A Survey on the Scalability of MMOGs

Delft University of Technology, The Netherlands

LieYen Cheung
liyuen@gmail.com
Olivier Hokke
oili4@wolfox.nl

Daniel van der Ende
daniel.vanderende@gmail.com
Ruud Visser
visser.rgm@gmail.com

ABSTRACT
An increasingly popular class of Networked Virtual Environments is Massively Multiplayer Online Games (MMOGs), where massive amounts of players interact simultaneously in virtual game worlds. As the demand increases for MMOG systems that support millions of players, so does the need for solutions to their scalability problems. The servers, and in some cases also clients, are required to work properly under heavy load, whilst maintaining an accurate game state, minimizing latency, and ensuring that bandwidth requirements do not exceed a maximum.

Although many scalability oriented solutions exist, few comprehensive surveys are available. In this article, we describe a taxonomy of scalability of MMOGs, and map surveyed solutions onto its classes. We find that many promising solutions are available, but insufficient research has been done on combinations of them. Therefore, in search of the best performing system, we recommend to combine promising solutions into a couple of different systems, to test each system’s performance.

1. INTRODUCTION
With hundreds of thousands of players, Massively Multiplayer Online Games (MMOGs) are becoming increasingly popular. However, they still suffer from scalability issues. Few comprehensive in-depth surveys on this matter are available, and therefore we contribute this article. In this work we survey promising solutions from over 80 references, on the following four major scalability related issues: Processing Power Allocation, Bandwidth Reduction, Latency Minimization, and Game State Synchronization. We briefly introduce each solution while explaining its working, and give a short survey with its important advantages and disadvantages.

Networked Virtual Environments (NVEs) [76] are artificial worlds that exist only digitally and allow many participants across the globe, to interact simultaneously. The first NVE, SIMNET [66], was created in the 1980s for military simulation purposes. NVEs have since then been adopted in many fields, such as online gaming.

MMOGs, are essentially Networked Virtual Environments, built for entertainment purposes. These type of games have become increasingly popular [2]. For example: World of Warcraft has globally more than 10 million subscribers and roughly 2 million concurrent users; Second Life has almost a million active accounts; Lineage II had a peak of more than 3 million subscriptions around 2004; etc. By now, the MMOG genre has become a multibillion dollar business [56].

Scalability is a major issue for MMOGs, as such games should be capable of handling hundreds of thousands of concurrent users. Thus, there is a need for architectures and methods that can handle the heavy resulting loads, whilst maintaining an accurate game state, minimizing latency, and ensuring that bandwidth requirements do not exceed a maximum.

To distribute heavy loads, common MMOGs use multiple servers, and even combine them in parallel, into a single physical supercomputer. These servers are often clustered in different geographical locations to minimize latency. Synchronizing game states among clients is usually managed by servers, and require a high speed internet connection in order to handle resulting bandwidth requirements.

Many more scalability oriented solutions exist. Therefore, in this work, we survey some promising solutions that improve scalability of MMOGs to some extent.

The remainder of this article is structured as follows. In Section 2 we explain the main definitions we use within this work. In Section 3 we define a taxonomy with four classes, after which solutions to scalability issues are surveyed in Section 4, by mapping related papers to the taxonomy. In Section 5 we point out some recommendations on future research. Section 6 concludes this work.

2. BACKGROUND
In this section we discuss a number of system architectures: Client-Server architectures, Peer-to-Peer architectures and Hybrid architectures. We describe each of these in detail to provide a clear definition of each of these.

2.1 Server-Client Architecture
In a traditional Server-Client architecture, the server is responsible for handling all game interactions for all players. These interactions include player-player interactions, such as firing a weapon at another player. The clients receive information from the server to update their game state as
necessary. A Server-Client architecture is, however, quite vulnerable to failures, as it contains a single point of failure. Server maintenance, for example, may lead to server downtime, leaving gamers unable to play for some time.

Sometimes in a Server-Client architecture, a server is not a single machine, but a cluster of machines. Otherwise, a single machine would be overloaded very easily, making online gaming almost impossible. For the purpose of clarity, we define a server to be one single machine in this article, unless indicated otherwise.

### 2.2 Peer-To-Peer Architecture

In a Peer-to-Peer (P2P) architecture, all machines function both as servers and as clients. Rather than depending on one server, or even a cluster of servers, a P2P architecture distributes the workload and tasks among the machines. In this manner, every machine involved in the game contributes some resources, which may be in the form of bandwidth, storage or processing power, to enable the proper operation of the game that is being played. Furthermore, a P2P architecture does not have a single point of failure, as the failure of one machine does not lead to the failure of the entire network, due to the network’s distributed character.

### 2.3 Hybrid Architecture

A Hybrid architecture combines elements from both the Server-Client and Peer-To-Peer architectures. Some machines in a hybrid architecture may function in a similar manner to servers as described in Section 2.1, whereas others may function as peers, as described in Section 2.2. This implies that a certain amount of workload can be assigned to servers whilst other computations can be executed in a distributed manner by peers. A Hybrid architecture, however, does reintroduce the single point of failure, which a P2P architecture does not have.

### 3. A TAXONOMY OF THE SCALABILITY OF MMOGS

As we mentioned earlier in Section 1, we focus in this article on the scalability problem of MMOGs and define a taxonomy for this. We identify four main classes for the taxonomy (See Figure 1): Processing Power Allocation, Latency Minimization, Bandwidth Reduction, and Game State Synchronization. Furthermore, we categorize the most promising solutions into two or more subclasses for each main class.

In this section, each class and its subclasses are introduced. For each subclass we identify its current challenges and sum up the most promising solutions that we will survey in Section 4.

#### 3.1 Processing Power Allocation

The developers of MMOGs face many challenges concerning the processing power of both clients and servers. When overloaded, a machine’s CPU may become a performance bottleneck for the whole network, since other nodes might depend on it. Overloaded CPUs fail to keep up the pace, and thus delay certain elements of a game. Consequently, players become dissatisfied either due to low response times of nodes in the network, or due to the overloaded CPU of the players’ own machine. Therefore, we need methods for allocating processing power among nodes in a network, to avoid performance bottlenecks. We define this as Processing Power Allocation (PPA).

We identify two different subclasses: the allocation of processing power among servers and among clients. In the following two subsections, we introduce each subclass, we describe its challenges to overcome, and sum up related solutions that we survey in Section 4.1.

**Server-Side PPA:** just like any other computer, servers have limited processing power. If a server struggles to perform all of its assigned tasks in time, some or all players within an MMOG may experience delays, due to low response times. Often, many game world updates can only be executed as fast as the slowest server, since other servers may depend on it. Such a server is a performance bottleneck and subsequently results in many dissatisfied players.

When a large number of clients connect to a server, the server often has to cope with heavy workloads. This might happen when two clans decide to have a war in a previously deserted area of the game world, suddenly creating a high player density within this area. However, if just a few players are connected to a server, unused processing power may be wasted when another server is overloaded.

Basic methods with small computational footprints exist, where the best allocations of processing power are predicted and manually configured. We categorize these methods as Manual Processing Power Allocation (MPPA), which forms the basis for the Server-Side subclass. However, problems may arise if the number of connected players exceeds the expectations. When more resources are needed, the game might have to stop in order to configure and/or install servers.

