to transfer the replica of a particular object to the client node, the IM checks if the objects fall within the AoI of the client node. If it does, the replica is sent. However, intelligent IM are also aware of impenetrable structures like walls and trees in the gaming world and do not send replicas of objects which are behind these obstacles. The two approaches used for such Interest Management are: Rectangular AoI based and Triangular Tile based approach, which would be discussed in detail as part of different Publish/Subscribe Architectures.

Replica Management (RM) is another important task of these architectures, since the master holder of the master copy of an object must share it with other clients or players for a successful game-play experience. This replica transfer to a participant node is guided by the IM mentioned above. If a client wants to make changes to an object and it has the master copy as well, it can simply update the object and all the other nodes can receive an update of this object if the object is in their AoI or if the node is subscribed to the topic or class of the object in Publish/Subscribe networks.

In traditional architectures Interest and Replica Management is a responsibility of the central server as they hold master copies of all objects. This introduces a lot of overhead as the changes must be communicated to each client in case of exclusive game-play AoI or sometimes only to a group of clients sharing the same AoI. Publish/Subscribe architectures introduce improvements to make the communication decentralized with no single point of failure.

4.3.4 Network Communication

When an update in an object happens, all the client nodes, which have their AoI encompassing the object must be updated. This update happens traditionally through point-to-point communication between the master and the client. In Publish/Subscribe MMOG architectures, this happens in a multi-cast way with the Master copy holder updating the object copy and publishing the copy for all requesting nodes through the subscribed channels on the basis of object classifiers like topic etc.

Only deltas (the actual change made) is sent to the client nodes as it keeps the messages short.

The procedure involves the client node requesting using a "Replica Request" message for an update upon discovery of a new object within its AoI and then the master sends an update using the "Replica Reply" Message.

4.3.5 Publish/Subscribe Engine Architectures

A lot of Publish/Subscribe networks in applications other than MMOG implement a mechanism where the client can retrieve all past updates for an interesting object. The workload generated is not heavy in case of some technologies like RSS feeds, but can generate significant overheads in case of MMOG.

Thus, in case of Publish/Subscribe Engines, the master copy holder notifies the clients about the update and leaves it upon the client to request for an object update, ensuring that the client receives only the necessary update and not the whole historical data about the object.

IM and RM are responsible for these actions in most engines, but in some designs, IM is done by the client itself by exploiting rich semantics of pub/sub systems. Communication can be sometimes point-to-point which is implemented by assigning a "topic" to each node. When nodes communicate with each other, they just have to publish a message on the specific topic channel.

4.3.6 Publish/Subscribe Engines

There are 3 Publish/Subscribe Engines which are popularly implemented in MMOGs, namely: 1) Object-Oriented 2) Tile Based and 3) Area Based.

Object-based Network Engines: This engine is topic based and interest in an object is made explicit by subscribing to the topic. The IM could be located on different servers which track the movement of the players in the game world. If the IM determines that a player is interested in an object, it makes the player's client subscribe to the topic associated with the object. When the master copy is updated, all interested clients also receive the updates through the object's channel. It also determines when the object stops being interesting to a client. Replica Transfer is done by notifying the client to request an update via the " Replica Request" Message to the Master Holder. The Master Holder determines if the update is valid and if yes, then it transfers the state update to the requesting node via the " Replica Reply" message of object's channel.

Tile-Based Engines: Tile-based engines divide the world map into tiles for better approximation of object interests. Tiles could be big or small but not configurable at run-time. It does not require a dedicated IM.

Master holders of object subscribe to "Replica Request" Topic of a tile if their objects are in the tile. A subscription by an interested node for these objects directly communicates with the master holder through this mechanism. The "Replica Reply" mechanism is similar to Object-based network engines, except that nodes are expected to subscribe to this channel. Nodes also subscribe to "Notify Update" channel to get notified if updates for objects within their AoI exist. This is a major difference from Object-based engine. Another difference is that updates are published on the tile and not on the object.

