TESTUMGEBUNG FÜR DIE AVATARBASIERTE LASTVERTEILUNG IN VERTEILTEN VIRTUELLEN SYSTEMEN

Sven Allers

Hamburg University of Applied Sciences, Dept. Computer Science,
Berliner Tor 7
20099 Hamburg, Germany
sven.allers@haw-hamburg.de

ABSTRACT

Bei verteilten virtuellen Systeme handelt es sich um 3D-Welten, die auf mehreren Knoten betrieben werden können. Ein essentieller Teil dieser Systeme ist die Lastverteilung über die Knoten. Diese kann unter anderem über die Verteilung der Avatare geregelt werden. Die folgende Arbeit setzt sich mit dem Heat Map Verfahren auseinander. Ein Verfahren zur Verteilung von Avataren in verteilten virtuellen Systemen, dass in einer vorausgehenden Arbeit entwickelt wurde. In dieser Arbeit wurde das System zur dynamischen Migration von Avataren erweitert, so dass zukünftig verschiedene Verfahren getestet werden können. Außerdem wurde eine Testumgebung zur Simulation von verteilten virtuellen Systemen geschaffen und erste Tests innerhalb der Umgebung durchgeführt.

Keywords: Lastverteilung, verteilte virtuelle Systeme, Client-Assignment

1. EINFÜHRUNG

Verteilte virtuelle Systeme (DVE) ermöglichen eine Kollaboration mehrerer Nutzer innerhalb einer 3D-Umgebung über ein Netzwerk. Die einzelnen Teilnehmer werden in dieser Umgebung üblicherweise mittels Avatare repräsentiert [33]. Der Einsatzbereich ist hierbei vielschichtig. So werden DVEs unter anderem im militärischen Bereich, für Simulationen sowie für Online Spiele eingesetzt [41]. Zur Realisierung können verschiedenen Architekturmuster verwendet werden. Während kommerzielle Anwendungen vor allem Client-/Server-Architekturen einsetzen [31, 40], werden im wissenschaftlichen Kontext auch Peer-to-Peer Systeme stark betrachtet [17, 20, 40]. Solche Systeme haben den Vorteil, dass diese sich mit einer höheren Last sehr gut skalieren lassen. Jedoch bringen diese Ansätze auch einige Probleme mit. So ist die Synchronisierung der Daten in einem Peer-to-Peer Netz aufwendiger und auch sicherheitskritische Aspekte müssen vor zugriffen geschützt werden [6, 29].

Insbesondere wenn es sich bei Online-Spielen um sogenannte Massive Multiplayer Online Games (MMOG) handelt, kann die Anzahl der zu verwaltenden Spieler auf über 100.000 wachsen. Ein Beispiel hierfür ist das Spiel EverQuest. Bereits im Jahr 2005 wurden in dem Spiel etwa 150.000 Spieler zeitgleich gehostet. Hierzu kamen etwa 1.500 Server zum Einsatz [21]. Damit diese Mengen verwaltet werden können ist es unabdingbar, dass ein geeignetes Verfahren zur Lastverteilung zum Einsatz kommt. Denn eine unpassende Verteilung der Last kann dazu führen, dass einzelne Server überlastet sind. So gab es bei Eve-Online durch eine schlechte Verteilung der Last starke Verzögerungen innerhalb des Spiels [42]. Durch solche Verzögerungen kann die Spielerfahrung stark beeinträchtigt werden.