We define the method of allocating processing power dynamically, and thus handling fluctuating player densities, as Dynamic Processing Power Allocation (DPPA). DPPA methods range from assigning unused servers, to re-dividing workloads over used servers. We consider these methods to be the most promising and they are therefore our main focus on PPA.

In Section 4.1.1 we discuss several solutions for the mentioned challenges. Only one MPPA method, called Server Cluster Expansion, is discussed, as it is a popular choice and has very low development costs. The following DPPA methods are discussed afterwards: Zoning, Instancing and Replication; Microcells; and On Demand Service Platform.
Client-Side PPA: Within an MMOG, it is preferable to allow players with inexpensive computers to have a fluent game experience. However, large numbers of players may be involved, and MMOGs have become more complex. Depending on the network architecture, there is a need for minimizing the workload that clients may experience with MMOGs.

MMOGs that use a traditional Client-Server architecture, often allow players to dedicate most of their processing power to local computations, that affect only the players’ own gaming experience. However, network communications also require the use of a players’ CPU. Depending on how the system deals with high player densities, this might pose a problem.

MMOGs based on P2P networks are completely dependent on the processing power of clients, as no traditional servers are used. All game entities have to be processed and hosted in a completely distributed manner by clients. Such a network would require a self-organizing system. Clients contribute resources to the network, because their machines can act as servers or they can take up some server-related roles. Although new challenges have to be overcome before such a P2P system may be used commercially, it promises scalability and affordability.

In Section 4.1.2, we survey the following solutions: Interest Management, The Voronoi-based Overlay Network, and Mediator.

### 3.2 Bandwidth Reduction

Bandwidth is a very important factor when considering the scalability of MMOGs. The amount of data sent, and the manner in which they are distributed, play a significant role when systems are to be implemented on a large scale. In this article, we focus on methods intended to achieve a reduction in bandwidth use. We define bandwidth reduction methods as methods that solely focus on ways to reduce the total number of bits sent over a network.

Players have access to limited network bandwidth, for example 4 Mbps. Interacting players must frequently exchange state information. This exchange of information results in considerable bandwidth use. High numbers of players would thus exceed the bandwidth available to players. As a consequence, to accommodate ever-increasing numbers of players, methods for reducing the amount of data transferred per player are necessary. In Section 4.2, we discuss a number of these methods: Interest Management, Dead Reckoning, and Delta Encoding.

**Interest Management** When playing an MMOG it is essential for players to be able to interact with other players. To do so, players must be aware of the position and state of other players. If played on a small scale, it is possible for players to maintain a full copy of all player information. When the game increases in scale, however, this becomes almost impossible to achieve. Therefore, it is important that only a selection of players send information regarding their position and state, to avoid information overflow and thus reduce the bandwidth used. We define Interest Management to be the process of determining which information is relevant to a player, and which information is not.

In Section 4.2, we discuss the following methods: Proximity-based techniques and Vision-based techniques.

**Dead Reckoning** is a technique that can be used to reduce the number of messages that need to be sent amongst players. By reducing the number of messages, the amount of bandwidth used is reduced as well. Dead reckoning is a technique used to extrapolate known information regarding other players to provide a more realistic game experience to the player. Dead Reckoning is based on small, infrequent updates sent to all players of the game. These updates contain information regarding a player’s position, speed and direction. By applying the Dead Reckoning algorithm, all players use these updates to estimate the position and other attributes of each player.

Pantel et al. [69] have shown the importance of the use of these algorithms. Without accurate algorithms, the deviation of the perceived position, relative to the actual position, increases dramatically. By applying the algorithm to its own data, each player is also aware of its own perceived position. Using this knowledge, each player can determine when the difference between its perceived and actual position would lead to inconsistencies. If a certain threshold value is exceeded, a new update is sent, regardless of whether this update is premature relative to the ‘scheduled update’. The choice of threshold value is therefore of vital importance to the effectiveness of Dead Reckoning, and implicitly also to the amount of bandwidth reduction that can be achieved by its use. A more detailed explanation of Dead Reckoning is provided by Jesse Aronson [6] including some code examples.

In Section 4.2, we discuss the following methods for determining the thresholds to be used: Fixed thresholds and Adaptive thresholds.

**Delta Encoding** is a technique used to considerably reduce the amount of data contained in each message within a game. Its basic premise is that if information has not changed since the previous message was sent, there is no need to retransmit the information. As such, Delta Encoding entails the omission of certain information depending on the differences between the previously sent information or the lack thereof. By removing this unaltered information from the message, the size of the message can be reduced significantly, especially in cases in which certain information changes at a very slow rate.

It is also important to consider possible package losses, which may occur due to a variety of reasons. Delta Encoding may not be sufficient in such cases. Therefore, it is also important to identify a method of coping with such situations.

In Section 4.2, we discuss the following Delta Encoding methods: Traditional Delta Encoding and Delta Encoding with robustness features.

### 3.3 Latency minimization

Latency forms a big obstacle when developing online games, as high latency can make it very difficult to play a game properly. We define latency as the time needed for a message to travel from one node in a network, to another.
Latency depends on a number of factors, such as the transmission length and the amount of routing required for sending a message over the network. Physical distances between nodes in a network may influence either of these 2 dependencies. As a result, latency compensation techniques usually intend to establish a compromise between consistency and responsiveness.

We identify two types of latency compensation techniques for MMOGs: synchronization techniques and optimistic techniques. Synchronization techniques are mainly used to improve the synchronization between the player and server, while optimistic techniques intend to execute a game optimistically (i.e. by assuming a game runs perfectly unless told otherwise) and solve inconsistencies when they arise.

In Section 4.3 we discuss a number of these methods: Forwarding Pool, Latency Equalization, Geographical Server Positioning and Voronoi-based Overlay Network.

The Forwarding Pool is used to help a client with sending its update messages to all other nodes that require it. This is necessary when a client does not have sufficient bandwidth capacity available to send all messages on its own.

For more details about this technique we refer to Section 4.3.1, where we discuss some approaches based on the forwarding mechanism, like the Forwarding pool and Forwarding Model.

Latency Equalization is a technique that can be used to minimize the latency difference among multiple clients. By making use of this technique, the interactive experience and game quality can be improved. Players will have a better play experience, when latency differences are minimized. A press release from AT&T [7] emphasizes the importance of reducing latency imbalances when wide geographical areas are being scaled.

Previous work [27, 30, 81] on online interactive application experiences, shows that equalized delay can be achieved by solutions either at the client-side or server-side.

Techniques at the client-side require all clients to exchange latency information, which makes it difficult to implement solutions for equalizing delay. Furthermore, these solutions are also vulnerable for cheating [14].

Server-side approaches are not sufficiently accurate in some scenarios, because they rely on the server to estimate the network delay. Computing the estimated delay also limits the number of client the server can support.

We discuss in Section 4.3.2 how latency equalization can be achieved, using the LEQ service proposed by Yu et al. [80]. We also look at the following approaches: buffering packets, source routing and MPLS.

Geographical server positioning There are various methods for reducing latency, however many of these may not be applicable in all cases. Basically there are two approaches to minimize latency. One approach is from the side of the server, which means the server uses algorithms or techniques to achieve the goal. Another approach is from the side of the client, in which clients may be able to choose to play on servers in geographical proximity to themselves to reduce latencies. Geographical server positioning is a technique that can be used to support the above mentioned approach. As many games are extremely sensitive to latency, clustering servers geographically close to players is extremely important.

