Massive Multiplayer Online Game Architectures

Tuesday 23 January

Martijn Moraal (0131903)

Abstract

This paper investigates the architecture of Massive Multiplayer Online Games. The main purpose is to explore the two different architectures that have surfaced, namely client-server and peer-to-peer, and analyze how these relate to each other. So far MMOG developers have almost exclusively used client server architectures to implement their games, but recently a lot of scientific research is being done into the use of peer to peer architectures for MMOGs. This paper will look at the feasibility of using these peer to peer technologies for operating MMOGs. This is done by first looking at the characteristics of the existing client server architectures and their advantages and disadvantages. After this peer to peer architectures will be examined and their implications for MMOGs will be discussed. Based on this the two architectures will be compared against each other, in an effort to determine which architecture will offer the best possibilities for future MMOGs. Finally the options for a hybrid architecture in which the best of both worlds is combined are discussed.

Contents

1 Introduction

As conclusion to the bachelor phase of computer science curriculum of the Radboud University of Nijmegen students have to write a bachelor thesis. The subject that I have chosen for this thesis is Massive Multiplayer Online Game architectures. Massive Multiplayer Online Games, commonly referred to as MMOGs, are computer games that are played by a massive amount of people simultaneously. They are complex games with 3D graphics and an immense virtual world where players can interact with each other in lots of ways.

In recent years Massive Multiplayer Online Games have become a popular research subject among scientists. A great number of articles have been published about issues involving these games. A lot of this research has been focused on the social aspects of these games. Players often spend a huge amount of time on them and build up relations and communities with other players. The reasoning why these players participate in these games for many hours a week, the inner workings of these communities and the real world consequences of this all has been a hot topic within the social studies for some time now. The more technical research in the area focuses on maximizing the possibilities and challenges they offer to the players, while minimizing the resources that are required to run the games.

In this bachelor thesis I will explore and discus the different architectures that are used in these games, and analyze and compare these in order to determine which is most suited for the purpose. The central question of this thesis therefore is:
"What different architectures are being used in Massive Multiplayer Online Games, and which is best suited to overcome the typical challenges that these games offer?"

Chapter 2 will give an introduction into Massive Multiplayer Online Games. A description of what MMOGs are will be given, and the research that is being done in this area will be discussed. Furthermore it will introduce the two possible MMOG architectures: client server and peer to peer.
In chapter 3 the characteristics of client server architectures in MMOGs will be described, and the advantages and disadvantages of them will be discussed.
Peer to peer architectures and the implications they have for MMOGs will be discussed in chapter 4, along with their benefits and downsides.
Chapter 5 will make a comparison between client server and peer to peer architectures, and will explore the possibilities for a hybrid architecture.
Finally, a conclusion will be made in chapter 6.

2 Massive Multiplayer Online Games

This chapter will give an introduction into what Massive Multiplayer Online games are, and why they are an interesting research subject.

2.1 What are Massive Multiplayer Online Games?

Massive Multiplayer Online Games are as the name suggests games that are played by a massive amount of players simultaneously online. There are two main characteristics that define these games.

- Ability to support a large amount of players simultaneously in a single game world, typically ranges from 1.000 to 10.000.
- Have a 'persistent' game world. This means that the game world maintained and developed around the clock, it can't be paused, halted or reset. It will even continue while some players are not active. Furthermore every action and every change to characters or objects have a lasting effect, and can't be reset or redone.

MMOGs originate out of very limited text based games called MUDs (Multi-User Dungeons). The first MUDs appeared in the late 1970s on university networks and later bulletin board. They were very simple text oriented games because bandwidth and computational power was very limited at the time. Through the 1980s and 1990s with the evolution of computer technologies MMOGs developed into graphical games with more and more possibilities. However no big commercial MMOGs had been developed until the late 1990s, when "Ultima Online", "Lineage" and "Everquest" brought the MMOG genre to the mainstream market. Today's MMOGs offer the player complex 3D graphics and a massive virtual world to explore. Although the interaction with other players is a huge part of these games they do not revolve around it solely, computer controlled characters and other computer controlled objects often play a big role.

The early MUDs were very simple applications based on a client server architecture, and would run on university mainframes and bulletin boards. With the increase of the complexity and possibilities of the games so does the strain they put on the servers that run them. Even with the rapid development of faster and more powerful computers, the resources that are required to run these games are simply too great to be handled a single machine. The MMOGs of today still use client server architectures like the early MUDs did, but the servers of today's games are big clusters, ranging from 10 to 100 state of the art machines.

2.2 Scientific research

Because MMOGs are complex and resource intensive they are a popular research topic. A lot of effort is put into optimizing them and into developing better and more efficient ways to distribute the load over multiple servers, so that they can achieve a maximum of player involvement and experience with a minimum of needed resources. As these games continue to attract more and more players, and the expectations of the quality, graphics and level of interaction that these games have to offer will continue to grow new and better solutions will have to be found in how they operate. A lot of research has therefore been focused on how to improve the effectiveness of the architectures and how to improve the amount of load that they can handle.

Two main streams can be identified in the research that is being done in this area. The first consists of the research that is being done in improving the existing client server architecture that is currently being used in the majority of the games. This can be in the form of a more efficient distributed server architecture or more efficient communication between the various parts. The second stream of research that can be identified focuses on creating a distributed peer to peer architecture for these games and with that eliminating the need of a heavy and complex server. The two streams will be discussed more in depth below.

2.2.1 Client server

The vast majority of the MMOGs that have been developed and brought to the market so far use the classic client-server architecture. This architecture poses some problems however. In this architecture all the load of handling the virtual world is carried by the server. No single machine can handle this, so complex architectures have to be developed to distribute the load over different servers. The research in this area is therefore mostly focused on how to distribute the different server functionality as efficiently as possible over different physical machines, by for instance: cloning the virtual world, dividing it in zones, putting different server functionality on different machines (database, login handling, etc).

Examples of the research that is being done in this area is for instance [1], where an architecture is proposed where the world is divided into zones that are dynamically assigned to different servers based on the load they experience. In [2] the use of generic middleware that can be used for multiple MMOG projects is promoted. An architecture for the inner workings of both server and client is presented in [3]. In [4] an algorithm is described for handling the interactions between different zones of the virtual world. A system for dynamically selecting the most optimal server for users is given in [5]. And a system for handling events based on different priorities is presented in [6].

In chapter 3 the characteristics of the client server architecture for Massive Multiplayer Online games will be described and explained.

2.2.2 Peer to peer

The development of a peer to peer architecture for a MMOG is a hot topic in the scientific world. A lot of articles have been published proposing architectures where clients communicate directly with each other and are all responsible for a small portion of the game state, and thus do not have the need for a central server. By eliminating the need for a central server you reduce the cost of running a game enormously. However a lot of problems present themselves when developing a peer to peer architecture for MMOGs. For instance, saving the game state will be a major problem, the distribution of updates and patches will be more troublesome, it is harder to counter cheating in the game, and the bandwidth use for the clients will be significantly higher compared to the client server architecture. Furthermore it will be harder for companies to exploit a peer to peer game commercially.
Scientists believe they can counter all these arguments and a lot of articles have been published presenting architectures or parts of architectures that are able to run peer to peer style MMOGs. However no MMOG based on a peer to peer architecture has ever been commercially developed, and the interest from the industry to do so doesn′t seem to be extremely high.

Research in this area is for instance [7], where a distributed peer to peer architecture is presented that guarantees a low latency for users and prevents cheating. A system that can be used to turn classic client server based games into peer to peer based ones is presented in [8]. And in [9] a publisher/subscriber model is described for developing a distributed MMOG architecture.