Area-Based Engines: Content-based engines are more expressive than topic-based engines in terms of publications and subscriptions as there exist filters for fine-grained multicast opportunities. A player moving in the game map needs to simply notify its location in order to get the shared variables in the game. The x and y co-ordinates defining the limits of the AoI rectangle also decides which node receive the updates on the player movements if their AoI co-incides with the co-ordinates of the player.

Client nodes subscribe to the "Notify Update Channel" for a particular x-y range within their AoI and whenever an object within this range gets updated, they receive a notification. Replica Request, Update Propagation and Replica Transfer are also similar to Tile-Based engine however, the object discovery is sometimes restricted by a periodic update instead of a continuous update when the player is moving. eg. update player position every 500 milliseconds.

Thus, it can be seen that replica request, replica transfer and update propagation in Publish Subscribe Engines is focused upon multicasting using subscribed channels to reduce point-to-point communication as much as possible. How this affects the performance would be discussed further in this essay.

4.4 Peer-to-peer architecture

The provision of suitable amount of game server resources is critical for success of MMOGs. Too many players could overload the servers and cause high delays. On the other hand, if there are fewer players than expected, the game company faces costly over-provisioning. [2] Peer-to-peer (P2P) architecture tackle these problems by significantly reducing server requirements by enabling client nodes communicating directly with each other or perform part of the game state computation.

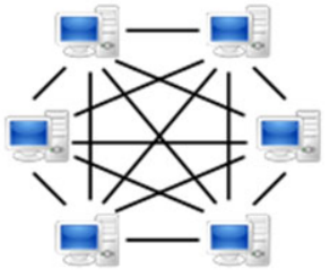

Figure 1: Peer-to-peer network

Peer-to-peer architectures are common computer network architectures where each workstation has the same capabilities and responsibilities. It partitions the workloads between peers that are equally privileged – equal participants in the application. The underlying structure is a k-connected network graph: each client is connected to all other clients in the network (Figure 1). The full connectivity is needed otherwise part of the network will not be able to receive events which are caused by unconnected clients. P2P enables high scalability since the load can be distributed among all nodes. New nodes also introduce new resources to the system at no extra costs for the game provider. Moreover, the failure of any individual peer would not effect many other players since responsibility is distributed across many nodes. Furthermore, it can lower latency as there is no need to send updates to central server and from the server to all other clients. Instead, it can only send updates on to interested peers.[3]

But P2P also have several drawbacks. P2P systems are harder to control and manage. This is due the fact that there is no central server that maintains global data and the state is distributed among peers. Consequently it is hard to provide consistency control and high security, especially against cheating. With the rise of nodes in the system the coordination overhead and complexity is also likely to rise. [3]

5 Comparison of architectures for sharing state

5.1 Availability

Traditional: Regions can be down independently, if the shared database is down everything is down. Multiple proxies may coexist for better availability.

Publish/Subscribe Architecture: Publish/Subscribe MMOG engines availability is dependent on the availability of the master holder of the object interesting the client node. If there is central holder which is down, whole gameplay get affected, where as if there are distributed master holders, only the clients interested in the object held by these master holders get affected.

P2P: The failure of any individual peer would not effect many other players since responsibility is distributed across many nodes. This makes P2P architecture highly available and robust.

5.2 Consistency

Traditional: Consistency is easy to enforce, all commands are validated by the region's server, and a total order of commands could be maintained within a region.

Publish/Subscribe Architecture: Consistency is maintained in Publish/Subscribe network engines through IM and RM which ensure that updates of objects reach the interested nodes through mechanisms like replica request, replica reply, notify update etc. This is a multicast operation with much better latency than point-to-point communication in traditional architectures. One broker systems have latency of up to 50 milliseconds for 170 players and up to 400 milliseconds for 230 players. However,with more brokers, more players can be added with reduced response times. More brokers but light use introduce delay in the system whereas heavy use and more brokers gives optimal performance.

P2P: Consistency is hard to enforce, since deleting and updating is non-trivial in a P2P network if we want to achieve consistency among copies.

5.3 Interactivity

Traditional: Depends on the performance of the region's server.