Previous work shows that lag [1] plays an important role in player experience and performance. It has also been shown that players tend to switch to near-by servers, if the latency increases [49, 50, 63]. It is important to determine how distributed data centers must be, because many Multiplayer games service providers cannot afford to have a data center in every location throughout the world. In addition, Multiplayer games service providers should define the maximum accepted delay, when determining the position to place a data center. The host game service requires more clusters in geographic location if the geographic distribution of players is tightly correlated with the geographic location of the server. Is this correlation is weak then the host game service can use few strategic locations to place those clusters.

In Section 4.3.3 we discuss several approaches for geographically server positioning. Like SEMMO [48], approaches proposed by Bescow et al. [8] and Jiang et al. [57].

Voronoi-based Overlay Network may be used to maintain a highly overlay topology consistency. This technique can be used for lowering latency. In a Voronoi diagram the latency with nearest neighbors is the lowest [17]. In Section 4.3.4 we discuss the Voronoi-bases Overlay Network. More information about Voronoi diagram can be found in the middle of Section 4.2.1.

3.4 Game State Synchronization

Game states are a collection of objects and properties you see in a game. Game state synchronization techniques ensure that every player has the same state and sees the same objects. It is undesirable to have different game states in one game, because players will see different versions of the game. Therefore, each player’s game state must be synchronized. There are various techniques to achieve this goal, but not every technique is suitable for a highly scalable game.

In this section, we discuss the following techniques: Event Based Synchronization and Voronoï Based Synchronization. Each of these techniques tries to ensure that the game state remains synchronized with a minimal amount of latency.

Event Based Synchronization is a technique to synchronize game states based on events. It uses events to keep the game up to date. Each event is sent to all the player’s in the game. All the players process the events in an optimistic way. Optimistic means that each received event is processed immediately, without checking if the events are processed in the right order. Events can have influence on each other, that way it is very important that each event is executed in the right order. Otherwise, the game
state would not be consistent anymore. If an event is triggered too late, the game state is not consistent anymore. Therefore, there is need for a rollback. When a rollback is performed, the state is reversed.

There are various ways to achieve this, in section 4 we elaborate on three methods, Time Warp Synchronization, Trail State Synchronization and Breathing Bucket Synchronization.

**Voronoi Based Synchronization** (VBS) is a Peer-to-Peer state management method. The game state is synchronized by updates instead of events. In VBS the game world is dynamically partitioned into a Voronoi diagrams [51], see Section 4.1.2 for definition of Voronoi Diagrams. These diagrams combine a group of users, the users are peers. The users who have spare capacity are arbitrators. Arbitrators can communicate with arbitrators in other cells.

There are three mechanisms in VBS: Consistency Control, Load Balancing and Fault Tolerance. Consistency control is in VSM update-based. Each Voronoi cell has its own arbitrator, events generated in the cell are first send to the arbitrator. The arbitrator processes and forwards the update. Load balancing is used to ensure that nodes do not exceed their capacity. If the arbitrator exceeds its capacity, he asks the server for peers with spare capacity. Because VSB relies on user supplied resources fault tolerance is important. Each arbitrator has a backup arbitrator, so if an arbitrator fails, a backup arbitrator can transfer the game updates.

In Section 4 we survey how the Voronoi diagrams are use in VBS and we elaborate on the method Voronoi State Management.

### 4. MAPPING SURVEYED METHODS ON OUR TAXONOMY

In this section we map a number of methods onto the taxonomy we described in Section 3. We elaborate on a number of challenges within each class, as defined in Section 3, and discuss a number of papers that address these challenges, providing a more in-depth view of each of these areas.

#### 4.1 Processing Power Allocation

We identify two different subclasses of Processing Power Allocation (PPA): **Server-Side PPA**, which is the allocation of processing power among servers; and **Client-Side PPA**, which is the allocation of processing power among clients. In Table 1 we map surveyed papers onto our taxonomy. Based on these papers, we identify and survey the most promising solutions on Processing Power Allocation throughout the next two subsections.

##### 4.1.1 Server-Side PPA

As explained in Section 3.1, methods for allocating processing power among servers are essential, since servers have limited processing power. Often many game world updates can be executed only as fast as the slowest server, since other servers may depend on it. In Section 3.1 we identify two types of PPA: **Manual Processing Power Allocation** (MPPA), and **Dynamic Processing Power Allocation** (DPPA). With MPPA methods, the allocation of processing power is manually defined. DPPA methods allocate processing power dynamically during runtime.

The remainder of this section is structured as follows. Firstly, we survey a simple MPPA method, **Server Cluster Expansion**, on balancing server load. Secondly, the three novel methods of **Zoning**, **Instancing and Replication** are surveyed, where zones, instances, and replicas of the game world are dynamically distributed over a server pool. Thirdly, we discuss the **Microcell** method, which dynamically distributes small subareas over a number of servers. Lastly, an **On Demand Service Platform** is surveyed, which provisions servers among multiple games and business applications. Each of these methods will be briefly explained, after which a survey will follow.

**Server Cluster Expansion**: The simplest method of ensuring that no performance bottlenecks arise, is by adding sufficient servers to the cluster. This has to be done manually, thus we consider this an MPPA method. DPPA methods provision processing power over the available servers, but these methods can not cope with workload requirements that exceed the total available processing power of a whole server cluster. Server Cluster Expansion (SCE) forms a basis of PPA.

A simple example of an SCE, is the expansion of the server cluster for the game Savage Eden: The Battle For Laghaim in 2007 [28]. More servers were needed to handle the increased player activities. However, the expansion created an opportunity for the developers to implement some new features in the game, so that server down times were used more efficiently.

We believe that SCE should be avoided if possible, because firstly, it is a very expensive and slow method. Secondly, an SCE does not solve the issue at the time it was needed, but for some time, it might prevent the issue from happen-

<table>
<thead>
<tr>
<th>Processing Power Allocation</th>
<th>Server-Side</th>
<th>Client-Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahmed et al. [4]</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Cross et al. [20]</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Fan et al. [33-35]</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Funkhouser [40]</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Frey et al. [50]</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Gamastra [41]</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Glinka et al. [44, 45]</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Hu et al. [51-53]</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Pluez et al. [70]</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Nae et al. [70]</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Shaikh et al. [74, 75]</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>De Vleeschauwer et al. [26]</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Zyda et al. [85]</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 1: Mapping of papers related to Processing Power Allocation onto the taxonomy. Abbreviations: **Server Cluster Expansion** (SCE), **Real-Time Framework** (RTF), **Microcells** (Mc), **On Demand Service Platform** (ODSP), **Interest Management** (IM), **Voronoi-based Overlay Network** (VON) and **Mediator** (Md).
ing again. Thirdly, it may be hard to predict the amount of servers that are needed to support unexpected peaks in server load. And lastly, an SCE may even require a temporary shutdown of the system, depending on the game design and implementation.

However, by combining server maintenance with SCE, server down times can be used more efficiently. Furthermore, when even DPPA methods fail to support peaks in server loads, due to server load requirements that exceed the total available processing power in a cluster, SCE might be the only best solution.

