A Survey of MMORPG Architectures

Sakib R Saikia

March 20, 2008

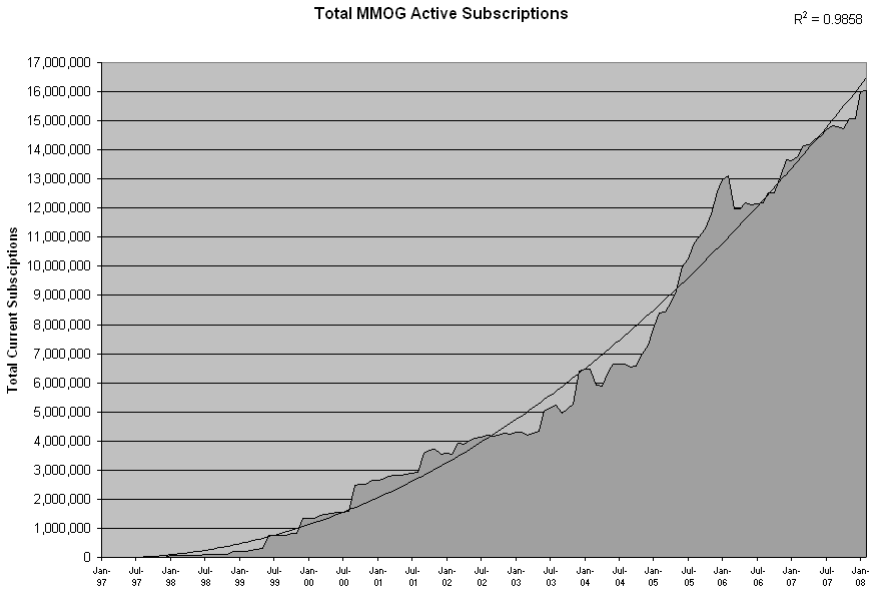
Abstract

This paper discusses the main architectural considerations for a Massively Multiplayer Online Role Playing Games. It outlines the goals that such a system needs to fulfill and the challenges faced in order to achieve them. The traditional prototypical approach to such a system is the client-server model. We shall review this model, as well as alternatives to this model and discuss their pros and cons.

1 Introduction

Multiplayer Online Games (MOG) present a classical problem of implementing a distributed system. There are certain constraints that need to be addressed to achience an efficient solution to this problem. At a higher lever, these can be classified as – performance, reliability, security, scalibility and maintainability[1]. Although the advances in networks, processing power and graphics hardware have helped MOGs achieve a new level of realism and playability, it has also resulted in an exponential increase in fan-following. The total number of active MMOG subscriptions currently stand at 16 million [2].

The traditional approach to host an MMOG is a client-server architecture. This, however, provides a single point of failure, increases latency and can cause possible bottleneck. A cluster of servers [3][4] can be used to circumvent this problem, but it needs to be decided whether to host a parallel game universe in each of the servers [9], or to split the game universe into regions and to assign each server to handle a unique region [6]. This raises the question of maintaining consistency between the game (sub)worlds. One also needs to consider that a static division of the game universe might not be the best solution – in that case one needs to consider how to dynamically divide the game universe and allocate it to servers depending on the game state.



A different approach to this problem is to form a P2P overlay network to exploit the *locality of interest* [5][6] exhibited by MMOGs, and distribute the game state to the peer players. The players thus contribute memory, CPU cycles and bandwidth to manage the shared game state.

In this paper, the terms Multiplayer Online Game (MOG), Massively Multiplayer Online Game (MMOG) and Massively Multiplayer Online Role Playing Game (MMORPG) are sometimes used interchangeably when a clear distinction among them is not necessary.

Section 2 discusses MMOGs and the key concepts that impact the architectural design. Section 3 discusses the challenges in implementing a MMORPG. Section 4 is a survey of the prevalent and proposed architectures. Section 5 discusses some of implementations of the architectures outlined in section 4. Section 6 is a discussion regarding the pros and cons of different architectures. Finally, section 7 concludes this survey.

2 Massively Multiplayer Online Games

MMOGs are characterized by having thousands of players in the game universe and allowing them to share the same game state so that they all have a consistent view of the world. There are typically two types of MMOGs - Real time strategy (RTS) games such as Warcraft III, and Role Playing Games (RPG) such as World of Warcraft. RPGs can be distinguished from other MMOGs in that they

allow a player to customize the world or the player characteristic depending on their role. Other than that, from a design architecture point of view they are mostly alike.