The characteristics of peer to peer MMOG architectures and how they counter all of the different problems will be described in chapter 4.

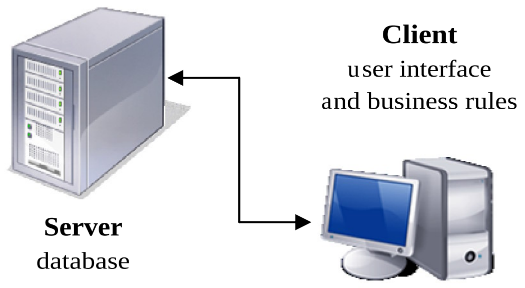
3 Client server architecture

This chapter will present a model of the general client server architecture that many of the MMOGs that are on the market today employ. Furthermore the benefits that these architectures have to offer and their downsides will be discussed.

3.1 Client server architectures in general

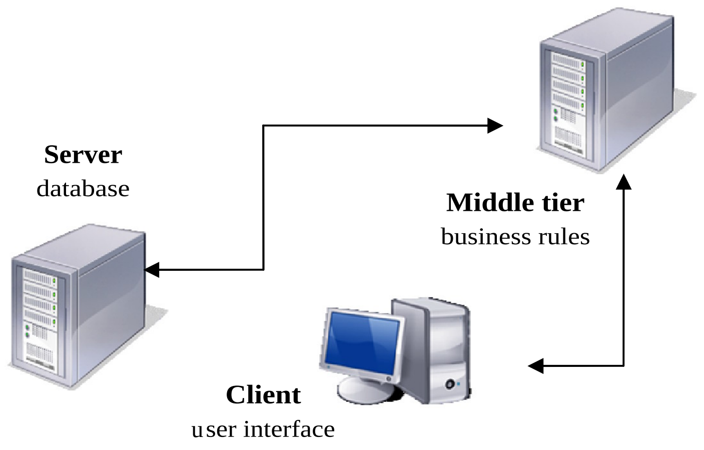
Client server architectures are not something that has been developed for Massive Multiplayer Games; they have been around for many years and are used in a wide variety of applications. They first started to appear in the 1980's when people began to face the limitations of the then popular mainframe/terminal architectures.

The first client server architectures were so called two-tier architectures. In this architecture there is an application running in the client machine which interacts with the server, most commonly a database management system. The client application, also known as a fat client, contained the presentation logic (user interface), the business rules and the database access. Every time the business rules were modified, the client application had to be changed, tested and redistributed, even when the user interface remained intact.



Two-tier architecture

To prevent having to alter the client application every time the business rules are modified the presentation logic and the business rules have to be separated. This separation is the fundamental principle of the three-tier, or multi-tier, architecture.



Three-tier architecture

In these multi-tier architectures the clients are often referred to as thin, or dumb, because they do not poses any of the business logic. The client only sends and receives requests from the server and presents it in a graphical way to the user.

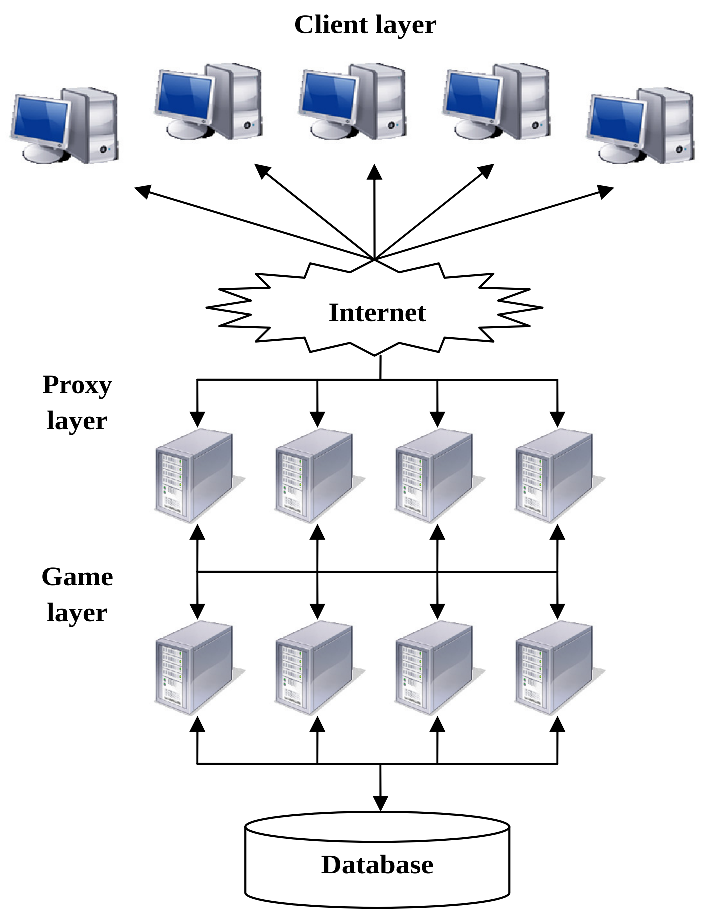
One example of a client server architecture is the world wide web. Web pages are stored on servers with all their structure and logic, the client simply requests them and presents them in a graphical way to the user. Client server architectures are very common and are used in wide variety of systems; Massive Multiplayer Online Games are just one application of them.

3.2 Client server architectures in MMOGs

Since the development of the first Massive Multiplayer Online Games they have almost exclusively used client server architectures. Each individual player connects to the server through the client application on their own PC and the server handles all the rules and the state of the virtual world. One of the first things you have to realize is that due to the sheer number of clients that will connect to the server and the load that they create it will be impossible for one single server machine to handle everything. Therefore some sort of distributed server architecture that balances the load over multiple machines will have to be used.

Because client server architectures are so widely used, both in MMOGs and in many other systems, a great deal of research has been done in the area and a lot of knowledge about how to best put them to use has been developed. This has led to the typical multi-tier architecture that is commonly used in MMOGs.

Most MMOGs today employ a four-tier server architecture. The four different layers that this architecture consists of are: client layer, proxy layer, application/game layer, database layer.



Typical four-tier MMOG architecture

3.2.1 Client layer

Each player in a MMOG is represented by a single client. All the players connect to the server through a client application on their own PC. The client presents the graphics and the user interface to the user, it signals all actions that the user wants to perform over to the server and shows all alterations in the game world that it receives from the server to the user. The clients are considered thin, or dumb, because they have no control over the 'business logic' of the game world. They only receive and request updates of the virtual world from and to the server, and present it in a nice graphical way to the user. This however does not mean that clients are simple applications. The latest MMOGs often employ complex 3D graphic engines, and use complex techniques to smoothen the game with a limited amount of client-server interaction. The number of interactions between the server and the client is kept to a minimum to not overload the server and because bandwidth for both server and client is limited. To accomplish this the client will have to use complex techniques like motion prediction to present a smooth playing experience even though it receives the world updates only on intervals.

3.2.2 Proxy layer

The proxy layer functions as a bridge between the client layer and the game layer. It handles all the connections to the clients and makes sure all the packets get forwarded to the appropriate servers in the game layer. Furthermore it can deal with security issues in the form of a firewall and make sure only paying costumers can actually play the game.

3.2.3 Application/game layer

This layer forms the heart of the game, as this is where the virtual world is housed, maintained and executed. All requests and commands sent by the players through their clients get forwarded to this layer. The game servers process these command, verify if they are legal, compute the changes to the virtual world, update the game state and forward the changes to all the clients that they affect. The main task of this layer is to maintain the state of all players and other objects in the world. It manages the position in the world of all players and controls monsters, non-player characters, treasures, the weather and all other dynamic aspects.