**Zoning, Instancing and Replication (ZIR)** are 3 novel DPPA methods on allocating processing power dynamically. They are introduced with the Real-Time Framework (RTF) proposed by Glinka et al. [44, 45, 70]. The Zoning method partitions the world into smaller areas, called zones, which reside on separate servers. Each server controls only the entities contained within its zone. The Instancing method distributes computational load by creating copies (or: instances) of (sub-)areas. Players entering one copy, can not see players on other copies, as instances are processed independently. The Replication method replicates an area onto multiple servers, that process the area cooperatively, while each server controls only a subset of entities. The three distribution methods may be arbitrarily combined, allowing more players in a fixed-sized area, as computational loads are balanced.

Although Zoning is traditionally an MPPA method, we consider the combination of all three methods a single DPPA method. After all, by combining Zoning with the Replication and Instancing methods, processing power can be allocated over multiple servers dynamically. However, Zoning may also be used dynamically, when, for instance, the size of the game world changes during runtime.

A closely related example is the MMOG Second Life, which uses a dynamic Zoning method to distribute all entities and calculations over several servers [72], while the game world changes constantly. Every server simulates physics, manages entities, runs scripts and manages the terrain for four fixed square regions of the game world, each of about 16 acres. The servers are topologically tiled in a grid, where each neighbor is connected to only its own four nearest neighbors, so that performance is not decreased when the game world becomes bigger.

The Real-Time Framework is developed at the University of Münster and is being used to develop some MMOG projects, such as: Hunter, Bioeclism and a Quake III version that is ported onto RTF [31].

We find that RTFs’ three main methods, Zoning, Instancing, and Replication, work very well for allocating processing power. Scalability is achieved by allocating processing power among a number of servers, by instancing and replicating (sub)areas or zones. Furthermore, developers experience flexibility, by arbitrarily combining three simple methods, according to the developers’ needs. Therefore, we strongly recommend the use of ZIR.

**Microcells** are proposed by De Vleeschauwer et al. [29], in order to cope with the allocation of processing power dynamically. Microcells are divisions of smaller areas of the game world, of any size or shape, that can be dynamically assigned to different servers. A server may own many microcells, but has just one database and one controller. Moving microcells from a loaded server to a much less loaded server, rebalances server loads and thus decreases chances of performance bottlenecks as it limits the variation of the maximum server load. This allows fluctuating player densities, without peaks in use of server processing power.

De Vleeschauwer et al. [29] show that performance tests need to be done, for an optimal configuration. Cells that are too small or too large could decrease efficiency significantly.

The work of De Vleeschauwer et al. has been continued by Van Den Bossche et al. [79] and Ahmed et al. [4]. Van Den Bossche et al. focus mostly on the implementation and evaluation of the actual system, while Ahmed et al. propose methods on how to select which microcell to move from an overloaded server to another server.

We believe that the Microucell method guarantees scalability and flexibility, as microcells, of any size or shape, can be assigned dynamically during runtime. However, different configurations have to be tested extensively, to avoid inefficiency.

**On Demand Service Platform:** The DPPA method described by Shaikh et al. [74, 75] allows the provisioning of servers among multiple games and business applications. As some games might become unpopular, processing power becomes available on some servers, permitting other games or applications to use it. The usage pattern of both online games and business applications exhibit a fairly predictable pattern. Online games experience peak times in the evening and weekends, whilst business application peak during common work hours. Sharing a server pool across games and business applications exploits these patterns, so that little processing power is wasted during different times.

Nae et al. [68] point out that most game operators mainly use MPPA methods to provision tens of thousands of geographically clustered machines, for a single MMOG. However, Nae et al. promise an efficient and dynamic platform, by extending the On Demand Service Platform. For efficiently predicting resource demands, they introduce a fast and accurate neural network.

We find that On Demand Service Platforms promise business opportunities and efficient provisioning of servers among multiple games and business applications. Nae et al. even show that their PPA algorithm performs five to ten times better than platforms used by most game operators. However, extensive development is required, as the On Demand Service Platform is a complex system compared to traditional systems.

### 4.1.2 Client-Side PPA

As explained in Section 3.1, it is preferable to allow players with inexpensive computers to have a fluent game experience. Since increasing numbers of players are involved, there is a need for minimizing the workload that clients may experience.

The remainder of this section is structured as follows. Firstly, we survey a commonly used method, called Interest Management, that is mainly used for bandwidth reduction (See
We find that Voronoi-based Overlay Networks allow extensive scalability, since no traditional servers are required. Furthermore, VON may be easily extended to allow even better and efficient ways of allocating processing power.

**Mediator** is a framework, designed by Fan et al. [33–35], that capitalizes on the potential of P2P networks by distributing functionalities of traditional game servers, so that MMOGs scale better in both communication and computation. The network is self-organizing and intended to be flexible and extensible. Mediator contains four main elements: a reward scheme, distributed resource discovery, load-management and super-peer selection.

Super-peers, or Mediators, may take up one of the following roles: the *Boot Mediator*, who handles the joining and leaving of players; the *Resource Mediator*, who discovers distributed resources and distributes jobs; the *Interest-Management Mediator*, who manages the interests of some players; and the *Zone Mediator*, who selects other super-peers and balances workloads among some super-peers.

Common-peers can contribute their processing power by claiming and managing some game objects. A Mediator’s workload is difficult to trace and measure, because it may vary over time, thus a reward scheme is used. This scheme is based on a Mediator’s hardware and online time. Strong peers are motivated to take Mediator jobs and to stay online for a longer time.

We believe that the Mediator framework promises scalability, as game objects are hosted and processed using players’ machines. We also expect flexibility and extensibility. However, we were unable to find sufficient research as to support these claims.

### 4.2 Bandwidth Reduction

We now discuss a number of techniques used to reduce the amount of network traffic used by games. Furthermore, we map a number of papers per method to illustrate the various uses and implementations of these techniques. We have identified three techniques to be discussed: *Interest Management*, *Dead Reckoning*, and *Delta Encoding*. In Table 2 we map surveyed papers onto our taxonomy. Using these papers, we identify and survey promising solutions on Bandwidth Reduction within the next two subsections.

#### 4.2.1 Interest Management

As discussed in Section 3.2, Interest Management is the process of determining which information is relevant to a player, and which information is not. There are many different methods to determine this relevance of information. Most techniques are based on distances between the player and other players or objects. Other techniques focus on visibility or in-game importance, which are all significant factors when determining the relevance of an object.

Bouilhan et al. [13] compare a number of Interest Management methods, though all are based only on proximity. We have identified two variants, namely proximity based techniques and vision based techniques. We discuss each of these in the following, in turn.

**Proximity based techniques** are based on the simple premise that other players who are spatially close to a player, in
the game world, are the most interesting to that player. Although more advanced techniques exist, it does result in a reduction of the amount of bandwidth used.

Abrams et al. [3] propose a three-tiered Interest Management system in which the game world is divided into regions that can be dynamically resized. This addresses the problem of clumping, referring to the phenomenon that players are rarely evenly distributed over the entire game world. By dynamically resizing the regions, this problem can be solved. It should be noted that the regions must have a minimum size to prevent the creation of many regions as a means for addressing the load balancing problem of clumping. Otherwise, an infinite number of regions could be created in an attempt to solve a load balancing problem.