2.1 Game Objects

A typical MOG game world is made up of objects which can be classified as [5]

- Immutable Objects - that cannot be changed, such as terrain
- Mutable Objects - whose state can be changed, such as food, tools, weapons
- Player Characters (PCs) - characters controlled by players
- Non-Player Characters (NPCs) - AI controlled characters

2.2 Game State

The game state is a snapshot of the game at a particular instant of time. This includes the position of the PCs and the NPCs as well as the state of the mutable objects. Immutable objects are generally not part of the game state, and are typically installed as part of the game client software, and updated through software update mechanisms. The game state can be changed through the following interactions between game objects

- position change
- player-object interaction
- player-player interaction

The game state for a MMORPG should be *persistent* so that the user experience is carried over seamlessly across different login sessions.

2.3 Locality

Although the game universe can be huge, the players' avatars have a limited area of interest, which is determined by their sensory capabilities. As such, the players do not need to know any information of the game universe outside this area of interest. This is the fact that is used to adapt MMOGs to P2P networks, such as in [5] and [10].

3 Challenges

The challenges faced by MMORPGs are:

- A customizable world[1].

[1] Not a requirement for other MMOGs

- One player's actions should be able to affect another player's game state. For example, if two players are facing each other in the game world, they should be able to see each other.
- Persistence. A player should be able to carry on from where he or she left in the next login session.
- Large number of concurrent users. A MMORPG should be able to support thousands of concurrent users while still maintaining an adequate frame-rate.
- The amount of network traffic generated to maintain game communication should be kept low.

4 Architectures

Section 4.1 discusses the client-server architecture to support MMORPGs. It also discusses variants of the traditional client-server model. Section 4.2 discusses peer-to-peer approaches and its variants.

4.1 Client-Server

At a higher level, this consists of a single server connected to one or many clients over the Internet. A larger game may require a cluster of servers.

Sergio et. al. [1] have suggested a *publisher-subscriber pattern* for an MMOPRG. The clients that require to track a certain state, subscribe for notifications of that state on the server. This forms a *notification list.* Whenver that particular state is updated, the server sends an update message to all clients in the notification list. The advantage is that the server does not need to maintian state information for each client (each client maintains its own state), and consequently reduces network traffic (caused by multicast update messages). This provides load balancing and scalability. The use of shared components, where components such as real-time graphics are replicated at each client, whereas authentication and global game state is maintained at the server also improve load balancing and scalaility. The architecture pattern used in the client side is that of a *model-view-controller* (MVC). They also introduce the concept of a *world module* which keeps a subset of the world database (game state) in direct access form. The improves speed and accessibility.

Assiotis et. al. [6] use the *locality of interest* principle as described in [5] to improve the client-server architecture discussed above. They split the world W into smaller, disjoint regions $w_1, w_2, ..., w_n \subset W$ and assign each region to a different server. This server is the main point of contact for all clients within that region. The clients communicate with the corresponding server using a *publisher-subscriber pattern.* A server itself is subscribed to its adjacent servers

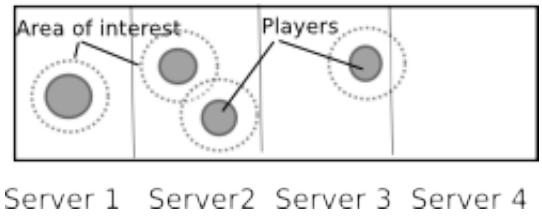


Figure 1: Different game regions assigned to different servers

for all points within the area of interest from the boundary. Contention between shared objects is resolved by implementing *region-level* and *object-level* locks. This scheme provides good scalibility and dynamic partition of the world in the case of hotspots. These are again implemented through locking mechanisms, to maintain consistency. The servers can be mirrorred to provide fault-tolerence against failstop failures.

4.2 Peer-to-Peer

MMORPGs display *locality of interest*, and thus are a good candidate for P2P implementation. Knutsson et. al. [5] suggest distributing the transient game state to the P2P hosts , while retaining persistent game state (user accounts, payment etc.) in a centralized server for security purposes. The P2P overlay is formed using a *distributed hash table* (DHT) implementaion using Pastry [7] and Scribe [8]. The game world is divided into regions and each region is mapped to a particular node based on a nearest-neighbor map to the Pastry key space. Players in the same region form an interest group, and updates relevant to that part are disseminated only within the group. Each player updates his or her own position and sends it as a multicast message to all members in the region at certain intervals. A coordinator-based mechanism is used to maintain consistency between shared object, and each coordinator is replicated at another node to accomodate failstop failures of the network and nodes.

Iimura et. al. [10] use a similar approach, but they abstract the DHT layer to a *zoning layer* and limit its use to backup data and for initial connection setup with an authorative node. A client node retains the connection to the authorative node for latest updates and modification requests, thus bypassing the DHT. The zone owner (who has write permission to the global data within that zone) modifies data on its own local storage, and uses DHT as a backup data storage. This reduces the latency because each time a message or modification needs to go through the DHT, it incurs an overhead.