The complexity of MMOGs lies for a great part in the efficient distribution of the game server functions over multiple machines. Different techniques to accomplish this are discussed in section 3.3.

3.2.4 Database layer

Every move a player makes, every item, object, and monster is a data element in the game′s database. However even the largest game worlds are miniscule compared to data warehouses employed by thousands of companies around the world. Furthermore the schema′s and structures of the game databases are also fairly simple. Because of this they can take advantage of parallel database technology and other large memory systems to speed up the processing of transactions.

One important thing you have to consider is: game transactions are not like business transactions. The transaction load that MMOGs generate is very specific. For the most part it consists of huge amounts of very small reads and updates, generated by players moving through the world and interacting with objects. What they will not need is huge joins on star schemas. Games are not at all like DSS or OLTP systems that many businesses use. The game work load is much more like LDAP or telephony directory lookup: lots and lots of small transactions over simple tables.

Setting up the database layer is relatively easy. The reason is that people have been designing and improving relational database technology for decades. Modern relational databases have incredible amounts of capacity and perform extremely well on multiple CPU systems. Once you have estimated the number of concurrent players, you can estimate the number of transactions per second. You can then simply get a system with enough CPU and memory to suit the transaction needs.

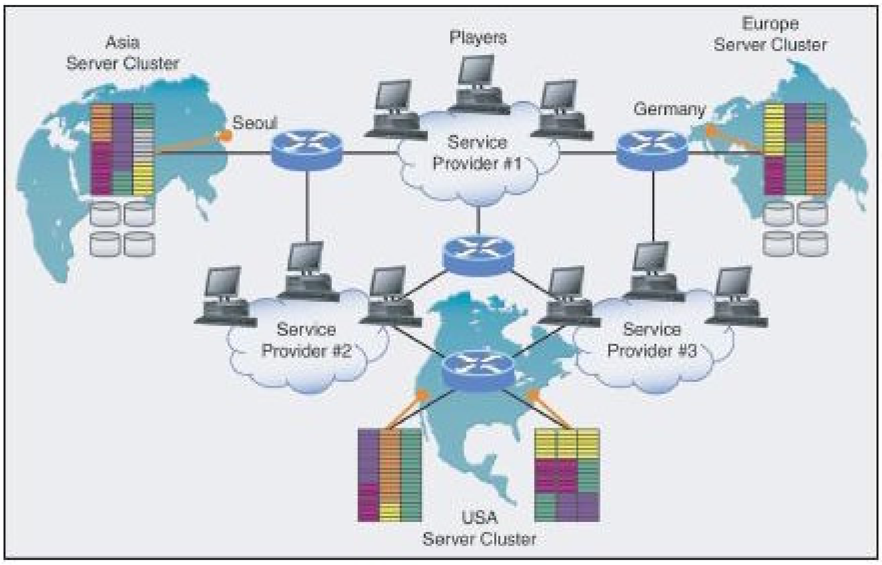
Ideally you would want to have the entire game database in memory. In this scenario, performance would be superb, and transactions would complete in times approaching those of a function call. But that would also mean having a system with a couple terabytes of physical memory. New databases systems are being developed that should potentially make it possible to run the entire database in the memory, but these kind of systems are not (yet) used by any MMOGs on the market.

3.3 Distributing techniques

The different server layers can easily be distributed over different servers, but each layer on its own will still require too many resources to be handled by a single machine. A lot of knowledge is available about efficiently setting up the proxy and database layer since these have been used for decades in a wide variety of applications. Distributing the game layer is more complex. A number of different techniques to split the game world over different servers will be discussed in this section.

3.3.1 Sharding

The first technique for distributing server resources across several machines is sharding. Apart from a few exceptions all Massive Multiplayer Games use this is in some form. Sharding means simply running different copies of the game. Each copy houses a number of players that can interact with each other but not with the players on the other shards. Different companies use different terms for these copies, such as worlds, realms or shards, but they all describe the same thing. Each copy of the game runs on a different server (-cluster) and will only support a limited number of players. This makes it very easy to scale the game. As the number of players increases you just add more shards and you remove them again just as easily when the number of players declines. Server load can be limited by decreasing the amount of players per shard.



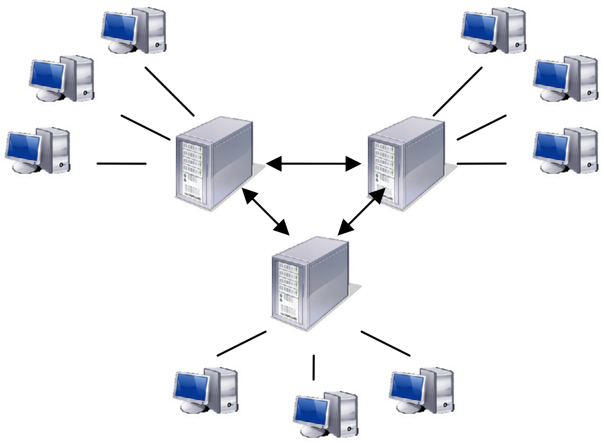
There are a few reasons why MMOGs use sharding.

- Firstly, because MMOGs strive to be worldwide applications you will have clients connect to your server from all over the world. This introduces the problem of the limited speed of light and the speed of electrons in copper. When you have a server on a single site in the world you will always have areas of the world that have a significant latency when connecting, which can severely hinder the playability for those clients. To counter this you have to setup different copies of the game at geographically disperse locations in the world so that there is a site for everyone to which they can connect with a reasonable latency.
- The second reason why sharding is an attractive technique is because it is a very easy and scalable way to distribute the load over multiple servers. This can also be achieved with the other techniques, but those are more complex and are harder to scale with increasing and declining numbers of players.
- Thirdly, it is attractive for playability reasons. When you have thousands or more players in a single world it can easily become overcrowded. Furthermore it takes a huge amount of work to create enough content for all the players. It is much more efficient to create copies of the same content so that different players can experience it at the same time.

The downside of sharding is however that it reduces the massiveness of the game. The players on the different shards can not interact with each other. So if the number of players per shard is too low it defeats the purpose of the Massive Multiplayer game. Sharding is widely used, but each shard still houses such a great amount of players that the load will still be too high for a single machine, and therefore other techniques will have to be used in combination with sharding to distribute the load over a cluster of servers.

3.3.2 Cloning

Cloning is running two identical copies of the game on different servers that are continually synchronized with each other so that the player experiences it as a single virtual world.

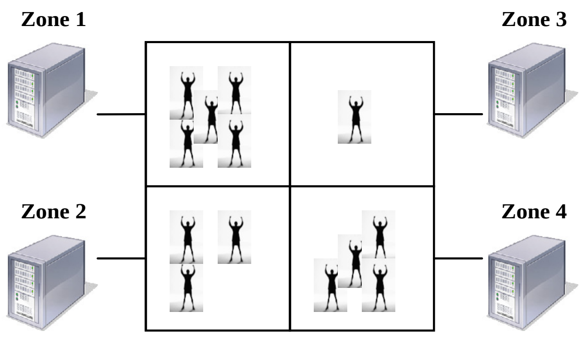


Cloning

The benefits of cloning are that you reduce the load of each server while still maintaining a single virtual world.
The downside is that the game state will constantly have to be synchronized between the two servers which will create a significant amount of overhead in terms of traffic and CPU load. Cronin et all [16] however believe that they designed a system that is more efficient than any other technique, using a cloned server architecture. Cloning is not used a lot in practice, the reason for this is not really clear and Cronin et all believe that in the future it will become a lot more popular for use in MMOGs.