Hu et al. [52] propose the use of Voronoi diagrams for the partitioning of the game world into regions. Hu et al. provide the following definition of Voronoi diagrams:

> Given a number of points on a 2D plane, a Voronoi diagram partitions the plane into the same number of Voronoi regions, such that each region contains all the points closer to the region's site than to any other site.

In Hu et al.'s approach, if each player is considered a node, a player's area of interest is a certain circle surrounding that node. The player receives updates from all neighboring Voronoi regions. This includes both directly neighboring nodes and nodes whose regions partially lie within the area of interest. To deal with the clumping problem, as described earlier, the area of interest of each node can be adjusted to keep the number of neighboring nodes within certain limits. This is a mechanism to prevent overloading one particular node.

Rather than dividing the game world into regions, Morgan et al. [67] propose making use of a player's aura. An aura is an area surrounding each player of a predetermined size. By determining when players' auras intersect or may intersect, it can be assessed whether a player's update information is relevant. In this way, it is not necessary to divide the game world into regions, which often can be a difficult task, as Morgan et al. [67] point out as well.

Greenhalgh [46] describes a similar partitioning using players' auras for use in the MASSIVE systems. Greenhalgh, however, does identify serious limitations of bandwidth reduction techniques based solely on proximity. The most important of these is an limited reduction in bandwidth. For instance, players that are not visible to a player, but are within a certain distance of the player, are contained within the interest set. This is inefficient, as interaction with these players is unlikely, and therefore their information is not as interesting. A solution to this is the use of vision based techniques, as described in the following.

We find that proximity based techniques to be somewhat efficient when reducing bandwidth use. However, as Greenhalgh [46] indicates, they do have serious limitations. These limitations are mainly due to the fact that proximity alone is not an accurate representation of the set of players a player is interested in. For instance, another player that is far away, but is visible, will be interesting to a player, even though the distance may be considerable. In light of these findings, we would recommend to combine proximity with some other factor to determine a player's interest set.

### Vision based techniques

Use a player's vision as the main factor for determining the relevance of other players' movements or actions. As indicated by Greenhalgh [46], proximity based techniques have certain limitations regarding the amount of bandwidth reduction they can possibly achieve.

Both Fiedler et al. [37] and Funkhouser [40] propose systems in which the interest set of each player is determined by what the player can actually see. There is, however, an important difference between the two approaches.

Funkhouser’s [40] approach is based on a pure Client-Server architecture, in which the server determines whether another player is interesting to a player. In this manner, Funkhouser claims the server is aware of the player’s interest set and as such can distribute this information to other servers and clients if necessary. Furthermore, Funkhouser’s approach assumes the game world is divided over a number of servers. Therefore, the fact that the server controls the interest sets of players smoothens possible handovers between servers when players traverse different regions.

Fiedler et al. [37], place more responsibility with the client itself, rather than with the server. Each player subscribes to updates from players and the environment within the region in which the player is located. Furthermore, each player is to subscribe to neighboring regions for environmental updates, to smoothen region transitions if and when they occur.

Knutsson et al [60] do not propose the creation of interest sets based solely on vision, but take other sensory capabilities into account as well. Players who are, for example, situated around a corner are included in the interest set, as they are likely to be heard by the player. This increases the number of factors taken into account and possibly improves the reliability of interest sets.

Similar to our findings on proximity based techniques, we find that vision based techniques are somewhat limited in their ability to appropriately select interesting players. Another player who, for instance, is situated around a corner may not be in a player’s vision, yet could still be interesting.

<table>
<thead>
<tr>
<th>Authors</th>
<th>IM</th>
<th>DE</th>
<th>DR</th>
<th>PB</th>
<th>VB</th>
<th>DT</th>
<th>OF</th>
<th>FB</th>
<th>ADT</th>
<th>AT</th>
<th>DG</th>
<th>TD</th>
<th>RD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abrams et al.</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bharambe et al. 2006</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bharambe et al. 2008</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cai et al.</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fiedler et al. [37]</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Funkhouser [40]</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Glinka et al. [44]</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greenhalgh [46]</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hu et al. [52]</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Knutsson et al. [60]</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Mapping of papers related to Bandwidth Reduction onto the taxonomy. Abbreviations: **Interest Management (IM), Dead Reckoning (DR), Delta Encoding (DE), Proximity Based (PB), Vision Based (VB), Other Factors (OF), Fixed Thresholds (FT), Adaptive Thresholds (AT), Doppełgänger (DG), Traditional Delta Encoding (TD) and Robust Delta Encoding (RD).**
due to its proximity. Therefore, as with proximity based techniques, we recommend that vision be used in combination with another technique, to improve its reliability.

**Other factors** that may play a role in Interest Management techniques include game-related attributes and elements such as interaction recency. Bharambe et al. [9] propose the use of interest sets based on proximity, aim and interaction recency. Interaction recency is a relevant factor in this case, as players that recently interacted are more likely to be interested in each other. By adding this factor, players that may have been removed from the interest set because they are too far away or cannot be seen are still considered interesting, which they probably are due to their recent interaction.

As indicated earlier in this section, proximity and vision alone are somewhat limited in their effectiveness. However, when combined, as Bharambe et al. [9] propose, we are of the opinion that they form a powerful method of bandwidth reduction. The addition of interaction recency strengthens the effectiveness and accuracy even further, thus improving the results obtained.

### 4.2.2 Dead Reckoning

As we discussed in Section 3.2, Dead Reckoning is a technique that aims to reduce bandwidth use by reducing the number of messages sent. It achieves this by extrapolating known information to determine players’ positions rather than require more information to be sent. Dead Reckoning can, however, be implemented in a number of different manners. The thresholds, as described in Section 3.2 can be fixed or can be adapted dynamically. In this section we discuss these two options along with a variation on Dead Reckoning proposed by Bharambe et al. [9].

**Fixed thresholds** are used in the traditional implementations of Dead Reckoning. Knutsson et al. [60] propose to use Dead Reckoning in this traditional sense as a back-up feature. If an update is missed, for whatever reason, Dead Reckoning is applied with a predetermined threshold value. Furthermore, Knutsson et al. also propose an alternative method of using Dead Reckoning, namely to only send positional updates if the position deviates significantly from the prediction provided by the Dead Reckoning algorithm. This, in essence, is the use of thresholds with Dead Reckoning.

Many other papers [43, 65] also use Dead Reckoning with traditional thresholds. As each of these implements a similar system to the system of Knutsson et al. [60], we do not believe it is necessary to reiterate or further discuss implementation details in this survey.

We find that the use of fixed thresholds should be avoided. This is mainly due to the fact that fixed thresholds cannot adapt to changing situations. Moreover, it takes a considerable amount of time, through trial-and-error testing, to find a suitable threshold. Thresholds also vary depending on the game, thus making it necessary to repeat this testing process many times.

**Adaptive thresholds** were proposed in an effort to improve the results given by Dead Reckoning. Cai et al. [15] propose an auto-adaptive algorithm for the determination of the threshold to be used. This algorithm is based on the linear distance between two players. When the two players are spatially far away from each other, a high threshold can be used, as a deviation from the true position would not lead to a collision. However, as the distance between the two players is reduced, the threshold needs to be lowered as large deviations would cause collisions. When two players are very close, the threshold is set to zero, as no deviations are permitted, because any deviation will lead to a collision.