Bei Lastverteilungsverfahren in DVEs kann zwischen statischen und dynamischen Verfahren unterschieden werden. Während statische Verfahren die Last über ein zu Beginn festgelegtes Kriterium über die Server verteilen [32], können dynamische Verfahren die Verteilung zur Laufzeit anpassen. Diese können dabei verschiedene Ziele verfolgen. So versuchen einige Verfahren die Last gleichmäßig über die bestehende Serverlandschaft zu verteilen [9], andere wiederum reagieren erst, wenn eine Überlastsituation droht [19, 28, 30]. Lee et al. gehen sogar einen Schritt weiter und versuchen die einzelnen Server möglichst stark auszulasten, um auf diesem Wege Energie im Rechenzentrum zu sparen [23], ein Ansatz der auch aus anderen Kontexten bekannt ist [15]. Um die Last zu verteilen gibt es verschiedene Strategien. Die am meisten genutzte Strategie ist die sogenannte Partitionierung. Hierbei wird die virtuelle Umgebung in mehrere Bereiche eingeteilt, die jeweils von einzelnen Server verwaltet werden. Dies können statische fest definierte Bereiche [32] oder auch dynamische Bereiche sein, die sich zur Laufzeit den Lastbedingungen anpassen [8, 9]. Vor allem statische Partitionierungen haben das Problem, dass die Last ungleichmäßig über die virtuelle Welt verteilt sein kann. So kann es einzelne Hot Spots geben, die dafür sorgen, dass einige Server überlastet sind, wohingegen andere Server nur eine geringe Auslastung haben [30]. Allerdings müssen einzelne Server nicht feste Bereiche verwalten. So werden

bei Lu et al. die Avatare mit einem Standardverfahren, wie z.B. Round Robin, über die Server verteilt [25]. Zuletzt gibt es das Sharding. Hierbei wird die virtuelle Welt mehrfach repliziert, so dass es mehrere eigenständige virtuelle Welten gibt. Darüber kann unter anderem gesteuert werden, wie viele Teilnehmer sich maximal in einer Umgebung befinden dürfen. Bei Bedarf kann eine neue Instanz gestartet werden, um mehr Teilnehmern den Zugang zu ermöglichen. Dieses kann auch parallel zu den vorher genannten Strategien stattfinden. Ein Beispiel hierfür ist World of Warcraft [23]. Hier findet innerhalb der einzelnen Shards eine zusätzliche Partitionierung der Spielwelt statt [1, 23]. Sobald eine dynamische Verteilung der Last vorliegt ist es das Ziel, Regionen oder Avatare so über die Server zu verteilen, als dass möglichst eine optimale Verteilung der Last vorliegt. Hierbei handelt es sich jedoch um ein NP-Hartes Problem [13, 37] und lässt sich somit nur über Heuristiken lösen.

Um die verschiedensten Verfahren zu evaluieren, wird ein Testsystem benötigt, welches in der Lage ist die Last zu simulieren die innerhalb eines solchen Systems entsteht. Die folgende Arbeit soll sich deshalb mit der Entwicklung einer geeigneten Testumgebung auseinandersetzen. Der Inhalt basiert dabei auf ein zuvor selbst entwickeltes Verfahren zur Zuweisung von Avataren auf verschiedene Server [3]. Das dazugehörige System wurde zudem erweitert damit verschiedene Migrationsstrategien effizient getestet werden können.

Diese Arbeit ist wie folgt aufgebaut. In Abschnitt 2 wird in die Hintergründe eingeführt. Hier werden bereits vorhandene Verfahren zur Lastverteilung in DVEs vorgestellt und aufgezeigt wie diese in anderen Umgebungen getestet wurden. Daraufhin wird die Architektur des Systems vorgestellt und im nächsten Abschnitt in das Verfahren eingeführt, das in der vorgestellten Umgebung zur Lastverteilung eingesetzt wird. Anschließend wird die Testumgebung dargestellt inklusive der Simulation von Spielern. Zudem werden erste Tests vorgestellt, die in dieser Umgebung ausgeführt wurden. Im letzten Abschnitt gibt es ein Fazit zu dieser Arbeit und es wird zusammengefasst was gemacht wurde. Außerdem gibt es einen Ausblick über zukünftige Arbeiten die getätigt werden sollen und was aufbauend auf dieser Arbeit getan werden kann.

