Increasing the Supported Number of Participants in Distributed Virtual Environments

Lutz Behnke

Thesis submitted in partial fulfilment of the requirements of the University of the West of Scotland for the award of Doctor of Philosophy

March 21, 2020

Dedication

To Anja, Svea and Lena

wife and daughters, who have supported, cajoled, chastised and loved me through all this time.

Contents

Abstract

Distributed Virtual Environments (DVEs) allow multiple users to interact in a synthetic environment. Within the DVE, users are represented by avatars. Depending on the type of DVE, a wide range of interactions of the avatars with each other and the environment is possible. The subset implemented in a specific DVE is largely dependent on the type of application or service that is provided by the DVE.

Massive Multiplayer Online-Games (MMOs) are not only a multi-billion dollar industry, but have also been the subject of research for quite some time. Historically, the majority of this research has focused on certain aspects of an overall DVE architecture.

But for DVE technologies to act as the foundation of this industry they have to display certain characteristics to be viable. These include high availability, requiring tolerance of faults in individual components. They also provide stable enforcement of game rules in order to limit misuse of the service by antagonistic users. Users flocking to locations of common interest also generate a specific pattern of high load that needs to be supported sufficiently.

This thesis describes a novel approach to completely decouple the location of processing and state retention from each other and the spatial position within the DVE. This approach not only allows the amount of computational resources to be scaled to a degree previously not supported, but it also does this while maintaining the above characteristics and also adding support for adaptively handling limited bandwidth between the DVE and client application.

The following particular problems are addressed by my contributions:

Scalability and Resilience: To escape the tyranny of the n^2 growth in communication overhead when trying to identify the set of observers for each change to the overall DVE state, this thesis

presents a novel approach to identify and strictly separate all axes of scalability in a DVE (e.g. client/server bandwidth, processing resources, inter-server bandwidth, I/O bandwidth, persistence processing overhead, persistent storage capacity), in order to allow for scaling each in isolation and providing overcapacity, which allows for fault tolerance of individual components without interrupting the availability of the overall service.

At the same time my approach eliminates the need for user-visible partitions completely and supports the creation of arbitrarily large fields of view without the DVE grinding to a halt from the overload of a single processing unit or the consumption of all of a server nodes bandwidth by a single client.

The result is support for a higher number of concurrent users in a more resilient way.

State Update Overload: Scaling the number of changing objects in a DVE, be it avatars or animated elements of the environment, could lead to the escalation of required bandwidth not only for providing these updates to the client application, but also in rendering these changes by that application, in such a way that it would not even be perceivable by a human user.

In this thesis I propose an aggregation framework that can dynamically adapt to the capabilities of transmitting, rendering and consuming the observable state updates of a DVE. This not only allows to save network bandwidth, but also offers the option to filter and convert the way the updates are presented for a specific client or user.

Player Flocking and similar Overload: As humans tend to concentrate in places of interest, the number of avatars per unit area can vary widely within a DVE, often by multiple orders of magnitude. This poses the problem of unused resources which will increase the cost of a DVE cluster for a given avatar population without improving its characteristics.

But worse, it may lead to a local overload of the DVE infrastructure, seriously decreasing the quality of the user experience. In the most extreme case, it will cause the DVE server to fail, evicting one or more users and, worst of all, cause loss DVE state.

This thesis introduces an architecture that allows the same locality within the DVE to be processed by multiple game servers, while still providing a single DVE instance to be maintained.

This avoids the common practice of sharding or instancing parts of the DVE in-world area. The latter would partition the user base, inhibiting interaction between these partitions.

Declaration

The research presented in this thesis was carried out by the undersigned. No part of the research has been submitted in support of an application for another degree or qualification at this or another university.

March 21, 2020
Date

Signature

Prior Publications

Parts of this thesis have been published previously in the following articles and conference proceedings:

Behnke, L., Grecos, C. and Luck, K. V. (2014), QuP: Graceful Degradation in State Propagation for DVEs, *in* 'Proceedings of International Workshop on Massively Multiuser Virtual Environments', ACM Press, Singapore.
URL: `http://dl.acm.org/citation.cfm?id=2577389`

Behnke, L., Wang, Q., Grecos, C. and von Luck, K. (2015), Budget based dynamic state update aggregation, *in* 'Proceedings of the 7th ACM International Workshop on Massively Multiuser Virtual Environments - MMVE '15', Portland, Oregon, pp. 25–26.
URL: `http://dl.acm.org/citation.cfm?doid=2723695.2723696`

Behnke, L., Allers, S., Wang, Q., Grecos, C. and von Luck, K. (2016), Avatar Density Based Client Assignment, *in* G. Wallner, S. Kriglstein, H. Hlavacs, R. Malaka, A. Lugmayr and H.-S. Yang, eds, '15th IFIP TC 14 International Conference', Vol. 9926 of *Lecture Notes in Computer Science*, Springer International Publishing, Vienna, pp. 137–148.
URL: `http://link.springer.com/10.1007/978-3-319-46100-7`

List of Figures

List of Tables

Chapter 1

Introduction

Distributed Virtual Environments (DVEs) are a type of networked software systems that allow human and non-human (i.e. computer controlled) users to connect to a virtual environment. The DVE usually supports the interaction of the users with each other as well as the environment in which are they placed. Users within the DVE are represented by avatars. Most often they represent the position of the users viewpoint. The avatars are the focal point of the interaction of the user with other users and the with the environment. What forms or methods these interactions can take vary widely between types of the DVEs and their intended use, be it for entertainment, training or even therapeutic uses. In Bartle (2004*b*) a number of uses-cases are discussed, as well as specific implementation that target each.

DVEs have been the subject of intense research since close to 20 years, with the roots of virtual environments reaching back even further to the early seventies of the last century. In the form of online games, DVEs are also a multi-billion dollar industry that has established some standards for operating large DVEs as a commercial service (for examples see section 2.10.3).

In the majority of DVEs developed between 2005-2015, the presentation of the content is done by client software that is separate to the actual DVE software system. The interaction between human users and the DVE may take a number of forms. From written text and typed commands at one end of the spectrum (Bartle, 2004*c*), to fully immersive virtual realities such as CAVE2 [1] at

[1] Introductory Video: `https://www.youtube.com/watch?v=yf0sllpZx3w`

the other end.

1.1 Research Objectives

The research presented in this thesis takes a holistic view of a DVE and considers it as a pipeline of components between the client software used by one user and the client software of another user and all components in between. Section 2.7 details components that make up an abstract DVE and discusses the resources each component requires. The aim is to increase the number of concurrent participants in an online game or similar virtual environment, while maintaining several properties:

- Provide interactive game speeds and latencies on state updates that are acceptable to players. The exact value for the reply time from a user input and the presentation of results is highly dependent on the type of application. The type of network connection, especially the WAN connection deployed by the user, has a large influence on this value in a real world scenario. Pertaining research is discussed in section 2.3.3.

  In general, minimising the amount of data sent to the computer of the end user will improve this value. This thesis covers this topic in chapter 5 in detail. Reducing or even controlling the latency of communication via an internet WAN connection is beyond the scope of this thesis. Therefore it is simply ignored, except by aiming to lower the bandwidth requirements of the connection between the DVE and the users computer.

- Make the game usable for consumers over the open internet, on consumer grade game hardware and consumer grade internet connections.

- Support the high avatar density of particularly favored locations in virtual environment with minimal service degradation. Examples include rock concerts or the places like the bank or auction house in most online games like the Elder Scrolls Online or World of Warcraft.

- If the proposed system requires a large number of hardware components, it must support fault tolerance and include self-healing capabilities to ensure that the failure of a component only results in minimal service degradation.

1.2 Thesis Contributions

My thesis is that *by fully decoupling spatial composition of elements in a DVE from the place of processing state change events, as well as the place or organisation of storage for state, a higher local avatar density can be achieved.*

It is my assumption that this will allow the creation of DVEs that are capable of supporting a higher number of concurrent participants than by other systems. I also assume that the decoupling will allow for better handling of component faults, as processing can be reassigned to other computing resources quickly and only have an effect on a subset of the current participants.

I show how to achieve the research goals through the following:

- Separating the partitioning of state data from the spatial partitioning of the DVE as well as the assignment of the processing resources required. This allows the scaling of each aspect independently.
- More dynamic interest management that allows the scope of the selected state changes to be optimised. It does not require static partitioning of the DVE, thus allowing for virtual worlds that can actually change instead of only allowing a few avatars to move within them.
- A mechanism for grouping client connections to re-use state data through pre-fetching. Due to the previously mentioned separation of concerns, multiple processing units can handle the same spatial location within the DVE. This enabled much high levels of local avatar density to be supported. This also allows a re-assignment to enable load shedding to protect agains server overload.
- Better resilience to component failure, by deploying and improving on existing failure-tolerant designs. The state data is persisted redundantly and the failure of a processing node can be quickly recovered from by re-assigning clients to existing processing nodes, which may have a large percentage of relevant state data already fetched.
- Handling of larger amounts of state data that are are identified by the interest management for transmission to the client. An aggregation system allows the dynamic adaption to changes in communication bandwidth, number of objects identified or amount of changes to these

objects. It can dynamically reduce the fidelity of the data transmitted to avoid overloading communication channels, be it between the DVE and the client application or the sensory capabilities of the user.

- Better resilience to prohibited state manipulation (cheating) through the design of the DVE: Rules can be enforced by the server due to the greater capacity for scaling the number of processing units available.

1.3 Thesis Outline

The thesis is organised into chapters as follows:

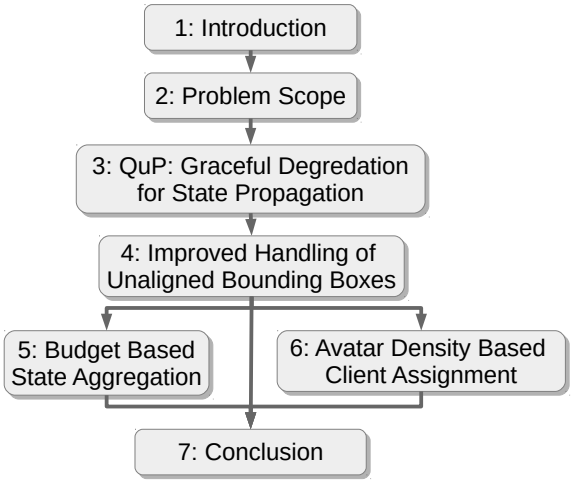


Figure 1.1: Chapter Structure

Introduction a starting point for the general context of work, as well as the observations and assumptions that led to stating the thesis.

Background and Related Research attempts to provide a general understanding of the structure of DVEs and the requirements placed on them. Based on prior publications, it identifies capabilities to characterize a DVE, and parameters that can be used to measure the performance of a given DVE and their impact in different types of applications that can be deployed

on various types of DVEs.

The chapter identifies some baseline values for the quantitative requirements placed on a DVE and the origins of these requirements. In addition it describes a set of optional additional features, that can extend the breadth of applications placed on it. The following chapter will focus on a range of proposed solutions to provide each of these features, while this one will focus on the source of the requirement.

The chapter introduces an abstract model for a DVE that serves as a reference for discussion on the features and characteristics of a potential service that is built on the methods described in this thesis. It is also intended to aid the reader in placing the individual research efforts into a larger context of a fully developed software system.

The chapter also traces a short history of the research of virtual worlds and DVEs to put the current work into context. The software experiments undertaken for this thesis utilise a specific architecture of DVEs (a distributed authoritative server with a strict client/server separation, see section 2.6.1) that was decried as outdated when I started research for my thesis. Recent publications give this architecture a revival, as well document a number of weaknesses of the other main school of architecture design (peer-to-peer, see section 2.6.3) that led to my decision to use a Client/Server design instead.

QuP: Graceful Degradation for State Propagation describes the method to manage the state of the DVE, in order to increase the number of supported concurrent participants beyond those documented in other publications. The state is maintained even in the face of peak avatar density as well as hardware and software component failure.

This work was published in Behnke et al. (2014) and has been presented at MMVE'2014 in Singapore.

Improved Handling of Unaligned Bounding Boxes takes a more in-depth look at some of the problems that existed in the solution of the initial QuP proposal. While the original approach is retained, complementary mechanisms are added to improve performance in the general case and to considerably reduce the impact of worst case scenarios.

Budget Based State Aggregation details a method to aggregate the state update messages that are sent to connected clients. This allows the dynamic adaption of the required bandwidth to the network connection between the client and the DVE. It also can adapt to the processing resources of the client. The goal is to strike a balance between reduction of transmitted details due to technical requirements and degradation of the user immersion.

My work only covers the technical capabilities to adapt the update message stream, not the psychological studies needed to fine tune this to the perception of human players.

This element of my research was published in Behnke et al. (2015) and has been presented at MMVE'15 in Portland, OR.

Avatar Density Based Client Assignment A wide range of research has been undertaken over the last decades on the problem of dividing the workload between the DVE parts of a cluster. Connecting clients to the most appropriate server of this cluster is as important. Clients will place widely varying, dynamically changing requirements on processing, storage and network bandwidth resource on the MMOG.

In this chapter I describe a mechanism for assigning clients to servers in a MMOG as part of a load balancing effort. It allows the optimisation of resource utilisation while being able to handle overload situations in the face of high avatar density and adapt to change over time.

This work was published in Behnke et al. (2016) and presented at ICEC'16 in Vienna.

Conclusion a summary of the work and a discussion of some questions that have been introduced by this thesis.

Chapters 3, 4, 5, 6 form a complete circle of processing and communication in a DVE: Starting with clients connecting to one of the numerous external endpoints of the system. Then continuously managing the large overall state and the high bandwidth of change events to this state. Finally, the dissemination of the state change messages to the connected clients, potentially requiring very high bandwidth which exceeds the capability of the connection (often a consumer grade WAN or mobile link). This circle is purposely started with the management of the state, as it is traditionally the central part to the design of a DVE and drives the characteristics of the application that is deployed

on the DVE.

The thesis does not cover the design of applications or games that would utilise the described platform. The literature review covers existing applications mainly to identify the requirements and features that would inform the type and scope of the supported DVE applications (see section 2.3).

Neither does this work cover the graphical rendering of the modeled environment. Early DVE research like Hagsand et al. (1997) were constructed to maintain and distribute the state of a 3D model. Later research usually separates the abstract state of the VE from any graphical representation. With the advent of DVEs using complex content directly provided by the users, like SecondLife (Inc. (n.d.)), this interest has resumed. Distributed Scene Graphs (DSGs) are one such research avenue that is currently investigated in parallel with the abstract model DVE that support/complement them (e.g. Valadares et al. (2015)) to support computational expensive object interactions like realistic physics simulation or modifiable objects with collision detection.

1.4 Out of Scope

Designing, implementing, scaling and continuously operating a DVE as a commercial service offering is a varied and far ranging undertaking that takes input from a enormous range of research efforts. In order to limit this to a manageable scope I have chosen to focus on the back-end infrastructure of the game server and the mechanism of state persistence. This results in the following subjects being beyond the scope of this thesis:

Game / application design Any discussion of the actual applications that are to utilise the described infrastructure, beyond the decision to focus on online role-playing games. This decisions stems from the context in which I started the research for this thesis.

The aim of the research is to provide the widest range of capabilities to enable the building of better applications, by providing additional tools to the designer. An overview of the required capabilities can be found in section 2.1.

Also, Bartle (2004*a*), Bartle (2015*a*) and Bartle (2015*b*) are priceless to help understand the requirements designers place on the infrastructure

Game client application Any serious game based on a DVE would require the implementation of a client application for that game and the creation of the visual and audio assets that the users would expect to support them in their immersion in the game.

This thesis treats the client application simply as a black box that generates and consumes a stream of events, which must be processed and generated respectively within in a strict timeframe.

Communication latencies in the public Internet The communication latencies observable by the client application (and thus the user) are the sum of a traversal of the public Internet, processing by the game server and a return trip via Wide Area Networks (WANs).

Even though each WAN traversal can be in the same order of magnitude as the time needed to process an incoming event by the DVE servers, optimising this beyon the scope of this thesis as it focuses on DVE server infrastructure.

Security It does provide a single source of truth to the application source needed to construct an authoritative system. This can be used to support to detect malicious use.

But the described system does not determine whether a request for a change or action by a client is permissible.

It does not protect network transmissions against malicious or accidental change, neither between the servers within the same data center, nor the servers and the clients connected via a the public Internet.

Advanced fault tolerance The level of fault tolerance discussed in this thesis only provides protection agains service interruption or data loss.

It does not cover the level of protection against bit-rot or faulty calculations that would be expected from fault-tolerant system used in avionics, vehicle or industrial control systems. Each computation is only done once and partitioning of resources is only done to support scalability on commodity hardware.

Processing efficiency As discussed in section 2.3.2, this thesis does not cover the question whether computing and network resources are used efficiently (i.e. the minimal amount of processing

per incoming game event).

It focuses on providing the scalability to reach high numbers of supported concurrent users, employing methods for optimisations towards this goal instead of cost reduction.

In addition, section 3.3.2 and section 3.3.4 discuss the adoption of eventual consistency, which I forego, as it would require a application specific functionality.

Chapter 2

Background and Related Research

This chapter aims to give a comprehensive introduction into DVEs and their characterisation in section 2.3, presenting a number of common applications of the technology in section 2.10.

Section 2.7 presents an abstract model to aid in comparing and evaluating different types of DVEs regarding their ability to support high avatar density situations.

2.1 DVE Applications

Software systems built from a relatively small set of technologies have been adapted to a very wide range of DVE applications. Widely different applications can share a similar software underpinning.

The most common one are online games, which is also the application with the greatest commercial scope. Yee (2006) finds a number of social and psychological motivations for participating in online games, especially Massive Multiplayer Online-games Roleplaying Games (MMORPGs) of various subjects. These motivations not only ensure prolonged use, but also in-depth immersion that will lead to an ever greater wish for technical finesse by the user base.

There is a trend towards more complex environments with higher level of immersion on the path to eventual high fidelity virtual reality while at the same time increasing the number of concurrent human participants; this has been discussed in White et al. (2007) as well. But DVEs have also been proposed as an aid in training and even used in therapeutic settings. The use of in-game entities as

tutors is described in Al-Gharaibeh and Jeffery (2010*a*) and Al-Gharaibeh and Jeffery (2010*b*) and outlines another, more educational use of the same underlying technology.

This thesis was started as the core of Timadorus, a greater research effort to create a platform for very large, highly dynamic MMORPGs. The vision of Timadorus is the realization of several long term goals to generally improve the quality and scope of a wide range of applications to be supported by a commodity middleware. The principal goals are:

Non-Stop The platform should support continuous operation, even when updating the software or introducing a wide range of changes to the state of the DVE, e.g. when introducing new content into a MMORPG.

This also includes the update of any client software and or assets installed to the local machine.

Regular planned or unplanned downtime is not acceptable. Overload situations should be made improbable if not impossible by design. Resources must be assigned automatically and be dynamically re-assigned if necessary. The only exception to this requirement is a complete overload of all available resources. Operators need to be warned of imminent overload.

No Single Point of Failure There must be no component whose failure will interrupt the service as a whole. Local levels of degradation in case of component failure must be acceptable to the vast majority of the users that are affected by it. All service degradation must be temporary and should heal itself automatically.

Continuous There must be no interruptions to user interactions with the game that may negatively impact immersion. Also, existing limits to the scope or type of interactions between multiple users due to technical reasons must not be visible to users.

This will disallow realms, shards, zones or similar concepts that either segregate the user base or place limits on the interactions between avatars. Naturally, there may be technical reasons to partition the game. But these must be completely transparent to the users.

Scalable No single component should limit the number of concurrent users of the system, regardless of whether they are human or technical.

The target size for the system is one million concurrent clients. Also the spatial size of the game world should not be restricted. With the the exception of *World of Warcraft*, this exceeds the number of players of any commercial MMO, regardless of commercial success.

Manipulation Resistant Cheating should either be so hard or be of such little benefit that nobody can extract a relevant gain. Cheating should be, if not impossible, at least be so difficult that it is more worthwhile for participants to contribute to the community rather than damage it. And if unwanted manipulation cannot be stopped, it should at least be detectable.

Vital for the discussion on this goal is the consideration of what actually constitutes cheating and what is a skillful manipulation of a game design. The base for further thought on this is the philosophy that *everything that is possible should be allowed and anything that is not allowed should not be possible.*

This requirement is vital to open the client of the game to 3rd party developers. Most current MMO games need strict control over the client to ensure a modicum of anti-cheat measures. Some also obfuscate or even properly encrypt the network traffic between the client and the rest of the MMO to make manipulation harder. But the current method of controlling cheating has shown itself to be a dead end by resulting in a constant arms race between authors of manipulation tools on one side and the game service providers trying to detect and stop them on the other side.

Future Proof There are few areas of computer science that develop as fast as the quality of computer generated images and 3D worlds. Decisions made today regarding acceptable detail fidelity ensure that rendering in real-time on current hardware can seem outdated just months later.

This requires creation and management of visual resources that is unlike present game industry practice of re-creating most assets for each product. Rather master models should be of the highest degree of detail possible. Processes, methods and tools to automatically downgrade assets for in-game use must be created and deployed. This processing must take place with as little human intervention as possible. But it should support such intervention if needed or wanted.

Especially the first and second item are requirements common to virtually all distributed systems acting as services. Based on personal experience few, if any, of the large commercial MMORPG offerings can deliver on all of these goals.

Research in DVEs has been very active for more than 20 years. In order to place the above requiremnts into the proper context of my research, I set out to answer the following questions

- What is the largest supported number of avatars reported for systems that pose the least amount of limitation or impedance to the interaction of any two or more avatars present in the DVE?

  Findings for this item will exclude many of the commercial offerings (e.g. World of Warcraft). These can boast a very impressive number of customers, but divide their user base among a large number of shards ("Servers" in WoW parlance) that represent disconnected copies of the VE. This violates the goal of a single continuous environment stated in section 2.1

- What other parameters and means of measurement have been proposed to characterise a DVE and its performance?

- What fundamental methods of partitioning the overall state of an DVE, the propagation of changes and the merging of versions of the state have been proposed and analysed?

- As preliminary research seemed to offer two fundamental software architectures - client/server and peer-to-peer - what are the guiding factors when making a design decision to choose one of them for further work?

This should be complemented with research on DVE applications and the networking and operations consideration that surround any large distributed system and sizeable software development project when implementing a complete system.

2.2 Terminology

In this thesis, some terms are used as presented in Delaney et al. (2006), with some additions and notable differences:

State. the sum of all information on all elements that make up the environment contained within the DVE. Usually this is contained in a set of objects within the DVE, often accompanied by the meta data needed to control the access to the state and/or manage the processes of changing the state.

Entity or Object. An identifiable element of the state. In contrast to Delaney et al. (2006) I will use the term Entity only to denote active objects that are controlled by some process to receive change messages. When perceived from within the VE, Entities will appear to initiate actions. Entities are a true subset of objects.

In the vision for Timadorus, there are no static objects as characterised in Lui (2001). Geographical features like mountains could be modelled as static objects which will never change. But that would block an application from providing support for manipulation of the virtual environment by the user.

Entities can be differentiated further into avatars and other entities. The former denote an entity that is controlled by a human player (or an AI that is sufficiently sophisticated). Many DVEs, especially MMORPGs also differentiate Non-Player-Characters (NPCs) or mobile objects (mobs). The latter name originates from the fact that in some MMOs like *World of Warcraft*, the only dynamic elements within the game world, beside the player controlled avatars, are the mobile objects. These serve as opposition and threat while the player explores the environment.

Other entities would include technical or magical effects that may initiate change in the state over time. A simple example would be a fire that would consume other objects unless extinguished.

Timadorus differentiates between Avatars that are controlled by software components connected via a network connection, while entities are maintained by the game server software, but this is entirely an implementation specific differentiation.

Environment In contrast to Delaney et al. (2006) as well, I use this to describe the conceptual world that exists within the DVE. Its graphical rendering in the user client would present the information contained in what Mauve et al. (2001) describes using this term.

Message This encompasses the concept of an event described by Delaney et al. (2006), but is considered as a *request* to change the state. Messages are processed by Nodes for an entity and are either generated by clients controlling avatars or internal processes that are controlling other entities.

Node a single processing element of the computing resources of the DVE. I will not differentiate between physical servers, virtual/abstract machines or individual processes running on them. A node is characterised by being able to hold a local state and to send and receive messages from and to other nodes.

2.3 DVE Characteristics

based on the rough goals for the Timadorus project, I needed to termine what the key features, characteristics and metrics of a DVE are. Regardless of the software architecture (see section 2.6), all DVEs share a number of characteristics with any distributed system, especially distributed simulations. Liu, Huaiyu et al. (2010) lists the following features that sets DVEs apart:

- Shared space: Users have the illusion of exploring and interacting in the same space.
- Graphical user interface: The world is represented to users visually, usually in 3D form.
- Interactivity: The world allows users to interact with each other and alter, develop, or submit customised content.
- Real-time: Users should be able to see changes in the world as they occur.
- Perpetual: The world continues to evolve regardless of whether users are logged in.

While the alteration of the environment, or even introduction of new elements is subject to the decisions of the designer of the actual application utilising the DVE, I concur with the authors on the influence this has on the widening of the creative envelope this grants to the designer.

By their very nature, all DVEs share the problem of all distributed Systems: Brewers Conjecture. It has been analised in Gilbert and Lynch (2002)

But beyond the commonalities, there are a number of capabilities and features that diffentiate DVEs amongst each other. They form the basis of what is can be experienced by the user when interacting with the DVE application. This more detailed set of requirements controls the stability and quality of service provided by a DVE platform. When traversing the full path of communication from user client to user client, not only dependent requirements must be discussed, but also the component that fulfils each of them. These will vary between the various architectures.

2.3.1 DVE Target Application

Not all DVEs are capable of performing the same type of operations as others. In the past, this may have often have been the result of a DVE being developed specifically to support a previously defined application or being developed in parallel with it. A few exceptions such as the OpenSimulator (OpenSim-Project (2013)) do exist, which was developed with the specific goal of supporting various research for virtual environments.

The two main parameters with which the applications drive the design of the DVE are a) the acceptable reply time for a network round trip, ranging from less than 50 msec for First Person Shooters (FPS) (Claypool and Claypool (2010)) to somewhere closer to 1 second for MMORPGs (Fritsch et al. (2005)). Even within any given game, the acceptable time may vary widely due to user action. An overview is given in Claypool and Claypool (2006)

2.3.2 Scalability

The scalability of DVEs is required on three, mostly orthogonal, dimensions as described by Müller-Iden (2007, sec. 2.3):

Number of Users that can connect to the DVE concurrently and are able to interact with each other. Often this is described as participating in the same *session*, as not all DVEs are continuously running all parts and may have their users split into multiple *shards*, each running an independent copy of the environment. Further considerations on the user population are discussed in section 2.10.1.

Spatial Size of the Environment describes the spatial extension of the virtual environment.

The amount of memory and storage capacity for a DVE is highly dependent on the method of storing the information on the physical structure of the virtual environment. This is greatly influenced by the density of features within that spatial expanse (e.g. compare inter-stellar space to a cave full of rock formations).

In contrast to Müller-Iden (2007), I do not consider the number of entities to be a direct characteristic of the size of the environment. Technical entities and NPCs/Mobs can be considered part of the connected user population with regards to the processing resources required to support them.

Avatar/Entity Density is defined as the number of avatars within a given volume of space.

The value may vary widely even within a comparatively small space of the overall DVE, thus peak values are the most relevant information. The creation of small areas of tight clusters of avatars is mostly driven by human users that congregate in places of interest (see section 2.4 for more discussion of user behaviour). Traditionally this has been the domain of human controlled dynamic entities, thus the term *Avatar Density* is used in the following, instead of the more accurate *Entity Density*. This may change if more multi-avatar DVEs like those described by Shen et al. (2015) are presented.

Further discussion on this attribute can be found in section 2.7.4

The need for scalability is discussed in Gupta et al. (2009), as well as spanning a space between computational complexity for different classes of applications and the observed scalability, mostly regarding the number of concurrent players.

Complementary to any measurement of the scalability of DVEs, commonly the effects in processing time, memory requirement or communication overhead due to the asymptotic complexity will increase beyond a threshhold of an interactive reply, thus placing a limit on the maximum of one or more of the metrics described above. Just as worthwhile would be the study of the efficiency of DVE, i.e. the measurement of the processing required for each additional user.

Due to the high complexity of the software involved and the quality of the actual implementation, this is area is beyond the scope of this thesis. Such analysis is probably done by every large commercial vendors, but the results are regarded as vital commercial secrets.

2.3.3 Interactive Reply

A DVE is commonly used by a real user in front of client software. For the immersion of the player, it is vital that the effects of actions are presented in a timely manner. Research on the appropriate meaning of 'timely' in this context has been part of the research into virtual environments from the beginning. Especially in CAVE like visual environments (Park and Kenyon, 1999), the ability to perform a task with respect to reaction delay has been analysed.

For game type DVEs, the amount of permissible latency is a function of game design and can take extreme measures ($>$ 1 sec in Everquest2, Fritsch et al. (2005)).

But for some game genres, an increase in reaction times can vastly change the game experiences as discussed by Claypool and Claypool (2010). In the most time-sensitive type of game, a First Person Shooter even delay and jitter can have a varying effect on the players performance and enjoyment of the DVE as shown in Quax et al. (2004).

As general aid in determining valid, Smed et al. (2002) states reply times between 0.1s and 1.0s for massive computer games (MCGs), as a rule of thumb. In the remainder of this thesis, a reply time of 100msec for the game server only will assumed. Latency experienced by the player will include an additional network round trip over the internet (from the user client to the server and back). The amount of additional delay accrued here is beyond this thesis, but assumed to be on the same order of magnitude as the internal delay of the DVE server, based on the table in Norvig (2001).

2.3.4 Consistency

A DVE requires the simultaneous application of two different perspectives on consistency: data consistency as information may be stored in multiple locations due to the distributed storage of the environments state. At the same time, changes to the state have to be handled consistently with respect to the rules of the virtual environment as well.

Zhou et al. (2004) discusses the problem of creating a consistent view of the state to external observers, in contrast to the consistency that cannot be guaranteed as shown by Fox and Brewer (1999)[1]

[1] some points have been updated in Brewer (2012)

Li and Cai (2012) proposes a schedule and protocol to enforce a global consistency in a multi-server DVE.

The possibility of using Eventual Consistency is dictated by the availability of a merging function for conflicting changes during a network partition. For most use cases, this can be achieved if a compensation mechanism is available (e.g. in trading with other players or computer controlled entities within the environment). For MMORPGs, this is the case for many instances without reducing the immersion of the user in the virtual environment. If the user observes a certain sequence of events and the resulting changes to the state of the environment, this change must not be undone, except for reasons within the ruleset of the environment.

2.3.5 Separation of Area of Effect and Area of Interest

SPEX (Najaran et al., 2014) presents a method to separate the scope of perception (managed by the Area of Interest, AoI, see section 2.9.3) from the position of entity as well as the Area of Effect (AoE):

> In most existing systems [...], the AoI of a user is tightly coupled with its position in the VE, i.e., a user publishes its location and its AoI is assumed as the area surrounding its position in the VE. SPEX allows users to fully decouple these two. A user can publish to any location within the VE, while its AoI could also be on another region in the VE.

While Najaran et al. (2014) mentions two elements (AoI and AoE) they actually support three points in the environment to be separated: a) the center of the AoI, b) the point of effect or center of the AoE (depending on the type of effect caused by the participant), and c) the position of the entity.

There is a classic example for applications that benefit greatly from this capability in a DVE: a sniper uses a high-powered telescope. This places the outer edge of his vision a great distance from his position. If this is solved by a large AoI, most of it would have to be filtered out, as the telescope actually only has a very limited view field. The sniper may now elect to shoot at a target in his view field, again far removed from his actual position, but limited in area. And during the whole time, the sniper may be attacked himself, thus having to subscribe to updates published to the area

of his position.

2.3.6 Object Dependencies

A majority of proposed DVE systems focus on one or two aspects when attempting to document the benefits of their approach: the distribution of movement events and updating of positional state of entities on the one hand and, especially in systems structured around AoI, the visibility of these actions to other entities on the other hand.