Lee et al. [61] elaborate further on this algorithm, and include a section regarding the automatic selection of the type of threshold to be used based on player movement. There are various Dead Reckoning formulas that can be applied, which are outside the scope of this article, that can also be varied. Depending on the type of motion, Lee et al. propose adapting the threshold and the formula used in the application of Dead Reckoning.

In comparison to the use of fixed thresholds, we believe the use of adaptive thresholds should be encouraged. By providing the capability for adaptation to varying circumstances, we find Dead Reckoning to deliver better results. This is mainly caused by the fact that dynamic thresholds offer the possibility to reduce the number of messages even further, only requiring updates when absolutely necessary.

**Doppelgängers** constitute a variant of traditional Dead Reckoning algorithms and have thus far only been proposed by Bharambe et al. [9]. Although this proposal is substantially different from Dead Reckoning, one can certainly identify certain similarities and as such we believe it is appropriate to discuss this system as a variant on Dead Reckoning.

Bharambe et al. suggest modeling players outside of a player’s interest set by using *bots* that make use of the in-game artificial intelligence (AI). However, rather than receiving updates regarding the ‘real’ player’s position and movement, Doppelgängers receive guidance updates. These guidance updates contain information regarding the predicted behavior for the period between the current guidance update and the next. Subsequently, Doppelgängers use this guidance to modify their behavior to simulate the player’s behavior in a more realistic manner.

Doppelgängers are a variation on Dead Reckoning that, in our view, improves the original Dead Reckoning algorithm considerably. By offering guidance, rather than raw information such as exact position and speed, the estimates of players’ positions will be improved, thus improving game experience. However, we do recognize that more work is necessary to explore the possibilities of Doppelgängers.

### 4.2.3 Delta Encoding

As we described in Section 3.2, Delta Encoding is based on the premise that if information has not changed since the previous transmission, there is no reason to retransmit the same information. However, if messages are lost during transmissions, Delta Encoding may result in unreliable results. In this section we discuss the traditional implementation of Delta Encoding, along with some methods that provide a more robust experience.
Traditional Delta Encoding is applied in many systems, as it is a simple, yet very efficient, manner of reducing network traffic. Glinka et al. [44] propose the use of Delta Encoding for cases in which parts of the game state are mirrored, either in a Peer-to-Peer fashion or in a more traditional Client-Server architecture. They also note that Delta Encoding is particularly useful for game processing, as only a small number of attributes actually change with each system clock tick.

Bharambe et al. [11] also make use of traditional Delta Encoding. As indicated in the paper, the use of Delta Encoding reduces the size of each message considerably, whilst providing the same end result.

Robustness is an important factor when considering the use of Delta Encoding, which is not considered by Bharambe et al. [11] or Glinka et al. [44]. The main challenge with regards to Delta Encoding is what happens when one or more messages are lost in transmission. In that case it is unclear how the system should deal with missing information.

Bharambe et al. [9] do provide a solution to this challenge by basing Delta Encoding on the previous two messages, rather than only the last update. This ensures the correctness of the messages, even if a message is lost or damaged, for whatever reason. Only when two or more messages are lost, a non-Delta encoded transmission is necessary to provide correct and up-to-date information.

Bharambe et al. [10] also proposed another method of ensuring consistent performance with Delta Encoding. Although this paper focuses on game spectating, the discussion on Delta Encoding is relevant for gameplay too. Bharambe et al. suggest a distributed Delta Encoding scheme, in which the differences are computed between individual nodes. Thus, a parent node computes the difference with respect to the most recent frame that it has in common with a child node. If a frame is lost, however, Bharambe et al. propose using an optimistic Delta Encoding scheme. This means that the system uses Delta Encoding with regards to only the previous message. If that message is lost, for whatever reason, the child sends a list of last received updates, so the parent can compute the differences with respect to one of those updates.

As we discussed, we believe that robustness features are an essential aspect of any implementation of Delta Encoding. Without these features, Delta Encoding is vulnerable to message loss, which could cause major inconsistencies within a game. Despite these drawbacks, we are of the opinion that Delta Encoding provides an excellent way of achieving bandwidth reduction without requiring major adjustments to the game architecture itself.

### 4.3 Latency minimization

In this section we discuss a number of latency compensation techniques: Forwarding Pool, Latency Equalization, Geographical Server positioning and Voronoi-based Overlay Network. In addition we also map several surveyed papers to these methods. See Table 3.

#### 4.3.1 Forwarding Pool

As mentioned in Section 3.3, the purpose of the Forwarding Pool is to support clients, with insufficient bandwidth, in sending update messages, using other clients with a more than sufficient bandwidth capacity. In the following section we discuss two approaches for forwarding messages: the Donnybrook and the Forwarding model mechanism.

Donnybrook is a system proposed by Bharambe et al, which enables large scale, high speed, Peer-to-Peer games [9]. Donnybrook uses a Forwarding Pool, a set of machines with good connectivity and sufficient capacity to meet the overall forwarding needs of all sources, to forward messages. Based on its latency and upload speed, a machine can be added to a Forwarding Pool. When a machine requires more capacity than available, to forward its update message, it will randomly choose a member of the Forwarding Pool that is capable to supply the required capacity. Depending on each machine’s available capacity, it may leave or enter the Forwarding Pool. Each member of the forwarding pool needs to advertise their available capacity periodically to keep all sources up to date. This information is included within guidance messages. See the paragraph Doppegleinters in Section 4.2.2. Based on these guidance messages, a source can determine whether a machine is still present in the Forwarding Pool, to adjust local threshold value. To avoid collisions and additional delay, pool members only advertise half of their capacity. By coordinating the allocation of forwarders, Donnybrook prevents multiple sources from choosing the same forwarder simultaneously.

Unfortunately, this approach has three disadvantages. First, it cannot handle large amount of update messages due the unpredictable spikes in bandwidth. Second, the system depends on failure of the coordination point. Third, the system needs more capacity to reduce collision likelihood. This approach, Forwarding Pool, compensates the latency by using available capacity of other machines for synchronization of guidance messages.
Another approach is the **Forwarding Model** proposed by Chen et al. [17]. The proposed forwarding model enables relaying messages from neighbor nodes and restricting each node's direct connections with only its nearest neighbors. This approach uses Voronoi-based Overlay Network (VON) to maintain a highly overlay topology consistency. Each node efficiently utilizes its bandwidth resources by using message forwarding and data compressions, thus the latency can also be reduced, because the amount of messages is limited. For more about VON we refer to Section 4.3.4 and 4.1.2.

### 4.3.2 Latency equalisation

In the following section we discuss another technique to reduce the overall latency, which is Latency Equalization. As mentioned in Section 3.3 Latency Equalization is being used to improve the user experience. There are several approaches to equalize the latency, one of these approach is the **Latency EQualization Service** proposed by Yu et al. [80]. A second approach is **buffering packets** at the edge router. A third approach is **source routing**. A fourth approach is to set up **Multiprotocol Label Switching (MPLS)** paths. In this section we discuss the above mentioned approaches.

**LEQ service:** Yu et al. [80] propose a latency EQualization (LEQ) service, which equalizes the perceived latency for all clients participating in an interactive network application. By providing equalized-latency paths between the clients and servers, LEQ services can assign a different route to the clients to minimize their delay difference. The equalized latency paths are determined by making use of a hub selection algorithm, which deals with the assignment of hubs, i.e. traditional router that are either in the core network or at the edge, to the client edge routers. Based on Akamai's SureRoute service [5] and Savage's research [73] Accordin to Savage latency can be reduced by using overlay paths. This means that LEQ routing can also minimize the delay difference without compromising the end-to-end delay.