3.3.3 Zoning

Zoning means dividing the virtual world into different zones that are all handled by their own server. The zones are often separated by geographical boundaries in the virtual world. Each zone server handles the events and maintains the state of that specific zone, it has no knowledge of the state of the virtual world outside that zone.



Zoning

The zones can be viewed as different worlds between which the players can move freely. In current MMOGs the borders are often not transparent to the players. A player standing at the border of zone 1 can not see or interact with a player standing at the same border in zone 2, and crossing the border is accompanied by a loading screen. It is however technologically possible to make the borders completely transparent to the players. [4] describes a technique to make the use of zones completely transparent for the player. This however introduces a lot of complexity in synchronizing the different zones over the different servers (players have to see each other and interact over border zones). The different zone servers have to update each other constantly with all actions that are happening in the border zone, which creates a lot of overhead.

Zones borders are often static in the game world and can not be altered, this makes it necessary to predict the load each zone will experience while developing the game. When your predictions are different from reality you can have a real problem with some overloaded zones and other zones hardly used at all. To counter this a system is proposed in [1] that uses micro-zones which are dynamically distributed over a set amount of servers, based on the load that they experience. These zones can not be too small however. When the zones get smaller the number of zone transfers (where a player moves from one zone to another) increases, these transfers create overhead on the server because all information about the player has to be sent from one zone handler to the other. When the zones get smaller and the amount of transfers increases this overhead can quickly outweigh the benefit you got from using them in the first place.

Zones are used in some form in almost all MMOGs that are on the market today. Mostly they use a small number of zones with static non-transparent borders along geographical boundaries in the virtual world, like mountains or water.

3.3.4 Instancing

Instancing is a mix between zoning and sharding. It means that you can have different ‘instances’ of the same zone. An instance is a copy of that specific zone in which only one, or a small group of players, can interact with each other and the virtual environment within that zone, but not with players that are in that same zone, but in a different ‘instance’ of that zone. Just like with shards different copies of the game are created that can not interact with each other. Because there is only a small number of players per instance they do not create a lot of server load and therefore several of these instances can be assigned to a single server. This assigning of instances to server can be done dynamically based on the load, because of that it is a very well scalable system.

With this technique some of the massiveness of MMOGs is lost, because you can only interact with a small amount of people within an instance, but as soon as you leave the instanced zone you get back into the whole virtual world again and can interact with everyone in that world (that is not in an instanced zone at that time). In most games only key area's like dungeons are instanced zones and the majority of the virtual world is a single copy where everybody resides.

Instancing is also very attractive for playability reasons because a lot of different players can experience the same content at the same time.

3.4 Benefits and downsides

The use of client server architectures in Massive Multiplayer Online Games brings a number of advantages and disadvantages with it, these benefits and downsides will be explored in this section.

3.4.1 Benefits

Client server architectures in MMOGs offer the same advantage as client server architectures in any application. They offer a clear separation of business logic and presentation logic. In MMOGs this separation is particularly interesting for a number of reason.

- Easy to update. Because the game rules and game logic are only handled by the server only the server has to be updated when changes to these rules and logic are made. This makes it far more easy to maintain the game and to keep adjusting the game to the needs of the players, than when you would have had to update the client software for all users every time. However, the client software in MMOGs is often not as 'dumb' as in more traditional client server applications. To limit the amount of interactions between the client and the server a lot of knowledge about the game world has been put into the client itself. Small changes can still be made on the server side alone, but for a lot of big or medium sized changes the client software has to be updated as well.
- Single game state on the server. Because the game state is completely maintained by the server it will always be the same for all users. Clients can go temporarily 'out of sync' with the game state when the connection to the server experiences problems, but there is never any question about what the real game state is because it is completely handled by the server. The server introduces a natural synchronization of the players that all display the same game at the same time.
- One governing authority to prevent cheating. All actions by the users are sent to the server which then alters the game state accordingly. This also allows for the server to authenticate if the requested actions are legal and if the user is allowed to do what he is trying to do. This introduces a natural 'referee' for the game, and makes cheating more difficult.

- Commercially exploitable business model. Client server architectures fits very well with the business model that companies employ to make a profit of their games. Most MMOGs that are on the market today require you to pay a monthly fee to be able to play them. Client server architectures allow the game companies to more easily restrict the game to those users that have paid the monthly fee, and counter piracy.

3.4.2 Downsides

Client server architectures also have a number of disadvantages. These disadvantages come from the mere fact that you need a server to be able to operate them, this server will represent a lot of costs and a certain strictness in your network. This leads to a number of disadvantages.

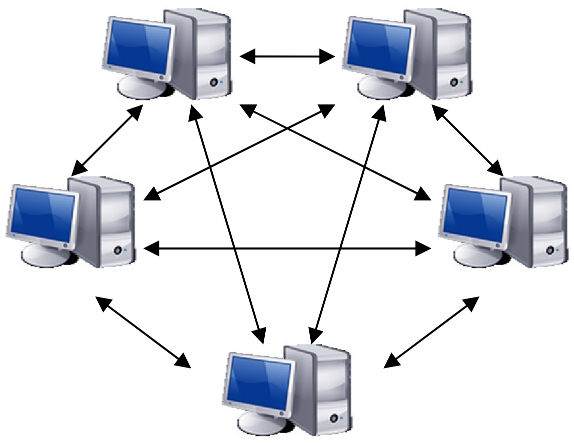
- Large cost for setting up and maintaining the server. Due to their complexity and the load they will experience MMOGs will require high end servers. These servers require a significant cost to acquire and maintain. Because of this the cost to develop these games increases and furthermore, a certain number of paying users has to be maintained to make it economically feasible to keep the game running. To be able to cover the cost of operating the server MMOGs will always need a large number of users, this makes it impossible for small niche market games to exist, or for players to keep playing the game after it has become unpopular among the majority of the users.
- Single point of failure. The entire game depends on the server and because of that the entire game will be unavailable when the server fails. This can make the network very fragile. One of the most common complains from players of MMOGs is that games are unavailable too often because the servers are either down or unstable.
- Limited scalability and load flexibility. The server forms a natural bottleneck. All data converges to it, and all computations are done on it. Once the server reaches its maximum load on CPU capacity, bandwidth, or storage capacity the game state computations and/or distribution will slow down or halt completely, reducing the playability for all users. The total server capacity is fixed given the architecture and the amount of machines. In the best case scenario machines can simply be added to the server cluster to increase the capacity. But even just adding machines will require reconfiguring of the server cluster and will take a significant amount of time. This makes it not very flexible. Therefore companies will have to accurately predict the load that will be put on their servers. If they predict too much load they will end up with more capacity then they need and will have wasted money. If they predict too little their servers will be unable to handle it, the game will be unplayable, and it will likely fail on the market.

4 Peer to peer architecture

This chapter will describe the use of peer to peer architecture in Massive Multiplayer Online Games. Furthermore the benefits that these architectures offer to MMOGs and their downsides will be discussed.

4.1 Peer to peer in general

In recent years peer to peer style network architectures have become more and more popular and are used in a wide variety of applications. Peer to peer architectures consist of only peer nodes with equal responsibilities and capabilities. In classic client server architectures the server possesses all or most of the resources like computational power and storage space, and the communication only flows from client to server and vice versa. None of the computations on the business logic is done by the clients and clients never communicate with each other directly. This differs greatly from peer to peer architectures. Because of the absence of a central server all resources lie in the peer nodes themselves and communications go from peers to peers directly.