But virtually all relevant applications also have some form of interaction of entities with objects, forming a dependency between objects needing special attention, as it is one of the contexts in which multi-object transactions as well as conflicting access may occur. Müller-Iden (2007), section 4.4.1, pp. 120-124 discusses the necessary considerations to ensure interactions are handled correctly, not only from a protocol perspective, but also in a manner that is expected and accepted by users of the DVE. Please note that this behaviour may be set to a particular method by designer of the DVE application. But the capabilities of the DVE platform influence the methods that are available to the designer.

Funkhouser (1995) first mentioned dependencies, proposing a model that prohibits mutual destruction. This poses a limitation to game design but eases the problem of ensuring causality.

2.3.7 Avatar Density

Providing a mechanism to handle high levels of avatar density has been discussed in in Lee and Lee (2003). But this solution uses a fixed spatial partitioning into cells, thus placing a limit on the granularity of the subdivision taking place. For example, if all users congregate into a single cell, there can be no further load shedding once all other cells have been removed from the server. Their approach also uses a zone-based approach, that will cause border effects when users are crossing zone borders, especially during cell migration from one server to the next.

2.3.8 Client Assignment

Distributing the workload resulting from clients contacting the DVE is of vital importance, not only to optimally distribute the workload, but also to provide fault-tolerance in the face of failed server nodes or communication links.

The general problem of optimal client assignment (or terminal assignment) is NP complete. An overview of terminal assignment algorithms, especially heuristic based, is given in Khuri and Chiu (1997)

A general discussion of achieving this through client assignment in games is given in Duong Nguyen Binh et al. (2006).

Client assignment need not be static, but recovering from faults as well as optimizing the client assignment dynamically may require a re-assignment of clients to servers.

2.3.9 Authoritative Control

Virtually all recently published games have a communication aspect built into them. This can be by original design, as a form of multiplayer game, or due to an integration into social platforms like Valve's Steam or Microsofts XBox Live. Many games suffer from a vulnerability of so called Cheats that enable some of the users to break the rules of the game. According to Blackburn et al. (2011):

> "cheats" are software components that implement game rule violations, such as seeing through walls or automatically targeting a moving character.

Breaking the games rules may result in reduction of the enjoyment of other players or even loss of resources with financial value in the real world. With the rise of in-game resources being available for purchase for real money, users may either spend considerable amounts or have generated relevant value by participating in the game. The wide range of feelings and aims of various groups of players and the aims of the industry producing the games has been described in detail in Consalvo (2007).

Individual games have tried to use social approaches of granting amnesties to users who do not follow game rules: GamesFirst (2011), but the comments in the article show that various groups of users have very different perceptions of what a *cheat* constitutes and some even see anti-cheat

measures and the hunt for cheaters only as a meta-game to manipulate the game while eluding the search for them by the games publishers.

From the perspective of a DVE middleware, there are only actors manipulating the state of the environment. Technical methods may attempt to enforce a filtering or limitation on which actor is allowed to introduce which change to the environment. Social protocol may dictate rules and conventions as to what constitutes breaking these rules and to what extent this is permissible.

The technical difficulty of protecting against an unwanted manipulation of the state of the environment contradictory gets increasingly difficult if user-supplied elements are introduced into a DVE. Not only is the user client software and the communication between client and server under the control of the user, but also part of the software components of the server. The Meru architecture described in Horn et al. (2009) (expanded in technical detail in Horn et al. (2010)) provides a method for enforcing authoritative control over the environment and delegating processing for individual objects of the environment to clearly defined owners.

2.3.10 Fault Tolerance

As DVEs are, by definition, constructed from multiple separate processing units, each of these may fail for numerous reasons, and thus must provide a method of handling such situations.

For P2P systems, fault tolerance may be part of normal node management. This is discussed as section 5.3 in Bharambe et al. (2008). They assume the full state of the game is maintained in each node, thus failure will not lead to data loss.

In Hu et al. (2010) the requirements of fault tolerance are more detailed, as nodes are stateful. The authors also position improvement in fault tolerance as measurement of improving the quality of a DVE.

In C/S, especially multi-server architectures, the fault tolerance can be provided by a separate data persistence layer, provided the actual game servers are stateless. This can be achieved by using various persistence techniques. One is Eventual Consistency, as described in Vogels (2009) and utilised in Amazon's Dynamo database (DeCandia et al., 2007) and hence implemented in a number of similar systems. Or by using the High-Availability or Load Balancing mechanisms in a number of commercial and open source databases.

But while the former requires a consolidation function to solve any conflicts that may arise, the latter usually requires all writes to be synchronous among all nodes of the database. Either choice may reduce the capabilities of the DVE, since they are write heavy by definition.

2.4 User Behavior

Since a DVE needs to be able to support the workload created by human users, understanding their characteristic requirements is a necessary prerequisite to its design.

Pittman and GauthierDickey (2007) has done an in-depth analysis of multiple large online games and have found two basic patterns for user behavior: a) a five-fold diurnal change in population closely matching typical prime time media hours. And b) concluded that: "The distribution of players in the virtual world also appears to follow a power-law distribution. This result is important because prior research has assumed the uniform distribution of players in the virtual world."

In Suznjevic and Matijasevic (2013) the authors have undertaken a thorough survey of different online games, including DVEs, such as MMORPGs. They find that the requirements are very varied and some state-of-the-art network protocols are not the optimal solution for handling DVE network requirements. Specific for DVEs, they conclude that: "Avatar distribution in the virtual world is not uniform, but rather conforms to the multiple hot spots model, which needs to be taken into account when performing simulations".

From this I derive two vital requisites for the DVE to support: a) being able to change the amount of available aggregated computational resources available for consumption by the system over time (i.e. scale out) and b) to freely assign these resources to any location within the space VE in order to support arbitrary avatar densities or any other fluctuations in processing needs that may arise (not all avatars will require the same amount of processing at a given time). To reduce cost, the ability to scale down, i. e. reducing the available computational resources may just be as important. The term computational resources above applies to CPU, memory as well as network bandwidth and storage bandwidth. Storage space is unlikely to impacted to the same extend, as the state of avatars will have to be stored, regardless whether they are active (i.e. a client is connected to them) or not.

2.5 DVEs as an Event Stream Problem

In order to better understand the requirements and bottlenecks of a DVE, I find it helpful to understand it as an event streaming system, wherein events are represented as messages, sent from one component to one or more other components. As all messages are of roughly the size, the number of messages represents a valid approximation of the communication bandwidth requirements. Reply latencies are the elapsed time between the start of an originating message being sent and the completion of the receipt of an appropriate reply message by the same component.

2.5.1 Naive Client/Server

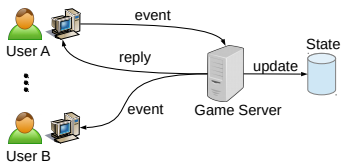


Figure 2.1: DVE as event stream: naive

Figure 2.1 shows the initial architecture: a single server machine receives messages from any of the client software operated by a user and will send back a reply message as well as informing other clients of the changes in the virtual environment. The server will also update the state maintained in the persistence back-end.

This is the starting point for all DVEs, described multiple times, e.g. in Macedonia and Zyda (1997). This approach can neither scale beyond a single machine, nor provide fault-tolerance agains hardware failure of the game server.

2.5.2 Multi-Server

The next step is applying scaling and fault tolerances with techniques that have been state of the art at since before 1990: multiple game servers and mirrored storage hardware.

This is shown in fig. 2.2. The main benefit is the scaling of the available processing resources beyond a single machine. This characteristic also provides limited fault-tolerance, as dead hardware

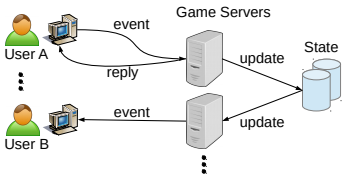


Figure 2.2: DVE as event stream: multi-server

can be switched, while service continues to operate. The rest of the system can run with reduced resources while the broken machine is missing.

The biggest drawback is the still limited scalability of the storage system as well as its limited fault-tolerance. The mirroring provides protection against catastrophic hardware failure, but not against single bit errors in storing the data.

But even worse is the fact that each update has to be sent to all servers, as there is no tracking which client to which server will actually consume which update.

This approach has been published multiple times, in its extreme form (large number of servers for a single logical database) in the Sun Gaming Server (SGS, Blackman and Waldo (2009))

2.5.3 Multi-Server with Partitioned State

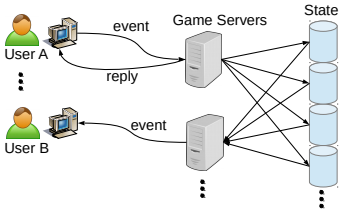


Figure 2.3: DVE as event stream: partitioned state

As the required read and write bandwidth of the storage backend becomes the bottleneck, it too can be partitioned. This is shown in fig. 2.3.

Above the multi-server solution of the previous section, it enjoys better scalability as well higher ability to handle faults in the storage layer as well. When combined with redundant storage, single storage nodes can be swapped out without data loss or service disruption.

The biggest drawback is the fully connected network between the nodes of the cluster, as all game servers need to communicate with potentially all storage nodes, which in turn need to forward updates to all server nodes, still suffering from the n^2 problem of a fully meshed network.

2.5.4 Filtering based on Area of Interest

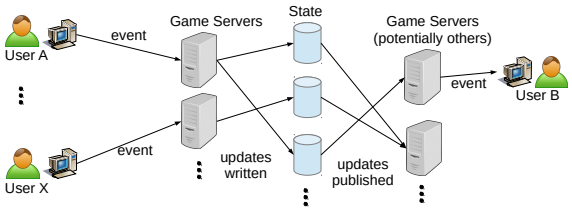


Figure 2.4: DVE as event stream: filtered based on Area of Interest

Figure 2.4 shows the approach taken in this thesis: in order to avoid the need for a fully meshed network, game servers subscribe only to the data and its updates that are relevant to the avatars present on the particular game server. Please note: the two columns of game servers are only shown to make the flow of messages more visible. All game servers are part of the same set, a single game server may even be shown twice in this representation.

The relevance of data for an avatar is computed by tracking the Area of Interest (AoI, section 2.9.3) of each avatar.

2.5.5 End-to-End Analysis

A previous analysis of complete DVE systems has been performed in considerable detail in Kwok (2006). But the author does focus on the overall performance characteristic, doing an in-depth analysis of the causes of latency. He neither considers areas of congestion nor overload situations in depth. Therefore, methods of their avoidance or mitigation are not discussed either.

An in-depth survey of methods for combining and distributing state changes in DVEs has been conducted by Liu et al. (2012), including methods for reduction in network traffic like AoIM. But they neither cover further network optimisations, nor limits or optimisations on bandwidth requirements for client links.

2.6 Software Architectures

The underlying architectures of DVEs researched in the last fifteen years are one of two basic designs: peer-to-peer (P2P) and client/server (C/S) systems. The former has received a very high level of attention in the last decade.

2.6.1 Client/Server

In its pure form, a Client/Server (C/S) architecture would imply a single physical server with large CPU, memory and networked bandwidth resources. But these are either non-existent or cost-prohibitive for supporting a player base larger than a few hundred.

In the following, C/S shall imply a multi-server system as characterised in Yahyavi and Kemme (2013): a set of multiple servers that are concentrated in one or at most a few physical locations. In most cases, these locations are dictated either by internet latency concerns and thus located on each real-world continent, or they are simply placed to observe the availability of large computational capacity like the regions of the Amazon Web Services.

The C/S system is characterised firstly by the central control, including physical access, informing its security model.

C/S systems have two vital drawbacks compared to P2P systems: a) the cost of communication, as the aggregate bandwidth of every client connection has to be paid by the operator of the DVE, and b) a single point of failure. The former was the stated motivation for a lot of the research on P2P DVEs. The latter can be reduced by redundant components in the processing as well as the networking infrastructure. But it also poses a single point of attack. This makes it attractive to various parties that are interested in causing a disruption of service. While this, in turn, can be reduced in relevance by using multiple points of connection to the internet at large when operated as a production service, this will drive up the costs further, aggravating the first problem even more.

Further downsides of a C/S, especially in its pure form outlined in Yahyavi and Kemme (2013) are mainly due to the type of partitioning used by some implementations and not necessary a characteristic of the software architecture, but more of product/service design decisions. For more on partitioning see section 2.8

2.6.2 Peer-to-Peer

For the last 15 years, DVE research has largely been focused on peer-to-peer (P2P) architectures in comparison to the C/S approach. The central aim was to make DVEs much more scalable: Ahmed and Shirmohammadi (2008), Bharambe et al. (2008) and many more.

The basic concepts of a peer-to-peer system and one of its commonly referenced protocol design have been published in Castro et al. (2002). Initially they were put forward as distributed storage mechanisms.

In Bharambe et al. (2002), another commonly referenced P2P protocol, CHORD, is used as design inspiration for a publish-subscribe infrastructure that routes state change information only to those participants that have subscribed to it. There is no further partitioning of the game world.

Among the first to propose the use of a P2P system for use in a partitioned DVE is Knutsson et al. (2004). For simplicity, they partition the VE into regions and share the state of those that are applicable to the participant.

But, as Assiotis and Tzanov (2006) and in more detail Miller (2011), point out, maintaining security in a P2P DVE is difficult and may never achieve the level that is expected by game production companies to protect their considerable investments.

In Miller and Crowcroft (2010), the same authors also cast doubt regarding the use of P2P techniques for private users. These are commonly connected to the internet via asymmetric connections (e.g. via DSL, or cable TV infrastructure). The leads to a restriction on the outgoing bandwidth of the node that may be insufficient to operate a P2P DVE, with its tight timing requirements.

But the research into techniques to solve specific problems of larger scale DVEs may still be applicable to other architectures, either by constructing a hybrid system or transferring the technique to other architectures.

2.6.3 Criticism of Peer-to-Peer Architectures

The decade of intensive research of P2P-based DVEs has been summarised in multiple surveys. One of the more complete ones can be found in Yahyavi and Kemme (2013).

A number of general problems in P2P systems are identified in Fan et al. (2009), but the two

main problems arise from the difficulty to maintain an authoritative state of the DVE: participant churn, which could potentially result in data loss, and manipulation of game state not permissible by the rules of the DVE or its user base (for a lengthy discussion of what may or may not be considered unwanted behavior by users or the operators of a DVE see Bartle (2015*b*) and Bartle (2015*a*)) .

In Stutzbach and Rejaie (2006), the churn of file distribution P2P systems is analysed. These P2P systems usually have a class of participants that are sometimes called *seeders*. These are nodes that will linger after they have completed downloading the data they requested, in order to provide segments of the files to other participants. But for this behavior to function, the data to be distributed must be static. It is the defining feature of DVE to have dynamic world state. In Pittman and GauthierDickey (2007) the authors state:

> Playing session durations appear to follow a power-law distribution where approximately 50% of the population remains online for 10 minutes or less. This impacts the design of MMOG architectures because it indicates frequent churn among players. Peer-to peer architectures will need to ensure that the frequent churn does not result in a large amount of overhead for overlay maintenance.

Similar to voluntary churn of users leaving the DVE application, a large part of the state can be lost due to a network partition. As it would be a common behavior of the user to restart their client program (and thus the DVE peer), complete data loss can be assumed unless special measures are taken.

Even one of the first publications to propose the use of P2Ps for DVEs, Knutsson et al. (2004) identifies security as one of the problems of this type of architecture, but explicitly chooses to ignore it: “Security - both the prevention of account thefts and the prevention of cheating during game play - should be considered. Distributing game states to the peers increases the opportunities for cheating.”

Requiring to trust each of the nodes seems to be the only option, according to Hu and Liao (2004), otherwise it would be very difficult to achieve stable security controls. Enforcing that trust to help overcome security issues of P2P architectures are presented in Wierzbicki (2006). More ideas on how to enforce rules are given in Schuster and Weis (2011). Unfortunately the author explicitly

ignores churn and describes a complex model, which does not guarantee a consistent state to be presented to the user.

New commercial large scale MMO deployments like Elder Scrolls Online, which are based on a C/S architecture, may indicate a lack of trust in P2P systems with commercial vendors.

Smaller problems of the P2P architecture include the potential for multiple network hops as shown in: Boukerche and Ahmad (2008). These may add latency of unknown size.

The problem of discovering the initial peer for a P2P architecture may be overcome by the use of an initial server at a well known address, to discover peers in P2P network (Rhalibi and Merabti, 2006). If the actual implementation consists of list of globally distributed hosts, that are not actually part of the resultant P2P overlay networks, this can be made very reliable as well resilient even to targeted attacks.

In general, P2P architectures lack of central entity leaves little opportunity for global aggregation (Hu and Liao, 2004).

2.6.4 Hybrid and Heterogeneous

In order to receive the advantages of C/S as well as P2P architectures, a number of publications have presented combinations of the two approaches, often as an implementation of the multi-server style architecture.

A separation of processing components from a component that primarily acts as the persistence backend is presented in Jardine and Zappala (2008). While this approach still retains some of the security issues that result from trusting the policy decisions of peers, it mostly avoids the data loss from uncontrolled peer churn and allows some security controls on the DVE state stored in the central server. Their analysis focuses primarily on the bandwidth savings when compared with a full P2P architecture.

The Meru architecture (Horn et al., 2009) separates three types of state in a DVE: objects, requiring processing resources and maintaining object state; spaces, holding spatial information; and persistent data, accessed read-mostly commonly media, mesh or texture data for visualisation. Each type of data is placed on dedicated servers and network infrastructure to allow optimisation for the differing usage patterns. The architecture presented in this thesis explicitly excludes the third

type, as the above paper already lists services like Content Delivery Networks (CDNs) as a valid method to provide such data to the client software.

2.6.5 Operations Considerations

A number of research efforts referenced in this chapter use simulation environments to conduct analysis, rather than actual implementations. Also most operate on a dedicated cluster of computers in a academic lab setting.

Running a production environment for a commercial service creates a number of challenges but also opportunities when operating on a public cloud service provider (Infrastructure as a Service, IaaS).

There has been at least one proposal for a system that has specifically been designed to operate on such a IaaS: Najaran and Krasic (2010), which is deploying a large sytem to AWS.

2.6.6 Conclusion and Architecture Selection

The research of my thesis is embedded in the creation of a truly massive multiplayer roleplaying game. It is envisioned to support the same size of player base as are enjoyed by the commercial market leaders as WoW, ESO or EVE. But in contrast to those services all players would be in a single, continous environment. This context drives some of the non-technical requirements for my design. Therefore, the problems of P2P solutions with accountability due to security issues and potential data loss are not acceptable.

But the techniques presented in some of the P2P research are applicable in a hybrid system, especially when treating the nodes of a multi-server cluster such as as a P2P system in itself. Control over the nodes eliminates the security concerns. Data loss through node failure can be effectively controlled through standard data center techniques (e.g. backups and monitored redundancy).

By the same token, a network partition within the server cluster is detectable in a straightforward manner (e.g. with likeness checks from multiple sensor points) when all nodes are in one or at most a handful of locations like one or more datacenters. It is not unreasonable to assume that human staff is actively monitoring the availability of the known number of nodes and takes appropriate action to choose one side of a partition, stop the other, and join them back to the cluster individually after the

cause for the partition has either been fixed, or a work-around has been devised. All this is possible as all elements of the server cluster (nodes as well as the network components between them) are under the control and observation of a single party.

2.7 Abstract Model

In order to compare the different methods of constructing a DVE described in section 2.8 and section 2.9, I will present an abstract model for all DVEs. The model will describe the general elements of most DVEs and how they relate to and communicate with each other. The basic building blocks are standard elements of distributed system design. Their application to DVEs has been outlined in Macedonia and Zyda (1997) for a long time. But only a few have received comparable level of interest in the two decades since then: variants of Client/Distributed Server (or Multi-Server) on the one hand and Peer-to-Peer solutions on the other, all communicating via unicast network messages.

2.7.1 Abstract Load

To model the network and computing workload placed on a DVE, I assume an abstract client C, connected to S_0, a specific server of the DVE. Depending on the structure of the DVE, this connection may vary widely. For details, see section 2.7.2. The client will send on average m/sec messages to request actions of its avatar (e.g move, pick up object, attack other avatar). Each message will be processed by the rules of the VE and result in zero or more events to either change the state of the VE or to notify other avatars. These events may either represent actual effects on other objects or avatars, may serve to update the global state of the VE, or they may represent another avatar observing the changes. All of these events result in zero or more messages sent to zero or more clients including acknowledgement sent to the client C_0, which sent the initial message.

Initially, when only very few avatars are present in a DVE, broadcasting each message to each client is possible. But this fully connected network will increase the required bandwidth quadratically. A time tested method is to filter messages with the goal of forwarding only relevant ones. A wide range of approaches are compared in section 2.6, section 2.8 and section 2.9. To represent

this in the abstract model, consider a DVE consisting of $C_0, \ldots, C_n$ connected to one of the servers $S_0, \ldots, S_m$. Each message sent[2] by the client C to the server S_0 it is connected to is filtered by the forwarding function, resulting in a set of events forwarded to zero or more nodes $S_1, S_2, \ldots$ in the DVE, based on the current world state W of the virtual environment:

$$fw(C_i, M, W) \rightarrow \{(S_1, \{E_{0,0}, E_{0,1}, \ldots\}), (S_2, \{E_{1,0}, E_{1,1}, \ldots\}), \ldots\}$$

When designing a DVE for a particular computing and network infrastructure, the maximum number of events sent or received must not be greater than the bandwidth of the link between two given nodes (S_i, S_j). If the network contains any broadcast or pseudo-broadcast elements (e.g. the backplane bandwidth of switching architecture), it must also be able to support the cumulative bandwidth of the links going through it. If the clients and server nodes are not placed on the same computer, the bandwidth between server and client to handle the replies must also be considered. See chapter 5 for more on managing this resource.

Most published DVE solutions require a certain amount of overhead to coordinate and control the forwarding function on each node. Generally the volume of this traffic will increase with the number of servers in the DVE as well as the number of the connected clients. This traffic is represented by the coordination and control function which will generate an additional set of events, sent to a subset of the nodes:

$$crtl(S_i) \rightarrow \{(S_0, \{E_{0,0}, E_{0,1}, \ldots\}), (S_1, \{E_{1,0}, E_{1,1}, \ldots\}), \ldots\}$$

The set of connected clients or their associated avatars are not an argument of the function, as a single node cannot be assumed to have a complete and correct set of avatars in the whole system.

The control traffic will reduce the bandwidth available to the transmission of the forwarded events. In a real system, the size of a network message conveying the event will vary widely. Not only between DVE implementations but also differing events within the same system. Therefore their individual or even average size is not considered here.

[2] In a DVE with a peer-to-peer architecture the client and server will be running on the same computer. They may even be simply elements of the same software component. There a server can be considered to have exactly one client connected to it.

2.7.2 Abstract Structure

Regardless of the actual colocation of the actual software components, DVEs can be divided into areas of responsibility described in the classical three tier pattern of representation, logic and model respectively (see fig. 2.5). The logic layer can be partitioned further into an application-specific layer and a generic middleware layer. A defining characteristic of any DVE is the distribution of the resource requirements among multiple instances of the software implementing the logic and model layer. This adds considerable complexity when compared with most client/server based 3-tier systems. Only in very large systems, usually as part of an online service, have partitioned layers two and three been taking hold during the last decade.

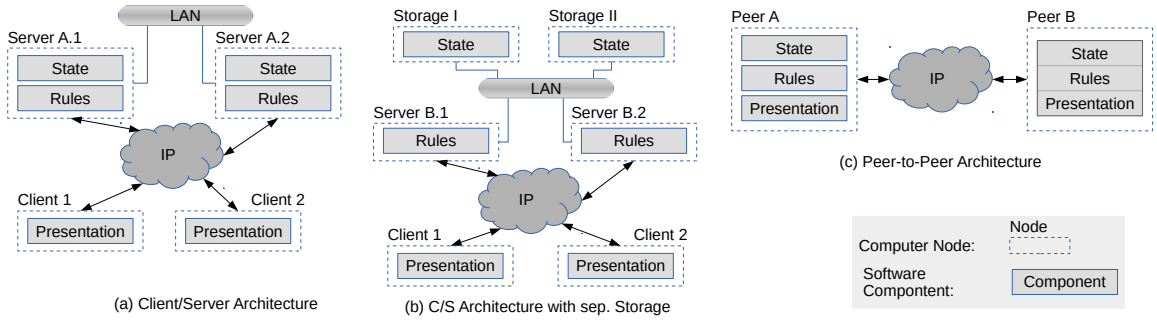


Figure 2.5: DVE Architectures

DVEs can also be categorised based on the tier responsible for exchanging data with other nodes. This is either the logic or the model tier. If the model tier is responsible for synchronizing the model between nodes, this is usually due to a general purpose mechanism being used to support a DVE system (e.g. Pastry and Scribe for P2P type systems). Some very early DVE implementations consisted of heterogeneous systems that would require explicit forwarding of parts of their individual state. Some examples are characterised in Morse et al. (2000). These systems have no common model, but each system maintains an individual model. These models may overlap, when observed from a global perspective. Exchange of state is done on bilateral basis. This is generalised in the High Level Architecture (HLA, *IEEE Standard for Modeling and Simulation High Level Architecture (HLA)* (2010)). It standardizes the interfaces and provides a common object model in order to construct a federated system. This is still used, mainly in military training and exercise envi-

ronments. In the context of games and other entertainment applications, homogeneous distributed systems using a centrally designed state model have been the common method. A range of methods maintaining these models has been collected in Liu et al. (2012).

Assuming the described form of distribution to be a correct generalisation, all DVEs can be considered as a set of n compute nodes $C_n = C_1 \dots C_n\}$ that potentially communicate with each other node in the set (e.g. they can form a fully connected net with a communication overhead $O(n^2)$). Each logic layer will support users. This would be exactly one for a peer-to-peer (P2P) architecture and usually a larger number for client/server solution.

The amount of required network bandwidth between two nodes varies with each combination of two nodes. For a bandwidth function $nb(C_x, C_y)$ for any two nodes, the internal aggregate bandwidth would be

$$\sum_{i,j=0}^{n} nb(i,j) \mid i \neq j$$

.

In a single node system, this internal bandwidth would be 0, and it would support multiple concurrent connected users.

A distributed system will place each joining user on one element of the partition set $P_k = \{p_1 \dots p_k\}$, with $k \geq n$. Partition p will support a maximum number of users of $mu(p)$. This number is the result of the amount of CPU processing, RAM and network bandwidth requirement for each user and the total available on each partition. For a given set of current users, each increase in the number of partitions will require fewer clients to be supported by each partition (or conversely, a growing number of partitions will allow a growing number of supported clients).

However, each new partition will increase the absolute amount of communication of the whole system, as described in the previous section.

If the scope of an avatar (usually the aura described in Benford (1993)) reaches into other partitions, it will create border effects (e.g. section 3 in Glinka (2007)), which in turn will increase the resource requirements. Adding more partitions to a highly loaded system may therefore actually *decrease* the total number clients it can support (assuming fixed QoS criteria).

As more and more overhead is accrued, the number of supported users per partitions is reduced, as

all other resources are used by coordinating and communicating with a set of neighboring partitions. Once this falls below a single user, the whole system has failed. A P2P type DVE, will (by definition in its purest form) never have anything other than a single user connected per partition. In contrast to a C/S system, the load of a node cannot be estimated from the number of connected users, but requires some other means to indicate it. Detecting and reacting to an overload scenario will be similar for both architectures, as more fundamental metrics like network or CPU load may indicate an imminent failure much earlier, providing more time to take measures to avoid it.

Partitions failed due to overloaded partitions may occur even if the DVE as a whole has not yet reached its combined limit of concurrent users. It fails to take the variation of local avatar density into consideration. As shown for the example of World of Warcraft in Shen et al. (2014), these may vary widely as users cluster around points of interest.

The system, which forms the central contribution of this thesis, can support the same spatial region on multiple server nodes (details are found in chapter 3). In contrast to sharding (see section 2.8), the avatars and objects are still part of the same environment and can interact with each other. To measure this, I introduce the metric of a *Weighted Minimal Supported Avatar Count* (WMSAC):

Minimum of $\frac{mu(p_u)}{nc(p_u)} u = 1 \dots k$, where $nc(p_u)$ is the number of copies of a partition that are being executed.

The WMSAC indicates the limit of overhead that a system may place on a specific partition without the most loaded partition failing.

2.7.3 Supported Clients

From the model above, two parameters can be derived to describe the performance capabilities of a DVE, which are usually are not documented.

But they can provide vital insight into the type of partition and the resulting limitations in avatar interaction when comparing the measured values. For example, while the game “World of Tanks” (WoT, a successful example of Massive Online Battle Arenas, MOBAs) has a similar player base in the millions as “World of Warcraft” (WoW), WoT will never allow more than 28 to interact with each other, while a WoW realm will support a few thousand.

The first parameter describes the number of supported clients within a single, continuous environment. Duplicating the environment (e.g. sharding in MMORPGs or battles in MOBAs) as a mean of increasing the aggregate number of supported clients is not the subject of this thesis.

Definition 1. *Oobservation delay (OD) is the elapsed time between a message from client C_i being received by a node of the DVE and a message reporting the resulting event being sent to client C_j.*

The OD is measured from the point of view of the DVE interfaces to the outside world, as WAN network delays are beyond the general focus of this thesis.

Definition 2. *A client C_{n+1} is considered to be supported, if adding it to a DVE currently supporting n clients will not increase the observation delay above a set threshold for the messages exchanged between any two clients in the DVE.*

The threshold is an arbitrary value that usually depends on the type of application provided by the DVE. For the rest of this thesis, I will assume a somewhat arbitrary number of 100 msec. This corresponds to an RPG type MMO game. Further discussion of this value is covered in section 2.3.3.

Furthermore, most real implementations of DVE systems will result in some amount of jitter and variability in the OD. Therefore, I will use the 90^{th} percentile as threshold instead of the absolute maximum.

2.7.4 Maximum Supported Avatar Density

As the density of avatars may greatly fluctuate between partitions (see Lui and Chan (2002) for commonly used test distribution and section 2.4 for a discussion of the causes), it is relevant not only to quantify the aggregate number of supported concurrent users of a DVE, but also to evaluate the maximum number of avatars within a given spatial segment of the environment.

The architecture of a DVE may influence the spatial sizes that inform the density limits. For example, a zone-based DVE (section 2.8.1) may place no limit on the density of avatars in a single spot, as long as a certain number is not exceeded for the avatars within one zone, while a solution based on tracking the Area of Interest of an avatar may support an unlimited number of avatars within the DVE or even placed within the vicinity of the avatar, as long as a limit of those visible within its aura is not reached.

From the standpoint of a single avatar, only those other avatars are relevant, with which it can exchange information (e.g. can sense them or be affected by them). Therefore, I define supported avatar density as follows:

Definition 3. *For a given avatar, the local supported avatars density is the number of other avatars it can interact with, meaning either sense or be affected by, without the* 90^{th} *percentile of the OD being greater than 100 msec.*

2.8 Partitioning

Using spatial partitioning of the game environment to allow the distribution among multiple communication or processing elements has been a well researched method for a long time. One of the earlier examples are presented is in Hagsand et al. (1997).

This covers the partition from the perspective of the game. Partition of computational resources or the persistent state is a result (even if often the intended result of an architectural decision). In the abstract model described shown in fig. 2.5, this would represent a partition in the logic layer, while a distributed persistence layer would be an example of a partition (and required synchronisation) within the persistence layer.

Pittman and GauthierDickey (2007) states:

> To date, all commercial MMOGs use a client/server architecture. Most commercial MMOGs are divided into realms. Each realm is managed by a logical server usually in the form of one or more physical machines acting as a cluster. Players in the MMOG play the game through an alter-ego, or character, represented on screen in a 3-dimensional world. They can interact with the world and with other players. A realm in a MMOG can handle several thousand players simultaneously. For example, in the popular MMOG called EverQuest, a typical cluster handles around 2,500 players concurrently, with around 10,000 players registered on each realm. To compensate for millions of subscribers, companies host hundreds of realms across the world.
>
> Realms are further divided into distinct areas called zones. Each zone ranges in difficulty, preventing new, less experienced players from traveling to more difficult zones. This

subdivision helps prevent players from amassing in a single zone which would result in processing difficulties for the server. Indeed, the processing cost for the server for n nearby players could be $O(n^2)$ if it must consider interactions between all pairs of players that are within a close distance to each other.