**Buffering packets** at the edge routers is an approach that uses the network to equalize delays [55]. This approach requires large buffers for each interactive application, which made the router expensive and power inefficient. Moreover, edge routers need to handle with packet delay requirements and determine how long packets will be buffered; and all this by using a complex packet scheduling mechanisms. This approach is less efficient if we compare it with the LEQ services, because the routing infrastructure needs to be modified after the delay difference has been reduced.

**Source routing** means that the client (or client edge router) itself may choose the path of a packet [71]. This approach is hard to implement, because each client in the network should be aware of the network topology and communicate with other clients to ensure that the chosen path contains the minimized the delay differences.

**MPLS Paths** can be used to equalize latency between clients and server edge routers [19]. This approach uses **label edge mobility agent (LEMA)** and **label switched path (LSP)** to determine which path should be chosen to reduce the latency. The principle of this approach is almost the same as the LEQ service, but this approach needs to configure more paths than the LEQ service.

In comparison with the other approaches mentioned above, we find that the LEQ service approach is most efficient. So we recommend this approach to be used in combination with other techniques to minimalize the latency.

### 4.3.3 Geographically server positioning

In the following section we continue our survey with discussing several geographical server positioning methods. We take a look at the approaches proposed by Gupta et al. [48], Bescow et al. [8] and Jiang et al. [57].

**Geographic characterization of game servers** Feng et al. [36] analyse the distribution of servers for several popular online games, such as **Counter-Strike** [22] and **Battlefield** [32]. Most online games use a centralized registry server that registers individual game servers. For each of the above mentioned game Feng et al. generates a ranked list of game servers based on the ping time, that is the round-trip latency from the client to each server in the list of each server. Then they use a commercial geographic mapping tool to map the IP addresses into geographic locations. But many online games are not enjoyable and playful when only a limited number of players are playing. For example playing with a few players in a large map can be quite boring, since it takes a significant amount of time before players find each other to shoot.

As result of their analysis, Counter-Strike servers appear to be primarily distributed across North America, Europe and Asia. The same goes for Battlefield 1942. But Unreal Tournament 2003 has more servers in North America than the other two games. In addition it also shows that for Counter-Strike and Battlefield 1942, most of the servers reside in North America and Europe and are evenly split between the two with approximately 40-45% of them residing in each continent.

**Geographically distributed servers** Bescow et al. [8] propose the use of **core selection**, distributed name server and **migration** for lowering the overall latency of the interacting players. First they select a server, then they assign a name to the server, and then they use their migration functionality to move the active region to its new location.

Previous work [47] shows that the migration middleware has potential to reduce latency. The time of the day plays an important role in this approach. Chambers et al. [16] analyze how server selection can be optimized for a single client. While Claypool [20] analyses how server selection can be optimized for a group of players, thus multiple clients.

**Core selection heuristics** are used to select the core. Karaman and Hassanein [58] proposed two heuristics: finding a server that is closest to the topological center of the **global graph** (Topology Center), and finding a proxy that is closest to the group center of the **group graph** (Group Center).

Another approach of server positioning is being demon-
strated by Gupta et al. [48]. They propose SEMMO, a consistency server for MMOGs. SEMMO consists of machines (computational farms), which are geographically dispersed all over the world and which are running on a high bandwidth low latency network. These computational farms are geographically located in a way that minimizes the average network latency for clients. The main advantage of this approach is that the virtual world is being partitioned which reduced the network traffic. Using a strategic geographical positioning of the computational farms the perceived network latency can be reduced.

Ta et al. [12] and Lui et al. [62] developed heuristics for distributed server system to improve performance. The main focus of those heuristics, is to assign players to servers, so that the communication costs between the players and the servers (e.g., latency) are low.

It is difficult to survey which method is the best, because we first discussed some approach to characterise game servers, then we talked approaches how to distribute servers geographically. But SEMMO [48] is in our perspective the most efficient approach for distributing servers.

4.3.4 Voronoi-based Overlay Network

Voronoi diagrams can be used to solve the neighbor discovery problem in a fully-distributed, bandwidth efficient, and low-latency manner. In the following section we are going to discuss the Voronoi overlay Network (VON).

To minimize latency, nodes (servers) should connect directly to Area-Of-Interest (see Section 3.2) neighbors [52]. Another approach to achieve low latency on data updates, is proposed by Zoned Federation [54], who inserts zonelayer between a game and the Distributed Hash Tables (DHT). Then they limit the use of DHT to the backup data storage. With other words Zoned Federation uses (DHT) for topology connectivity to reduce latency.

Solipsis [59] maintains direct connections among neighbours, so latency is minimized. To ensure global connectivity, each node needs to be inside the polygon formed by its outmost neighbors.

4.4 Game State Synchronization

In this section, we discuss various techniques for game state synchronization. There are many techniques to manage game states, we elaborate the following techniques, Event Based Synchronization and Voronoi Based Synchronization. In Table 4 we map surveyed papers onto our taxonomy.

4.4.1 Event Based Synchronization

Firstly, we survey the Time Warp Synchronization algorithm, which is a well-known optimistic state synchronization algorithm [39]. Secondly we discuss a variant on the Time Warp Synchronization, the Trailing State Synchronization. Thirdly we discuss a variant on the Time Bucket algorithm [42], the Breathing Bucket Synchronization.

Time Warp Synchronization is event based as explained in section 3.4, which means every server processes the game events itself. Every event must be processed in the same order, otherwise state inconsistencies could be possible.

Every event has a timestamp, which are used to check if the events are processed in the right order. An event causing an inconsistency, is called a strangler [78], which can happen when the event is processed too early or too late. To undo this fault, a rollback is necessary. An event that has to be rolled back, could have changed the state of the game or it might have processed new events, which are called messages. State rollback is achieved by restoring an old saved state. New events, which are processed by the event, have to be undone by sending a message. This is called anti-messages. When an anti-message is received, the new event will not be processed. In the case the new event is already processed, another rollback necessary. This rollback can also send out an anti-message. Recursively repeating these rollbacks can cause enormous computations, which can lead to processors overloads and memory issues.

Two other methods of rolling back states are, Lazy Cancellation [39] and Rollback Relaxation [78]. The method described as above uses aggressive anti-messaging. When a rollback is executed, every anti-message is directly sent. Lazy Cancellation differs in sending anti-messages. It does not automatically process anti-messages. First, the strangler event is processed. Then a check is done if this event generates the same messages as the rolled back events. If there are similar messages, there is no need for an anti-message for the same messages. Depending on the system used, Lazy Cancellation can be faster or slower than the original Time Warp Synchronization. Because Lazy Cancellation does not send anti-messages directly, the strangler events can produce more easily more stranger events. This can lead to recursively repeating rollbacks. On the other hand, Lazy Cancellation can be faster if the messages processed from a stranger event are the same as the messages processed in a rollback. In that case, there is no need for an anti-message.

The second method, Rollback Relaxation, differs in the rollback method. Normally every event that is processed before the strangler event and should be processed after the strangler event is rolled back. With Rollback Relaxation, not every event is automatically rolled back. For instance, an event that has nothing to do with the stranger event, does not need a rollback, because the strangler event has no influence on the event. This can save many rollbacks and it decreases the chance recursively repeating rollbacks.