Peer to peer architecture

Peer to peer architectures are used in many different situations and in many different forms. In recent years peer to peer file sharing systems have gained great popularity. With these systems users can download files directly from other users. However these systems often still use a central server to provide an index for the available files, and in that sense they can't be classified as 'pure' peer to peer but employ a more hybrid architecture.

Peer to peer architectures offer a number of advantages. Because all the resources like network, bandwidth, storage space, and computing power are shared between the

nodes the entire load on the network is also shared between them. Furthermore when new nodes connect not only the load on the network will increase, but also the capacity of the network because of the new resources that these new nodes bring. This makes peer to peer networks very scalable.
Another advantage of peer to peer networks is their robustness. In client server architectures the entire network is dependant on the server, when the server fails the entire network fails. Peer to peer networks can be setup so that when one node fails the rest of the network will continue to operate as normal.

4.2 Peer to peer architectures in MMOGs

A great deal of research has been done on the use of peer to peer style architectures in MMOGs, but so far this has not lead to it being used in any major MMOG that is on the market today. There are some multiplayer games that are based on distributed peer to peer architectures, like MiMaze [14] and Age of Empires. However MiMaze is little more than a research project and Age of Empires can not be called a Massive Multiplayer Game with only support for 8 players in a game at once.

Because the use of peer to peer architectures in multiplayer games is a relatively new phenomenon and hasn't yet reached the mainstream consumer market there is no single dominant 'best practice' architecture. A number of possible architectures have been proposed by researchers in articles, but these vary substantially in their implementation. Most of these architectures still make use of central servers to index the peers that are active in the game at any time and to handle all the account and other administrative functions that might be needed. However all the computations on the game rules and the maintaining of the entire game state are done by the peers themselves.

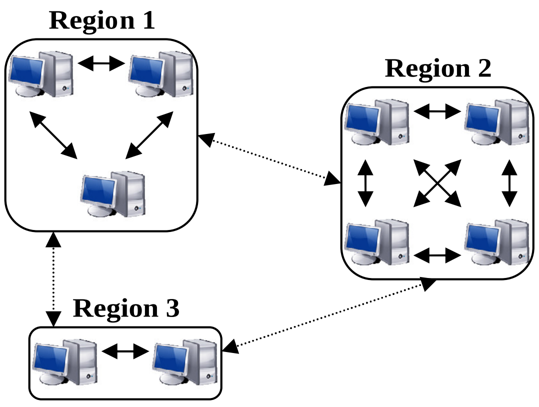
Client server architectures have the advantage that a single authority orders events, resolves conflicts in the simulation and acts as a central repository for data. Peer to peer architectures do not have this single authority and therefore it becomes much harder to present one consistent virtual world to all users. The game state will be distributed over all the peers so there will have to be some efficient way to assign different elements of the game world to different peers. Furthermore these elements of the game world have to be consistent and accessible for all other peers in the network. It becomes even more complicated when you consider that peers can not be completely trusted as they might want to cheat to gain an advantage in the game, or that they can randomly crash, losing that part of the game state that they were responsible for.
The different issues that you face when using a peer to peer architecture for a MMOG, and the possible solutions for them are discussed in the following section.

4.3 Issues with peer to peer architectures

4.3.1 Interactions

The first issue that immediately presents itself is that bandwidth limitations simply make it impossible for users to send and receive all updates to and from all other users. The players in a MMOG usually do not have the kind of high end connections to allow for the bandwidth that would be needed for that. Fortunately players are usually not interested in the whole game world. The characters usually have limited sensory capabilities and movement speed, and therefore only need information about a small part of the world in their direct vicinity. This can be exploited with interest management in a number of ways. The world is often divided into regions, much like the zoning technique in client server architectures, as described in section 3.3.3. The peers in the specific region only need to communicate with each other and not with the peers in other regions.

Knutsson et al. in [13] divide their game world into small regions. Players in the same region form an interest group for that portion of the world, so that updates relevant to that part are communicated only within the group. A player changes group when he moves from one region to another. Additionally, objects in a given region only need to communicate the part of their state that is visible to players. For example, a chest must communicate its location and appearance to players, but not its status as locked or unlocked, or its content. Fiedler, Wallner and Weber describe a somewhat different system in [9]. Their architecture is based on a publisher-subscriber model, where peers subscribe to local interest groups when they enter an area. The details of interest management systems are highly dependent on the inner workings and specifics of the game.

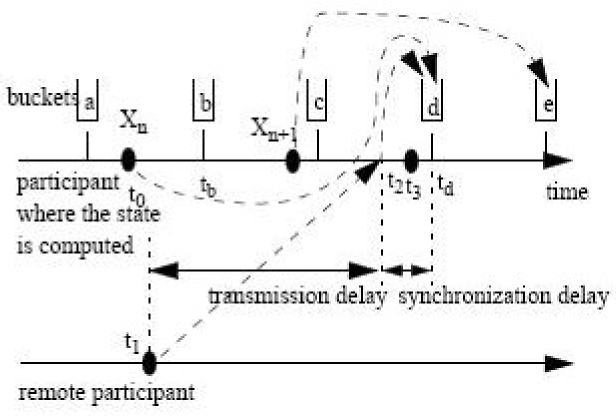


Peers divided into regions

4.3.2 Consistency and synchronization

To allow the different players to play together and interact with each other the game world has to be consistent for everybody. If, for example, player A opens a chest and picks up its contents, then that chest has to be empty for player B who arrives at it later on. In client server architectures this unique game state is computed and maintained by the server. With a peer to peer architecture you do not have the advantage of a single server to administer the game state, instead each peer computes its own local view of the game state using information received from the other peers. Because network delays are different for all peers some sort of synchronization protocol will have to be used to provide a game state as timely and consistent as possible for all peers.

Diot and Gautier [12] described the first protocol for distributed games and built a game called MiMaze to demonstrate its feasibility. Their protocol, called bucket synchronization, divides the game time in ‘buckets’ so that actions issued at the same time will be processed together by all peers. The synchronization mechanism makes sure that actions processed to evaluate the global state of the game at time t were all issued “close to” t, or in the same evaluation period. For this purpose, a bucket is associated to each evaluation period. This bucket receives data that will be used by the game to compute the game state when it will be time to compute this bucket.



Bucket synchronization

Consider the above figure, without synchronization the action issued by the local peer at t_0 would be processed much earlier then the action issued by a remote peer at t_1 (but received at t_2), while in reality both actions were issued at the same time. Bucket synchronization delays actions by putting them in buckets to be processed at the same time. In the above example the action issued at t_0 is put in bucket d together with the action issued at t_1, and both will be processed together at time t_d. Because of this synchronization all peers should display the same game state at the same time. The synchronization delay is computed by the receiver to determine in which bucket the incoming information should be stored. A global clock is used to evaluate the delay with which information reaches the participants. If the synchronization delay becomes too long it will severely hinder the playability of the game. To counter this a maximum has to be defined with which actions can be delayed, and a mechanism to deal with actions whose transmission delay exceeds the maximum delay.