The terms realm and shard are used synonymously in differing publications. In this thesis I will use the term shard.

The most obvious method of partitioning is sharding, which is not only used in WoW but also fundamental fundamentally approached in the entire genre of Massive Online Battle Arenas (MOBAs.) It is used as one aspect of multi-server in Yahyavi and Kemme (2013)

2.8.1 Zones

Zones were introduced by Ultima Online as evolvement from rooms in MUDs according to Kesselman (2011). He also lists some of the inherent problems of zones: a) overcrowded individual zones, b) zones are subject to server overload by other zones on server. Higher server power may lead to under-utilised resources.

There are some examples of zones that are born out of the design of the application that is deployed on the DVE, e.g. the individual solar systems of EVE Online. When mapped directly to sufficiently powerful server nodes, they can avoid the problems of overloading or underutilisation as they may migrate zones between servers (X, 2013). Due to the design of the game, the players actually expect travel from one solar system to the next to take some time. Thus border effects and zone switching workload that are commonly inherent in a zone based design do not break immersion or the game flow expected by the players. But in the case of massive battles (CCP Inc., 2013), the local avatar density can overwhelm a single server node and require techniques like time-dilation (CCP Veritas, 2011) that are not covered by the definition of supported users presented in section 2.7.3.

A variant of the zone is the instance (described in Glinka (2007), but also widely used in many commercial MMOs), which duplicates the properties of a shard for a small set of zones (usually a single zone) instead of the full set of zones that make up the spatial area of a DVE.

In Iimura et al. (2004), zone-based partitioning is coupled with local cached data to reduce the amount of state to be held by each peer in a P2P network. A leader election and conciliation allows the DVE to handle node churn while minimizing the data loss. While it tackles some of the problems of P2P systems (see section 2.6.3), it does not address the specific problems of zones.

In Assiotis and Tzanov (2006), a locking protocol is discussed to handle hand-off between zones, which are potentially on different servers. The publication also discusses the need to handle border effects and shows the strength of their approach in avoiding additional overhead for objects close to the borders. The authors propose increasing the number of zones to handle high load situations by splitting the zones. But they also incorporate a queue to order events whose growth is unbound as zones get ever smaller.

In Ahmed et al. (2009), more issues with handing objects between zones and the need to establish a specialised border zone are discussed to reduce the necessary communication overhead to not only handle objects that traverse from one zone to another, but especially objects being visible from one zone to the next.

Zones do allow interest based filtering, especially with small zones, and allow filtering and forwarding based on the zones that contain at least one interested observer (Shirmohammadi et al., 2008). But section 2.2.3 of the same paper also documents the problems inherent to zone based approaches: The need to handle border situations where an avatar is present in one zone, but interested in events of a neighbour zone. This causes a possible instability in object position and network bandwidth requirements. This is compounded, when zones are equivalent to servers or overlay network nodes e.g. in a Peer-to-Peer topology

Dynamic Zone Partitioning

JoHNUM (Farooq and Glauert, 2009) proposes a tree of recursively partitioned zones, which are able to react dynamically to hot spot situations as well as general player density imbalance. Since the newly created regions are then assigned to arbitrary servers, this can help avoid a wide range of overload scenarios with regards to processing. But it does not address the problem of increased cross zone communication that will result from arbitrary partitioning.

A similar approach is used by Micro Cells as sub zone partitioning in De Vleeschauwer et al.

(2005), which focuses more on the organisation of intentionally small zones to servers. But the authors focus solely on the processing elements of the problem.

Zone Replication

In order to avoid the communications and processing overhead of border effects as the size of individual zones is reached, the use of multiple server nodes to support the same spatial area have been proposed.

Variations of replicating the same spatial areas have been shown in Bharambe et al. (2006) and Müller-Iden (2007), chapter 3.

Especially Glinka et al. (2008) illustrates a wide number of possible mechanisms to handle high avatar counts for specific spatial areas (including sharding/instancing and non Euclidian topologies).

This approach offers a good method to aggregate processing and memory resources to be utilised for a single location with high avatar density, but it still requires a rigid partitioning, which results in border effects as Glinka et al. (2008) also documents.

2.8.2 Tree Based

In GauthierDickey et al. (2005) a N-Tree mechanism is presented that allows nodes in a P2P network to subscribe to groups for event propagation. The tree is iteratively deepened to evenly distribute the avatars among peers. Unfortunately, the publication assumes the avatars to be evenly distributed among the spatial expanse of the virtual environment. In contrast to *QuP*, their approach does not provide for small objects to overlap space borders high up in the N-Tree, thus unfavorable positioning may result in very large areas being selected for event forwarding. Also, a given spatial area may not be maintained by multiple peers to share the processing load.

2.8.3 Distributed Scene Graph

Liu and Bowman (2010) proposes a dynamic partitioning mechanism based on binary space partition (BSP). Processing units can subscribe to events in multiple partitions. In Lake et al. (2010) this is described in more detail. It allows the dynamic assignment of servers to areas based on differing load patterns. the use an OpenSim based model of treating animate and in-animate objects alike

("prims"), some of which having a mechanism to manipulate them, which may be external to the system.

The system a) can handle a high local avatar (or in this case prim) density by splitting space and assigning it to multiple servers, and b) removes communication overhead for cross border visibility, as object managers like action scripts or physics engines can simply subscribe to multiple servers.

Unfortunately, the link to the storage infrastructure and resulting communication for state update distribution may incur hot spots. Their impact can be reduced by occasional reorganisation to ideally utilise resources, and this is described in the above publications. But the authors also mention the danger of initiating reorganisation when and where the load is the highest.

2.8.4 Other Spatial Partitioning

Alternatives to using regular intervals on a Cartesian plane for partitioning have been proposed, with Voronoi being the most prevalent. Among the first this was proposed by Hu and Liao (2004).

Boukerche and Ahmad (2008) proposes a Voronoi based partitioning for use in AoI calculation, and routing of messages based on this partitioning. A more generalised spatial publish subscribe mechanism based on a Voronoi tesselation is shown in Hu (2009).

Hu and Liao (2004) describes a general problem with AoI based filtering in an Voronoi based partioning for small areas of interest when compared to the size of the Voronoi cell, as all messages for all AoI areas within the cell need to be forwarded to the cell (Figure 8).

A disucssion of using hexagon tiles to partition the plane is presented in Shirmohammadi et al. (2008) and Kazem et al. (2007) in it is developed into a system of dynamic and hierarchical subdivisions to allow AoI based filtering to reduce the effect of cross-border traffic and visibility between tiles.

2.9 State Update Distribution

The update distribution shapes the capabilities of the DVE and also dictates the requirements of the network bandwidth needed to support it.

An overview of the basic network topologies has been presented in Funkhouser (1996). Two of

them have been out of favour for a number of years. The first is broadcasting, as it quickly overwhelms the network as the number of participants increases, especially if they are connected via the public Internet. The other is network level multi-cast. This is mostly due to the limited availability in the wide Internet, especially for consumer-level services and in the face of organisational firewall and consumer grade Network Address Translation (NAT).

Some of the methods analysed for multi-cast networks have since been applied to P2P based methods. These usually implement some form of application level multicast.

2.9.1 Hybrid Approaches

Combining multiple approaches to update distribution were analysed to reduce the drawbacks of individual ones.

Chan et al. (2007) proposes Hydra, a P2P backend with a client, which connects to a proxy system as client/server from outside.

This is extended in Quax et al. (2008) and Quax et al. (2009) to separate network management concerns like security, load balancing and failure recovery from the actual game server.

Debeauvais et al. (2011) offers a layered approach abiding to a RESTful design principles, but no implementation to actually test the resultant latencies.

Horn et al. (2009) discusses the use of multiple nodes with separate roles and responsibilities that create communication links as needed by ongoing existence of virtual world.

2.9.2 Prefetching

Prefetching has been used for long time, often to support loading some or all of the visual characteristics of an object with the aim of providing a balance between visual fidelity actually perceived by the user and the amount data transferred between components in a DVE (especially to the graphical client) Lau et al. (2001). In *QuP*, a comparable approach is used to provide world state for processing, controlled by the AoI. A separate prefetching, governed by similar but distinct requirements and control methods, would be required for a graphical client attached to DVE application that operates on *QuP*.

Especially in P2P based DVEs, pre-fetching has been identified very early as a central Requirement. This has been demonstrated by Park et al. (2001) for the inclusion in ATLAS (Lee et al. (2007), a zone based framework that supports P2P, C/S and hybrid communication. ATLAS has been used as basis for various DVE research efforts, e.g. Lee and Lee (2003)).

Model-based predictions for object requests to optimise the control of pre-fetching have been discussed in Chan et al. (2001).

Lu et al. (2006) discusses the prediction of data to be prefectched, and also uses AoI (defined via aura and nimbus) to predict the required objects. In *QuP*, AoI acts as prefetching controller. See section 3.3.3 for details.

2.9.3 Area of Interest Management

Selecting state information and applying change based on the spatial area that the avatar is interested in (i.e. that it can observe) has been a well published method in DVEs for a long time. Initial introduction of the underlying concepts of discerning types of perception are covered in Greenhalgh (1998). Utilizing this information for clusters of servers was described in Funkhouser (1995).

An in-depth comparison of a number of AoI algorithms has been presented in Boulanger (2006), but these are all based on a static, a-priori partitioning of the DVE.

As separation of spatial position within the DVE and AoI is presented in Ahmed and Shirmohammadi (2008), where a zoneless AoI allows the creation of a convex hull over the position of interactive avatars to serve as a overestimating heuristic. But the publication is based on a simulation and does not cover problems of churn and node discovery, which exists in real P2P network.

The AoI relationship of any two avatars in a DVE need not be symmetric. This can either result from a differing view range, or from the handling of the avatars focus by the AoI mechanism of the DVE server (rather than the displaying client.) Ahmed and Shirmohammadi (2012) presents a solution that takes these characteristics into account and uses them to choose a medium of communication to be used.

2.9.4 Interest based aggregation

Reducing the bandwidth requirements of distributing state updates can be achieved by aggregating changes. Bharambe et al. (2008) describes a system that will augment other AoI mechanisms by varying the frequency of updates to a specific client. It achieves this by strictly limiting the number of other avatars that can be observed at full update frequency (5 in this example). Other avatars, even if in the AoI are only updated at a reduced rate.

In Najaran and Krasic (2010) this is simplified for a low-latency FPS game, where only a small set of avatars in the interest set of a player have their state updated at full frequency and the rest of the avatars in the DVE at a reduced frequency. This publication also provides an implementation that has been tested in an actual cloud-based implementation.

2.9.5 Load shedding

With any technique of dynamically assigning workloads to specific servers comes the need to shed this load to be able to handle overload scenarios, where the requirement of any resource (CPU, memory, network bandwidth) exceeds the available maximum. If the workload is partitioned into a number of parts greater than the number processing nodes, workload may be reassigned to a nodes wich has a lighter current load.

In Chen et al. (2005) this topic is addressed for the case of a zone based architecture. But it suffers the same increased communication overhead as avatars moving from one zone (and thus possibly another node) in other approaches of this type. This means the node has the highest communication, and possibly processing, requirements at a time where its load is at its highest.

2.10 Relevant Existing Systems

These are testbeds and commercial systems that have been investigated thoroughly, either due to their research nature (OpenSim) or their commercial success. They will be referenced throughout the thesis, mostly as a baseline to compare other systems and works of research against.

2.10.1 Concurrent Users in Existing DVEs

Measuring the number of concurrent users or even giving an informed guess can be difficult for most commercial products, as the service providers consider this information a trade secret.

According to Kain (2014), World of Warcraft (WoW) had around 10 million subscribers at the end of 2014. They have subsequently stopped publishing the number of subscribers as these started to dwindle. These users are divided among ca 350 shards (see section 2.8.1), called "Realms" in WoW parlance), each in turn divided into 100+ zones that limit interaction across their borders to a certain extent. In some special cases, the zones are actually areas of interaction between the shards (connected cross-realm regions).

Personal observations in actual gameplay lead me to believe that there can only be about 5000 players per realm being active at any one time.

2.10.2 OpenSim

OpenSim is an open source re-implementation of the Second Life (Inc., n.d.) server system, allowing the use of the official Second Life client or any other third party client implementation.

It has not only been used as a basis for analyzing implementation details to show a relevant impact on performance (Bowman et al., 2010), but also as the basis for further studies into DVEs and DVE applications in general Valadares et al. (2015).

2.10.3 Relevant Commercial Products

Commercial MMOGs are the only applications that are large enough to observe asymptotic effects in behaviour in actual implementations. They are also the source of tracking datasets to analyze user behavior.

According to Kesselman (2011), the commercial offerings are usually not the source of much innovation, since the adoption of such new techniques would mean an increase in risk. And due to the investments costs far exceeding 100 million US$ for top-of-the-line productions, business control for the development is very risk averse. More often they strive for incremental improvements by using different mixes of techniques. Also, corporate research and investment is mostly focused on

client experience, as that is most visible to the potential buyers and easily transported in tradeshows and teaser videos.

Game development is a me-too business; technical evolution happens slowly due to risk. Architectural innovation happens elsewhere. The biggest leaps are usually the adoption of techniques already proven elsewhere.

World of Warcraft

One of the largest commercial MMORPG, World-of-Warcraft (WoW), is likely to have provided a large part of the 2.4 billion US$ revenue reported by the business unit responsible for maintaining it (*BlizzardActivision Inc. 2016 Annual Report*, 2016).

Unfortunately, Blizzard has stopped reporting its user numbers when they started to decline from its peak at 11 million users in 2010.

WoW is a game that allows players to explore a fantasy world which is partitioned into a number of zones that can handle around 1000 concurrent players peak load each. These zones run continuously as player enter or leave through connecting to the game or moving to and from other zones. Gameplay mostly focuses around combat against beings in the world (player-versus-environment, PvE) or other Players (player-versus-player, PvP). Upon killing opponents player characters receive goods that can be used as equipment for further combat or sold. Beings in the world hand out tasks to complete, which usually provide rewards in the form in items, money and experience points upon completion.

Special parts of the the world (called "Instances") are separated to be entered by small groups of 5 to 40 players. These are reset for each group entering that part of the world together and promise greater rewards but also higher risks to the health of the character. Upon reset the same monsters and obstacles will wait in the same location each time. Upon killing the monsters, a random reward is given from a specific set of items that has to be distributed between the members of the group.

The player base is segregated among 729 shards as of 2017 (called Realms in WoW, current list: `http://www.wowwiki.com/Realms_list`.), While some exchange between Realms is possible through so called connected zones as well as connected realms and merged realms, these are mainly technical measures for optimising the utilisation of the server hardware the game is deployed on.

Due to its sheer size, and longevity, it has been the source of a wide range of datasets for analyzing user behavior.

Eve Online

In Eve Online, the player base is effectively split into zones as well. But these are not as arbitrary as in WoW, but rather part of the immersion, since each zone represents a complete stellar system. Travel between systems requires a jump gate due to the vast distances of interstellar travel.

In Eve Online the players operate ships (one at a time, but a player can own multiple ones) and perform most functions of a multi-stellar society (e.g. exploration, mining, trading, transport, production and research and development). The game supports the building of player organisations and these have, over time, create basically any form of human organisation, past or present (e.g. clans, companies, conglomerates, countries holding space, military alliances, research groups)

Withing a single star system Eve models movement in a physically correct model with mass, velocity, inertia and acceleration. This can lead to requirements for large computational resources in cases of high avatar density. This is compensated by giving up a real-time reply, but rather introducing time dilation on the simulation, as it is executed on the game server.

As described in CCP Veritas (2011), that can occur at about 1200 avatars in one location. However, it allows the game to handle larger battles in a manner that is acceptable to the users (CCP Inc., 2013).

2.11 Comparison To Other Distributed Systems

A DVE is not the only type of large heterogeneous distributed application. The following overview of a range of such systems is intended to show the requirements and architectural patterns that separate them from DVEs. There is considerable area of overlap in the techniques used. This makes it worthwhile to consider the applicability to DVEs of development in the respective areas of research.

2.11.1 Parallel Discrete Event Simulations

Parallel Discrete Event Simulation (PDES) are described in depth in Fujimoto (2000).

The main difference to DVEs is the strict control of simulation time, as this may deviate from wall-clock time and thus does not provide an interactive feedback. Also, the consistency requirements are higher to ensure deterministic behaviour that may not be as important as in a game type DVE. In order to ensure repeatability of simulations, human users usually cannot provide direct input into a running simulation.

But especially if the simulation is structured as an agent based system which will interact via a common simulated environment, the similarities are sufficient for many techniques to be shared between PDESs and DVEs.

2.11.2 Stream- and Complex Event Processing

Stream Processing (SP) and Complex Event Processing (CEP) are two approaches that are receiving some added interest with the further rise of the Internet of Things. Recent years have seen a lot of research in order to improve the programmability of processing large amounts of events in as short a time as possible, with (soft) real-time results as the goal.

Similar to the user commands in a DVE, SP receives events from a large number of external sources, sensors in parallel. Then further processing elements subscribe to these event streams, processing the information and creating resulting events that are passed on in the pipeline.

Research currently focuses on optimising the detection of events and controlling the subscription to specific event streams. In the past, more work was done on dynamic partitioning of event streams and assigning streams to computing and networking resources (Abadi et al., 2005)

Using CEP as aggregators for user input in social interaction has been tried in Roughton et al. (2014)

What separates SP and CEP from DVEs is the lack of high-bandwidth channel back to event sources. There may be some control of the sensors, but most data is forwarded to separate entities to analyze and observe the information gained from the sensor events.

2.12 Problem Summary

I have documented the long history of research on virtual environments, covering a steady progression to increase the size of the DVEs with regards to the number of participants as well as potential spatial size, support for computing power, network bandwidth and storage capacity. This has made available a large solution space by combining different techniques.

The consequence is a steady progression to highly distributed system, with all the costs and benefits that result from such an fundamental architecture. Most of the difficulties in building such a system have been accepted in general computer science for a long time (Hoogen (2004) or Gilbert and Lynch (2002)). The research history shows the step-wise separation of concerns in order to apply standard solutions to either circumvent or abstract away the problems outlined above. This is exemplified by Najaran et al. (2014).

The available metrics, I will focus on the number of concurrent users as the central metric to be improved, while ensuring the following conditions are met: a) the system will reply in 100 msec or less, b) will provide security through an authoritative entity to maintain state correctness according to the rules of the game, and c) provide a continuous service, maintaining an availability of 99% or better with very little to no planed downtime.

As shown by the research of others, listed in section 2.4 avatars will often congregate on points of interest. In order to support a high number of users, a high local avatar density will have to be supported as well.

Over a long time research was conducted on the optimal method of partitioning the state information of the DVE (see section 2.8), culminating in mechanisms to dynamically partition the DVE and assign them to computing resources that depend on the avatar density instead of network topology or characteristics of the server. As mentioned in section 2.8.3 this may eliminate the threat of overloading a computing instance, but is likely to increase communication overhead when state updates must be forwarded to a number of infinitesimal small spatial area fragments. My work aims to strike a balance between communication overhead and the ability to assign varying amount of processing power to shifting avatar densities.

With the research on managing state update information based upon the area of interest of an

avatar there also have been publications on aggregating update information on the periphery of the area of interest to more closely mimic human perception (section 2.9.4). But this has only been targeted on the dynamic elements of the DVE with the assumptions that all other data is fully available to the clients beforehand. This restricts the type of changes that can be made to the virtual environment, but also eliminates the option aggregate on changes spatially.

The literature review has led me to select the following as an initial design and architectural approach:

Partition game world based on capabilities and perception range Each object is represented multiple times in the octree used to partition the VE. Each capability and each form of perception creates a marker in the lowest tree node that encompasses the sphere of influence by capability or mode of perception.

The state partition is created by navigating upward from the node representing the avatar of the connected client. The complete subtree of the highest node that contains at least one entry will be fetched by the node that contains the client connection.

The rationale here is that all objects that can be affected by any actions of a given avatar or observe any action by another object.

Aggressive Prefetching All readings of object state should be local access on the compute node that handles the client connection, as the objects are prefetched in totality.

prefetched objects are also registered for later update through a publish/subscribe mechanism.

In Lau et al. (2001), a prediction mechanism is presented that aims to predict the movement pattern of avatars in order to derive the visual representation of objects that needed to be pre-fetched by the client. My approach, described from chapter 3 onwards, does not provide a graphical representation of objects, but rather object data as a set of key/value pairs. But is uses an AoI computation as an overestimating heuristic to select a set of objects that are likely to be accessed by a process representing an avatar, and therefore should be pre-fetched.

Quorum based write The transactional behaviour is relaxed from correct to consistent and fair, meaning that two conflicting commands entering the system at the same time (to the granu-

larity of not having reported the result of the first message before the second message enters the system at any node) have an equal chance of succeeding, and that differing nodes reply with a globally consistent result.

The persistence nodes should support a protocol that supports specific operations for manipulating lists or incrementing/decrementing counters to change object attributes without the need for reading data first.

In order to parallelise the processing of messages, incoming messages are forwarded to each observer since the filtering capability is the sensory power of the receiver. Targets of actions are processed by the originating client process, as all information is available to the originator.

2.13 Process of Literature Review

To structure the process of a literature review, I followed a method of collecting seeding publications (Guru Documents) from a range of sources: long-time personal experience with online games, results of naive web search, and hints from research contacts made during the course of my research. In total, these were about 20 publications.

These are the initial entries into a pool of publications. Each reference of these was added to the pool. In addition, the results of the searches in the online library of the ACM, the IEEE and Elsevier for a number of initial nave keywords were added. This ultimately resulted in about 2000 documents.

These were filtered by title, abstract and keywords given by the authors to a total of about 500 papers considered for an initial reading. Each paper that I considered relevant was again added to the publication pool and recursively all references were extracted. From this, several research groups, primary researchers and conferences and workshops were identified. The publications of each of the researchers, their post-graduate students as well as the conference proceedings were systematically added to the queue for triage (relevant/not relevant).

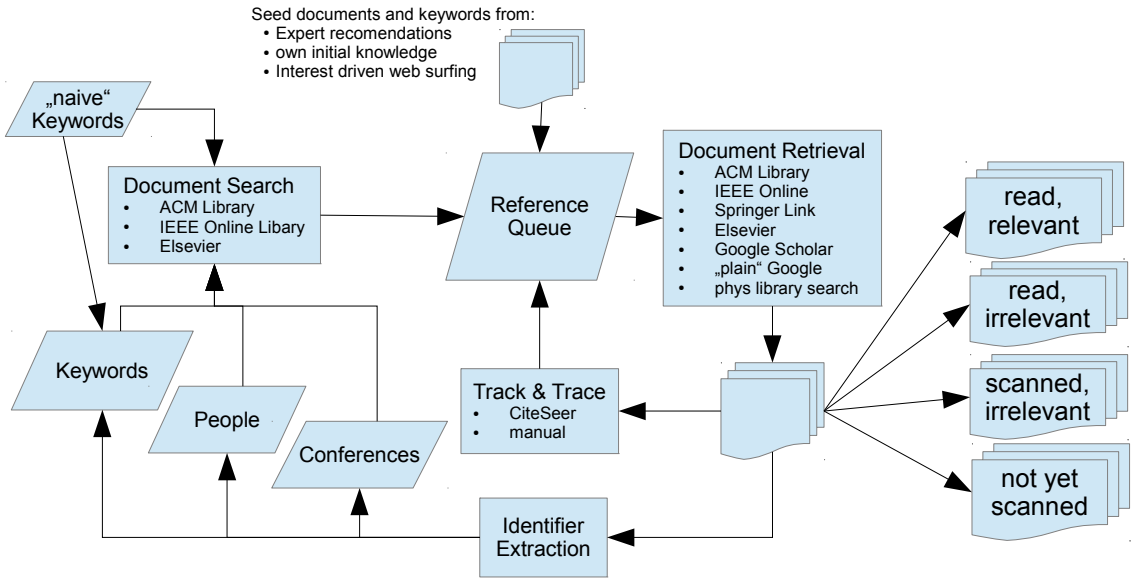


Figure 2.6: Literature Review Process

2.13.1 Research Focus Points

While several areas of research applicable to games are undertaken, only a small subset of this is focused on the backend of distributed systems specific to games and other virtual environments. The following are specific sources that have at least a partial focus on distributed systems as a base for creating a virtual environment.

Conferences and Workshops:

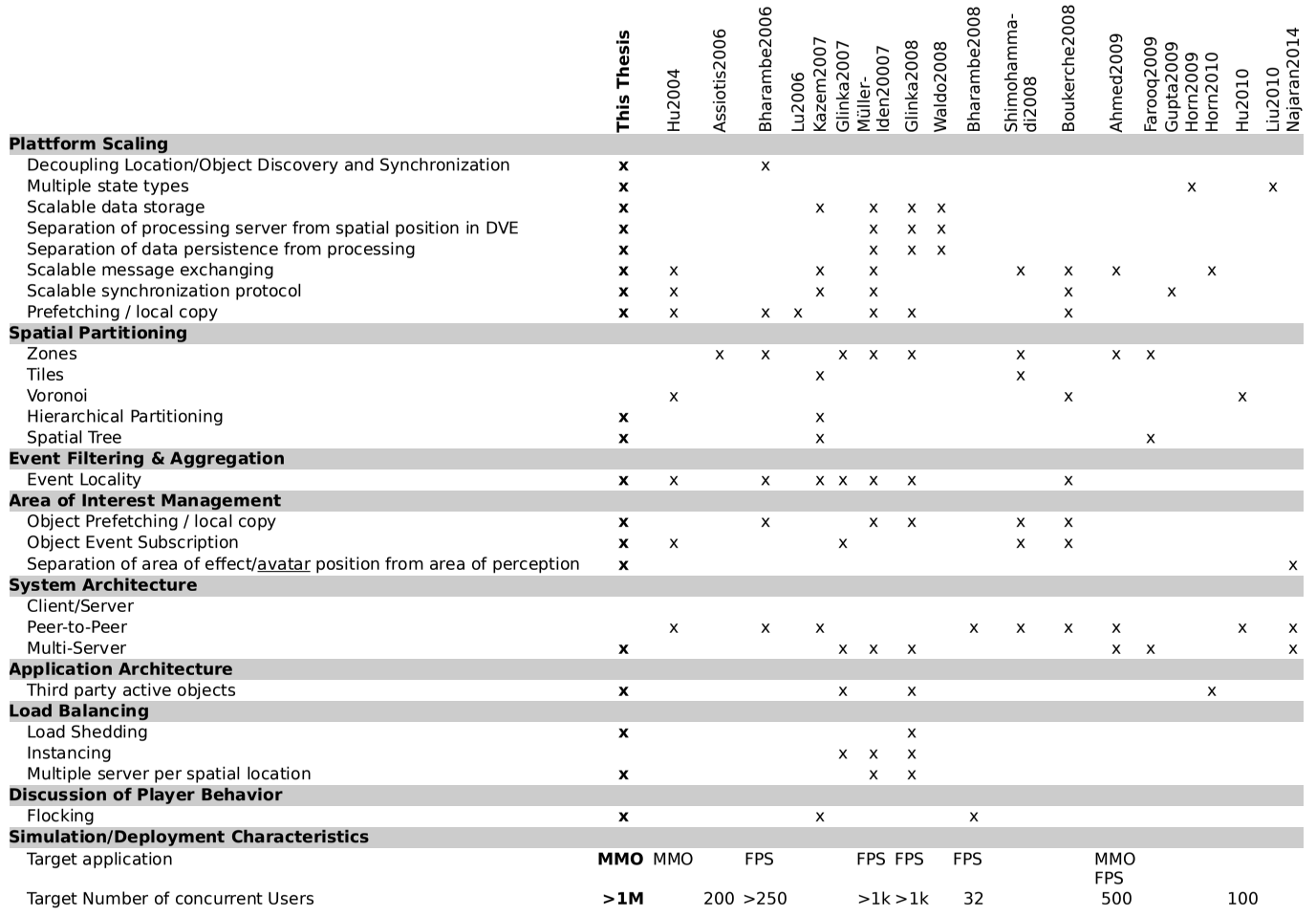
- Workshop for Massive Multiuser Virtual Environments (MMVE)
- Netgames
- ACM Computer in Entertainment
- ACM Int. Conf. on Advances in Computer Entertainment Technology
- IFIP IECE

Journals:

- IEEE Transactions on Parallel and Distributed Systems (Scimago JR: 94)
- IEEE Transactions on Multimedia (Scimago JR: 88)
- International Journal of Computer Games Technology (Scimago JR: 12)

Websites:

- Digital Games Research Association(DiGRA). It is also a conference, but the website holds or references additional material.



	This Thesis	Hu2004	Assiotis2006	Bharambe2006	Lu2006	Kazem2007	Glinka2007	Müller-Iden20007	Glinka2008	Waldo2008	Bharambe2008	Shimohamma-di2008	Boukerche2008	Ahmed2009	Farooq2009	Gupta2009	Horn2009	Horn2010	Hu2010	Liu2010	Najaran2014
Plattform Scaling																					
Decoupling Location/Object Discovery and Synchronization	**x**			x																	
Multiple state types	**x**																x			x	
Scalable data storage	**x**					x		x	x	x											
Separation of processing server from spatial position in DVE	**x**							x	x	x											
Separation of data persistence from processing	**x**							x	x	x											
Scalable message exchanging	**x**	x				x		x				x	x	x			x				
Scalable synchronization protocol	**x**	x				x		x					x			x					
Prefetching / local copy	**x**	x		x	x			x	x				x								
Spatial Partitioning																					
Zones			x	x			x	x	x			x		x	x						
Tiles						x						x									
Voronoi		x											x						x		
Hierarchical Partitioning	**x**					x															
Spatial Tree	**x**					x									x						
Event Filtering & Aggregation																					
Event Locality	**x**	x		x		x	x	x	x				x								
Area of Interest Management																					
Object Prefetching / local copy	**x**			x				x	x			x	x								
Object Event Subscription	**x**	x					x					x	x								
Separation of area of effect/avatar position from area of perception	**x**																				x
System Architecture																					
Client/Server																					
Peer-to-Peer		x		x		x					x	x	x	x					x		x
Multi-Server	**x**						x	x	x					x	x						x
Application Architecture																					
Third party active objects	**x**						x		x									x			
Load Balancing																					
Load Shedding	**x**								x												
Instancing							x	x	x												
Multiple server per spatial location	**x**							x	x												
Discussion of Player Behavior																					
Flocking	**x**					x					x										
Simulation/Deployment Characteristics																					
Target application	**MMO**	MMO		FPS				FPS	FPS		FPS			MMO FPS							
Target Number of concurrent Users	**>1M**		200	>250				>1k	>1k		32			500				100			

Figure 2.7: References Overview

Chapter 3

QuP: Graceful Degradation for State Propagation

As shown by the large body of research examined in section 2.8, various forms of partitioning have been proposed and examined. Many of them require the restriction of interaction between entities to be efficient. Numerous solutions have been proposed to (a) minimise the impact of these limitations, and (b) to handle zone overload and server overload (when an individual zone can handle the load, but the number of loaded zones overload the server).

The following discusses a system called *QuP*, that extends on the concept of spatial publish/subscribe, described initially in Hu (2009), to completely separate the partitioning of computing resources from the partitioning of the state storage, as well as the assignment of the state update processing to the computing partitions. This allows the dynamic (re-)assignment to handle activity hot spots as well as failure of computing partitions.

QuP is the result to a systematic analysis of DVEs with the aim to eliminate any bottlenecks or Single Points of Failure (SPoF) in its architecture. The resultant system should not have any elements that that would hinder scaling to web-scale levels (in excess of 100k concurrent users) while ensuring its suitability for production level availability (> 99.0) and reliability. While the basis of the resulting solutions combines best-of-breed solutions in a novel combination, not only is the end-

to-end nature of the search for scalability bottlenecks a vital contribution (a large part of the prior work focuses on reducing communication latencies), but chapter 4 also adds new techniques to solve problems, that are not sufficiently covered by any of the existing solutions, when operated at the intended scales.