We find that the Time Warp Synchronization and some
methods based on it can easily be used for game state synchronization. However, it is not the best method to use, there is a high chance of recursively repeating rollbacks, which can lead to processors overloads and memory issues.

Trailing State Synchronization Cronin et al. [27] propose an efficient technique for mirrored server architectures called Trailing state synchronization (TSS). The technique is both Server-Client and Peer-to-Peer based. Each client connects to the best server with the lowest latency. To connect these servers, they need to update their game states with each other, this could be done in multiple ways. TSS is one of these techniques. TSS is an event-based state management technique. Similar to Time warp, TSS is also an optimistic algorithm. This means that if an event is received, the event is immediately processed. When a second event is received, which had to be executed before the other, TSS uses rollbacks to make up for the resulting inconsistency in the game state. Furthermore, TSS does not suffer from processor overloads and memory issues, like Time Warp. Instead of storing all the events [64], there are multiple delayed copies of the game state, each with a different delay. These so-called states execute every event that is received after a certain delay. The leading state is the state with the shortest delay, this is the state which is used in the game. The other states are trailing states, these states detect inconsistencies.

To let TSS perform optimally, the number of trailing states and the delay for each state is important. If few states are used, the delay between the states must be large. This has the consequence that inconsistencies are detected after a longer period of time, which leads to larger and more noticeable rollbacks. On the other hand, if many states are used, the advantage of memory saving will be eliminated.

We find that the Trailing State is a good alternative for the Time Warp Synchronization. It can save a lot of memory issues. However, when there is not an optimal number of trailing states, the synchronization can lead to noticeable rollbacks. An optimal number of trailing states should be tested for each game.

Breathing Bucket Synchronization The Breathing Bucket Synchronization is very similar to Time Bucket Synchronization. The Time Bucket Synchronization algorithm is design for the game MiMaze [42]. Time is divided into periods of time T, these periods are called buckets. Every event is put in the bucket, before it is processed. Therefore, events are processed after a delay. Because some event messages might arrive at incorrect times, this increases the chance of executing events in the right order. Whenever the period T is not long enough for events that arrive late, there is no inconsistency detected. This is not desirable for games where game state consistency is very important.

Steinman [77] tries to solve the problem of game inconsistencies with rollbacks like in the Time Warp algorithm. He proposes the Breathing Time Bucket, which also uses periods of time. The only difference is that it does not use fixed periods, but variable periods. The period is determined by the event horizon. An event horizon is determined by the timestamp of the first next event. There is a local and a global event horizon, the global horizon is the smallest of all the local event horizons. If the true horizon is known, every event with less or equal to this time are processed. If an event is processed that was beyond the true horizon, a rollback is needed for consistency. Rollback involves discarding the messages generated by the event and restoring the state to the previous state. The messages have not yet been released because the time stamp of the event is greater than the true horizon. Because there are no bad messages released, there is no need for anti-messages. This also means there is no chance getting into a recursive computations.

We believe the Breathing Bucket Algorithm is a good alternative to the Time Warp algorithm. It also uses a rollback system for inconsistencies. The advantage to Time Warp is that there is no chance of getting into recursive computations.

4.4.2 Voronoi Based Synchronization Hu et al. [51] propose a Peer-to-Peer based game state synchronization technique, called Voronoi State Management (VSM). In VSM, the virtual environment is partitioned into Voronoi cells. Within a cell, the user receives object updates in a Client-Server based model. As we already addressed in section 3.4 user can have two different roles, peer or arbitrator. An arbitrator is like a server, it has the authority over a Voronoi cell, and it manages the game state of the cells it controls. A peer is like a client, it generates events and simply reads the game state from the arbitrator. In a cell, every user is a peer. Arbitrators on the other hand are only capable users, or servers if needed. An aggregator is used as a special arbitrator, if user nodes become crowded, a more with high capacities user is required. If needed, servers may be added to the system, which can also act as arbitrators. An aggregator is then acts as a special arbitrator with a fixed location and a circular sphere of control. In case a peer approaches multiple aggregators, it would join the closest one.

VSM is update based, peers in a cell receive update from its arbitrator. Peers send events to the arbitrator and the arbitrator processes them according to the game logic. Interesting objects for other cells are updated, by a state update between arbitrators. Due user movements bound, Voronoi cell boundaries can change. For consistence update changes, ownership about object events is required for arbitrators. Events sent from an object to arbitrators without ownership of that object are forwarded to the right arbitrator.

VSM has two types of updates, basic update and transactional update. A basic update is sent whenever a user walks or picks up a weapon(an event that affects one or more objects), in that case an event is sent to the arbitrator. The arbitrator processes the event and updates it to all the interested other arbitrators. The transactional update is whenever for instance two users trade a weapon. This will update the state of the object at the same time. When such event is sent, the affected objects are first locked, before both objects are updated. This way another event cannot update the object between the transactions. When both arbitrators acknowledge a confirmation that both objects are updated, the objects are unlocked.

Although, Voronoi State Management can be improved on some features, we believe that Voronoi State Management is a promising technique to improve the scalability of MMOGs.
5. RECOMMENDATIONS
Due to increasing popularity MMOGs, the scalability problems that come with them, become even more important to address. Thus, we think it is essential to continue current research in all four areas we discussed in this article. We do believe certain aspects can be improved more than others can.

As discussed in Section 4.2.2, the introduction of Dop pelgängers [9] has yielded promising results, and therefore merits further research into the use of artificial intelligents to reduce bandwidth use and make this an even more useful technique.

Furthermore, more benchmark testing is required to provide a full overview of the effectiveness of the discussed techniques for various game types and environments. Each paper we have surveyed has either developed a custom system or modified an existing system to such an extent that it is unclear what the real-world gains are of the proposed combined techniques. We thus found it quite difficult to provide an objective comparison of these techniques as each game has specific challenges.

Finally, we think that the research in scalability of games might open doors for new game genres. Methods may be benefitted from, by thinking of yet unknown gameplay elements that could potentially lead to new genres and ideas.

6. CONCLUSION
The demand for MMOG systems that support millions of players has been increased, and so has the need for solutions to their scalability problems. In this work, we have provided a survey of solutions for scalability issues of MMOGs. Processing Power Allocation, Bandwidth Reduction, Latency Minimalization, and Game State Synchronization are the four primary aspects on which this survey is based. First, we have created a taxonomy of the scalability problem of MMOGs with the four aspects mentioned earlier. Second, we have surveyed several solutions for each of these aspects and mapped it on our taxonomy. We have found that many promising solutions are available, but insufficient research had been done on the combination of them. Therefore, in search of the best performing system, we recommend to continue research in all four areas we discussed in this article and combine the promising solutions into a couple of different systems, to test each system’s performance.

7. ACKNOWLEDGEMENTS
We would like to thank our supervisor Dr. Alexandru Io-sup for his constructive comments and valuable suggestions which help to improve the quality and presentation of this survey. Also we want to thank Delft University of Technology for giving us the opportunity to write this article.

8. REFERENCES


[33] FAN, L. Solving key design issues for massively multiplayer online games on peer-to-peer architectures, 2009.


NetGames ’04, ACM, pp. 106–110.