Bucket synchronization does not address the problem of cheating by the peers. To prevent cheating Baughman and Levine [14] designed the lockstep protocol. Lockstep uses rounds for time, which are broken into two steps: first, all peers reliably send a cryptographic hash of their action, then all peers send the plain-text version of their action. This forces the peers to commit their move, without revealing it, thereby preventing anyone from knowing someone else′s action ahead of time. However, because of the added interactions in lockstep the effective latency is three times the latency of the slowest link.
Building further on lockstep Cronin et al. designed the Sliding Pipeline (SP) protocol [15]. They add an adaptive pipeline that allows players to send out several moves in advance without waiting for ACKs from the other players.

4.3.3 Game state distribution

The game world of MMOGs typically consists of four different types of elements: static landscape, intractable objects, player controlled characters and computer controlled characters. The landscape consists of all the static objects and terrains that will not change during the course of the game. Because it will always be the same for everyone it is usually part of the client program and does not contribute to the game state that has to be shared and saved among the peers. The intractable objects and both the player controlled characters and the computer controlled characters make up the dynamic game state that constantly changes. This game state has to be consistent for all players, but players do not need to know the state of every object in the game world all the time. It would take far too many resources in memory, computing power and bandwidth usage to have every player constantly keep track of the entire game state and receive every update on it. A better solution is to distribute the state of all the objects in the game over the different peers, so that for each object there is a peer that is responsible for keeping the state of that object. Peers that then want to interact with that object have to request the current state of the object and submit changes to it to the peer that is responsible for it.

To make this possible some sort of method will have to be used to efficiently distribute all the elements of the game world over the peers. Knutsson et al. in [13] describe an architecture where the game world is divided into small regions, each peer and each region is then assigned an ID, for each region the peer with the closest matching ID is then assigned the coordinator for that region. The coordinator is responsible for all objects in that region. Putting the responsibility for all object in a region on one peer can create a high load on that peer, to counter this objects can be given their own ID and can be assigned to different peers.
Because of the random mapping, the coordinator of a region is unlikely to be a member of the region, but this is an advantage for a number of reasons. First, it reduces the opportunities for cheating by separating the objects from the players that access them. Second, the coordinator of a region doesn′t have to be changed every time the corresponding peer leaves the region, instead it is limited to when peers join

or leave the game. Finally, random mapping improves robustness by reducing the impact of localized events. For example, multiple disconnects in the same region do not typically result in losing the region′s state.

When distributing the games objects over the different peers one problem presents itself: What if there is only a very small number of peers active in the game, or even none at all? When only a very small number of peers is active they will have a very high number of objects assigned to them, it is imaginable that the resources required for this will exceed what they have available. And when no peers are active at all there is no saved game state and the state of all object and characters in the game world may be lost. Knutsson et al. in [13] do not address this problem, but it is imaginable that there is a critical mass of peers that is required to make the game playable. This critical mass can possibly be provided by the game developer itself.

4.3.4 Fault tolerance

Unfortunately peers do not always leave the game in a controllable fashion that you want them too. They can simply crash or disconnect without handing over their responsibilities and game state to the rest of the network, which can lead to the loss of elements of the game state. When for instance in the game state distribution system as described in section 4.3.3 a coordinator of a region crashes the state of that entire region will be lost. To prevent this some backup system has to be used so that the state of every object will always be available on multiple peers.

In [13] Kuntsson et al. extend their algorithm for distributing the game state by assigning a backup coordinator for each object, next to the main coordinator. This backup is the peer that has the ID closest to the object ID after the coordinator. Given an object with key K, then the numerically closest node N will be its coordinator. Similarly the next numerically closest node M will be the objects backup coordinator.
$(N, M, \forall I : (|N - K| \leq |M - K| \leq |I - K|))$
When the coordinator fails the backup automatically has the closest ID and therefore is the new coordinator, only a new backup coordinator will have to be assigned which will be the peer with the ID closest to the object ID after the new coordinator. The coordinator and the backup will continually have to keep synchronizing the states of their objects. This system only uses one backup peer, but it can easily be extended to use multiple.
This system is of course not 100% failsafe, the coordinator and all backups may crash at the same time. But based on the assumptions that peer failures are independent and that the failure frequency is relatively low the chance of loss of game state due to multiple peer failures can be reduced to acceptable levels.

4.3.5 Latency

In client server architectures the server is responsible for the game state, and clients only connect to the server to receive and send updates. This means that when a player has a high latency it will be slow in sending and receiving its updates, possibly hindering the playability. However, the high latency of this one player does not affect the game for all the other players, they can still send and receive their updates to and from the server without any delay. Even interactions with the player with the high latency will not pose any problem since the server administrates its state and changes to it can still be made.

In a peer to peer architecture this is more troublesome. With synchronization protocols as described in section 4.3.2 the pace of the entire game is dictated by the slowest peer. The protocols can be adjusted so that the consistency of the slow peers is sacrificed to ensure the playability for the other peers. However then you still have the problem that these slow peers can be responsible for the state objects in the game world and of the character that they represent, and that the interaction with these objects and the character will be slow for all other peers. Algorithms are imaginable that will prevent slow peers to be responsible for any objects or even for the state of their own character, but this will hinder the playability for that peer to an extend that it is questionable if it is playable at all. Therefore GauthierDickey et all. in [7] propose using a minimum latency requirement for all peers, so that users with a connection that is too slow simply won't be able to play at all.

4.3.6 Cheating

One very specific issue that all multiplayer games have to deal with is cheating. In client server architectures there is a single governing authority that can prevent players from gaining an unfair advantage over others. In peer to peer architectures this is a lot harder.
Gauthier et all. [7] distinguish three different types of cheats: protocol level cheats, game level cheats and application level cheats.

- Protocol level cheats occur by modifying the protocol, such as changing the contents of packets. There are five common protocol level cheats.
  Fixed-Delay Cheat: Add a fixed amount of delay to your outgoing packets, so that you can react faster to the actions of other then they can on yours
  Timestamp Cheat: Send wrong timestamps on your actions to make them appear having happened earlier. For example send a move action after having received a 'player x shoots you' action, with a timestamp before it, so that the shot will miss.
  Suppressed Update Cheat: Don't send updates of your own actions while still receiving updates of others, to 'hide' from them.
  Inconsistency Cheat: Send different updates to different players so that the game world will go out of sync.
  Collusion Cheat: Players sharing updates to gain information from the other that they should not have received themselves.

These cheats can be prevented by using a protocol that eliminates the possibility for them, an example of a protocol that eliminates most of these cheats is described in [7]. However these protocols add delays en bandwidth use for the users.

- Game level cheats occur by breaking the rules of the game. For example, a player could move from place A to B in one step, while this should take him a long journey. Peer to peer games are especially vulnerable to this because there is no governing authority that handles all actions and can reject them if it should not be possible. Also because the peers themselves are responsible for the state of the objects they can alter them to get an advantage. For example, right after you loot a chest you modify it so that it is full again. These cheats can be partially prevented by assigning objects randomly to peers, as for instance in [13], so that peers have no control over what part of the game state they are responsible for. Another way is to replicate the state of all objects over multiple peers, and have them check for each update if it is legal.

- Application level cheats occur by modifying the code of the game. A common cheat that is used is for example modifying the graphics engine so that walls become invisible so that you can easily locate objects and players. Peer to peer architectures are not more vulnerable to this type of cheats then other architectures are.

4.4 Benefits and downsides

The use of peer to peer architectures in MMOGs brings a number of advantages and disadvantages with it, these benefits and downsides will be discussed in this section.

4.4.1 Benefits

By eliminating a central server and distributing the load over all users in the network you gain a number of benefits.