QuP is a mechanism to filter message exchanges within the DVE by using redundant persistence, and an overestimating heuristic, to forward required information to the users software client. In the case of server node failure, the DVE will remain consistent and show only local service degradation, as clients connected to the faulty node will be reconnected to other nodes.

The main advantage of QuP over other approaches is the capability to process the same spatial locale on multiple computing partitions. It can handle situations of very high avatar density (see section 2.3.2). In contrast to replicated zones as described in Müller-Iden (2007), QuP does not use a fixed partition like zones or cells.

QuP implements an AoI based pre-fetch mechanism, but does not rely on pre-computed structures (e.g. as in Boulanger et al. (2006)), thus allowing a dynamic change of the virtual environment. Rather QuP uses an octree (Meagher, 1980) that partitions the space of the VE in a per axis binary tree, thus allowing a direct computation of the storage location from the Cartesian coordinates. The space of the VE is also normalised to Cartesian coordinates of the interval $[0, 1)]$ in three dimensions. This allows positional computations to be expressed as bit-operations on per-axis fixed-point number (the value is the significant of a base 2^{-62} number, see section 3.3.1 for defaults). Objects are tracked as sets of octree nodes marked to contain an object (the actual object data is maintained as a map of key-value pairs identified by a unique number.

Tracking these sets of spatial areas enables the filtering of change. As a result, the users network bandwidth is not flooded. In addition, only information to which the user is entitled is forwarded to the client, as per game rules. This reduces the opportunity for a large subset of forms of manipulation.

QuP was developed as part of the server backend for the project Timadorus (`http://www.timadorus.org`), an effort to design and implement a complete massive multi-player online role-playing game (MMORPG), that provides a number of improvements over existing solutions. Firstly, all players participate in a single instance of a given game. This puts the central focus on the scalability of the system in the face of potentially hundreds of thousands of players. The second

improvement is "four nines" (99.99%) of availability, resulting in a service design around error isolation and restriction of fault propagation to a temporary local service degradation rather than the server failures and weekly maintenance intervals that are common to many MMO offerings today. The final improvement is authoritative management of game state by the server only. Many successful MMO offerings are plagued by an arms race to build and detect and suppress forms of unwanted manipulation (i.e. cheating). Timadorus follows the design principle *"What is possible is permissible, what is not permissible is not possible"* when accepting commands from clients. This requires the verification of every request by clients to ensure that their avatar has the capabilities to execute that action. This design goal of Timadorus will require considerable CPU resources, thus again driving scalability needs.

As a pure research effort, Timadorus does not consider the cost effectiveness of its design, and the needed computing resources to actually implement it on the scale of a successful commercial MMO product. The aim is to show that a system can be scaled to such range while supporting normal performance requirements and the stability mentioned above.

3.1 Definitions

This chapter discusses the exchange of update information for the state of a game world in a distributed virtual environment. The world state is composed of a potentially large ($> 100k$) number of objects that have a unique identifier and an arbitrary number of named attributes with potentially structured values.

Entities are objects that are processed by one or more compute process and can initiate actions. Either due to build-in programm code (e.g. scripting, physics or game AI) or an external connection (e.g. a client program for use by a human or an external AI or real world sensor). An action is expressed as a message which requests the DVE to state the game state. The game rules verify the requested action, possibly resulting in a change of the game state (the state having possibly been changed from the original request).

Human users connect to the server running the game world via clients running on their computers. The client controls a specific entity called an avatar that represents the user in the game. Other

entities may be controlled by local processes or also be controlled via a remote process, but one that is not controlled by a human.

3.2 Related Research

Optimizing the exchange of messages for state change propagation through the use of Areas of Interest (AoI) has been done since Greenhalgh (1998) and Funkhouser (1995)). In the past AoI approaches have focused on reducing the number of messages sent to the client (or among peers in a P2P solution). In contrast, QuP focuses on the elimination of single point of failures and sources of contention within a cluster of game servers to allow for maximum scalability.

Liu and Theodoropoulos (2011) offers an interest matching algorithm that will work on nodes of a cluster individual computers. As they state, most prior work has focused on DVEs with a single server, or at least a server dedicated to interest matching. But their solution does not support resilience to node failure.

GauthierDickey et al. (2005) introduce the use of a hierarchical data structure (generalised as N-Trees) to construct a set of participants that need to be informed of an event. They formulate their approach from the perspective of the effect, rather than the range of perception, but I hold them to be equivalent (more on the Area of Interest representing perception and the Area of Effect representing the scope of possible interaction with objects of the environment are discussed in section 2.3.5). Unfortunately, their publication is a pure theoretical study that neither includes a simulation nor an implementation. They do not address the problem of high level node borders for areas that are not align to the spaces of individual nodes, common to all N-Trees. This problem will occur for the placement of entities (in GauthierDickey et al. (2005) placement of peers is identical to the position of the avatars they control) as well as the scope and placement of the effects that will cause the generation of events to be propagated to the peers. The authors of Backhaus and Krause (2010) present QuOn, a refinement of the protocol. They also conduct a limited simulation, especially to compare their approach to VAST (Hu and Liao, 2004). A more detailed simulation has been undertaken for Liu et al. (2015) showing the generally exceedingly good scalability for both approaches in regard to participant count and environment size, even in the face of high

avatar density. But both QuOn and VAST only support change messages for entities that are directly linked to peers registered with the local node. Any static objects would have to be shared via another protocol or be pre-shared to all peers. The latter would prohibit any fog-of-war style applications, as the peer poses the complete information about the environment. See section 2.6.3 for further discussion.

I present a system which can handle scaling to a very large set of server nodes, and continues to work in the face of server node failure with only local service degradation.

The use of the spatial proximity to control the amount of object-to-object bandwidth shown in Horn et al. (2010) does provide a method of forwarding messages between objects in a manner that does suffer from n^2 communication difficulty, while still allowing a certain degree of global exchange, which is commonly lacking from zone-based solutions. But it does not support an efficient method of distributing state change to many observers. Nor does it exploit the connection of multiple clients to the same server node, beyond the caching of routing information. Horn et al. (2010) also does not enforce game rules except through trusted objects. But as they place emphasis on the fact that their solution can operate in collaborative mode of multiple object implementers, they make implementing a number of scenarios difficult if not impossible. For instance, a simple "Fog of War", in which messages between objects would be mixed with an appropriate form of "noise" to make perception more difficult, could easily be circumvented, as objects can be discovered by spatial closeness. QuP provides similar capabilities, but also allows the manipulation of state exchange between objects according to game rules.

A wide range of research has attempted to utilize P2P architectures to make DVEs more scalable. As covered in section 2.6.3, many P2P systems have a great difficulty maintaining tight security controls on the DVE state and the permissions to effect changes. QuP enables tight security controls by acting as an authoritative server.

The authors of Gupta et al. (2009) point out that visibility may not be sufficient to cover all entities that may interact with each other. QuP allows arbitrary range per object to be applied when computing the interest set. For example, if only some entities are sensitive to radio transmission, and objects may or may not transmit, but all objects are visible, then radio receivers have their range of "vision" set by the range of their radio reception, while all other only by the range of their

visual perception. In the following, optical visibility is used as an example, because it is the standard mechanism in DVEs. QuP can support multiple perception ranges in parallel.

The design of QuP's persistence part is strongly inspired by DeCandia et al. (2007) and Basho Inc. (2013), using consistent hashing to manage the distribution of storage elements between multiple storage backends to support high scalability and fault resilience through redundant copies and eventual consistency. QuP differs from both in that it does not provide a `get()` function, but rather an entity must subscribe to a subset of objects by providing a spatial bound. All states of the contained objects are then copied to the server node that the client controlling this entity is connected to.

QuP has similar aims of being usable in a cloud type environment to Najaran and Krasic (2010), but focuses on slower paced MMO role-playing games (MMORPGs). Not only do these tend to have player bases of similar or larger size than most FPS games, but also these have much larger user bases. More importantly, many aim to bring the maximum number of users together in a single world or at least a system of linked environments. Examples for a single world presented to the users include the megaserver of https://www.elderscrollsonline.info/mega-server or the connected solar systems of https://www.eveonline.com/ described in X (2013).

3.3 Initial Approach

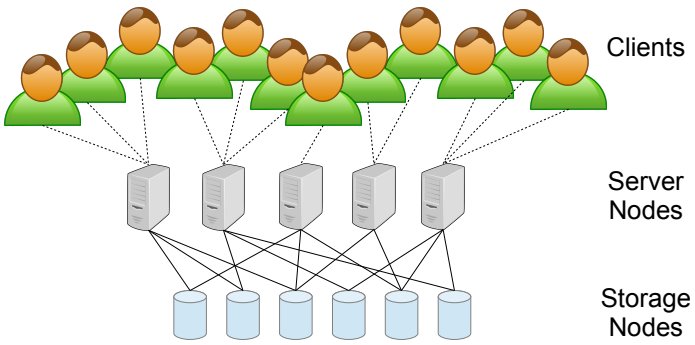


Figure 3.1: Architecture Overview

QuP assumes a specific architecture for a large game service installation that is shown in fig. 3.1.

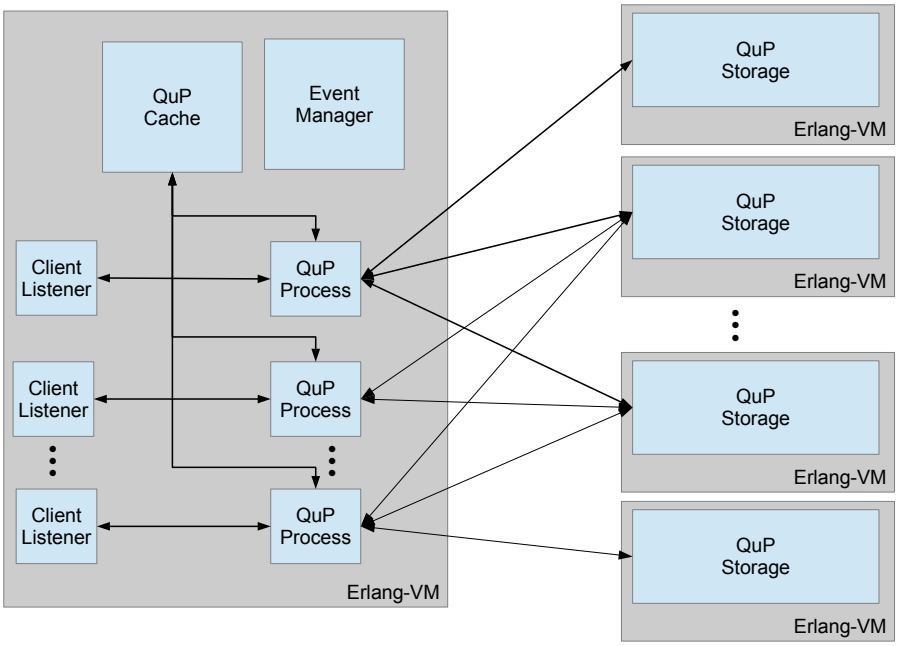


Figure 3.2: Components of Architecture

It is comprised of three types of machines: 1) Client computers, of which there will be roughly one for each user of the DVE. They are likely to be connected via a DSL or similar WAN connection affordable to private users. The individual end-to-end bandwidth is likely to be somewhat limited when compared to LAN technologies like high end Ethernet. 2) Server nodes, which run the actual game code and serve as connection points for the client computers. 3) Storage Nodes. These hold the game state in partitioned, redundant copies. The game server and storage nodes are connected via fully switched Ethernet and have a bandwidth of at least 1 Gb/sec per link. Each node may have more than one link. Network round-trip latencies are less than 1msec for the 90^{th} percentile.

From a very high level overview, QuP behaves as publish/subscribe platform that allows the server nodes to subscribe to state and state updates for objects in the game state. They act as proxy for the game clients that are connected to the servers. An octree is used to allow the identification of other objects in spatial proximity and the subscription to their updates as well. Additionally, the game state is structured to avoid the need for global information and the ability to handle hardware and software faults through redundant data.

Compared to others designs, I focus on eliminating all single point of failures and aim to minimise sources of contention. While this may ignore the overall computational cost of running the server, it allows scaling the number of concurrent participants to a degree that lies far beyond other approaches.

In RING (Funkhouser, 1995) and many derived works, messages are routed among the server nodes and then forwarded to the clients. State is mainly held in the client. QuP holds the authoritative copy of the state, to be queried by the clients. While clients *may* hold partial world state, e.g. in order to improve their performance or user experience, they don't strictly need to do so.

In contrast to Riak and Dynamo, QuP does not implement a synchronous `get()` primitive, for reading a value from storage. But rather state is requested from the QuP client process, and an asynchronous message is sent to the game client (or rather the listener process, that acts as the local proxy of the game client connected over the internet.) Whether the lookup of the data was hit in the cache or not is transparent to the listener process.

In order to change the state, a listener process would send a request to update a value to the QuP client. This request will be forwarded to the storage servers, which are indicated by the consistent hashing (QuP uses SHA1 as does Riak). Provided the request does not collide with another update, the changed value is stored, and update messages are sent to all subscribers of the object (including the writing process, which will be auto-subscribed, if need be.) There it will update the local caches (one per node) and forward the message to the client listener that initiated it and all other listeners that are locally subscribed to that object.

QuP's design is heavily based on the assumption that the majority of all state information within the DVE is static or experiences little change. For example, Horn et al. (2010) showed that in SecondLife, only 8% of all objects are mobile. QuP exploits this by pre-loading all state information within the visibility envelope of an avatar. Thus, all state required for updates in reaction to a command message sent by a client is already available to the server node that the client is connected to. This state change is then propagated concurrently to all other server nodes that share a spatial proximity.

QuPs design is capable of connecting external code as described in Horn et al. (2010), which they state as one vital design benefit of their solution. QuP only manages the state of the game world,

not the rules on how to manipulate that state. As the change requests and notification of failure or success are sent as messages throughout the system, they can easily be forwarded to an external program via a publicly defined protocol, like HTTP/JSON or similar (possibly after some filtering to enforce access permissions or similar restrictions).

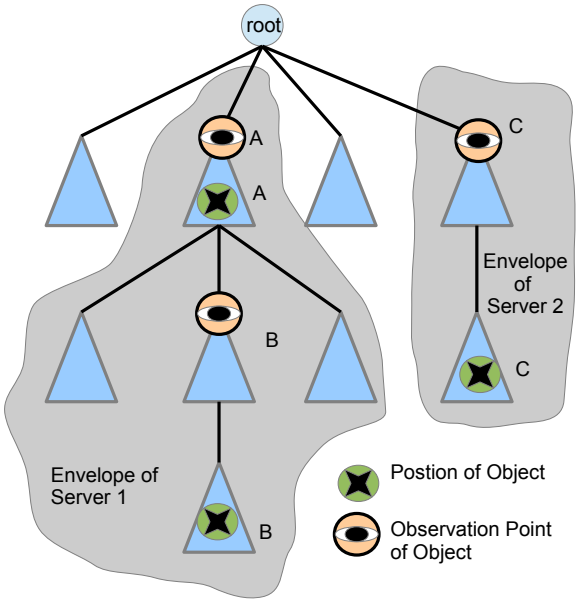


Figure 3.3: Example of fetch envelopes. Clients A and B are connected to server 1, client C is connected to server 2

The subset of world state loaded onto a server node is dynamically determined by the position and size of the visibility envelope of the avatar (see fig. 3.3). In an unloaded server with no initial client connections, the subtree of octree that models the world space will need to be fetched from the storage servers once a client connects. Depending on the structure of the DVE and position of the view box of the avatar, a naive calculation of this envelope may contain the root node of the the octree, i.e. the whole virtual world. See section 3.3.6 and chapter 4 for further discussion to avoid this necessity.

As additional clients connect, only the difference between the subtree of client A and client B need to be read from the server. This allows fast handling of connection of new clients to a server node, provided that the avatar of the client is positioned in a similar spatial region as most of the

other avatars connected to the server. But as (re-)connection is cheap, clients may easily be moved to the server most optimal for the partitioning of clients among the game world.

QuP does not take into account view frustum nor culling nor any other directed AoI approaches. I assume that by rotating, the avatar may quickly change its set of visible objects. As QuP is intended as an overestimating heuristic, objects are prefetched to a certain envelope as described in Ahmed and Shirmohammadi (2008).

Furthermore, QuP only filters the intra-server traffic. Additional reduction of the data sent to the clients may be done by the actual game code that runs on the game server. A possible method for this is discussed in chapter 5.

QuP uses an octree because positions in it can be calculated without any pre-processing or the need to have access to global information. This is in contrast to other methods like BSP-Trees as proposed in Rhalibi and Merabti (2006).

3.3.1 Structure

QuP maintains two sets of objects, each with an arbitrary set of named attributes, containing Erlang terms as values. Each object is identified by a sequence of byte: a) the game state objects form the content of the game state, and b) the octree nodes. They have only two relevant attributes: the list of objects contained in this node and the list of objects that can observe the contents of the subtree with this node as the root. The system maintains these lists automatically.

The identifier n of an octree node is computed from the Cartesian coordinates of a point $p = (x, y, z,)$ in VE of the dimensions $U = (X, Y, Z, D)$ by concatenating the bitstring result the following steps for each axis a, followed by D as an 8 bit integer (see section 4.1 for further information on improved axis alignment):

$$w_a = p_a / U_a \qquad \text{normalise to norm-cube}$$
$$n_a = floor(w_a * (1 << D)) \qquad \text{into an integer of max D length}$$

Objects of both types can be subscribed to and will be copied to a server node initially as well as when the objects change. These transfers are done automatically, without need for intervention by the application author.

An object is contained in an octree node when there is no smaller node (i.e. further down the tree) which will fully encompass the spatial location and size of the object. The visibility envelope forms a type of pseudo-objects, thus is registered in the containing octree node just as well. All positions and sizes are normalised to be between 0 and 1.0 in order to make the implementation independent from the actual size and resolution of the game world. The maximum depth of the tree is limited to a fixed to an arbitrary depth (see section 4.2.1 for discussion on depth in implementation).

The octree nodes are only created if needed, thus saving large amounts of storage (at depth of 64, the tree would consist of 2^{193} nodes, thus about 10^{60} elements when fully instantiated).

3.3.2 Server Selection

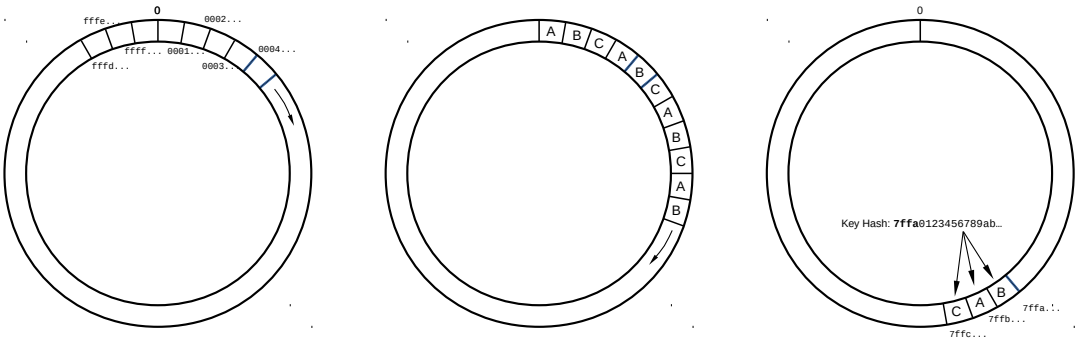
Objects and octree nodes alike are stored in a number of copies in the storage nodes. Objects and tree nodes each have a unique identifier: an arbitrary globally unique 128 bit identifier for objects and a 200 bit positional code for the tree nodes. These identifiers are hashed using a cryptographic hash (the prototype uses SHA1).

The resultant hashes identify a position on a ring, which is split into a number of fixed length segments called chunks (fig. 3.4a), identified by a prefix of the hashed object keys they hold. The length of this prefix is set globally. Initially these chunks are assigned to the servers in a round robin manner fig. 3.4b. During operation, chunks will get reassigned when recovering from node failure, new nodes are added or chunks are redistributed to spread peak load. A gossip protocol is used to distributed the node-to-server mapping to all clients and all servers. A separate control node acts as a seed node for the gossip protocol as well as the assignment of the initial order of server nodes for the chunk distribution: server nodes register with the control node and get a sequence number in the order of registration.

To write an object with replication n, the chunk that matches the prefix of the key hash and the n-1 following ones are selected. The servers that currently contain these are retrieved from the mapping table and the data is written to each. A write is considered complete if m servers ($m \leq n$) have replied.

Reading uses the same mechanism of r consecutive chunks being read from ($r \leq n$). If the replies to a read operation returns a quorum of $(r/2) + 1$ identical values the read operation is considered

successful.



(a) Hash Ring (b) Chunk Assignment (c) Write Replication

Figure 3.4: Consistent Hashing and Server Assignment

QuP utilises a variant of the fixed length chunks of multiple virtual storage nodes as described in DeCandia et al. (2007). This publication also covers Eventual Consistency by using values for m and r that are less then n. The prototype of QuP uses the same value for all three, to avoid the need for a merge function, as that is highly application specific and beyond the scope of this research.

3.3.3 AoI Algorithm

QuP will automatically detect changes in all the objects within the observation aura of an avatar. This is accomplished by the following process:

1. For each object in environment, find octree node large enough to fully contain bounding box for an object.
2. Find analog node to fully encompass the aura.
3. Subscribe to the Aura Subscription Set (ASS), consisting of each octree node in the subtree beneath the aura node.
4. Subscribe to the set of objects listed in each of the contained lists of the ASS.

As a subscription always is node based, an implementation of QuP must track successful subscriptions to minimise network traffic. In QuPerl this is done by the subscription manager subsystem.

If an object update event is received by game server node, the observing object(s) can be identified by selecting all entities avatars whose ASS contains the object node. The event will be forwarded to all observers.

3.3.4 Consistency Model

QuP envisions an architecture similar to the one presented Gupta et al. (2009): clients maintain a local (usually partial) state of the environment and optimistically change this state according to a) the user input and b) the updates sent by the server. Any change due to user input is framed as a change request (analog to the action described in Gupta et al. (2009)). Upon receiving the change, the server will reply with a new state of the object in question. The new state may be a) the state requested by the client, b) a changed state as a result of the game rules that is computed from the old state and the requested state, or c) the state set by a previous request of another client, which has not been taken into account by the request of the first client. It is the task of the client to present the new state in a suitable form to the user.

QuP does not try to be correct when compared to distributed systems that support applications like stock trading or managing shopping baskets (DeCandia et al., 2007). It relaxes several constraints specifically to provide a behaviour of least surprise when utilised in DVE applications.

QuP is not fair. The order of messages arriving at the server is not necessarily the order in which the messages are executed (i.e. the resulting state changes are performed). For conflicting changes, one is arbitrarily chosen to succeed (the one whose change request is processed first); the others will simply fail. Thus in theory, a faster client could can successfully change the state of an object so in such quick succession, that all requests for change by a slower client will fail as it can never present the current value to the server.

But in a DVE, this situation should only arise if two clients are trying to manipulate the same object. It is my assumption that most state changes are to the avatar and the objects in its possession. These usually are not competitive in nature (only one client will control one or more avatars). To further reduce the problem, QuP provides dedicated change-request operations (add-to-set, remove-from-set, increment counter) in order to support applications implementing higher level synchronisation protocols.

Also, it is my assumption is that most often competitive access is not directly detectable by the users. E.g. if two or more avatars try to grasp a unique item, the order of completion of the processing of the request to do so will control which avatar (and thus user) will get the item. While the users can see that multiple avatars have grasped for the item, they cannot perceive when each of them started to do so. Only the end-result of one of the holding the item is visible. Should the starting time be visible to users (e.g. all are members of a group and the item became available to all of them, when some pre-condition was met), then a common technique should be used: actually taking control of the item takes some time and is shown to the users as some visible effect. The computer randomly selects one avatar from the pool of candidates, ideally at the beginning of the effect, but only reveals this information to the client, and hence to the user, once the time of the effect has lapsed. But as all of this highly dependent on the type and design of the application that is running on the QuP system, this is part of game/application design and specifically beyond the scope of my work.

Strong Eventual Consistency

In theory, QuP could be configured to provide strong eventual consistency (Shapiro et al., 2011). Given suitable settings to the generation of quorum, QuP could behave similar to Dynamo (DeCandia et al., 2007) and be able to provide partition tolerance in addition to consistency and availability.

However, this would require the definition of a merge function to reconcile deviated state. I have chosen not to provide this capability as the chosen level of abstraction of QuP (generic named objects that hold an arbitrary set of key-value attributes) does not lend itself to a generic merge function applicable to all types of objects within every DVE application. Any generic function may violate the principle of least surprise intended for the system.

Multi-Object Transactions

QuP does not natively support multi-object transactions common to most Relational Database Management Systems (RDBMS). This is a intentional design decision, rather than an oversight or chosen as a performance optimisation. The decision was made, as the majority of changes to the state in an DVE only involve a single object (e.g. movement, observing state) or are not dependent

on each other (e.g. taking damage from an attack in an MMO): while multiple avatars may interact, the transactions don't need atomic change of all of them, in contrast to e.g. an money transfer between banking accounts.

In addition, DeCandia et al. (2007) describes the need for application level reconciliation when providing eventual consistency. In the case of a DVE, this may either mean recompensation in a business sense by the service provider (e.g. free game time for perceived victim of a transaction failure) or the inclusion of the form of mechanic into the virtual environment that eliminates the need for multi-object transactions (e.g. the trading window in WoW, which is basically placing the objects to be traded into escrow, to be released only once both parties have agreed to the trade.) Some preliminary work in Klauß (2015) has shown that all technical solutions incur some restrictions that limit the design space.

3.3.5 High Load

QuP supports the processing of the same spatial segment of the DVE on more than one game server. But in contrast to P2P style architectures, the number of servers is very small compared to the number of avatars served by each server. As each game server only needs a single update event for each change, the amount of network traffic between the storage nodes to the game servers is reduced.

The number of server per spatial segment can be increased dynamically. As the size of the segment is governed only by the set of avatars that are connected to a game server, reducing the size of that segment is an optimisation that is provided by optimally assigning the clients to the game servers. Details on how this can be achieved are covered in chapter 6

3.3.6 Limits and Weaknesses

The initial approach has a number of inherent problems and limitations that needed to be addressed. They were given only minimal treatment in the initial publication.

Selected Node Level

For the view box to be rendered by the client application, a level within the octree must be selected, in order to notify the client of every object within this view box.

But the objects that are larger than the view box may also need to be rendered as they are visible to the avatar and thus require to be selected. The problem of identifying *neighbouring* objects is ignored as an implementation concern between the objects selected by the view-box on the game server and the objects in the frustrum actually being rendered by the client application.

Initially I looked at two forms to view this: Either the object *is* the surrounding space or at least larger than the view box. Whether it is visible is a game design issue. But all objects from the root of the octree to the contents of the view box must be selected and filtered for display either by the game logic on the game server of by the client application.

Alternatively I considered treating any object placed higher than the view box as an error to be addressed in the game code. But this will quickly lead to very large view boxes, that pose problems of their own (see section 3.3.6 for further discussion).

I have chosen to handle this by selecting any object, as long as it lies between the root to the octree and the selection depth. See section 4.2 on page 87 for more.

Very Large View Boxes

One of the design goals of QuP was the support of arbitrary sized view boxes. But if an VE application that uses QuP contains entities that allow view boxes close to omniscience (concurrent perception of all events/state changes within the environment), the resultant rate of messages would quickly overwhelm the bandwidth of the network connection to the client consuming these messages.

To a certain extent this is exactly the problem that AoI is intended to solve. It partitions the stream of update message in such a way, that only the relevant messages are transmitted to interested parties, e.g. a given client. But as the DVE is increased in scale and scope, the message rate will continue to climb. Proposals for dynamically adapting the size of the AoI dependent on the message rate have been proposed as far back as de Oliveira and Georganas (2002). But this publication also discusses the *Degree of Blindness* (de Oliveira and Georganas, 2002, sec. 3.2.4) as an impact on the entity vision, and thus the user experience, when technical limitations reasons reduce the space

covered by the AoI too much. The fundamental reason for this is the fact that adapting the size of the AoI will not consider the relevance of an object, but only its placement. Size may have an impact only if it causes the object to overlap the AoI at least partially. But this only applies if the chosen AoI mechanism includes partial objects.

QuP approaches this problem from a single assumption: the consumer of update information (regardless whether a human player or software component controlling an NPC) has a limit in the rate of information it can consume. For this, it is irrelevant whether this limit is created by the available network bandwidth, the display fidelity, or the amount of situational comprehension available to actually understand the meaning of the updates.

From this assumption I have explored two methods of solving this problem: aggregation, discussed in detail in chapter 5, and limiting the size difference between the largest and smallest object, introduced in section 3.3.6 and explored in more depth in section 4.2

Elephant and Ant Restrictions

Given that the objects identified by the QuP mechanism are likely to be displayed in a graphical user client, there is a physical limit between the largest and smallest object shown. The actual screen sizes and the size of a single pixel present the upper and lower bounds respectively. In an actual client, the lower bound are likely to be somewhat higher, since single pixel objects are almost impossible to identify by most users.

The obvious solution is to limit the size difference between the largest and smallest object to be shown. For small objects this is simple. If they cannot be displayed, or more likely, be perceived, I assume that simply ignoring them, will not change the behavior of the DVE application.

In QuP this will reduce the size of the ASS by setting two parameters that computing effort spent on determining the set of visible objects. The first is the split depth, controlling the *fidelity* of the selected volume when converting from an AABB to a set of octree nodes. This is a tradeoff between computing resources and time on one side and the size of the selected object set on the other. The mechanism is described in section 4.1. The other setting is the selection depth. It controls the number of levels in the octree downward. From the lowest (and thus smallest) node of the octree set selected by the split depth, the search should go looking for more (smaller) objects.

3.4 Prototype

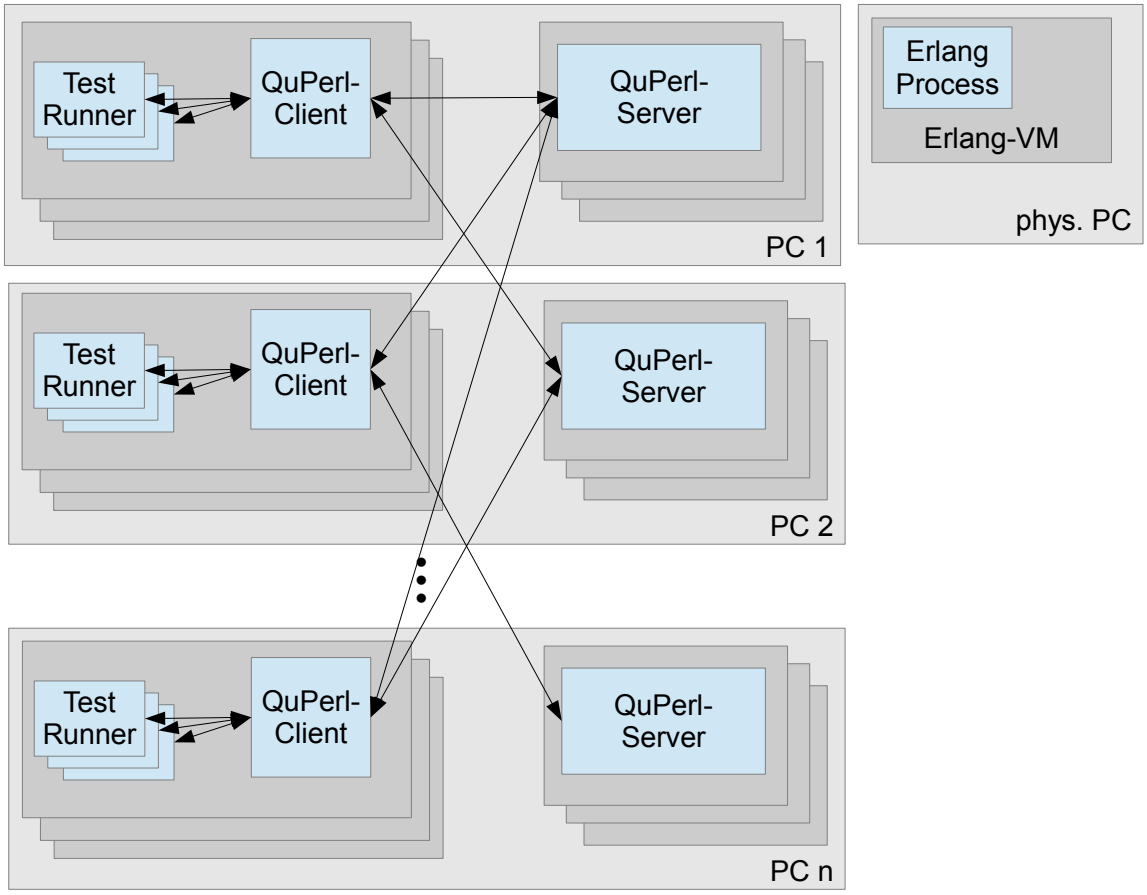


Figure 3.5: Experiment Setup

In order to verify our approach, I have implemented a prototype in Erlang. Erlang was chosen to reduce the effort to send messages over the network, and in order to reduce the complexity of writing software that handles a large number of threads of execution in a concurrent manner.