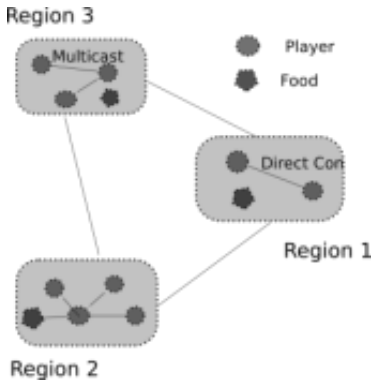


Figure 2: A game world divided into regions

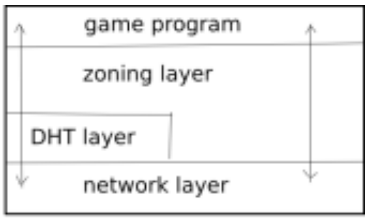


Figure 3: Zoned Federation Model

5 Implementation

Knutsson et. al. [5] implemented a simple game called SimMud to demonstrate the performance of the P2P architecture. They found that 99% of the messages were position updates, thus necessitating a lot of multicast messages. But, since all multicast messages were within the same region, this doesn't clog the network. SimMud was tested successfully on 4000 nodes. The Zoned Federation model [10] is suitable for only medium sized MOGs, where the number of nodes is less than 500. Marios et. al. [6] demonstrated their architecture on a game called *Kosmos*. The game performed well for near-boundary (region) interactions and showed good scalibility for their test set of 400 clients. However, it remains to be seen if it performs equally well for thousands of clients.

While evaluating architectures for MMOGs, it should be noted that the requirements are different for different types of games. In general, first-person-shooter games can tolerate much less latency than role-playing games.

6 Discussion

A client-server model provides a simpler implementation for an MMOG. There are not extraneous requirements to maintain consistency and it avoids the problem of a decentralized game clock. It is also more secure than a distributed model. The downside is that it may generate more network traffic and be a bottleneck for the game performance. A client-server model also presents a predicament for game publishers - should they over-provision the server only to see all that resource go to waste if the game doens't do well, or should they go ahead with a regular server and later do patch-work if the game becomes very popular (and slow as a result)? In other words, the client-server model does not scale dynamically.

A peer-to-peer model alleviates the problem mentioned above. Also, becuase the avatars have limited sensory abilities, they do not need to know of the states of all the other players. This self-organizing (or self-grouping) property also seems to favor a P2P approach. There is a risk of security getting compromised in a P2P setting, but keeping user account data in a server and distributing the game data alone to peers solves this problem. P2P models use a distibuted hash table to locate authorative nodes and to route messages. This incurs an extra overhead. Also, multicasting updates to all peer nodes consumes extra network bandwidth.

7 Conclusion

Peer-to-peer architectures provide a good alternative to host MMORPGs. Although they provide novel mechanisms for load distribution and scalability, they are not very pragmatic in a real world commercial setting [6]. P2P systems are not under the centralized control of the game publisher, and this is the reason that major titles such as World of Warcraft still follow the more traditional client-server approach despite the masssive number of subscriptions.

References

[1] S. Caltagirone, B. Schlief, M. Keys, M. J. Willshire. Architecture for a massively multiplayer online role playing game engine. *Journal of Computing Sciences in Colleges.* 18(2):105-116, 2002.

[2] www.mmogchart.com

[3] Zona Inc. Terazona: Zona application frame work white paper, 2002.

[4] Butterfly.net, Inc. The butterfly grid: A distributer platform for online games, 2003.

[5] B. Knutsson, H. Lu, W. Xu, B. Hopkins. Peer-to-peer support for massively multiplayer games. *IEEE INFOCOM.* 2004

[6] M. Assiotis, V. Tzanov. A Distributed Architecture for MMORPG. *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*. Article No. 4, 2006. ACM New York, NY, USA

[7] A. Rowstron and P. Druschel. Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. *Proceedings of 18^{th} IFIP/ACM International Conference on Distributed Systems Platforms (Middleware),* November 2001.

[8] M. Castro, M. B. Jones, A. Kermarrec, A. Rowstron, M. Theimer, H. Wang, A. Wolman. An evaluation of scalable application-level multicast using peer-to-peer overlays. *Infocom '03*, April 2003.

[9] E. Cronin, B. Filstrup, A. R. Kurc, S. Jamin. An efficient synchronization mechanism for mirrorred game architectures. *NETGAMES '02: Proceedings of the 1^{st} workshop on Network and system support for games,* pages 67-73, New York, NY, USA, 2002. ACM Press.

[10] T. Iimura, H. Hazeyama, Y. Kadobayashi. Zoned Federation of Game Servers: a Peer-to-peer Approach to Scalable Multi-player Online Games. *SIGCOMM'04 Workshops, Aug 30-Sep 3, 2004, Portland, Oregon, USA. ACM Press.*