- Robustness. Peer to peer architectures have the ability to offer a very robust network. When one peer node fails the rest of the network can continue to operate normally. This isn't necessarily true for any peer to peer architecture, but it is a typical advantage of peer to peer architectures to make it possible that node crashes have no significant impact on the network as a whole. For MMOGs this means that the game will always be available. As mentioned previously most proposed peer to peer architectures for MMOGs still employ some kind of central server to keep track of the nodes that are active in the network. This server would still form a 'single point of failure', however this server has a low complexity and low load and can be implemented redundantly so that it will still offer a great robustness.

- Scalability and flexibility. Arguably the greatest benefit of peer to peer architectures in MMOGs is that they are extremely scalable and flexible. The load on the game and the network is directly related to the number of users that are active on it. The more users that are active in the game, the higher the load on the network will be. In a peer to peer architecture the total capacity of the network is equal to the sum of the capacities of all users. This means that when more users become active in the game not only the load will increase but also the total capacity of the network. If the amount of load that each new user adds to the network does not exceed the amount of capacity that he brings, then the network will theoretically be able to handle an infinite amount of users. However, if peers have to send their updates to too many other peers this can exceed their bandwidth resources. If the game world becomes too overcrowded even locality management as described in section 4.3.1won't be able to prevent this. Furthermore, playability issues with overpopulating the game world will still exist.
- Low cost for running the game. Peer to peer architectures do not require large infrastructure investment to keep them running. To operate the game you won't need any expensive servers or system administrators. The development and maintenance process for the game developers will essentially be the same as for any single player game. The game software can simply be sold to the users through normal distribution channels, and updates for it can be made available. No added network infrastructure is needed. However, as previously argued it will be difficult to make a MMOG 'pure' peer to peer. Some kind of central server will almost always have to be in place, but due to the low complexity and low load on it it will most likely not form a huge cost.

4.4.2 Downsides

The lack of a central server and a central authority in peer to peer architectures brings a number of disadvantages for the operation of the game. These disadvantages will be discussed below.

- No governing authority to counter cheating. One of the main disadvantages of peer to peer architectures is that there is no single authority that sees to it that the rules of the game are applied correctly. In a distributed architecture each peer is taking its own decisions, and there is no authority to check the legality of the actions and identify potential cheaters. For instance, players moving faster then they should be able to or gaining access to areas, objects or skills that should not be accessible to them. To prevent cheating in peer to peer architectures other systems have to be set up. Partially this can be done by developing 'cheat-proof' protocols, and other systems to check or double check users′ actions such as software agents. But it is questionable if cheating can be eliminated completely without some sort of independent global referee.

- No easy identifiable global game state. In peer to peer architectures the game state is distributed over all the peers in the network. This means that there is no single entity that has the complete game state, because of this it is much harder to ensure that all users always have the view of the game state. When the state of objects is dynamically assigned to peers corruptions and inconsistencies can easily develop. For instance two peers can think that they are responsible for the same object at the same time, when both objects are changed in different ways it leads to two different game states. Without some sort of referee to decide which object is the true one it will be hard to merge these again.
- Not easily commercially exploitable. Peer to peer architectures do not easily fit in with the business model that most MMOGs use today. Most MMOGs require you to pay a monthly fee to be able to play. With a peer to peer architecture this becomes much harder to enforce because users simply form their own network and don′t need the developer anymore. The developer can choose to operate a central server to manage the accounts and ensure that only users who have paid get access to the network, but because all of the game logic lies in the client software and the server will technically be very simply, it will be very vulnerable for piracy. Another solution would be to change the business model and charge a price for the software and no monthly fee, like most single player games. However, MMOGs typically are games which intend to have longtime playability for the users by continually developing new features and content. These new updates and features can be sold again to the players as expansions, but the lack of a monthly subscription will bring more uncertainty to the developers and will make it less economically feasible to continue to develop new content for the game.
- Software complexity. Because of all the issues mentioned previously the software complexity of a peer to peer based MMOG will be very high. MMOGs already are very complex software systems that require years to develop, doing it with a peer to peer architecture greatly increases the number of ‘things that can go wrong′. It can be argued that a lot of the added complexity of peer to peer comes from the fact that it is fairly new and unused, and that therefore there are no best practices yet. While this certainly is true peer to peer also has a fundamental higher complexity then client server architectures because of all the added issues it has to deal with, such as: game state consistency, fault tolerance, cheating.

5 Client Server versus Peer to Peer

When you review the scientific research that is being done in the area of Massive Multiplayer Online Games one of the first things that you will notice is that a relatively low amount of research is being done into the client server architectures that the vast majority of the current games use, and a relatively high amount into peer to peer architectures that almost no game uses. The question that automatically comes to mind is whether or not peer to peer architectures will make it into the mainstream games and if it will possibly take over the dominant position of client server architectures. This chapter will attempt to answer this question and will explore the possibilities of a hybrid architecture that offers the best of both worlds.

5.1 Client server or peer to peer

To answer whether or not peer to peer architectures will make it into mainstream games you have to look at the interests of the people that develop these games. Their motivations for choosing an architecture will not only be of a technological nature but economical motivations will also play an important role. Creating a MMOG typically takes two to three times as long as creating and launching a single-player game (2 to 3 years versus 9 to 12 months [2]). MMOG projects from major PC games studios can require a five-year timetable and hundreds of people′s efforts prior to release. For example Electronic Arts/Westwood Studio′s Earth & Beyond MMOG didn′t appear until 2003, even though its development began in 1998. Besides this long development time MMOGs are also expected to have a long lifetime in which development will continue in the form of fixing bugs and adding content. Developing a MMOG is a large investment for a development studio, and because of this it is no surprise that they rather use existing technologies and proven concepts than experiment with new technologies.

Development studios have a number of reasons to be hesitant in using peer to peer architectures for their MMOG projects.

- Software complexity/uncertainty. No major MMOG has been developed yet using a peer to peer architecture, so there are no proven concepts or best practices. Of course this argument only apply to the first MMOGs that will be developed using peer to peer, but even when best practices start to appear the fundamental complexity of peer to peer architectures will remain higher, adding to the development time and cost and/or amount of bugs in the game.
- Cheating. Because MMOGs are continuous games with a persistent state cheating has a much bigger impact on them than on other games. If somebody cheats in a game of chess he will just win that game, but the benefit of the cheat will not carry over to any of the next games he will play. When somebody in a MMOG gains an object through a cheat he will have that object

until the end of the lifetime of the game. It is fundamentally easier to cheat in games with peer to peer architectures. Developers will have to continuously try to counter all the new cheats players develop, requiring a lot of resources.
- Business model. Almost all MMOGs that are on the market today rely on a business model where players have to pay a monthly subscription fee to play the game. This model works well with client server architectures, but much less so with peer to peer ones. Peer to peer style MMOGs will be much more vulnerable to software piracy just like single-player games and ′small′ multiplayer games (16 to 64 players) are, which will reduce the profitability for the developer.

The benefit of not having to setup and maintain a server and everything that comes with it (costs, low scalability, single point of failure), will have to weigh up against the problems that peer to peer architectures bring with them. At the moment, for most development studios, the benefits of peer to peer do not weigh up against the downsides. The disadvantages of client server architectures mainly relate to costs. If you put enough money into it then limited scalability can be countered by having a lot of over capacity and the impact of a single point of failure can be reduced by introducing redundancy and continuous server monitoring and maintenance. Developing and maintaining a MMOG brings large costs with it, but even with these costs it is a very profitable enterprise to exploit successful MMOGs. Because of this there is little incentive in the industry to switch from their proven concept to more risky peer to peer architectures.