One vital drawback of Erlang, the much lower speed of execution, when compared to well written C/C++ code is simply ignored, as only the scalability, not the single user performance, is currently focused on by QuP.

Currently the project has only limited hardware resources, thus the test were done on five standard desktop PCs (twin core, 2 to 4 Gig RAM each). But the use of Erlang allows starting several

virtual machines to show how the message latency for state updates increased with added connected clients, distributed over a growing number of server cluster components.

As shown in fig. 3.5, each physical machine runs a number of server nodes as well as a number of storage nodes, each in separate virtual machines. On each server node, a number of load generator simulate connected clients by generating a movement update every $500ms$. Through appropriate distribution of load generators between the server nodes, and the use of consistent hashing, 80% of all message will have to be transmitted over the LAN, instead of being received locally on the physical machine.

A fixed number of load 100 generators was started on an increasing number of virtual machines in order to establish the overhead of setting up the octree and copying the object to the individual VM. Each load generator would perform 50 steps and then terminate.

The client assignment of the prototype (fixed number assigned directly with each additional server node) was chosen as a simplified representation of an increase in server nodes to handle more and more clients connecting. As the location of the clients within the VE is determined randomly with a uniform distribution, I assume that this is also a valid representation of connecting clients being assigned to a set of pre-started server nodes using either a round-robbin or least-loaded client assignment mechanism. Any of these are far from optimal and chapter 6 discusses a mechanism of optimising the client assignment to make use of possible optimisations that QuP offers.

3.4.1 Hash Ring Partitioning

I have opted to use strategy 3 for QuP: *Q/S partitions per node, equally sized partitions.* This splits the name space of the hash values into Q equally sized partitions. This not only reduces the necessary meta data (mainly the start points of the chunks) that needs to be kept track of, but also all start values for the chunks can be computed once during system startup and stay the same during the lifetime of the setup. For a number of S storage nodes, each node is assigned (Q/S) chunks on average.

The partitioning and distribution strategy above can fail if $0 < (QmodS) < N$, N being the replication rate. In this case, the wrap from highest to lowest value in the hash ring may generate a sequence of N nodes that contains less than N unique node ids. My implementation simply chooses

as many further nodes as necessary to produce N unique node Ids.

3.4.2 Results

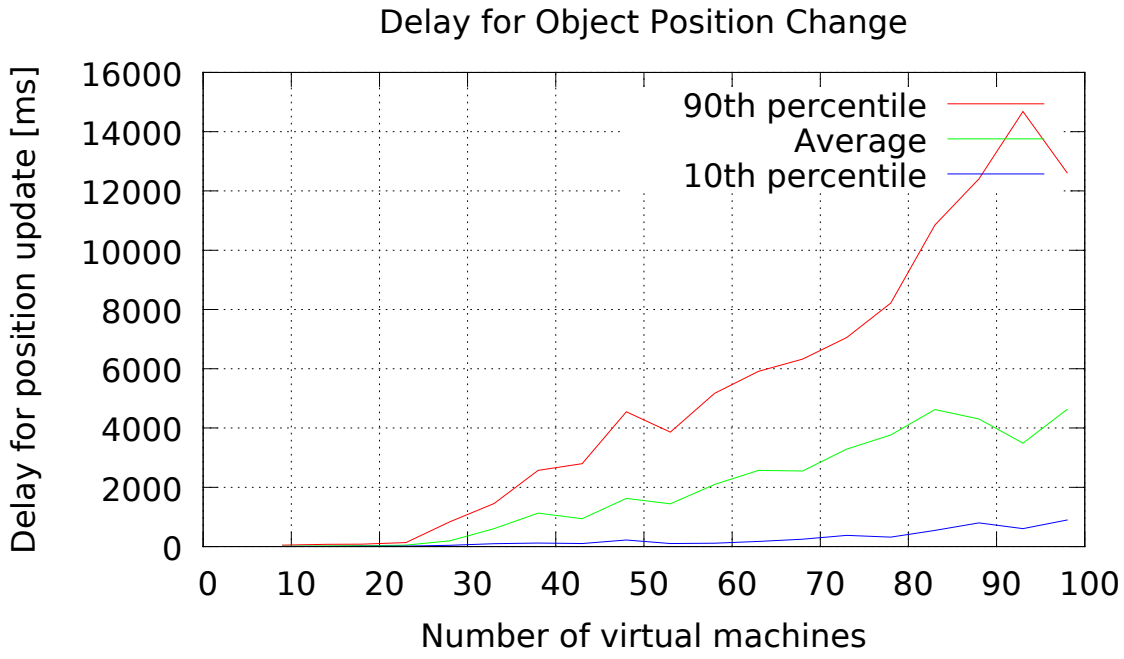


Figure 3.6: Update Delays

As shown in fig. 3.6, the average latency, while increasing due to the fixed hardware resources, increases dramatically less than the n^2 for a full distribution.

The amount of latency, especially for the $90th$ percentile shown by this is extremely high. With values of up to 16 seconds, this is more than two orders of magnitude greater than the permissible 100msec initially planned. The partition of work among 100 virtual nodes on a fixed hardware would account for an increase in reply times due to the increased overhead and increase number of clients.

But it would not explain the roughly one order of magnitude between the slowest and fastest reply times. This would require further investigation.

3.5 Summary

The first attempt at QuP showed that almost arbitrary partitioning with high server counts were possible and would require much less communication overhead than n^2, but could not provide the required100msec reply at higher server node counts.

A question by Mahdi Tayarani Najaran, one of the authors of Najaran et al. (2014) sent me to investigate the volume of octree subscribed to, and as a result, the number of covered objects subscribed to as well, as a result of the position of the bounding box of the view area. This quickly led to the realization that the naive selection shown in this chapter was insufficient to provide acceptable performance. The steps taken to remedy this problem are covered in the next chapter.

Chapter 4

Improved Handling of Unaligned Bounding Boxes

This is an extension of the basic QuP mechanisms published in Behnke et al. (2014). It addresses one of the problems that were discovered in the initial implementation.

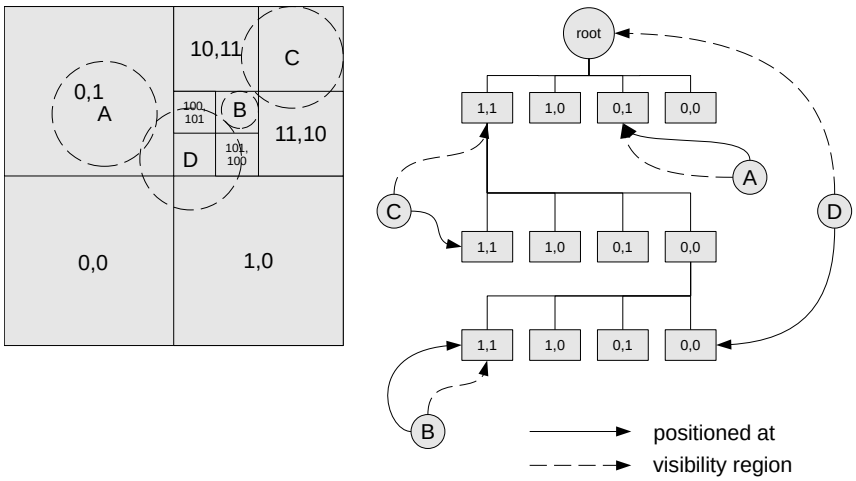


Figure 4.1: Primitive quadtree node selection

The view box of each avatar is described as an axis aligned bounding box (AABB), spanned by two arbitrary points within the world cube of the VE. As AABBs are usually not identical to a

single octree node, the first design of QuP, described in Behnke et al. (2014) traversed the octree upwards to find a node that would completely envelop the AABB. While very fast to compute, it is likely to select a node much further up the octree, holding a much greater volume than that of the actual AABB, if the points spanning it are on either side of a higher level divider (compare the node for the position of object D vs. the node for the view box in fig. 4.1).

This will result in an extreme expansion of the selected space. While the result was correct, since QuP is intended as an over-estimating heuristic of perceived objects, it would not deliver the required performance. This can be seen in the difference of reply time of multiple orders of magnitude shown in figure fig. 3.6. It will also quickly run afoul of the limit stated in section 3.3.6.

To reduce the volume of the octree chosen for object selection, the AABB is transformed into a View Box selection Set (VBSet) , a set of octree nodes that ideally exactly match the volume of the view box, using the algorithm in section 4.1. This set is comparable to the selected nodes of a classical octree for space partitioning of arbitrary objects or point cloud data, but positioned within the global tree of the VE. But in QuP, it is assumed to have a bound volume that is multiple orders of magnitude smaller than the volume proscribed by the whole tree in most cases.

Further processing of QuP will take place on the VBSet and consider each element as a separate subtree as described in fig. 3.3. The VBSet is also limited in its generation depth to allow strict control of the time and resource requirements. Due to the exponential nature of the algorithm, handling the set would quickly render its benefits void.

In addition to the view box, an AABB encompassing the Aura of the avatar (Ahmed and Shirmohammadi, 2008), QuP also tracks the object bounding box, an AABB which encompasses the simulated physical extension of the object. An object is considered visible to the avatar, if the avatars view box and the objects bouding box intersect. From the two $3D$ points identifying the bounding box the Bounding Box Node Set (BBNSet) is computed, adding each element of the BBNSet to the appropriate octree node that it resides in. Again, to ensure the over-estimation heuristic behaviour of the whole protocol, all new entries will be inserted prior to removing the old to changing the placement of an object.

Implementing unaligned bounding boxes will result in the following refined AoI algorithm (from section 3.3.3):

1. For each object in virtual world, compute its BBNSet. Mark each node in this set as containing the object. These nodes will be of differing levels within the octree of the full virtual environment.

   The BBNSet will be re-computed each time an object is moved. Nodes that do not overlap anymore with the object after the movement will be removed.

2. Compute the VBSet for the aura. This is re-computed every time an entity changes its view box.

3. Subscribe to the Aura Subscription Set (ASS), consisting of (a) each tree node within the VBSet of the aura, (b) each node below those of in the BBNSet, and (c) each node to the root from those in (a). The resultant list will not be the full subtree, as nodes that have not previously been instantiated, by either setting any attribute, or any child being instantiated, are excluded. In section 4.2, further restrictions are discussed.

4. compute the Contained Object Set (COS), consisting of each object contained in each node of the ASS. An object is considered contained if at least one node of the ASS is marked as holding the object. Subscribe to each object in the COS.

4.1 Convert AABB to Octree Node Set

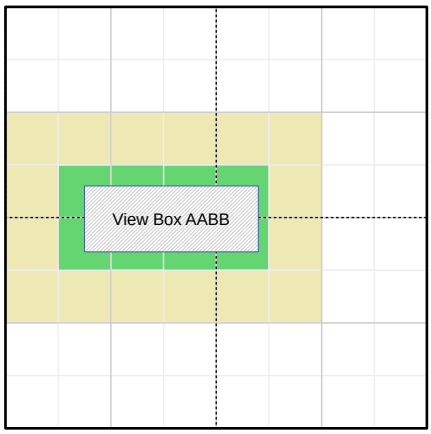
This is the algorithm to partition volumes spanned by two points (P1, P2) into sets of octree nodes with arbitrary precision. In addition to two Cartesian points of three values $0 \leq v < 1$ each, it takes a maximum tree depth D and a maximum split depth S, two non-negative integers as arguments. By assigning the axis to correspond to the numbering of the child nodes (as described by section 7.3 in Ericson (2005)), a node id can be computed directly from the 3D Cartesian value:

$c_a = floor(v_a * 2^D)$ with D the maximum depth, $a \epsilon (x, y, z)$

In combination with the depth D, this will result in a unique integer code for the octree node that contains the point, with a result of $1/2^D$ on each axis.

Now the points are normalised, so that P1 holds the lower value for each dimension, P2 the greater value respectively).

4.1.1 Calculating Node Set



Split level	VBSet	Precision
0	[]	256 / 16.5 = 15.52
1	[1],[2],[3],[4]	4x64 / 16.5 = 15.52
2	[1,3],[1,4],[2,3],[3,1], [3,2],[4,1],	6x16 / 16.5 = 5.82
3	[1,3,4],[1,4,3],[1,4,4], [2,3,3],[3,1,2],[3,2,1], [3,2,2],[4,1,1]	8x4 / 16.5 = 1.94

View Box: 6.6x2.5 = 16.5 units
Virtual Environment: 16x16 = 256 units

Figure 4.2: Splitting selected nodes to improve fit to AABB

To map from objects to octree node sets, the naive implementation of QuP selected the lowest node that an AABB was completely encompassed by. If the AABB was overlapping the border of two or more higher nodes, this could result in selecting volumes that are orders of magnitude larger than the AABB. Using algorithm 1 this is optimised by recursively considering only the part of the AABB that intersects with a given node and selecting only those child nodes that intersect themselves. Figure 4.2 illustrates this for a quad-tree

4.1.2 Algorithm validation

To validate the algorithm and determine its performance characteristics, I have written a test to select 47.000 pairs of points, each within the norm cube. The box defined by these two points is then approximated by a set of octree nodes for maximum split depths 0-10 each.

4.1.3 Choosing Maximum Split Depth

As shown in fig. 4.3a increases the required time to determine the bounding set of nodes for an AABB on average exponentially with the maximum split depth. I define the precision of the selection as the volume enclosed by the selected set of octree nodes, divided by the volume of the AABB. As fig. 4.3b shows, the precision of the selection will asymptotically approach 1.

Algorithm 1: Partition bounding box into node set

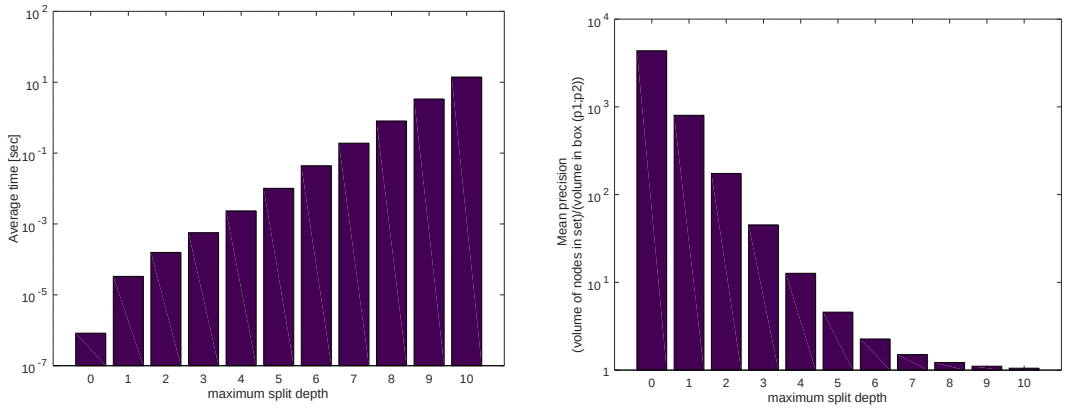
```
Function partition(workList: list of {p1: point, p2: point}, depth, maxSplitDepth: int,
 maxDepth: int): list of point →
begin
    foreach box ∈ workList do
        {p1, p2 } ← box
        if maxDepth = 0 then
            result ← list_add(result, make_parent(p1, depth))
            continue
        end
        if maxSplitDepth = 0 then
            result ← list_add(result, make_parent(p1, depth))
            continue
        end
        if get_node_at(p1, depth) equals get_node_at(p1, depth) then
            result ← list_add(result, partition([{p1, p2 }], depth +1 , maxSplitDepth,
              maxDepth- 1))
            continue
        end
        else
            if no_bits_on_below(p1, depth- 1) and all_bits_on_below(p2, depth- 1) then
                /* parent completely covers sub volume                          */
                result ← list_add(result, make_parent( p1, depth- 1))
                continue
            end
            else
                /* consider the split nodes for further partitioning            */
                List = split(P1, P2, Depth)
                result ← list_add(result, partition(List, Depth + 1, MaxSplitDepth - 1,
                  MaxDepth - 1))
                continue
            end
        end
    end
end
Function make_parent(n1: node, depth: integer): node) →
begin
    return the parent of n1 at given depth
end
```

Algorithm 2: Compute included sub-nodes from partial AABB

Function `split` *({p1: point, p2: point}, depth: int): OutSet: array of {p1, p2 } →*
- n1 ← `node_code(p1, depth)`
- n2 ← `node_code(p2, depth)`
- (nodes, splitIdx) ← `lookup_children(n1, n2)`
- nodeCount ← length(nodes)
- **for** *i: 0 to* nodeCount **do**
  - masks ← `make_masks(i, splitIdx, depth)`
  - **for** dim $\in \{x, y, z\}$ **do**
    - $\text{outBox}_{\text{p1}_{\text{dim}}} \leftarrow \text{p1}_{\text{dim}}$ & $\text{masks}_{1_{\text{dim}}}$
    - $\text{outBox}_{\text{p2}_{\text{dim}}} \leftarrow \text{p2}_{\text{dim}}$ | $\text{masks}_{2_{\text{dim}}}$
  - **end**
  - outSet ← `list_add(outSet, outBox)`
- **end**
- **return** outSet

Function `lookup_children` *(n1: 0-7, n2: 0-7): (array of 0-7, 0-7)* →
- return line number (n1 * 8) + n2 from table child nodes

Function `make_masks` *(i: 0-7, splitIdx: 0-7, depth: int): mask structure* →
- return mask structure (see code for details)



(a) Mean time

(b) Mean precision

Figure 4.3: Splitting AABB to volume set

Algorithm 3: Computing difference between Node Sets

```
Function diff(FromNodes, ToNodes, Add, Remove: List of node) →
    Data: Add, Remove are initially empty
    Result: (List of node, List of node)
    if len(FromNodes) = 0 then
        return (Remove, list_add(Add, ToNodes))
    end
    if len(ToNodes) = 0 then
        return (list_add(Remove, FromNodes), Add)
    end
    F ← head(FromNodes)
    FRest ← tail(FromNodes)
    T ← head(ToNodes)
    TRest ← tail(ToNodes)
    if morton_less_than(F, T) then
        if is_ancestor_of(F, T) then
            diff(FRest, TRest, list_add(Remove, F), list_add(Add, T))
        else
            diff(FRest, ToNodes, list_add(Remove, F), Add)
        end
    else if morton_less_than (T, F) then
        if is_ancestor_of (T, F) then
            diff(FRest, TRest, list_add(Remove, F), list_add(Add, T))
        else
            diff(FromNodes, TRest, Remove, list_add(Add, T))
        end
    else
        /* neither is less, they are equal, they remain unchanged          */
        diff(FRest, TRest, Remove, Add)
    end
```

0	0	0
1	0	1
2	0, 2	2
3	0, 1, 2, 3	3
4	0, 4	4
5	0, 1, 4, 5	5
6	0, 2, 4, 6	6
7	0, 1, 2, 3, 4, 5, 6, 7	7
9	1	0
11	1, 3	2
13	1, 5	4
15	1, 3, 5, 7	6
18	2	0
19	2, 3	1
22	2, 6	4
23	2, 3, 6, 7	5
27	3	0
31	3, 7	4
36	4	0
37	4, 5	1
38	4, 6	2
39	4, 5, 6, 7	3
45	5	0
47	5, 7	2
54	6	0
55	6, 7	1
63	7	0

(a) child nodes (empty rows omitted)

0	((1, 1, 1), (0, 0, 0))
1	((1, 1, 1), (0, 0, 1)), ((1, 1, 0), (0, 0, 0))
2	((1, 1, 1), (0, 1, 0)), ((1, 0, 1), (0, 0, 0))
3	((1, 1, 1), (0, 1, 1)), ((1, 1, 0), (0, 1, 0)), ((1, 0, 1), (0, 0, 1)), ((1, 0, 0), (0, 0, 0))
4	((1, 1, 1), (1, 0, 0)), ((0, 1, 1), (0, 0, 0))
5	((1, 1, 1), (1, 0, 1)), ((1, 1, 0), (1, 0, 0)), ((0, 1, 1), (0, 0, 1)), ((0, 1, 0), (0, 0, 0))
6	((1, 1, 1), (1, 1, 0)), ((1, 0, 1), (1, 0, 0)), ((0, 1, 1), (0, 1, 0)), ((0, 0, 1), (0, 0, 0))
7	((1, 1, 1), (1, 1, 1)), ((1, 1, 0), (1, 1, 0)), ((1, 0, 1), (1, 0, 1)), ((1, 0, 0), (1, 0, 0)), ((0, 1, 1), (0, 1, 1)), ((0, 1, 0), (0, 1, 0)), ((0, 0, 1), (0, 0, 1)), ((0, 0, 0), (0, 0, 0))

(b) wall alignment lookup

Table 4.1: Split lookup tables

Considering the 100msec time budget for sending a reply to client, split depths between 4 and 6 seem appropriate. As the computation will need to be performed for each movement update of an entity, overestimating the volume of perception of an entity by almost an order of magnitude may be appropriate.

The exact amount for the maximum split depth is an manually set parameter and its optimal values is application dependent. For an optimatimal value, a specifica application must be tested by running the application and comparing the results of differing max split depths. As the view-box recalculation will have to be performed each time the viewer moves more than what the overestimation of the selection covers this may happen multiple times per second, but not necessary every time slice for all active objects.

The direct benefit of increasing the split depth is not only the reduction of selected octree nodes for the VBSet, but more importantly reducing the the number of objects that are subscribed to, which are contained in the subtrees of the VBSet, as there is less spatial area being selected, due to the better precision. Everything else being equal, less subscribed to octree nodes and contained objects equals less latency maximums when compared with fig. 3.6, especially the upper percentiles.

4.2 Bound Selection Depth for View Box Elements

This is the second extension of the basic QuP mechanisms published in Behnke et al. (2014). It solved the other problems that were discovered in the initial implementation: an unbound amount of RAM needed to track the octree node subscriptions. This not only would test the physical limits of the hardware, but could also stop other clients, connected to the same machine, from being processed.

The bounding also provides a mechanism of selecting objects that are relevant for the perception abilities of users as well as the rendering capabilities of the client software they employ. The current prototype implementation allows the definition of a complete solar system at 1mm accuracy:

$$2^{62} = 10^{18.6639}$$

$$10^{18.6639} mm \approx 30830 AU$$

This equates to a point somewhere in the Öpik-Oort Cloud in our solar system. For comparison, the orbit of the dwarf planet Pluto varies between 30 and 49 AUs. Even at nanometer scale (10^{-9} m) this would still allow the complete Earth-Moon system to be modelled.

While the positions in these vast spaces can be enumerated by the system, modelling such an environment would obviously exceed the storage capacity of a real system by multiple orders of magnitude. But in a very sparsely populated environment like the solar system, detailed information about objects within human perception in a few isolated places on a small number of planets should be feasible.

4.2.1 Selection Generation

For each nodes in the VBSet, the complete subtree, including all tree nodes and all objects contained within these nodes are subscribed to. The number of selected nodes and objects balloons to a large number of elements if the root of the selection is above all but the lowest levels of the tree. The current implementation of QuP uses a tree depth of 62 due to the observation that modern computers will have a word length of 64 Bit anyway (the remaining two bits are used for technical details of the Erlang implementation), so using smaller depths will result in a reduction of neither storage nor computational requirements. In contrast, retaining the maximum depth may provide great flexibility in the size of the DVE, as well as the range of contained objects.

This maximum depth is a configurable value, but requires a recompilation of the code due to bitmap optimisations. Persisted environment state cannot be exchanged between software set up for different tree sizes without some prior processing.

This maximum size of the trees selected from VBSet is reduced to a certain degree, as tree nodes are marked as leafs (by setting the attribute `leaf` to true), if their children do not contain objects. These empty nodes are not instanced on the server. Empty nodes will be removed and their parent checked for further reduction. But in a VE of widely varying sizes of objects, this may still result in very large selected trees, as objects may be placed very far down the octree. This is compounded by the fact that object location information is partitioned with the same method as the view box (see chapter 4).

QuP overcomes this problem by placing a bound on two elements: the maximum split depth

when determining the node set covered by a bounding box or view box, and the selection depth when selecting objects from nodes within the subtree beneath view boxes.

For a given VBSet, the maximum selection depth is set to the minimum of $\forall node \epsilon VBSet, minimum(depth(node)+F))$. F represents the fidelity of perception that results from the selection. It governs the smallest object included in the selection for a view box. Due to the method of partition, a Fidelity of 10 would mean that objects may be $2^{10} \approx 2^4$ times smaller than the view box itself and still be selected. Due to the partitioned placement of the OBB the actual selection may vary by up to 2 levels (variation for both the VBSet as well as the BBSet).

Fidelity is currently set globally via the configuration variable `MaxSelectionDepth`. It could be dynamically adjusted during the runtime of the DVE or vary for different entities. This would require some further control mechanism that currently is not part of QuP.

4.3 Tracking visibility

In order to provide an overestimating heuristic as the basis of selecting relevant objects for AoIM, QuP performs a box intersection selection within an octree.

Two steps to separate: a) an object entering a given node and b) a change event of that object, which will have to be propagated to all other entities that are observing this object.

4.3.1 Special Cases

A number of extreme cases have to be considered when constructing the selection mechanism:

objects larger than the view box As these objects can potentially encompass the whole VE, this requires the selection algorithm to traverse up to the root of the tree. This is not a great problem, since there cannot be a large number present high up in the tree, as this would otherwise violate the stated limits discussed in section 3.3.6 on page 71. As all the subscriptions are managed per client node, this does not lead to added network traffic either. The selection is provided inclusion of parent nodes, described in section 4.3.3.

4.3.2 Maintaining Position

For each node that contains part of the bounding box of an object, this object is marked as contained (in the field `contained`). The nodes are determined by the algorithm 4:

The `contained` sets are updated for each object when its bounding box or position are changed.

4.3.3 Maintaining Watchers

Entities are objects that are able to observe their surroundings. They require a local CPU thread (in Erlang represented as a process) to consume the events produced by the creation and changes of other objects. The avatar of a player is an entity, that is processed by the thread maintaining the connection to the client application.

Watchers are subscriptions by entities to nodes in the octree to track the objects that are contained within that node.

To track these watches, whenever an entity is created or its view box of two points (P_{v1}, P_{v2}) is changed, the view box is converted into a point set, using algorithm 2. Then change sets are computed, using algorithm 3, to either create one watcher per node in the add set or remove the appropriate one for node in the remove set respectively.

Each watcher will recursively subscribe to all nodes in the octree from a given start node, limited by the `MaxSelectionDepth` (see section 4.2.1), and the `leaf` attribute of a node. As long as the `ancestor_has_data` attribute of a node, starting with the initial one, is set, the watcher will also recursively subscribe to those as well.

The watcher will then subscribe to all objects given in the `contained` attribtes in each node listed above.

4.3.4 Deleting Objects

The octree nodes will never be removed, since the spatial volume with in the DVE does not cease to exist.

Normal objects may be deleted by the application, but QuP does not provide any consistency checks, should one object reference another by its object Id.

Algorithm 4: Position Tracking

Data: ObjectId, MaxSplitDepth, newBBox: $\{N1, N2\}$, oldBBox: $\{O1, O2\}$; with $N1, N2, O1, O2 : \{p_y, p_y, p_z\}, 0 \leq p_n < 1$.

```
begin
    newSet ⟵ split(newBBox, MaxSplitDepth)
    oldSet ⟵ split(oldBBox, MaxSplitDepth)
    (removeList, addList) ⟵ diff(oldSet, newSet)
    for node ∈ addList do
        add_to_set(node, contained, ObjectId)
        mark_ancestor_data (node)
    end
    for node ∈ removeList do
        remove_from_set( node, contained, ObjectId)
        if isEmpty (node, contained) & ! isSet (node, ancestor_has_content) then
            cleanup_ancestor_data (node)
        end
    end
end

Function mark_ancestor_data (node: nodeId) →
    begin
        if ! isSet (node, leaf) then
            forall child ⟵ children (node) do
                setValue (child, ancestor_has_data, true)
                mark_ancestor_data (child)
            end
        end
    end

Function cleanup_ancestor_data (node: nodeId) →
    begin
        if ! isSet (node, leaf) then
            forall child ⟵ children (node) do
                setValue (child, ancestor_has_data, false)
                cleanup_ancestor_data (child)
            end
        end
    end
```

4.4 Summary

In QuP I present a system that allows scaling DVEs without running into the square law communication overhead.

By the nature of being a multi-server architecture, the proposed system provides the enforcement of global security controls as required by Hu and Liao (2004). It also supports the requirement for differing interest ranges with minimal overhead as proposed in Hu et al. (2010).

By not being a P2P system, QuP also allows tighter control over the nodes to avoid losses due to churn. State is replicated among multiple nodes and these are controlled by a single entity, so that all forms of state of the art system administration like backups and local tools can be used to bring a failed node back on line, to avoid lengthy re-synchronisation of the complete state of a node.

As Najaran and Krasic (2010), a pub-sub subsystem acts as the core to AoI based state distribution, but supports much finer grained subscription subjects based on a spatial area, rather than the other avatars. This allows not only for state update on objects that are not the avatars, but also a much higher number of avatars.

Unlike Hu et al. (2010) (which only covers a single Second Life region, basically a shard), QuP does not partition the DVE and allows spatial resolution and total size that are far beyond any other system. The only examples that cover a greater spatial area like are sharded as well, and may provide interaction outside of the DVE (e.g. trading in addition to star systems in Eve Online).

As SPEX (Najaran et al., 2014), QuP supports hierarchical partitioning of the DVE. But the octree of QuP avoids the need for synchronising the splitting or merging of partitions. As SPEX, QuP also supports the separation of area of perception from the area of presence. This allows new features for DVEs to be designed (e.g. observing a distant location through a telescope). As there is no direct coupling between the network topology, the spatial layout and partitioning of the DVE and the assignment to computational resources in QuP, an avatar could also have multiple view points, or even support multiple octrees (e.g. support radar cross section and optical visibility).

The performance of the system is governed by a tradeoff between high fidelity in the positioning of view-boxes and object bounding box, represented by the maximum split depth (see section 4.1.3), and the subscription to ever smaller objects governed by the maximum selection depth as described

in section 4.2 on one side. On the other side is the increasing time as well as memory and network bandwidth it takes to generate and transmit the ever larger set of individual octree nodes and their content data. Balancing these needs must be conducted for each DVE application and may also have to be changed dynamically, as the optimal solution for this tradeoff may vary widely within the same DVE instance.

Chapter 5

Budget Based State Aggregation

Many of the wide range of AoI techniques proposed (see section 2.9.3), select the active objects relevant to a given client, in order to partition the state change as well as the state data sent to that client.

This partition ensures that a given server node of the DVE can handle the necessary processing. It also ensures that the relevant changes can be forwarded to the client without overloading the potentially slower WAN network connection (when compared to the inter-server LAN) with information on all the objects present in the DVE.

In Suznjevic and Matijasevic (2013) and Shen and Iosup (2014), the authors have shown that avatars are not evenly distributed over the spatial area of the DVE, but rather will cluster tightly on hotspots of interest to the individual users, whether to achieve goals in the application or simply to gather and socialize.