Publish/Subscribe Architecture: Interactivity in Publish/Subscribe engines depends on the master holder and client nodes performance and connectivity because if these individual nodes are under performing, the interactivity of gameplay would decrease considerably.It could also be affected by the congestion in the information broker's queue.Most update messages are received when there are lots of moving players that are close together. When player move at twice the speed, the number of subscriptions issued by clients for the tile and area-based engines doubles as well. This is because faster avatars cross tile boundaries more often, and hence their AoI changes more frequently.

P2P: It can lower latency as there is no need to send updates to central server and from the server to all other clients. Instead, it can only send updates on to interested peer.

5.4 Scalability

Traditional: Hard to load balance with fixed regions. Could duplicate the world but is not nice.

Publish/Subscribe Architecture: Scalability is easy to implement in Publish/Subscribe engines as the management is no more centralised and many players can be supported at the same time due to subscription to topics instead of a central manager. Object-based engines support upto 750 players whereas Tile and Area-based support upto 900 players. As brokers of information increase, scalability and performance increase but not lineraly.

P2P: Enables high scalability since the load can be distributed among all nodes. New nodes also introduce new resources to the system at no extra costs for the game provid

5.5 Security

Traditional: All commands are validated by a trusted server. This is the ideal scenario from a security perspective.

Publish/Subscribe Architecture: The channel can be compromised in the Publish/Subscribe architectures or the broker all the information flows through the broker. If the broker is secure, the information exchange is secure. If any of the client nodes are compromise they can propagate wrong information on the subscribed channels. Thus there are multiple points of failures.

P2P: High level of security is hard to achieve due the nature of the architectural style. However, there are solutions to most specific attack scenarios. Cheating attacks can be countered by integrating special voting algorithms that game events are checked by number of peers before executing it.

5.6 Persistence

Traditional: If deemed necessary, the entire state of a region could be stored in an ACID database. And commands could only return success when they are persisted.

Publish/Subscribe Architecture: Persistence is a key feature of the Publish/subscribe network architecture where clients joining a repaired broken down network receive the update of the objects the client is interested in, as long the object is within the client's AoI. This is ensured by periodically saving game state and also master copies of object states at master holder. Updates are notified regularly to clients and when requested, these updates are provided to clients. Master copies could be with a central server or with specific nodes. This ensures distributed risk and topic based update propagation through topic channels.

P2P: In pure P2P network a data item disappears when the last peer offering it leaves the network.

6 Conclusion

From the papers that we have read we can conclude that P2P architectures have many benefits related to upkeep and performance. However, the associated engineering cost of the additional complexities introduced by P2P architectures such as eventual consistency, redundancy and cheat prevention is most likely a driving factor behind the still imminently prevalent traditional-style architectures.

Material used: For formulating the requirements, we have briefly looked over all papers in order to see what they consider important. We have of course also used the dedicated paper that discusses requirements[2]. [5]

References

[1] T.-Y. Hsiao and S.-M. Yuan, "Practical middleware for massively multiplayer online games," *IEEE Internet Computing*, vol. 9, pp. 47–54, Sept 2005.

[2] G. Schiele, R. Suselbeck, A. Wacker, J. Hahner, C. Becker, and T. Weis, "Requirements of peer-to-peer-based massively multiplayer online gaming," in *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, CCGRID '07, (Washington, DC, USA), pp. 773–782, IEEE Computer Society, 2007.

[3] A. Yahyavi and B. Kemme, "Peer-to-peer architectures for massively multiplayer online games: A survey," *ACM Comput. Surv.*, vol. 46, pp. 9:1–9:51, July 2013.

[4] "The Bloodbath of B-R5RB, gaming's most destructive battle ever." `https://www.eveonline.com/article/the-bloodbath-of-b-r5rb/`. Accessed: 2018-10-30.

[5] C. Cañas, K. Zhang, B. Kemme, J. Kienzle, and H.-A. Jacobsen, "Publish/subscribe network designs for multiplayer games," in *Proceedings of the 15th International Middleware Conference*, Middleware '14, (New York, NY, USA), pp. 241–252, ACM, 2014.