It can be said that the lower the number of players in a game, the more attractive it is to use peer to peer instead of client server. When you divide the costs of a serer infrastructure by a huge number of players, it doesn′t amount to much, but when the amount of players becomes less the costs get more significant. Furthermore the complexity of the software for a peer to peer architecture will be much lower when there is only a limited amount of players. There are less people you have to communicate with so interest management won′t be needed, synchronization will be less of an issue because there are less people to synchronize with, and it is easier to keep track of everybody else in the game to detect and prevent cheating. For this reason peer to peer technologies are getting more and more popular for multiplayer games of 16 to 64 players. In these games, that typically do not carry a game state over from one session to the other, almost all of the benefits of peer to peer architectures apply (no cost of running a server, high scalability, high robustness/availability) while the downsides do not apply so much since these games have a relative low complexity, cheating is less of an issue because it is not carried over from game to game, and these games already use a ′one-time-purchase′ business model.
If this development will scale to large multiplayer games and MMOGs remains to be seen. With the number of players the complexity of the software will increase

significantly and with the current successfulness of client server based MMOGs there is little reason to switch for the major development studios. However it is likely that researchers and small development studios will continue developing peer to peer technologies and that this will lead to small MMOGs for limited numbers of players.

5.2 Combining architectures

When looking at the advantages and disadvantages of both client server architectures and peer to peer architectures you automatically wonder if both architectures can be combined in some way to offer all of the benefits with none of the downsides.

Creating an 'optimal' combination with all benefits and no downsides is impossible. The main disadvantages of client server architectures come from having the server and everything it brings with it: the cost for setting up/maintaining, limited scalability/flexibility and single point of failure. While the disadvantages of peer to peer architectures come forth from not having a server: no 'referee' to counter cheating, no easily identifiable global game state. These are almost mutual exclusive. When you try to remove a disadvantage from peer to peer architectures with an element from client server, you automatically gain a disadvantage from client server. For instance, you want to counter the disadvantage of peer to peer architectures that there is no 'referee' to counter cheating. You could do this by adding such a referee. However the question then raises itself who should run the software for this referee. If it is run by peers then it is susceptible to cheating by the very peers that run them, if you run it on an independent machine you have created a server that brings costs, a single point of failure and loss of scalability.
When you want to increase the scalability of the server by having some process be handled by the peers then those processes automatically are susceptible to cheating and you gain consistency issues.

This however does not mean that combining peer to peer elements with client server elements is never beneficial. As already mentioned previously the disadvantages of peer to peer architectures do not apply so much to games with a limited number of players or games that do not carry over a game state from one session to another. It depends on the games specifics what elements of what architecture will be most beneficial for it. For MMOGs it might not be feasible to eliminate a central server completely, and it is probably best to have the server maintain the entire game state, but certain elements like for instance the AI of monsters might be done by peers. This makes it susceptible to cheating in some degree, but with a high enough amount of peers and random assignment of responsibilities the risk of a player being able to manipulate things in his advantage can be reduced to an acceptable level. What elements exactly will be possible to do in a peer to peer fashion is so dependent on the game's specifics that no general judgments can be made on it.

6 Conclusion

This bachelor thesis has described the use of client server architectures and peer to peer architectures for Massive Multiplayer Online Games, and has compared these two architectures in order to determine which is suited best for the task.
The central question of this bachelor thesis was:
"What different architectures are being used in Massive Multiplayer Online Games, and which is best suited to overcome the typical challenges that these games offer?"

The initial study of the relevant literature showed that the MMOGs that currently exist almost exclusively use client server architectures. However, a great deal of scientific research is being done into developing a peer to peer style architecture for these games. Therefore the first part of the question can be answered with: client server and peer to peer.
Client server architectures have been used in both MMOGs and other applications for many years. Therefore a lot of knowledge has been developed about their use, which has led to the typical four-tier architecture that are commonly used in MMOGs. This common architecture has been described in chapter 3, along with the techniques that are used to distribute the server load.
The use of peer to peer architectures for MMOG style games is fairly new, there are no best practices or common architectures for it (yet). There are a number of issues that the use of a peer to peer architecture brings that all these architectures have to deal with. These issues have been described in chapter 4.

To order to answer the second part of the research question, the advantages and disadvantages of both client server and peer to peer architectures have been discussed and these have been compared against each other in chapter 5. The main advantage of peer to peer architectures is that they reduce the cost of running a MMOG by eliminating the cost of setting up and maintaining a central server. However by doing so they greatly increase the complexity of the software, increase the possibilities for cheating and decrease the possibilities to commercially exploit the game. This in turn increases the cost and potentially lowers the income for the game developer and with that cancels out the benefit they got from using a peer to peer architecture in the first place. Therefore the conclusion of this comparison is that at this moment client server architectures are better suited for use in big MMOGs. However, the downsides of peer to peer architectures do not apply as much to smaller games with less players, and therefore peer to peer architectures are the better choice for those games. Whether future developments will eliminate the disadvantages of peer to peer architectures for the big commercial MMOGs remains to be seen, but for the foreseeable future client server architectures will remain the favorable choice.

7 References

[1] Vleeschauwer, Van Den Bossche, Verdickt, Turck, Dhoedt, Demeester “**Dynamic microcell assignment for massively multiplayer online gaming”** in Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games NetGames '05

[2] Tsun-Yu Hsiao, Shyan-Ming Yuan “**Practical Middleware for Massively Multiplayer Online Games”** in IEEE Internet Computing, Volume 9 Issue 5

[3]Caltagirone, Keys, Schlief, Willshire “**Architecture for a massively multiplayer online role playing game engine”** in Journal of Computing Sciences in Colleges, Volume 18 Issue 2

[4] Assiotis, Tzanov **“A Distributed Architecture for Massive Multiplayer Online Role-Playing Games”**

[5] Kang-Won Lee, Bong-Jun Ko, Seraphin Calo **“Adaptive server selection for large scale interactive online games”** in Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video

[6] Griwodz **“State replication for multiplayer games”** in Proceedings of the 1st workshop on Network and system support for games NetGames '02

[7] Gauthier, Dickey, Zappala, Lo, Marr **“Low latency and cheat-proof event ordering for peer-to-peer games”** in Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video

[8] Kaneda, Takahashi, Saito, Aida, Tokuda **“A challenge for reusing multiplayer online games without modifying binaries”** in Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games NetGames '05

[9] Fiedler, Wallner, Weber **“A communication architecture for massive multiplayer games”** in Proceedings of the 1st workshop on Network and system support for games NetGames '02

[10] Kim, Choi, Chang, Kwon, Choi, Yuk **“Traffic characteristics of a massively multi-player online role playing game”** in Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games NetGames '05

[11] Fritsch, Ritter, Schiller **“The effect of latency and network limitations on MMORPGs: a field study of everquest2”** in Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games NetGames '05

[12] Gautier, Diot “**Design and Evaluation of MiMaze, a Multi-Player Game on the Internet**” Proceedings of the IEEE International Conference on Multimedia Computing and Systems ICMCS '98

[13] Knutsson, Lu, Xu, Hopkins **“Peer-to-Peer Support for Massively Multiplayer Games”** In Proceedings. of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM) 2004.

[14] Baughman, Levine **“Cheat-proof Playout for Centralized and Distributed Online Games”** in IEEE Infocom, 2001.

[15] Cronin, Filstrup, Jamin **“Cheat-Proofing Dead Reckoned Multiplayer Games”** in Intl. Conf. on Application and Development of Computer Games, January 2003.

[16] Cronin, Filstrup, Kurc **“A Distributed Multiplayer Game Server System”** 2001