If the avatars are distributed evenly throughout the virtual environment, AoI would provide the capability to support very large populations of participants. But once a sufficient number of avatars gather in a small area of the virtual space, communication overhead grows on a n^2 scale as they observe each other. This problem is compounded when QuP scales as well as described in chapter 4 and processing of a single spatial area is distributed among multiple servers, allowing even higher local avatar densities.

Instead of optimising the overall number of messages, I limit the number of messages sent to a

single client. In Behnke et al. (2015) I proposed a novel approach to dynamically aggregate state update messages. It can support a very high numbers of avatars in restricted spaces. We decouple the management of the global state of the DVE from amount of detail sent to the client. This approach exploits information about the specific DVE and its current state to maximise the ratio of bandwidth reduction to state update fidelity. It is intended to strike a balance between providing as rich a user experience as possible for a client that has connected to the DVE via limited network link (e.g. mobile users or consumer grade DSL.) as possible. The number of of network messages are assumed to be a sufficiently valid representation of the bandwidth needs for providing the client with this experience.

5.1 Introduction

Multiple techniques for managing Areas of Interest (AoI) and filtering state change messages based on these areas have been proposed in recent years. A good overview is provided by Liu et al. (2012). Approaches for client/(multi-)server, as well as peer-to-peer architectures have been proposed. Independent of the underlying architecture, they usually aim to reduce the state update messages sent to a given client by limiting them to those that can actually be observed by an avatar and thus are required by the client.

Morillo et al. (2006) considers the need to verify MMO Architectures for varying avatar concentration and movement patterns. From his work, we take the concept of the Avatar Density (AD), the number of avatars in a given spatial area. The global AD represents the total number of client connections and thus avatars supported by a DVE. In contrast, we focus on the maximum supported local AD, the number of the avatars in high density situation. Work like Shen and Iosup (2014) has shown that the tight clustering of human players is the norm rather than the exception.

In de Oliveira and Georganas (2002) the adapting of the size AoI dynamically to respond to a wide range of metrics has been discussed in detail. But this results in a binary decision to include the updates for a given avatar or to be forwarded to a client or not. This is in contrast to human perception where artifacts in peripheral vision may still be observed, even if with reduced fidelity.

In Gascon-Samson et al. (2015) the problem of managing the required bandwidth is mainly

viewed from the overall bandwidth required by the complete DVE. This is interesting for planning and budgeting the associated cost of providing a public service especially. In their paper, they use a filtering that represents a dynamic adaption of the view range. In contrast, my approach will focus on the bandwidth requirements of each individual client connection and allow a much more varied adaption in order to minimize the perceived impact on the user.

For a high local AD, the nimbus' of the avatars overlap and they can each see each other, resulting a n^2 communication overhead. Handling this by dynamically reducing the observation range may render avatars effectively blind. It also results in a restriction of the avatars abilities due to technical reasons and is likely to be perceived as a service degradation by the user.

I present Dynamic Budget Based State Aggregation (*DyBuBSA*), a mechanism for fast discovery of applicable aggregators from a repository to handle local overload scenarios. While most aggregators are based on common algorithms as reducing the update ratio or the detail of information sent, *DyBuBSA* allows the integration of DVE specific aggregators, that exploit elements of the content within the DVE (see section 5.2.2 for some examples).

I also introduce a novel way to select applicable aggregators and apply them to specific avatars and their context.

This work is complementary to aggregate network packets, like Jinzhong (2011), that allow the combination of multiple messages into single network packets.

5.2 Approach

The prototype of *DyBuBSA* is based on *QuP* (Behnke et al. (2014)), a system for persisting a unified DVE state, distributing state updates to networked game servers, using eventual consistency to provide resilience to component failure. Figure 5.1 shows the software elements of a single instance of a set of game server nodes.

The aggregators behave as avatars, subscribing to state updates that are elements of the aggregate. The aggregators will only actually subscribe to update messages and process these if at least one avatar or other object (e.g. another aggregator) is subscribed to the event stream that the aggregator produces. QuP makes this simple, as the list of subscribed entities is a published

attribute available to other subscribers.

Upon activating an aggregator, the avatar will be informed of the covered objects, will unsubscribe from those objects and subscribe to the aggregator. By subscribing to it, the aggregator will be activated, should it have been dormant.

5.2.1 Budget

For each client connection, a message budget is set. Appropriate values of connection can be obtained from various sources, e.g. a) an arbitrary value set by the DVE operator, to limit bandwidth costs; b) information on end-to-end network bandwidth, as congestion messages or an IP address block assigned to mobile service provider. The budgets can also be updated dynamically and separately for each client connection.

DyBuBSA tracks the message rate sent to each client. For our prototype, the length of each message is fixed and thus the number of messages provides a sufficient approximation of the required bandwidth.

5.2.2 Types of aggregators

DyBuBSA supports three basic classes of aggregators. The first are global aggregators, which are applicable to any object and can be dynamically instantiated and applied to any object generating update messages. They usually support sending fewer updates per time or ignoring certain types of updates. The most basic of these aggregators simply forwards only every n^{th} update message, or only update the position, but suppress any more detailed attributes.

The second class of aggregators are application specific and support the collation of multiple objects into a compound object that generates resultant update messages on a much more coarse level. Good examples for this are tight military formations of many soldiers, acting as a single entity. Or a train composed of a locomotive and a number of carriages.

> Example for a military unit: The aggregator subscribes to all members of the unit. This requires some tracking of the unit, but would be the result either of a prior setting as an attribute of the individual element (individual soldier, vehicle, or sub-unit), registering

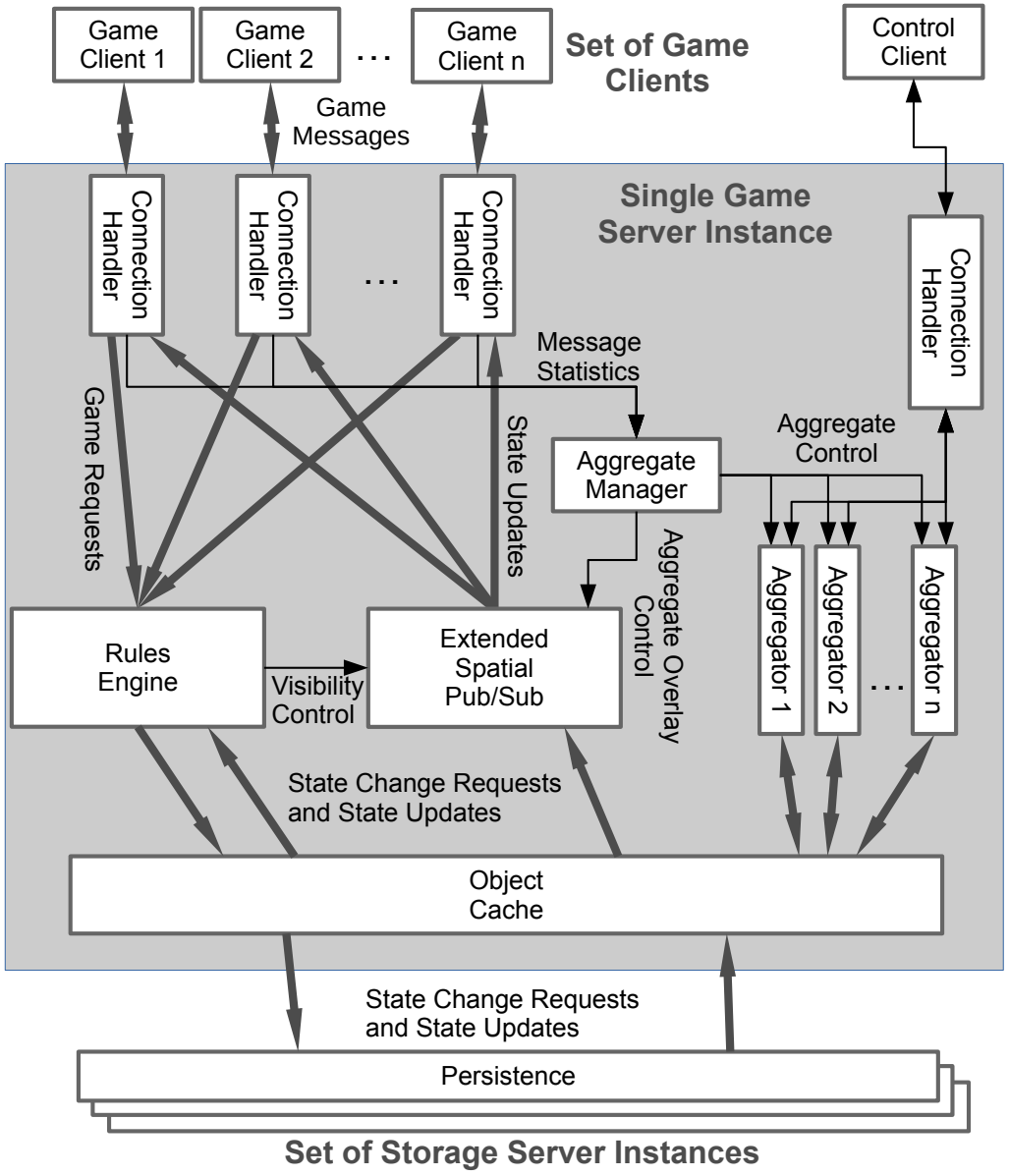


Figure 5.1: software components of system

itself with the aggregator, or some user interaction assigning the element to the unit. The aggregator then maintains a minimum and maximum for each axis of the spatial extent of all elements of the unit as the basis of a bounding box of the unit. The unit resultant bounding box is positioned in the octree according to the algorithm described in section 4.3.2. Depending of the level of detail supported by the client and the network link between the server and the client, either the aggregator is displayed (possibly via visually indicators like military unit signs) or the elements that make up the units as fully animated soldiers or vehicles.

In Heger (2013) a method of describing avatars through pre-defined commands has been published. These pose an ideal fit for abstracted groups of avatars. This can be supported by BuBSA by defining volumetric areas of space within the virtual space that processes the number of avatars within this space and simply measures the avatar density within the volume and reports that value to the appropriate render element within the client, transmitting sets of predefined actions to provide a believable simulacrum of e.g. the milling mass of people in a market square. As the client needs to have the geometric information on all of the stalls, enclosing walls and moving shoppers to be able to render them, it could provide a lively market scene with standardised animations by simply avoiding collisions; this would thus greatly reduce the needed network traffic. The server side algorithm would simply register a viewBox with the extent of the market square, thus subscribing to all those avatars that enter the square. Then avatar density is simply the number of avatars divided by the size of the square.

The third class is application specific too, but enables the visibility based on importance. The bound octree selection depth described in section 4.2 results in objects being ignored that are too small when compared to the vision range of an entity.

Example: consider a initial user perspective of a space exploration game. The user will be shown a complete solar system. Due to the vastness of space, any object other than the central star will be too small to render and will thus be ignored by the object selection mechanism. In order to provide any visual clues and starting points for further interaction, the planets and other modelled objects could be represented by a marker

object that would actually be updated when certain aspects of the planets change. This would take place not only within the graphical client, but be reflected in instanced objects within the server back-end of the DVE.

Objects, or even other aggregators, can be aggregated themselves to be placed much further up the octree to ensure they stay visible, regardless of the selection depth. Consider a marker for each planet in the above example.

5.2.3 Aggregator Selection

Aggregators act as any other active object: the have view box that represents their range of perception for activity that may be part of the the aggregate, as well as a bounding box that represents the spatial extend of the aggregate. For both of these Axis Aligned Bounding Boxes (AABB) a node set is computed (as described in section 4.1). The aggregators select objects from the subtrees of their View Box node Set (VBSet) and compute updates of the aggregate

Active objects perceiving the space in which the aggregates are located, will subscribe to them as any other objects placed in the octree. By choosing the applicable aggregator with the greatest distance in octree depth to the avatar, the impact on the user experience can be minimised as well, as details further away are displayed smaller and are not as likely to be in the focus of the user.

Each aggregator also publishes a message reduction ratio as well. Additional aggregators are selected until the projected aggregates message rate is below the intended rate.

Global aggregators are registered with the top node of the octree and will perform their filtering for each avatar that subscribes to them instead of a specific original object. As the root of the tree represents the complete VE, these aggregators may either manage globally visible objects, or metadata that applies to the complete virtual world.

5.3 Message Propagation Partitioning

Timadorus uses the following design pattern for discovering interested parties in effects and updates: a) observers can register with QuP and receive update information based on the space their view box coves. QuP will automatically discover all interested observers and notify in case an object

changes. b) for one entity to affect a change in an object (including other entities), they need to pro-actively find the objects that are contained within the volume of effect. QuP will aid in this, but has no automatic discovery process.

Without partitioning, this discovery of effected objects may lead to high aggregated messages bandwidth, as would broadcasting object state updates.

As a reverse to the budget based aggregation, updates may also be distributed to segments of the set of affected objects. As the state update (optionally filtered) will be set as attribute of the aggregator object, the aggregator may act as a proxy object to the originator of the message. This will accrue considerable delay; as setting and reading the value of the proxy adds a full roundtrip through the QuP system, it allows the partitioning of an update message stream.

5.4 Bandwidth Reductions and Computational Costs

This sections does not provide any numbers on bandwidth reduction or the amount of computational overhead, as they are not only completely application dependent, but may differ from client to client even for a given application. First there is the amount of computation for calculating the hull of an aggregate; or the ratio between the view box of the aggregator and the bounding box of the resultant aggregate. A larger view box will automatically observer more objects, resulting in more messages that need to be processed, even if the the majority are discarded as irrelevant to the aggregator and the aggregate.

Secondly there is this the reduction in user experience that the aggregation actually produces. If rendering capabilities are below par for the type of the network connection (again both depending on the type of client hardware used and the form of rendering used by the application), then even a fairly small message budget due to a mobile network connection may not be visible to the user at all.

Last, but very much not least, there are the perception capabilities of user herself. With the advent of Retina Displays, which can output image pixels smaller than the ability of a *young and healthy* human eye to differentiate, then users may not even be able to contiguously perceive the reduction of update rate and detail. Much less react with a reduction of immersion that would lessen

the game experience.

DyBuBSA provides a mechanism to utilise the aggregators in adapting the rate of game messages to the capabilities of the client network connection and rendering hardware. And it minimises the computational overhead of the aggregators by automatically turning off the maintenance of the aggregate when no avatar is subscribed to it. But I know of no way to state a function of computation cost to message reduction that would be applicable to all DVE applications.

5.5 Fault Tolerance and Scaling

Aggregators are instantiated as any other component in the game server: either on a per server basis, as separate active objects or as part of a client connection handler. They all use the same API to access persisted objects to ensure that state and its change is maintained even in the face of component failure.

As servers or client connections fail, their associated aggregators will be restarted as either replacement servers are brought back on-line or a client reconnects. The active objects are maintained with a supervision hierarchy that is a well tested central feature of the Erlang run-time environment. They will be restarted should they fail and retrieve their state via *QuP*.

Thus BuBSA shares the same resilience to component failure and will scale along with the other components of the system. BuBSA does not provide protection against memory corruption or faulty computation as part of the aggregation. As the intended target type of application are MMORPGs, absolute correctness is not a primary design goal. Such guarantees could be provided fairly easily by calculating the aggregates redundantly by multiple instances of the same aggregator (using the same program code multiple times or even differing implementations). The persistence layers provides redundancy as fault protected, thus multiple independent code paths could be realised. But I consider it unlikely that a MMO service provider will be willing to pay the price of computational overhead for protection against this type of error.

Chapter 6

Avatar Density Based Client Assignment

Scaling the number of supported users for DVEs allows more users to experience its content together. Enabling this scale needs to consider the chain of all components between the client software of any two users within that DVE.

A large body of research has been created over the last decades on the problem of dividing the resultant workload of a DVE to specific nodes in a cluster of server. Connecting clients to the most appropriate server of this cluster is as important. Clients will place widely varying, dynamically changing requirements on processing, storage and network bandwidth resource on the DVE.

In this chapter I describe a mechanism for assigning clients to servers in a DVE, specifically a Massive Multiplayer Online Game (MMOG) as part of a load balancing effort. It allows the optimisation of resource utilisation while being able to handle overload situations in the face of high avatar density and adapt to change over time.

6.1 Introduction

Managing the amount of processing, memory and network resources to provide the platform for any MMMOG follows the proven divide-and-conquer approach. When trying to scale up the number of

players that can play the game concurrently, many bottlenecks have to be considered and overcome. The following work is part of a larger effort to create a complete MMO-RPG system that a) will allow the creation of a single virtual environment with support for far more than 100k concurrent users; b) will provide a non-stop service with no planed down-time, c) will handle component failure (hard- and software) with minimal service degradation; d) will allow unparalleled level of flexibility of the contained environment.

Prior publications have targeted the partitioning of the actual server workload, the resultant storage requirements, and the aggregate network bandwidth that is required for the communication between the components (see chapter 3). The platform presented, *QuP*, will support a high number of concurrent players, as it allows large (> 20) clusters of servers to maintain a single, continuous game environment.

QuP will also support a high local avatar density (the number of avatars in a given space), as it allows the same spatial area to be maintained on more than one server. This may lead to the next bottleneck: sending environment updates to the client computer, as this connection is usually the weakest link. To provide a minimal reduction user experience while greatly reducing the amount of change messages needed, I introduced dynamic budget based state aggregation in chapter 5. While the concept is a generalised one, it will mostly benefit from situations that are present in many MMOs, where crowds of people or blocks of military units are moving in a similar fashion. Their action can easily be described as a group, with massive reduction of required messages.

QuP and DyBuBSA both benefit from an optimal assignment of clients to servers in order to group information about avatars that are in spatial proximity to a small subset of the whole cluster, ideally to a single server. This is due to the prefetching and pre-processing nature of both techniques. Client Assignment (CA) is a NP-complete problem Chen et al. (2005); if encompassing a grouping problem, it is even NP-hard Falkenauer (1994).

Many existing solutions to CA in MMOGs are very application dependent. In many large-scale games, they range from the primitive, as in MMOBA-type (Massive Multiplayer Online Battle Arena) games like World-of-Tanks or StarCraft, where very small groups of players (usually 4-40) are grouped into a large number of world instances. These instances only exist for a short time (usually less than 60 min) while a single battle lasts. MOBA CA usually follows a few computationally simple

rules to group players of similar strength in order to optimize the entertainment value of the game.

In MMO-RPGs, the assignment is often done via a static shard selection (e.g. World of Warcraft). This selection may be changed by the user if the vendor allows it. The selection is usually subject to capacity planning by the service provider. These shards, or replicated instances of the virtual world, can then be distributed statically to pre-planned server instances.

When attempting to construct a MMOG that models a continuous or at least connected space, in which all users interact in a single, integrated, continuous environment, the CA is more difficult to pre-plan and requires constant monitoring and updating while the game is running. To increase the number of concurrently supported players within the world, MMORPGs often use zoning to partition the game population together within the game world. Unfortunately, this often fails to support scenarios where the whole population gathers explicitly to partake in a single event (e.g. massive battles in EVE online CCP Inc. (2013))

In chapter 3 (published in Behnke et al. (2014)), I presented a MMOG platform that is able to separate the spatial location of the avatar within the game environment from the game server (GS) to which it is assigned. Within it, large synergistic effects may be gained by grouping avatars in spatial proximity on the same server while avoiding overload situations before they develop. To minimise the required network traffic further, I introduce a method to reduce the average number of replicas of game objects on the GSs. I aim to achieve this goal by attempting to group avatars of similar AoI on the smallest subset of servers as possible. To handle overload of a single server, QuP does allow for the same spatial area of the game to be represented by more than one server, albeit with a certain amount of additional network overhead.

Striking a balance between tight grouping and avoiding overload is the task of the client assignment manager (CAM).

6.2 Related Work

A number of approaches to partitioning the client population prior to assigning them to servers have been discussed by commercial vendors as well as scientific publications. Most commercial vendors opt for primarily static or pre-computed assignment.

A wide range of work has been done on the subject of load distribution and load balancing. Most publications propose the partitioning of the game world and thus the avatar population. These partitions are then assigned to servers within the system. Ng et al. (2002) proposes to assign one region to one server, with the region being partitioned further into cells, which can be ceded to other servers in high load situations. The load information of each server is regularly submitted to a central load collector server. As Lau (2010) observes, the cells are only moved to neighbouring servers and global state is not considered, thus underutilised servers may be ignored since they are not adjacent to the server in overload. To order to address these shortcomings, Lau (2010) proposes a hybrid approach in which the load is shed not only to neighbors, but appropriate servers are also chosen from a global set.

The work in Lui and Chan (2002) approaches load balancing via linear optimisation, looking not only at server processing load, but the required communication overhead among the servers as well. The aim is to distribute the load evenly among servers while minimising communication at the same time. They construct a graph based on the area of interest (AoI) of each avatar, connecting the graph when the AoIs overlap. Morillo, Fernandez and Orduna (2003) improve this by using an ant colony optimisation algorithm, but also showed that there is a minimal size of avatar population and server cluster that limits the cost efficiency of more complex load balancing approaches.

Chen et al. (2005) does not attempt to distribute load evenly, but rather to ensure that pre-defined levels of quality of service (QoS) are met by each server. These QoS include CPU utilisation, network bandwidth consumption. They propose an appropriate partitioning and merging scheme.

Static partitioning is the state of the art of many commercial online games, but does not match the dynamics of an MMO-RPG. De Vleeschauwer et al. (2005) investigate a number of algorithms and how well they approach optimal distribution of clients to server. They show linear programming to be able to provide an optimal solution, but also that the required cost of computation is prohibitive for practical use. Yunhua Deng and Lau (2012) show that all the other algorithms are costly as well. In Deng and Lau (2014) they propose a method that uses a simulated heat flow model, based solely on local information.

All above approaches assume that all client connections and thus avatars produce a similar load and thus partition the avatar population by partitioning the environment space. A different approach

is taken by Morillo, Orduña, Fernández and Duato (2003). Here a server is focused on the avatars that are assigned to it. The scheme first sorts the avatars by the number of AoIs they are present in. It then repeatedly moves the most interesting avatar until a number of iterations is reached.

Lu et al. (2006) states that regions do not reduce the inter-server communication to a sufficient degree. Instead they favor a simple load balancing scheme. They propose the use of a NAT-Server between server and client and the use of standard load balancing methods like round-robin. The proxy approach is also proposed by Quax et al. (2009) to isolate the region-to-server assignment from the client-to-server assignment.

The client assignment method presented below does not take into account the network delays incurred between the clients and the servers. Details of this problem are discussed in Ucar et al. (2013). Selecting an appropriate *cluster* of processing nodes to minimise the network latency (or other restrictions between client and server) would be complementary to the assignment discussed below. Integrating both types of delays are subject of further study (see section 6.7.3).

6.3 Requirements

As this work is based on the QuP platform for large scale MMOGs, the AoI of each avatar is represented by its view box, an axis aligned bounding box (AABB). The CAM should group the assignments to the game servers according to the spatial grouping of their avatars within the virtual environment. This is expected to allow a reduction of retrieved state and subsequent state update messages to a given server as multiple avatars will access the same prefetched object and environment data through a local lookup rather than costly remote query.

The CAM must also ensure that the CPU load on a given server is within acceptable parameters. As the load created by a single client connection can vary widely, this will require feedback from the GS. Available main memory and network bandwidth must be considered as well.

Based on the requirements, the following cost function can be derived to govern the client assignment. The client will be assigned to the server that will return the minimal value from the cost function below:

$$cost(a,s) = w_0 * -g(a,s) + w_1 * Load(s) - w_2 * MemFree(s) \tag{6.1}$$

For an avatar a and the serve s, the function $g(a,s)$ will calculate the grouping factor (GF). The GF represents similarity of the AoI of the new avatar to the AoI of the avatars that are already present on a given server. It is assumed that most of the objects in the AoI of the other avatars have already been loaded and that further updates on those objects will be relevant to both the existing and the new avatars. The actual computation of the GF will be discussed in section 6.4.2

The memory is not given as the percentage of used memory, but the amount of remaining memory, so that servers that have large remaining amounts of memory are favored, regardless of their total amount of memory. w_0, w_1, w_2 are weights to control the priority of the potentially contradictory requirements of maximizing the grouping factor and not surpassing limits on CPU load and memory utilisation. The GF can be considered a divisor of the average per-object network bandwidth requirement, through not having to load a given object multiple times. Thus I assume network utilisation relates directly to GF. It is therefore not considered as a separate parameter in server selection. More fine-grained modeling on this may be an area for further study.

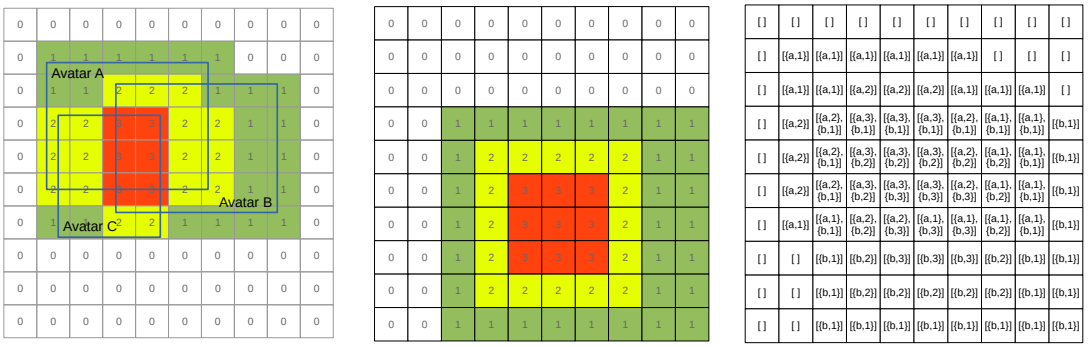
As the avatars move over time, the client assignment must be able to track validity of the calculation results over time. This may lead to the need for a reassignment of clients to new GS over time. Also any CAM solution should be able to scale with the MMOG to support a user population far in excess of 100k concurrent users, including the resultant churn, expected clustering of user log-in and number resultant number of GS.

6.4 Proposed Solution

I propose calculating the grouping factor using a heat map. To construct the heat map, the virtual environment is partitioned into a set of blocks of equal size. This is similar to partitioning of the actual game environment into micro cells De Vleeschauwer et al. (2005), but uses these blocks only for the grouping calculation. Due to the architecture of QuP, this is done in three dimensions. The diagrams below are restricted to two dimensions for the benefit of illustration only.

As the AoI of an avatar governs which objects in the game have to be prefetched to a server, the

view boxes and their overlap are used to determine the grouping factor when considering a server for assignment.



(a) Viewbox overlap and resultant heatmap for server A

(b) Heatmap for server B

(c) Combined heatmap for servers A and B

Figure 6.1: View box overlap

Each block holds a set of servers that serve at least one avatar whose view box overlaps with the block. A tally of the number of intersecting view boxes is kept, as shown in fig. 6.1c

In QuP and the LB, all object coordinates are within a norm cube ($0 <= x, y, z < 1$). This cube is partitioned into n blocks per axis. For each block (identified by it coordinate between 0 and n-1) and each server, $T(x, y, z, s)$, a tally of avatar view boxes that overlap this block is maintained. This allows the calculation of the grouping factor for each server in a block.

As each client is requesting a server assignment, the view box of the avatar is queried from the world state (maintained by the QuP servers). As the server is chosen, its tally is updated in each of the blocks that the clients view box intersects with. As a server is considered to cover the view box of an avatar even if the view box only intersects the spatial area covered by the server, this method forms an overestimating heuristic. It allows the computation of the GF used in eq. (6.1).

6.4.1 CAM Components

The CAM is partitioned into two different components: the load balancer (LB) and the server monitor (SM). In the experiments, only a single LB instance is used (for further scalability see section 6.7). One SM is assigned to each GS. See fig. 6.3 for communication links and integration into the full system. The actual CAM components are shown in darker color.

As the central component, the LB holds the actual heat map and meta information on the known servers and the previously assigned clients. The authentication server queries the LB to receive a server assignment. The LB in turn maintains a lease on the client/server assignment to update the tallies for the heat map. This lease will be confirmed if an acknowledgement message is received by the LB. This message will be sent by the server once the client has connected to it.

The LB continuously monitors the servers for any messages send to it. This may either be heart beat messages by the server or responses to client assignment requests. Should a server not have sent any message longer than a configurable threshold, it is considered dead and will be removed completely from the state of the LB. It may be reintroduced only by administrator command. If there are pending assignments that have not been confirmed yet, they are rolled back in the heat map as well.

The SM running in each GS will inform the LB of any successful client connections. As the avatars move through the environment, the SM will track which heat map regions are being covered and uncovered by the shifting view boxes of each avatar. The list of changes is reduced by applying inverse operations (e.g. the number of new avatars in a region is subtracted from the number of avatars that have left the region). The resulting difference list is sent to the LB in regular intervals.

6.4.2 Calculating the Grouping Factor

The assignment algorithm has to weigh between placing an avatar on a server that already covers a spatial region or placing it on a new server to avoid overloading the former server. There are two special cases to consider when selecting servers for the client: a) empty servers (i.e. without any clients) and b) server whose covered area does not intersect with the avatar view box. Empty servers will not only occur after starting the system, but can also result from avatars congregating on small spatial areas of the game, leaving others completely empty and thus the corresponding servers idle, as reassignment incrementally groups the avatars on fewer and fewer servers to optimize the grouping factor.

Both cases are similar, in that they result in a grouping factor of 0, as no view boxes overlap. The client assignment would only be controlled by the CPU and memory resource utilisation values.

For empty servers, the grouping factor is computed to represent a server in an idealised state,

containing only equidistant avatars that are evenly distributed among the servers. For this, the distance from an ideal field count to the current average field count of the actual servers in the running game is calculated:

$$IdealFieldCount = g_0 \frac{P^3}{ServerCount} + g_1 \frac{UsedFields}{ServerCount}$$

The heat map is split into precision P, given as the number of regions per dimensiong_0, g_1 are weights so that $(0 < g \leq 1, g_0 + g_1 = 1)$. The weights represent the balance between two objectives: the even partitioning of the whole virtual environment among the servers and the even distribution of the regions already used among the servers. The optimal values are application specific.

From the ideal field count, a distance value can be computed.

$$Distance = \frac{\sqrt[3]{IdealFieldCount} - \sqrt[3]{AvgFieldCount}}{2} * \frac{1}{P}$$

As a large distance would be bad for grouping avatars, the grouping factor for empty servers is $g(a, s) = -Distance$

All servers that are neither empty nor have a server view box that overlaps with the avatar view box are considered in turn for assignment by setting the cost to the Manhattan Distance from the fields that contain the avatars view box. The position of a given non-empty server is calculated for each field it is assigned in.

6.4.3 Reassignment

After each avatar is assigned to a server, it is likely to move through the virtual environment. The grouping factor of the servers will therefore deteriorate over time. To counter this, my approach supports a protocol for client reassignment. Executing this protocol produces a small overhead. Reassignment may result in a negative impact on the user experience, even if only for a limited time. A certain amount of hysteresis is therefore required to regulate the frequency of reassignments of a given client connection.

In order to track the quality of each assignment, as well as to monitor a user session, the LB maintains a (lightweight Erlang) monitor process for each connected client. In regular intervals, each

process is activated in order to recalculate the server assigned to the client. Should the calculation result in a different server, the client is reassigned.

As the new avatar shifts the aggregate view box of the newly assigned server, the probability of assigning the same server on the recalculation increases. If needed, an additional weight against reassignment could easily be applied to the calculation.

The server will notify the LB when the client has disconnected. This will remove the monitoring process for the client session.

If the distance is greater than a pre-set threshold, a client reassignment process will be triggered. If the assignment results in a different server, the re-connection protocol is executed (see fig. 6.2).

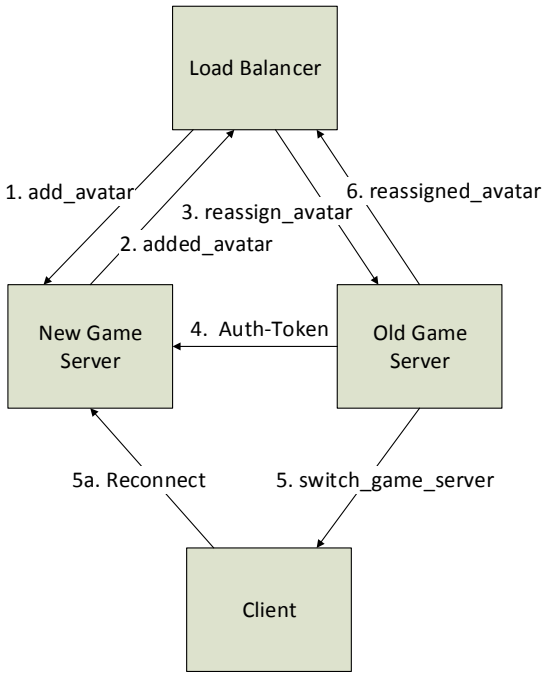


Figure 6.2: Reassignment process

The LB will also re-check the assignment of each avatar after a pre-set interval. This will slowly, but continuously, improve the assignment in the face of avatar movement. While this will regularly query the position of the avatar from the QuP subsystem, it will not suffer the very large bandwidth requirement of tracking each position change of each avatar.

A further method of initiating an assignment reevaluation will be triggered by the server directly. If it detects insufficient CPU, memory or bandwidth resources, it will send an emergency signal to the LB to trigger immediate assignment re-evaluation.

The reassignment protocol will notify the new server to start prefetching all object data required by the avatar. Then the client is informed by the old server of the need to switch. The message will include the contact details of the new server as well as a token string that allows the client to authenticate to the new server.

6.5 Experiments

My research group has implemented a prototype of each of the components shown in fig. 6.3. For the tests, the AS and the game clients were replaced by a test server in order to isolate the LB for measurements. Also, the LB was configured to contact a special QuP mock server in order to separate the time required for computing the server assignment from the time for the QuP look up. The mock was pre-loaded with the position of the avatars, provided no redundancy and was co-located with the LB on the same machine.

The tests were done on three PC type virtual machines. The VM manager was configured to allow full CPU utilisation during the test. The systems were running Erlang version OTP 18 on Ubuntu Linux.

Tests reflect the three types of avatar distribution introduced by Lui and Chan (2002). I consider them sufficient representations of avatar behaviour in a MMOG. The distributions were adapted to three dimensions as supported by QuP. All avatars are placed in a norm cube. Skewed distribution was set to 60% of avatars in a cube occupying the lower 0.3 of each dimension; clustered set to 75% of the avatars being in 10 evenly distributed groups; the remaining avatars were uniformly distributed in the whole environment.

Two test series with 10,000 and 100,000 avatars were set up. The heat map was partitioned with a precision of P=10 blocks per dimension. The smaller scale test was conducted with only three servers in order to allow manual verification of the assignment. The larger one was to test the time requirements based on the number of clients previously assigned. The larger scale test used seven

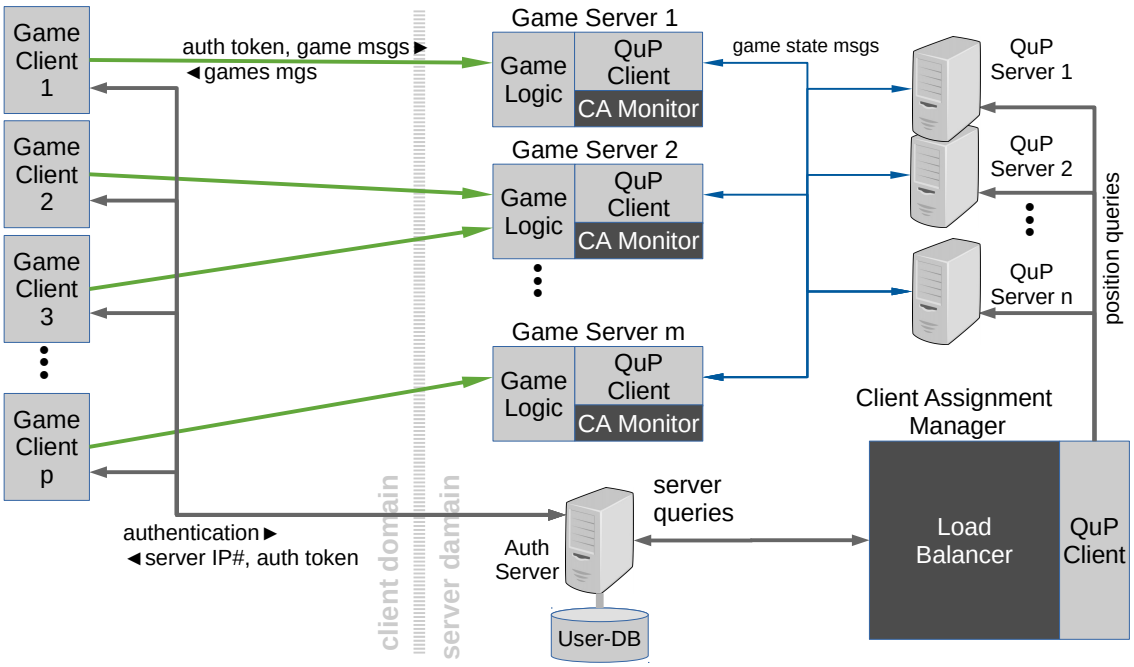


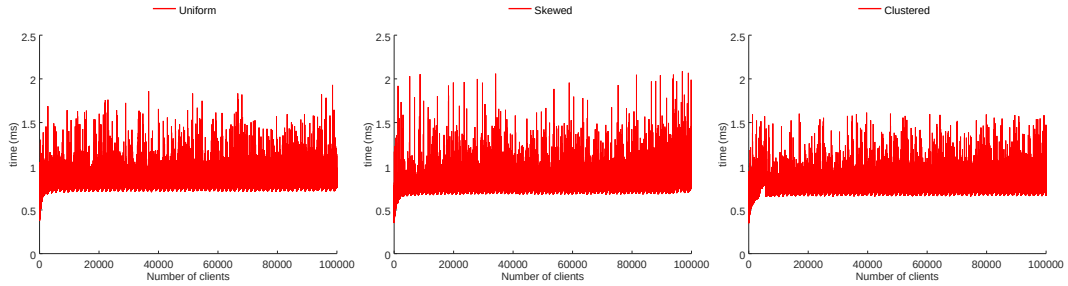
Figure 6.3: proposed Timadorus Architecture

servers to inspect the increase of computation time for an increased number of avatars.

Increasing P will result in a cubic increase of the memory requirements of the LB. I assume that P=10 will partition the space sufficiently to effectively distribute clients to a number of game servers in the order of 10 to 100. Any installation that outgrows this should be able to provide enough resources to support much values of P.

6.6 Results

Figure 6.4 shows the separate results for each of the three distributions when requesting an assignment for each of 100,000 clients to three servers; fig. 6.5 combines them for easier comparison. For each the 99.9th percentile for the time elapsed from a placement request being sent, to the reply given by the LB. Each uses P=10, thus partitioned the VE into 1000 blocks. The measurement does not include the network communication to the QuP servers. Once the required data structures have been set up (due the implementation they are instanced lazily), the mean reply latency is close to



(a) uniform (b) skewed (c) clustered

Figure 6.4: Reply latencies per avatar distribution

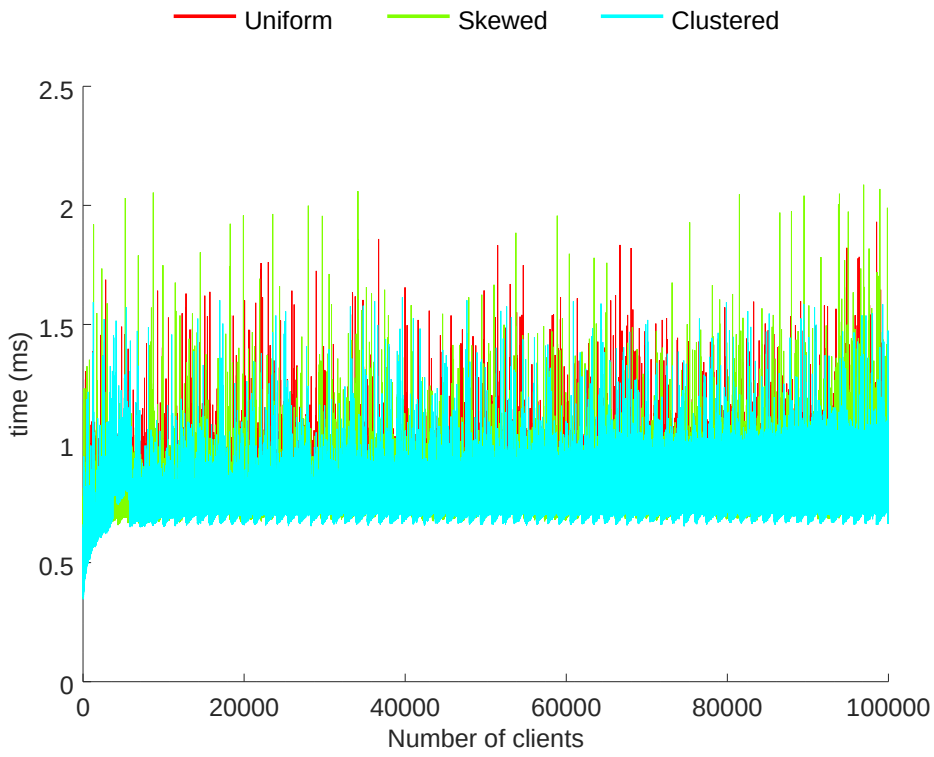


Figure 6.5: Reply times for all distributions, P=10

constant for an increasing number of avatars.

Figure 6.6 shows the distribution of the avatars to three servers in the smaller scale test environment for each of the three distributions to illustrate the utilisation of the same server for avatars that use the same spatial region of the VE.

As the tested number of clients out-paces the previously documented number of avatars in a single instance DVE by roughly one order of magnitude, even a higher than constant latency would still be usable for numbers of avatars far in excess of the number of avatars tested here.

Performing an actual client reassignment incurs a communication overhead for the client and to some extend for both of the participating servers as well. But since the selection of the optimal server is neither in the codepath of the actual game play, not does it require the participation of the client, it may be worthwhile to continuously re-compute the optimal server for each client in a round-robin manner. Whether to act on an updated selection or not is a tuneable parameter and is likely dependent on the applications that is running on the VE. There may also be reasons to never do this as discussed in section 6.7.2

6.7 Conclusion and Future Considerations

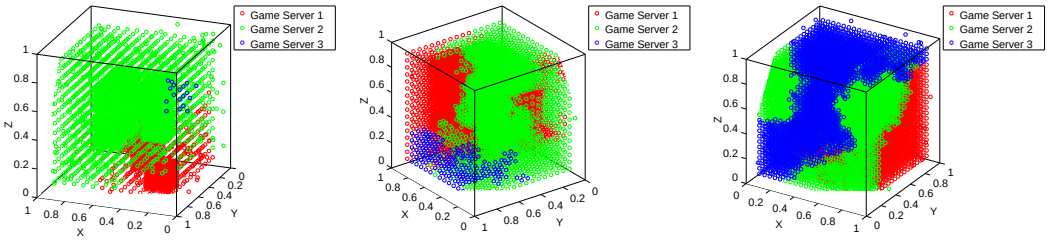
I have shown that in combination with QuP and DyBuBSA (Behnke et al. (2014), Behnke et al. (2015)) this approach to client assignment provides the needed capabilities to drastically scale up the number of supported concurrent users in an MMOG with a single continuous game environment, or distributed virtual environment in general.

For a given set of servers, the load balancer will return an assignment in constant time, regardless of the number of avatars currently active. The only prerequisite is a sufficient amount of memory to maintain the heat map, but this is constant and can be controlled by the number of blocks per dimension.

The quality of the assignment will improve the utilisation rate of the prefetched data on the target server in the normal case. In the worst case, if data in the LB has drifted too much, a less than optimal assignment is given, further increasing the drift distance. If a sufficient drift distance has been detected, a reassignment will be initiated for all avatars concerned.

Sub-optimal assignments will only cause the CPU, memory and bandwidth resources of the server to be depleted by a reduced number of client connection compared to the optimal case. However, the individual client connections even in the sub-optimal case will still receive full quality service.

The reassignment process will minimise the impact on the user experience, but may create small interruptions as the network connection is switched from one server to another.



(a) clustered (b) skewed (c) uniform

Figure 6.6: Avatar Grouping (P=10, 3 servers, 1000 avatars)

6.7.1 Scalability and Fault Tolerance

When scaling up the system, the number assignment requests and reassignment protocol messages will increase with the number of connected game servers and the number of clients connecting to the system concurrently. This will result in an increased network bandwidth requirement placed on the LB. Access to the data structure that holds the heat map and its ancillary data may pose a bottleneck, since it is held on a single compute node and not replicated. It receives updates sent by the monitor processes of each server, as well as all the client assignment requests. More importantly, it creates a single point of failure for the whole system with regards to connecting new users and the continuous improvement of the resource utilisation for the already connected clients.

One member of my research group is currently undertaking further work to eliminate this source of system failure and to support even larger user bases. One possible method is to use a distributed hash table (DHT) to maintain the heat map in a small set of servers, as the changes to the map are generally small and localised. The authentication server would contact the LBs in a round-robin fashion. Thus, the ASs can be duplicated and contacted by the clients in a round robin fashion

themselves. The re-calculation processes and possible re-assignment processes can be separated onto further server nodes as well.

6.7.2 Cost of Reassignment and Operational Considerations

As mentioned in section 6.4.3, it will require considerable computational overhead to continuously update the client assignment and what is probably much worse, each reassignment may result in a user detectable interruption in the event flow, which could result in a jarring break of immersion.

In a service of sufficient size running continuously (24/7) this may not be necessary to do at all. Given the assumption that the game server nodes need to be restarted in order to deploy updates for the game software, each server nodes would have to be drained of all client connections and restarted with the new game code. The communication steps shown in fig. 6.2) still apply, as they be used to drain a running server in a controlled manner, to minimise the impact on the game experience of the clients.

Even if the game code can be build in such a way that a restart is not strictly required (Erlang supports this for use in carrier grade telephony equipment that prove 99.999% of availability), this is hard to do correctly. And this still does not cover the need to update the operating system or any ancillary software components. Modern operations processes (eg. SLOs as described in Beyer et al. (2016)) advise to restart services components on purpose to identify consumers of that services which cannot react to an outage appropriately.

6.7.3 Higher Dimension Heat-Map

We implemented the prototype only to use three dimension for spatial position in Euclidian space. But the data structures would allow the use of higher dimensions. These could be used to further separate servers.

Additional dimensions can be discrete or continuous, the former would eliminate the need for a partitioning. In Rehner et al. (2014) possible applications for higher dimensions are discussed in detail for a P2P type DVE architecture.

Potential applications include separate sets of servers for varying forms of perception (including technical senses like radar or sonar), as not all objects and avatars would be present in all modes

of perception. Or to choose servers that have optimal characteristics external to the DVE, e.g. be based on the location of the user client in the real world. This would allow the assignment of globally distributed datacenters to clients that are in physical proximity, but retain the ability for automatic reassignment in the case of a server or network failure.

Chapter 7

Conclusion

I have shown how to construct a DVE that can support a very large number of concurrent users, while ensuring an upper limit to the bandwidth requirement of each communication channel within the DVE, including the connection to the end client. These limits are configurable parameters within the proposed DVE architecture, thus allowing it to be tailored to a specific application. Depending on the equipment of the users the application attracts, the bandwidth requirements can be optimised, even dynamically (e.g. for mobile users), should the available bandwidth change rapidly.

My system allows the creation of vast virtual vistas and large user populations in seamless environments.

7.1 Achievements

I have presented a system that can overcome the limitations of prior approaches by systematically eliminating a) singletons, which may either become single points of failure or resource bottlenecks, and b) presented a solution to dynamically and locally reduce the amount of required bandwidth for transmitting state updates.

These features help to solve a number of weaknesses identified in section 2.12:

No n^2 communication overhead while maintaining global interaction There is no need to choose between isolating users in shards or zones on one hand and being limited in the maxi-

mum number of cluster nodes before observing diminishing returns.

Large AoI nimbuses and auras handled by build-in aggregation Overloading communication channels, including human perception capabilities, can be avoided by dynamically aggregating data as needed within the nimbus of avatars.

Scaling along multiple independent axis By completely separating the storage location from the processing location as well as the in-DVE spatial position, the DVE can not only be tailored to the needs of application running on it, but also based on the usage characteristics. Re-distribution, not implemented in the prototype, would also allow dynamic scaling with changing usage over time.

Support for very high avatar densities The decoupling of location of processing from the location of storage in *QuP* also allows the handling of the same in-DVE spatial location on multiple server nodes, thus allowing for very high avatar densities in the same location, and aggregation allows the forwarding of state updates, even if all these avatars are all active and are continuously changing the state of the DVE.

One of few forms of interaction that *QuP* and *DyBuBSA* cannot handle are true physically correct n-body systems as can be found e.g. in EVE Online.

7.1.1 Handling Failure

As data in the described system is persisted in a redundant fashion, component failure should lead to data loss only very seldom (e.g. if the data had not been committed to the persistence system yet. If any of the storage nodes fail, they will recover the data as the placement of the chunks is redistributed and reconstructed from the remaining copies. In the case of a network partition, the smaller part will not be able to achieve quorum and thus will fail writing updates.

The system does not aim to provide any protection against bit-rot or software errors as a redundant system in avionics or industrial control systems might do. It rather provides multiple units of processing to the same spatial location within the DVE as a means of supporting more concurrent users within that location than a single node would be able to provide. As the partition of the

storage nodes is independent to the spatial position within the DVE, one or more highly tasked game servers are independent to the peak load of any storage nodes.

If a client connection fails (either due to failing client software or intermittent network failure), it can be reconnected within a few hundred milliseconds to any of the servers that currently handle the spatial area of the AoI and AoE of the avatar. Most of this will time probably be spend reestablishing network connections and handling client authentication, compare to the time spend actually selecting a appropriate server (see section 6.6)). This assumes that the particular servers have available computing and communicating resources remaining. In case this is not the case, the Load Balancer (LB) will start assigning newly (re-)connecting to another available server.

Should a server node fail, the same need for re-connection applies its full set of client connections, as they try to reconnect. Since the LB will also monitor the health of servers, it will simply mark the failed machine as dead assign (re-)connecting clients to other machines.

7.2 Position in Computer Science

With the advent of cloud computing, clusters of mid-sized computing nodes are widely available as an ubiquitous and cost-efficient resource through a pay-for-time procurement. This makes it especially attractive for distributed applications, that are able to scale in size as user traffic increases. And almost as importantly, reduce its resource footprint as user traffic wanes.

But that requires a systematic un-bundling of individual components, in order to eliminate bottlenecks, which would prohibit scaling or squander opportunities for parallel processing or communication. My work furthers the separation of concerns by fully decoupling the game processing node from the persistence node as well as the spatial location within the DVE and node of the cluster to which the client connects.

Through the combination of the dynamic client (re-)assignment, the partitioning of processing is informed, but not controlled by DVE spatial locality, as well as separated and redundant persistence of state, the architecture can scale up and down as well as react to failure of individual components. The persistence utilises established protocols (see section 2.3.10) for robust distributed storage as a foundation for an resilient application.

While *QuP* game server nodes are not completely stateless, this state can be recreated from the persistence layer and information provided by the client application connecting to it. In combination with a redundant load balancer for client assignment, as described in section 6.7.1, the whole system becomes completely fault tolerant to the failure of any individual components.

7.3 Transferability and Generalisability

There are three basic avenues of further work that open with the current results:

First the current prototype offers a number of areas for further optimisation that will provide faster replies and more efficient use of computational resource. For comparison, QuP will require around 2 msec for a round trip to write an object attribute. Riak, a commercial database system, using a similar mechanism for distributing changes and ensuring fault tolerance, required around $450\mu sec$ in benchmarks. This effort primarily poses an engineering and development problem, rather than an actual research effort. But it would allow QuP to be used to serve as the foundation for DVE research in higher abstraction layers for creating DVE applications. Especially research in programming models, methods and tools to operate in DVEs have multiple tens of thousands of entities.

Current work like the OpenSimulation (OpenSim-Project, 2013) often is restricted by the fact that active objects in the same spatial area have to be processed by the same process (see section 2.8.3 discusses work to alleviate this problem). QuP does not suffer from this, but does not provide an easy-to.use programming model or tools to support application development. The asynchronous interface does not lend itself to a programming style that is commonly used, like imperative or object orient programming.

Secondly, QuP is intended and designed to be an overestimating heuristic. For any gaming application, it should be sufficient when this is true for a high percentage of select operations (far beyond 99%). But if it can be shown that this is always true, even when faced with high relative spatial velocities within the VE, then QuP could be used as a selection mechanism for Parallel Discrete Event Simulations like MARS (Hüning, 2016). This would allow the automatic partitioning of the problem and computation of interactions only for simulation elements that are

actually within their respective zones of interaction. It is also expected to allow simpler modeling of simulation environments or at least the required programming, as the partition and distribution on multiple computational resources can be abstracted away completely. Current humanitarian oriented research like Collins and Frydenlund (2016) often does not include the effects of geography and vegetation, because the required data is unwieldy to use in the tools used by the researchers. As the required information on topology is available from satellite data, *QuP* could aid in restricting data sets by spatial locality.

Thirdly, the distributed nature of the aggregation framework described in chapter 5 would lend itself ideally to generating specific media streams for avatars in the same spatial location. (e.g. to support applications like Chen et al. (2014)). As the aggregation can be turned on and off, based on subscription, the use of processing and network resources is optimised. By weighing the input to the aggregation, a rock concert scenario is straightforward to implement: each audio mixer would receive the band playing, as its audio volume is sufficiently high. But (yelled) discussion between visitors would only be audible within a very small area and thus need not be transmitted between many spatial regions within the DVE.

Generalising the rock concert scenario as input to the *DyBuBSA* control mechanism, clearly documents the utility of a multi-modal AoI system, which *QuP* supports (by instantiating multiple octrees and tracking the appropriate object value fields). The audio aura of the band (or rather the speakers) would be very large, its nimbus would be rather small: members of the band can only hear each other and the crowd only as a whole), while on the other hand, the visually, the band can percieve the crowd only as amalgamation of the individual visitors, but crowd and band can see each other over quite distance. Both have large nimbuses and auras that correspond to the physical extend of either.

7.4 Further Work

One area my current work does not support is the establishment of a single DVE instance for a global audience. This is hard to support due to the resultant latency issues. The speed of light takes ¿100msec for a single round trip of the world without any additional processing overhead. This

makes the delay observable to most humans. As a result, current commercial services usually deploy instances of their DVEs in North America, Europe and south-east Asia, to enable latencies below that threshold.

Current AAA commercial game offering still rely on a number technical tricks to hide the fact that subsequent parts of the game world are loaded from local storage or over the network, while the player is traversing it. Often this is not done with primitives like the Prisms used in OpenSim-Project (2013), but rather using more complex geometric objects. The reason for this is the faster rendering as well as more space efficient encoding for transport, using modern GPUs. But this firmly entrenches the split between pre-baked, static objects with complex geometries and high definition surface definitions on one hand, and the relatively few objects that actually change within the game world. With the aid of *DyBuBSA* this can be overcome as objects that are not currently being changed could be baked on the server for future transmission to the game client to be rendered efficiently.

Since the start of the implementation of the prototype, a number of improvements to distributed consensus algorithms with Raft (Ongaro, 2014) or gossip protocols like SWIM (Das et al., 2002) with Lifeguard (Dadgar et al., 2018) have been published. They promise numerous improvement to the stability and performance of the storage backend, especially scaled out installation. The distributed persistence backend of *QuP* had not been the main topic of this thesis and could benefit from a more in-depth analysis. But Raft does not protected against stale reads. Further work to evaluate if this is really needed to support the correctness requirements described in section 3.3.4.

Bibliography

Abadi, D., Ahmad, Y. and Balazinska, M. (2005), The design of the borealis stream processing engine, *in* 'Conference on Innovative Data Systems Research'.
URL: `http://www.cidrdb.org/cidr2005/papers/P23.pdf`

Ahmed, D. T. and Shirmohammadi, S. (2008), A dynamic area of interest management and collaboration model for P2P MMOGs, *in* 'Proceedings - 12th 2008 IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2008', Ieee, pp. 27–34.
URL: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4700100`

Ahmed, D. T. and Shirmohammadi, S. (2012), 'Improving online gaming experience using location awareness and interaction details', *Multimedia Tools and Applications* **61**(1), 163–180.
URL: `http://www.springerlink.com/index/A853Q65037474514.pdf`

Ahmed, D. T., Shirmohammadi, S. and De Oliveira, J. C. (2009), 'A hybrid P2P communications architecture for zonal MMOGs', *Multimedia Tools and Applications* **45**(1-3), 313–345.

Al-Gharaibeh, J. and Jeffery, C. (2010*a*), 'PNQ: Portable Non-player Characters with Quests', *2010 International Conference on Cyberworlds* pp. 294–301.
URL: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5655000`

Al-Gharaibeh, J. and Jeffery, C. (2010*b*), 'Portable non-player character tutors with quest activities', *2010 IEEE Virtual Reality Conference (VR)* pp. 253–254.
URL: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5444779`

Assiotis, M. and Tzanov, V. (2006), A distributed architecture for MMORPG, *in* 'Proceedings of

5th ACM SIGCOMM workshop on Network and system support for games', ACM Press, New York, New York, USA, p. 4.

URL: http://portal.acm.org/citation.cfm?doid=1230040.1230067http://portal.acm.org/citation.cfm?id=1230067http://portal.acm.org/citation.cfm?id=1230040.1230067

Backhaus, H. and Krause, S. (2010), 'QuON: a quad-tree-based overlay protocol for distributed virtual worlds', *International Journal of Advanced Media and Communication* **4**(2), 126.

URL: http://www.inderscience.com/link.php?id=32139

Bartle, R. (2004*a*), *Designing Virtual Worlds*, Vol. p of *New Riders Games Series*, New Riders Games.

Bartle, R. (2004*b*), Its Not a Game, Its a..., *in* 'Designing Virtual Worlds', first edn, New Riders Publishing, Berkeley, California, chapter Chapter 6, pp. 3–21.

Bartle, R. (2004*c*), What They Re and Whence They Came, *in* 'Designing Virtual Worlds', first edn, New Riders Publishing, Berkeley, California, chapter Chapter 1, pp. 3–21.

Bartle, R. E. (2015*a*), *MMOs from the Inside Out: The History, Design, Fun, and Art of Massively-multiplayer Online Role-playing Games*, 1st. edn, Apress.

Bartle, R. E. (2015*b*), *MMOs from the Outside In: The Massively-Multiplayer Online Role-Playing Games of Psychology, Law, Government, and Real Life*, Apress.

Basho Inc. (2013), 'Riak'.

URL: http://basho.com/riak/

Behnke, L., Allers, S., Wang, Q., Grecos, C. and von Luck, K. (2016), Avatar Density Based Client Assignment, *in* G. Wallner, S. Kriglstein, H. Hlavacs, R. Malaka, A. Lugmayr and H.-S. Yang, eds, '15th IFIP TC 14 International Conference', Vol. 9926 of *Lecture Notes in Computer Science*, Springer International Publishing, Vienna, pp. 137–148.

URL: http://link.springer.com/10.1007/978-3-319-46100-7

Behnke, L., Grecos, C. and Luck, K. V. (2014), QuP: Graceful Degradation in State Propagation for DVEs, *in* 'Proceedings of International Workshop on Massively Multiuser Virtual Environments',

ACM Press, Singapore.

URL: http://dl.acm.org/citation.cfm?id=2577389

Behnke, L., Wang, Q., Grecos, C. and von Luck, K. (2015), Budget based dynamic state update aggregation, *in* 'Proceedings of the 7th ACM International Workshop on Massively Multiuser Virtual Environments - MMVE '15', Portland, Oregon, pp. 25–26.

URL: http://dl.acm.org/citation.cfm?doid=2723695.2723696

Benford, S. (1993), A Spatial Model of Interaction in Large Virtual Environments, *in* G. de Michelis, C. Simone and K. Schmidt, eds, 'Proceedings of the 3rd European Conference on Computer-Supported Cooperative Work (ECSCW'93)', Springer Netherlands, Dordrecht, pp. 109–124.

URL: http://link.springer.com/10.1007/978-94-011-2094-4

Beyer, B., Jones, C., Petoff, J. and Murphy, N. R. (2016), *Site Reliability Engineering: How Google Runs Production Systems*, 1st edn, OReilly Media, Inc.

Bharambe, A., Douceur, J., Lorch, J., Moscibroda, T., Pang, J., Seshan, S. and Zhuang, X. (2008), 'Donnybrook: Enabling large-scale, high-speed, peer-to-peer games', *ACM SIGCOMM Computer Communication Review* **38**(4), 389–400.

URL: http://dl.acm.org/citation.cfm?id=1403002

Bharambe, A., Pang, J. and Seshan, S. (2006), Colyseus: A Distributed Architecture for Online Multiplayer Games., *in* 'NSDI'06 Proceedings of the 3rd conference on Networked Systems Design & Implementation', Vol. 3, pp. 12–12.

URL: http://www.usenix.org/events/nsdi06/tech/full_papers/bharambe/bharambe_html/http://www.usenix.org/event/nsdi06/tech/full_papers/bharambe/bharambe_html/

Bharambe, A., Rao, S. and Seshan, S. (2002), Mercury: a scalable publish-subscribe system for internet games, *in* '1st Workshop on Network and Systems Support for Games (NetGames '02)', ACM, pp. 3–9.

URL: http://dl.acm.org/citation.cfm?id=566501

Blackburn, J., Simha, R., Kourtellis, N., Zuo, X., Long, C., Ripeanu, M., Skvoretz, J. and Iamnitchi, A. (2011), 'Cheaters in the Steam Community Gaming Social Network', *ArXiv e-prints* (dec).

URL: http://adsabs.harvard.edu/abs/2011arXiv1112.4915Bhttp://arxiv.org/abs/1112.4915http://adsabs.harvard.edu/abs/2011arXiv1112.4915B://arxiv.org/abs/1112.4915

Blackman, T. and Waldo, J. (2009), Scalable Data Storage in Project Darkstar, Technical report, Sun Microsystems Laboratories.

URL: https://dl.acm.org/doi/book/10.5555/1698219

BlizzardActivision Inc. 2016 Annual Report (2016).

URL: http://investor.activision.com/common/download/download.cfm?companyid=ACTI&fileid=938514&filekey=5075E521-4A3F-456C-8C2B-9EEF50DD69FE&filename=Activision_2016_AR_FINAL_Spreads.pdf

Boukerche, A. and Ahmad, L. (2008), A Scalable Message Passing and Synchronized Technique for Large Scale Distributed Virtual Environments Based on Peer to Peer Networks, *in* 'International Workshop on Haptic AudioVisual Environments and Games'.

Boulanger, J. (2006), Interest management for massively multiplayer games, Master thesis, McGill University.

URL: http://cs.mcgill.ca/$\sim$jboula2/thesis.pdf

Boulanger, J., Kienzle, J. and Verbrugge, C. (2006), Comparing interest management algorithms for massively multiplayer games, *in* 'Workshop on Network and Systems Support for Games (NetGames '06)'.

URL: http://dl.acm.org/citation.cfm?id=1230069

Bowman, C., Lake, D. and Hurliman, J. (2010), 'Designing Extensible and Scalable Virtual World Platforms', *Workshop on Extensible Virtual Worlds* .

URL: http://vw.ddns.uark.edu/X10/content/ARCHITECTURE--Designing-Extensible-and-Scalable-Vir
pdf

Brewer, E. (2012), 'CAP twelve years later: How the "rules" have changed', *Computer* **45**(2), 23–29.

URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6133253https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed

Castro, M., Druschel, P., Kermarrec, A.-M. and Rowstron, A. (2002), 'SCRIBE : A large-scale and decentralized application-level multicast infrastructure', *Selected Areas in Communications, IEEE Journal on* **20**(8), 1489 – 1499.
URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1038579&queryText=scribe

CCP Inc. (2013), 'A Weekend of Epic Destruction in EVE Online'.
URL: http://community.eveonline.com/news/dev-blogs/a-weekend-of-epic-destruction-in-eve-online/ http://www.webcitation.org/6dK9gvgpD

CCP Veritas (2011), 'Introducing Time Dilation'.
URL: https://community.eveonline.com/news/dev-blogs/introducing-time-dilation-tidi/

Chan, A., Lau, R. W. H. and Ng, B. (2001), A hybrid motion prediction method for caching and prefetching in distributed virtual environments, *in* 'Proceedings of the ACM symposium on Virtual reality software and technology - VRST '01', ACM Press, New York, New York, USA, p. 135.
URL: http://portal.acm.org/citation.cfm?doid=505008.505035

Chan, L., Yong, J., Bai, J., Leong, B. and Tan, R. (2007), Hydra: a massively-multiplayer peer-to-peer architecture for the game developer, *in* 'Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games', ACM, pp. 37–42.
URL: http://portal.acm.org/citation.cfm?id=1326264

Chen, J., Wu, B., Delap, M., Knutsson, B., Lu, H. and Amza, C. (2005), 'Locality aware dynamic load management for massively multiplayer games', *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming - PPoPP '05* p. 289.
URL: http://portal.acm.org/citation.cfm?doid=1065944.1065982

Chen, S., Nahrstedt, K. and Gupta, I. (2014), 3dti amphitheater: A manageable 3dti environment with hierarchical stream prioritization, *in* 'Proceedings of the 5th ACM Multimedia Systems Conference', MMSys '14, ACM, New York, NY, USA, pp. 70–80.
URL: http://doi.acm.org/10.1145/2557642.2557654

Claypool, M. and Claypool, K. (2006), 'Latency and player actions in online games', *Communications of the ACM* **49**(11), 40.
URL: http://portal.acm.org/citation.cfm?doid=1167838.1167860

Claypool, M. and Claypool, K. (2010), Latency can kill: precision and deadline in online games, *in* 'Proceedings of the first annual ACM SIGMMSys 2010'.
URL: http://dl.acm.org/citation.cfm?id=1730863

Collins, A. J. and Frydenlund, E. (2016), Agent-based modeling and strategic group formation: A refugee case study, *in* '2016 Winter Simulation Conference (WSC)', IEEE, pp. 1289–1300.
URL: http://ieeexplore.ieee.org/document/7822184/

Consalvo, M. (2007), *Cheating: Gaining Advantage in Videogames*, 1st edn, MIT Press, Cambridge, Mass.
URL: https://books.google.de/books?hl=de{&}lr={&}id=XbcNAUxUajAC{&}oi=fnd{&}pg=PR5{&}dq=cheating+consalvo{&}ots=F34iEkhPEY{&}sig=bCy2Mb42azaTY{_}FVH4XvdMP0dFY

Dadgar, A., Phillips, J. and Currey, J. (2018), 'Lifeguard: Local Health Awareness for More Accurate Failure Detection', *Proceedings - 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN-W 2018* pp. 22–25.

Das, A., Gupta, I. and Motivala, A. (2002), SWIM: scalable weakly-consistent infection-style process group membership protocol, *in* 'Proceedings International Conference on Dependable Systems and Networks', IEEE Comput. Soc, pp. 303–312.
URL: http://ieeexplore.ieee.org/document/1028914/

de Oliveira, J. and Georganas, N. (2002), 'VELVET: an adaptive hybrid architecture for very large virtual environments', *2002 IEEE International Conference on Communications. Conference Proceedings. ICC 2002 (Cat. No.02CH37333)* **4**, 2491 – 2495 vol.4.

De Vleeschauwer, B., Van Den Bossche, B., Verdickt, T., De Turck, F., Dhoedt, B. and Demeester, P. (2005), Dynamic microcell assignment for massively multiplayer online gaming, *in* 'Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games - NetGames '05',

ACM Press, New York, New York, USA, p. 1.
URL: http://portal.acm.org/citation.cfm?doid=1103599.1103611

Debeauvais, T., Valadares, A. and Lopes, C. V. (2011), 'RCAT: A RESTful client-scalable architecture', *2011 10th Annual Workshop on Network and Systems Support for Games* pp. 1–2.
URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6080987

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. and Vogels, W. (2007), 'Dynamo: amazon's highly available key-value store', *ACM SIGOPS Operating Systems Review* **41**(6), 205–220.
URL: http://dl.acm.org/citation.cfm?id=1294281

Delaney, D., Ward, T. and McLoone, S. (2006), 'On Consistency and Network Latency in Distributed Interactive Applications: A SurveyPart I', *Presence: Teleoperators and Virtual Environments* **15**(2), 218–234.
URL: http://www.mitpressjournals.org/doi/abs/10.1162/pres.2006.15.2.218http://www.mitpressjournals.org/doi/10.1162/pres.2006.15.2.218

Deng, Y. and Lau, R. W. H. (2014), 'Dynamic load balancing in distributed virtual environments using heat diffusion', *ACM Transactions on Multimedia Computing, Communications, and Applications* **10**(2), 1–19.
URL: http://dl.acm.org/citation.cfm?doid=2579228.2499906

Duong Nguyen Binh, T., Suiping, Z. and Haifeng Shen, S. (2006), 'Greedy Algorithms for Client Assignment in Large-Scale Distributed Virtual Environments', *Principles of Advanced and Distributed Simulation 2006 PADS 2006 20th Workshop on* **84**(10-11), 103–110.
URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1630714

Ericson, C. (2005), *Real-Time Collision Detection*, Morgan Kaufmann, San Francisco.

Falkenauer, E. (1994), 'A New Representation and Operators for Genetic Algorithms Applied to Grouping Problems', *Evolutionary Computation* **2**(2), 123–144.
URL: http://www.mitpressjournals.org/doi/abs/10.1162/evco.1994.2.2.123

Fan, L., Trinder, P., Taylor, H. and Management, A. I. (2009), Design Issues for Peer-to-Peer Massively Multiplayer Online Games, *in* 'The 2nd International Workshop on Massively Multiuser Virtual Environments at IEEE Virtual Reality 2009'.

Farooq, U. and Glauert, J. (2009), 'Joint Hierarchical Nodes based User Management (JoHNUM) Infrastructure for the Development of Scalable and Consistent Virtual Worlds', *System* .

Fox, A. and Brewer, E. (1999), Harvest, yield, and scalable tolerant systems, *in* 'Proceedings of the Seventh Workshop on Hot Topics in Operating Systems', IEEE Comput. Soc, pp. 174–178.
URL: `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=798396&matchBoolean=true&rowsPerPage=30&searchField=Search_All&queryText=(p_DOI:10.1109/HOTOS.1999.798396)http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=798396`

Fritsch, T., Ritter, H. and Schiller, J. (2005), 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', ACM, pp. 1–9.
URL: `http://dl.acm.org/citation.cfm?id=1103623`

Fujimoto, R. M. (2000), *Parallel and distributed simulation systems*, John Wiley & Sons, Inc.

Funkhouser, T. (1996), Network topologies for scalable multi-user virtual environments, *in* 'VRAIS '96 Proceedings of the 1996 Virtual Reality Annual International Symposium (VRAIS 96)', p. 222.
URL: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.50.77&rep=rep1&type=pdf`

Funkhouser, T. A. (1995), RING : A Client-Server System for Multi-User Virtual Environments, *in* 'ACM Symposium on Interactive 3D Graphics', pp. 85–92.

GamesFirst (2011), 'Do we need some sort of 'cheater amnesty' as we crack down on cheaters?'.
URL: `http://apbreloaded.gamersfirst.com/2011/09/do-we-need-some-sort-of-cheater-amnesty_30.html`

Gascon-Samson, J., Kienzle, J. and Kemme, B. (2015), DynFilter: Limiting bandwidth of online games using adaptive pub/sub message filtering, *in* '2015 International Workshop on Network and Systems Support for Games (NetGames)', IEEE, pp. 1–6.

URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7382988

GauthierDickey, C., Lo, V. and Zappala, D. (2005), Using n-trees for scalable event ordering in peer-to-peer games, *in* 'Proceedings of the international workshop on Network and operating systems support for digital audio and video - NOSSDAV '05', ACM Press, New York, New York, USA, p. 87.

URL: http://dx.doi.org/10.1145/1065983.1066005http://portal.acm.org/citation.cfm?doid=1065983.1066005

Gilbert, S. and Lynch, N. (2002), 'Brewer s Conjecture and the Feasibility of Consistent , Available , Partition-Tolerant Web Services', *SIGACT News* **33**(2), 48–51.

URL: http://lpd.epfl.ch/sgilbert/pubs/BrewersConjecture-SigAct.pdf

Glinka, F. (2007), RTF: a real-time framework for developing scalable multiplayer online games, *in* 'NetGames 2007', pp. 81–86.

URL: http://portal.acm.org/citation.cfm?id=1326272

Glinka, F., Ploss, A., Gorlatch, S. and Müller-Iden, J. (2008), 'High-Level Development of Multi-server Online Games', *International Journal of Computer Games Technology* **2008**, 1–16.

URL: http://www.hindawi.com/journals/ijcgt/2008/327387/

Greenhalgh, C. (1998), 'Awareness-based communication management in the MASSIVE systems', *Distributed Systems Engineering* **129**, 129–137.

URL: http://iopscience.iop.org/0967-1846/5/3/006

Gupta, N., Demers, A., Gehrke, J., Unterbrunner, P. and White, W. (2009), 'Scalability for Virtual Worlds', *2009 IEEE 25th International Conference on Data Engineering* pp. 1311–1314.

URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4812528

Hagsand, O., Lea, R. and Stenius, M. (1997), Using Spatial Techniques to Decrease Message Passing

in a Distributed VR System, *in* 'VRML 97 Second Symposium on the Virtual Reality Modeling Language', pp. 7–15.

Heger, F. (2013), Scalable Propagation of Continuous Actions in Peer-to-Peer-based Massively Multiuser Virtual Environments : The Continuous Events Approach, PhD thesis, Univ. Mannheim.

Hoogen, I. V. D. (2004), 'Deutsch's Fallacies, 10 Years After'.
URL: `https://web.archive.org/web/20171015074801/http://java.sys-con.com/read/38665.htm`

Horn, D., Cheslack-Postava, E., Azim, T., Freedman, M. J. and Levis, P. (2009), 'Scaling Virtual Worlds with a Physical Metaphor', *IEEE Pervasive Computing* pp. 50–54.

Horn, D., Cheslack-Postava, E. and Mistree, B. (2010), To infinity and not beyond: Scaling communication in virtual worlds with Meru, Technical report, Stanford University.
URL: `http://ewencp.net/papers/to_infinity_and_not_beyond_tr.pdf`

Hu, S. Y. (2009), Spatial publish subscribe, *in* 'IEEE Virtual Reality (IEEE VR) Workshop MMVE'.
URL: `http://pap.vs.uni-due.de/MMVE09/papers/p8.pdf`

Hu, S.-Y. and Liao, G.-M. (2004), Scalable peer-to-peer networked virtual environment, *in* 'ACM SIGCOMM 2004 workshops on NetGames '04', ACM, pp. 129–133.

Hu, S.-Y., Wu, C., Buyukkaya, E., Chien, C.-h., Lin, T.-H., Abdallah, M., Jiang, J.-R. and Chen, K.-T. (2010), A spatial publish subscribe overlay for massively multiuser virtual environments, *in* '2010 International Conference on Electronics and Information Engineering', Vol. 2, IEEE, pp. V2–314–V2–318.
URL: `http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:VAST+:+A+Spatial+Publish+Subscribe+Overlay+for+Massively+Multiuser+Virtual+Environments#0http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5559789`

Hüning, C. (2016), Analysis of Performance and Scalability of the Cloud-Based Multi-Agent System MARS, Master's thesis, HAW Hamburg, Berliner Tor 5, 20099 Hamburg.

IEEE Standard for Modeling and Simulation High Level Architecture (HLA) (2010).

Iimura, T., Hazeyama, H. and Kadobayashi, Y. (2004), Zoned federation of game servers, *in* 'Proceedings of ACM SIGCOMM 2004 workshops on NetGames '04 Network and system support for games - SIGCOMM 2004 Workshops', ACM Press, New York, New York, USA, p. 116.
URL: http://portal.acm.org/citation.cfm?doid=1016540.1016549

Inc., L. (n.d.), 'SecondLife'.
URL: http://www.secondlife.com

Jardine, J. and Zappala, D. (2008), A hybrid architecture for massively multiplayer online games, *in* 'Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games - NetGames '08', ACM Press, New York, New York, USA, p. 60.
URL: http://portal.acm.org/citation.cfm?doid=1517494.1517507

Jinzhong, W. (2011), 'On packet aggregation mechanism for improving Massive Multiplayer Online Game performance in P2P network', *2011 3rd International Conference on Computer Research and Development* pp. 337–339.
URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5764145

Kain, E. (2014), ''World Of Warcraft' Tops 10 Million Subscribers Following 'Warlords Of Draenor' Expansion'.
URL: http://www.forbes.com/sites/erikkain/2014/11/19/world-of-warcraft-tops-10-million-subscribers

Kazem, I., Ahmed, D. T. and Shirmohammadi, S. (2007), A Visibility-Driven Approach to Managing Interest in Distributed Simulations with Dynamic Load Balancing, *in* '11th IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'07)', IEEE, pp. 31–38.
URL: http://ieeexplore.ieee.org/document/4384527/

Kesselman, J. (2011), 'Networked Game Development'.

Khuri, S. and Chiu, T. (1997), 'Heuristic algorithms for the terminal assignment problem', *Proceedings of the 1997 ACM symposium on Applied computing - SAC '97* pp. 247–251.
URL: http://portal.acm.org/citation.cfm?doid=331697.331748

Klauß, F. (2015), Verteilte Transaktionen auf Basis von asynchronen Nachrichten in einem Eventual Consistency Persistenzsystem, Bachelor, HAW Hamburg.

URL: http://edoc.sub.uni-hamburg.de/haw/volltexte/2015/3042/

Knutsson, B., Lu, H., Xu, W. and Hopkins, B. (2004), Peer-to-peer support for massively multiplayer games, *in* 'INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies', Vol. 1, Ieee, pp. 96–107.

URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1354485http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1354485

Kwok, M. (2006), Performance Analysis of Distributed Virtual Environments, Doctoral, University of Waterloo.

URL: http://www.collectionscanada.gc.ca/obj/s4/f2/dsk3/OWTU/TC-OWTU-879.pdf

Lake, D., Bowman, M. and Liu, H. (2010), 'Distributed scene graph to enable thousands of interacting users in a virtual environment', *2010 9th Annual Workshop on Network and Systems Support for Games, NetGames 2010* pp. 2–7.

Lau, R. W. (2010), Hybrid load balancing for online games, *in* 'Proceedings of the international conference on Multimedia - MM '10', ACM Press, New York, New York, USA, p. 1231.

URL: http://dl.acm.org/citation.cfm?doid=1873951.1874194

Lau, R. W. H., Chim, J. H. P., Green, M., Leong, H. V. and Si, A. (2001), 'Object caching and prefetching in distributed virtual walkthrough', *Real-Time Systems* **21**(1/2), 143–164.

URL: http://www.scopus.com/inward/record.url?eid=2-s2.0-0035395899&partnerID=40&md5=5ca6a659fe572312f9bae218ba4b7c62http://link.springer.com/10.1023/A:1011199405471

Lee, D., Lim, M., Han, S. and Lee, K. (2007), 'ATLAS: A Scalable Network Framework for Distributed Virtual Environments', *Presence Teleoperators Virtual Environments* **16**(2), 125–156.

URL: http://www.mitpressjournals.org/doi/abs/10.1162/pres.16.2.125

Lee, K. and Lee, D. (2003), A scalable dynamic load distribution scheme for multi-server distributed virtual environment systems with highly-skewed user distribution, *in* 'Proceedings of the ACM

symposium on Virtual reality software and technology - VRST '03', ACM Press, New York, New York, USA, p. 160.

URL: http://portal.acm.org/citation.cfm?doid=1008653.1008681

Li, Y. and Cai, W. (2012), Consistency aware update schedule in multi-server Distributed Virtual Environments, *in* 'Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques', ICST, p. 3.

URL: http://dl.acm.org/citation.cfm?id=1808143.1808147http://eudl.eu/doi/10.4108/ICST.SIMUTOOLS2010.8669

Liu, E. S. and Theodoropoulos, G. K. (2011), 'A Parallel Interest Matching Algorithm for Distributed-Memory Systems', *2011 IEEE/ACM 15th International Symposium on Distributed Simulation and Real Time Applications* pp. 36–43.

URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6051801

Liu, H., Bowman, M. and Chang, F. (2012), 'Survey of state melding in virtual worlds', *ACM Computing Surveys* **44**(4), 1–25.

URL: http://dl.acm.org/citation.cfm?doid=2333112.2333116

Liu, H. L. H. and Bowman, M. (2010), 'Scale Virtual Worlds through Dynamic Load Balancing', *Distributed Simulation and Real Time Applications (DS-RT), 2010 IEEE/ACM 14th International Symposium on* pp. 43–52.

URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5636721

Liu, Huaiyu, Bowman, Mic, Adams, Robert, Hurliman, John and Lake, Dan (2010), Scaling virtual worlds: Simulation requirements and challenges, *in* B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan and E. Yücesan, eds, 'Proceedings of the 2010 Winter Simulation Conference', pp. 778–790.

URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5679112

Liu, L., Jones, A., Antonopoulos, N., Ding, Z. and Zhan, Y. (2015), 'Performance evaluation and simulation of peer-to-peer protocols for Massively Multiplayer Online Games', *Multimedia Tools and Applications* **74**(8), 2763–2780.

Lu, F., Parkin, S. and Morgan, G. (2006), Load balancing for massively multiplayer online games, *in* 'Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games - NetGames '06', ACM Press, New York, New York, USA, p. 1.

URL: http://doi.acm.org/10.1145/1230040.1230064http://portal.acm.org/citation.cfm?doid=1230040.1230064

Lui, J. (2001), 'Constructing communication subgraphs and deriving an optimal synchronization interval for distributed virtual environment systems', *IEEE Transactions on Knowledge and Data Engineering* **13**(5), 778–792.

URL: http://ieeexplore.ieee.org/document/956100/

Lui, J. and Chan, M. (2002), 'An efficient partitioning algorithm for distributed virtual environment systems', *IEEE Transactions on Parallel and Distributed Systems* **13**(3), 193–211.

Macedonia, M. and Zyda, M. (1997), 'A taxonomy for networked virtual environments', *IEEE Multimedia* **4**(1), 48–56.

URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=580395

Mauve, M., Hilt, V., Kuhmunch, C. and Effelsberg, W. (2001), 'RTP/I-toward a common application level protocol for distributed interactive media', *IEEE Transactions on Multimedia* **3**(1), 152–161.

URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.7302http://ieeexplore.ieee.org/document/909602/

Meagher, D. (1980), Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer, Technical Report Technical Report IPL-TR-80-111, Rensselaer Polytechnic Institute.

Miller, J. L. (2011), Distributed virtual environment scalability and security, Technical Report UCAM-CL-TR-809, University of Cambridge, Computer Laboratory.

URL: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-809.pdf

Miller, J. L. and Crowcroft, J. (2010), The near-term feasibility of P2P MMOG's, *in* '2010 9th Annual Workshop on Network and Systems Support for Games', IEEE, pp. 1–6.

URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5679578http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5679578

Morillo, P., Fernandez, M. and Orduna, J. (2003), An ACS-based partitioning method for distributed virtual environment systems, *in* 'Proceedings International Parallel and Distributed Processing Symposium', Vol. 00, IEEE Comput. Soc, p. 8.
URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1213283

Morillo, P., Orduna, J. M. and Duato, J. (2006), A scalable synchronization technique for distributed virtual environments based on networked-server architectures, *in* 'Proceedings of the International Conference on Parallel Processing Workshops', RAND Europe/Ofcom, pp. 74–81.

Morillo, P., Orduña, J. M., Fernández, M. and Duato, J. (2003), An adaptive load balancing technique for distributed virtual environment systems, *in* 'Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems (PCDS'03)', Vol. 15, pp. 256–261.
URL: http://www.scopus.com/inward/record.url?eid=2-s2.0-1542537865{&}partnerID=tZOtx3y1

Morse, K. L., Bic, L. and Dillencourt, M. (2000), 'Interest Management in Large-Scale Virtual Environments', *Presence: Teleoperators and Virtual Environments* **9**(1), 52–68.
URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.50.2493&rep=rep1&type=pdfhttp://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1242993http://www.mitpressjournals.org/doi/abs/10.1162/105474600566619

Müller-Iden, J. (2007), Replication-based Scalable Parallelization of Virtual Environments, PhD thesis.

Najaran, M., Hu, S. and Hutchinson, N. (2014), SPEX: scalable spatial publish/subscribe for distributed virtual worlds without borders, *in* 'Proceedings of the 5th ACM Multimedia Systems Conference (MMSys)', ACM Press.
URL: http://dl.acm.org/citation.cfm?id=2557655

Najaran, M. T. and Krasic, C. (2010), 'Scaling online games with adaptive interest management in

the cloud', *2010 9th Annual Workshop on Network and Systems Support for Games* pp. 1–6.
URL: http://portal.acm.org/citation.cfm?id=1944805

Ng, B., Si, A., Lau, R. W. and Li, F. W. (2002), A multi-server architecture for distributed virtual walkthrough, *in* 'Proceedings of the ACM symposium on Virtual reality software and technology - VRST '02', ACM Press, New York, New York, USA, p. 163.
URL: http://portal.acm.org/citation.cfm?doid=585740.585768

Norvig, P. (2001), 'Teach Yourself Programming in Ten Years'.
URL: http://norvig.com/21-days.html

Ongaro, D. (2014), Consensus: Bridging Theory and Practice, PhD thesis, Standord University.
URL: http://purl.stanford.edu/qr033xr6097https://github.com/ongardie/dissertation/

OpenSim-Project (2013), 'Open Simulator Project'.
URL: http://www.opensim.org

Park, K. and Kenyon, R. (1999), Effects of network characteristics on human performance in a collaborative virtual environment, *in* 'Virtual Reality, 1999. Proceedings., IEEE', IEEE, pp. 104–111.
URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=756940

Park, S., Lee, D., Lim, M. and Yu, C. (2001), Scalable data management using user-based caching and prefetching in distributed virtual environments, *in* 'Proceedings of the ACM symposium on Virtual reality software and technology - VRST '01', ACM Press, New York, New York, USA, p. 121.
URL: http://www.scopus.com/inward/record.url?eid=2-s2.0-0035551779{&}partnerID=40{&}md5=349d84a5df11583931cdf338875983dchttp://portal.acm.org/citation.cfm?doid=505008.505033

Pittman, D. and GauthierDickey, C. (2007), 'A measurement study of virtual populations in massively multiplayer online games', *Proceedings of the 6th ACM SIGCOMM workshop on Network*

and system support for games - NetGames '07 pp. 25–30.

URL: http://portal.acm.org/citation.cfm?doid=1326257.1326262

Quax, P., Cornelissen, B., Dierckx, J., Vansichem, G. and Lamotte, W. (2009), 'ALVIC-NG: state management and immersive communication for massively multiplayer online games and communities', *Multimedia Tools and Applications* **45**(1-3), 109–131.

URL: http://www.springerlink.com/index/10.1007/s11042-009-0299-3

Quax, P., Dierckx, J., Cornelissen, B., Vansichem, G. and Lamotte, W. (2008), Dynamic server allocation in a real-life deployable communications architecture for networked games, *in* 'Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games', ACM, pp. 66–71.

URL: http://portal.acm.org/ft_gateway.cfm?id=1517508&type=pdfhttp://dl.acm.org/citation.cfm?id=1517508

Quax, P., Monsieurs, P., Lamotte, W., De Vleeschauwer, D. and Degrande, N. (2004), Objective and subjective evaluation of the influence of small amounts of delay and jitter on a recent first person shooter game, *in* 'Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games (NetGames '04)', ACM Press, New York, New York, USA, p. 152.

URL: http://dl.acm.org/citation.cfm?id=1016557http://portal.acm.org/citation.cfm?doid=1016540.1016557

Rehner, R., Castro, M. Z. and Buchmann, A. (2014), nSense : Decentralized Interest Management in Higher Dimensions through Mutual Notification, *in* '13th Annual Workshop on Network and Systems Support for Games (NetGames'14)', pp. 4–6.

Rhalibi, A. and Merabti, M. (2006), 'Interest management and scalability issues in P2P MMOG', *Consumer Communications and Networking Conference* pp. 1188–1192.

URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1593226

Roughton, A., Warren, I. and Plimmer, B. (2014), 'Concentus: Applying Stream Processing to Online Collective Interaction', *2014 23rd Australian Software Engineering Conference* pp. 70–79.

URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6824110

Schuster, S. and Weis, T. (2011), Enforcing Game Rules in Untrusted P2P-based MMVEs, *in* 'Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques', ACM, pp. 288–295.
URL: http://dl.acm.org/citation.cfm?id=2151054.2151106http://eudl.eu/doi/10.4108/icst.simutools.2011.245550

Shapiro, M., Preguiça, N., Baquero, C. and Zawirski, M. (2011), Conflict-Free Replicated Data Types, *in* 'Distributed Computing', Vol. 6976, pp. 386–400.
URL: http://dl.acm.org/citation.cfm?id=2050613.2050642http://link.springer.com/10.1007/978-3-642-24550-3_29

Shen, S., Brouwers, N., Iosup, A. and Epema, D. (2014), Characterization of human mobility in networked virtual environments, *in* 'Proceedings of Network and Operating System Support on Digital Audio and Video Workshop', NOSSDAV '14, ACM, New York, NY, USA, pp. 13:13–13:18.
URL: http://doi.acm.org/10.1145/2578260.2578272

Shen, S., Hu, S.-Y., Iosup, A. and Epema, D. (2015), 'Area of Simulation: Mechanism and Architecture for Multi-Avatar Virtual Environments', *ACM Transactions on Multimedia Computing, Communications, and Applications* **12**(1), 1–24.
URL: http://dl.acm.org/citation.cfm?doid=2816987.2764463

Shen, S. and Iosup, A. (2014), Modeling Avatar Mobility of Networked Virtual Environments, *in* 'Proceedings of International Workshop on Massively Multiuser Virtual Environments (MMVE'14)', ACM Press, Singagapore, pp. 1–6.
URL: http://dl.acm.org/citation.cfm?doid=2594448.2577396

Shirmohammadi, S., Kazem, I., Tanvir Ahmed, D., El-Badaoui, M. and De Oliveira, J. C. (2008), 'A Visibility-Driven Approach for Zone Management in Simulations', *Simulation* **84**(5), 215–229.
URL: http://sim.sagepub.com/cgi/doi/10.1177/0037549708092832

Smed, J., Kaukoranta, T. and Hakonen, H. (2002), 'Aspects of networking in multiplayer computer games'.
URL: http://www.emeraldinsight.com/10.1108/02640470210424392

Stutzbach, D. and Rejaie, R. (2006), Understanding churn in peer-to-peer networks, *in* 'Proceedings of the 6th ACM SIGCOMM on Internet measurement - IMC '06', ACM Press, New York, New York, USA, p. 189.

URL: http://portal.acm.org/citation.cfm?doid=1177080.1177105

Suznjevic, M. and Matijasevic, M. (2013), 'Player behavior and traffic characterization for MMORPGs: a survey', *Multimedia Systems* **19**(3), 199–220.

URL: http://link.springer.com/10.1007/s00530-012-0270-4

Ucar, S., Guler, H. and Ozkasap, O. (2013), Online Client Assignment in Dynamic Real-Time Distributed Interactive Applications, *in* '2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications', IEEE, pp. 65–71.

URL: http://ieeexplore.ieee.org/document/6690495/

Valadares, A., Gabrielova, E. and Lopes, C. V. (2015), 'On Designing and Testing Distributed Virtual Environments'.

URL: http://arxiv.org/abs/1508.04465

Vogels, W. (2009), 'Eventually consistent', *Communications of the ACM* **52**(1), 40.

URL: http://portal.acm.org/citation.cfm?doid=1466443.1466448http://portal.acm.org/citation.cfm?doid=1435417.1435432

White, W., Demers, A., Koch, C., Gehrke, J. and Rajagopalan, R. (2007), 'Scaling games to epic proportions', *Proceedings of the 2007 ACM SIGMOD international conference on Management of data - SIGMOD '07* p. 31.

URL: http://portal.acm.org/citation.cfm?doid=1247480.1247486

Wierzbicki, A. (2006), Trust enforcement in peer-to-peer massive multi-player online games, *in* 'Proc. Grid computing, high-performAnce and Distributed Applications (GADA06), Springer, LNCS, 2006'.

URL: http://link.springer.com/chapter/10.1007/11914952_7

X, C. P. (2013), 'Building a Balanced Universe'.

URL: https://community.eveonline.com/news/dev-blogs/building-a-balanced-universe/

Yahyavi, A. and Kemme, B. (2013), 'Peer-to-peer architectures for massively multiplayer online games', *ACM Computing Surveys* **46**(1), 1–51.

URL: http://dl.acm.org/citation.cfm?doid=2522968.2522977

Yee, N. (2006), 'Motivations for play in online games', *CyberPsychology & Behavior* **9**(6), 772–775.

URL: http://www.ncbi.nlm.nih.gov/pubmed/21780935http://www.liebertonline.com/doi/abs/10.1089/cpb.2006.9.772

Yunhua Deng and Lau, R. W. H. (2012), 'On Delay Adjustment for Dynamic Load Balancing in Distributed Virtual Environments', *IEEE Transactions on Visualization and Computer Graphics* **18**(4), 529–537.

URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6165133

Zhou, S., Cai, W., Lee, B.-s. and Turner, S. J. (2004), 'Time-space consistency in large-scale distributed virtual environments', *ACM Transactions on Modeling and Computer Simulation* **14**(1), 31–47.

URL: http://portal.acm.org/citation.cfm?doid=974734.974736

Appendix A

Other Sources

Net Computer https://openclipart.org/detail/17668/net-computer

Netserver https://openclipart.org/detail/17664/netserver

Appendix B

A DVE Feature Checklist

This is a short overview of the features an advanced DVE should have. While originally created to help me ensure I have covered all points, it may help guide the analysis of any other proposed DVE solution.

B.1 Required

Fullfilment of these requirements are what actually defines a DVE.

Interactive Reply a DVE shall reply to each command send to it. 90% of these replies shall be returned within an interval appropriate to the type of application running on said DVE (usually 20-100 msec).

Persisted State the state of the DVE shall be maintained persistently.

Scalable The DVE must be extensible to multiple processing units to support an arbitrary number of concurrent users. For a definition of *supported*, see section 2.7.3.

B.2 Needed for Sustained Service

These describe the working of a well-behaved network service that can support a (potentially very) large concurrent user base.

Fault Tolerant The service must keep running, despite the failure of multiple software or hardware components, including the partitioning of any of the networks utilized. Reduction in service quality to users is permissible, but should be restricted to as small a set of users as possible and to be as light as possible for those affected.

Rule Enforcement As the client application (or the local node in the case of a P2P system) will be under the control of a potentially adversarial party, the DVE must have mechanism to ensure that all participants are bound by the same rules. See section 2.3.9 for more in-depth discussion.

Support arbitrary avatar densities As users will flock into areas of interest, the system must be able to support any number of avatars moving into a single spatial location within the environment.

Research into such behaviour is found in section 2.4, the required DVE capability is described in section 2.7.4.

B.3 Beneficial

These functions support widening of the design envelope of the applications deployed on the DVE system.

Non-avatar objects are dynamic allows the alteration of objects other than the avatars (user-controller and computer-controlled) within the environment, while the game is running. The alternative usually is the re-deployment of the game client and the restart of the game server found in many commercial MMOs.

fine grained AoI modelling AoIs may be simple areas around the avatar or may be much more focused and detailed. A discussion can be found in section 2.9.3

Support separation of AoI and AoE the spatial position of the avatar, thus the area of effects that it can enact or is subject to should be separate from the one or more areas of perception from which the avatar receives information.

A final note

this document was produced using LaTeX