2. HINTERGRUND

Verteilte virtuelle Systeme können mittels verschiedener Architekturen realisiert werden. Am häufigsten eingesetzt wird eine Client/Server-Architektur [31, 40]. Hierbei wird die virtuelle Umgebung alleine von den Servern verwaltet. Ihre Avatare können die Teilnehmer mittels der Clients kontrollieren. Eine weitere Architekturmöglichkeit ist Peer-to-Peer [17, 20, 40]. Der Vorteil in dieser Architektur liegt darin, dass sie sehr gut skaliert, da die einsetzbaren Knoten mit jedem angemeldeten Nutzer steigt. Jedoch ist die Synchronisierung von Daten aufwendiger. Zudem müssen alle sicherheitskritischen Aspekte vor unberechtigten Zugriffen geschützt werden [6, 29]. Eine Möglichkeit zum Schützen der Daten wäre es, einen hybriden Ansatz zu wählen. Bei diesem können sicherheitsrelevante Daten auf einem zentralen Server liegen und die Peers nur die einzelnen Zonen verwalten [6]. Doch auch bei einem solchen Ansatz ist es für einzelne Peers möglich die Umgebung zu ihrem Vorteil manipulieren.

Bei der Verwaltung der virtuellen Umgebung ist ein wichtiger Aspekt, welche Elemente einem Avatar bekannt sein müssen, da dieser sie sehen oder potentiell mit diesen interagieren kann. Ein Konzept um diese zu ermitteln ist der sogenannte Area of Interest (AoI). Bei diesem Konzept wird ein Bereich um die Avatare definiert, in dem sich potentiell interessante Objekte befinden. In der Folge muss der Avatar über alle Veränderungen von Objekten in diesem Bereich informiert werden [26].

2.1. Lastverteilung in verteilten virtuellen Systemen

Lastverteilung in verteilten virtuellen Systemen kann auf verschiedenen Arten erfolgen. Deswegen werden im Folgenden verschiedene Verfahren vorgestellt. Diese fokussieren sich hierbei vor allem darauf, wie Avatare verschiedenen Servern zugeordnet werden können, um die Auslastung der Server zu optimieren. Es kann hierbei zwischen Verfahren unterschieden werden, bei denen die virtuelle Welt partitioniert wird und bei denen die Avatare unabhängig von ihrer lokalen Position verteilt werden. Eine Unterteilung der virtuellen Welt kann auf zwei Arten geschehen. Erstens in Form einer geographischen Unterteilung, bei der die virtuelle Welt in Zonen unterteilt wird und zweitens über das Verhalten der Avatare, bei dem Avatare über ihre Interaktionsmuster verteilt werden [25]. Wenn im Folgenden von Partitionierung gesprochen wird so ist ersteres gemeint, wenn nichts anderes gesagt wird.

Bei Partitionierungsstrategien kann wiederum zwischen statischen und dynamischen Strategien unterschieden werden. Während statische Verfahren mit fest definierten Partitionen arbeiten [32], setzen dynamische Verfahren oft Subpartitionen oder Zellen ein. Hierbei handelt es sich um kleinere Bereiche der virtuellen Welt, die auf die verschiedenen Server verteilt werden können [8, 9, 30]. Weiterhin kann zwischen globalen und lokalen Verfahren unterschieden werden. Bei globalen Verfahren werden Lastverteilungsentscheidung über global gesammelte Informationen entschieden. Dazu werden in irgendeiner Form die Zustandsinformationen aller Server gesammelt und darüber Entscheidungen

zur Verteilung der Last getroffen. In der Regel gibt es hierbei eine zentrale Instanz, welche die Zustandsinformationen sammelt und auswertet [9,30]. Bei lokalen Verfahren treffen einzelne Server selbstständig die Entscheidung ob Last zu verteilen ist. Dazu haben diese in der Regel nur ihre eigenen Informationen sowie Informationen von benachbarten Servern zu Verfügung, um die Last effizient zu verteilen [19].

In vielen Verfahren ist es vor allem das Ziel, die Last gleichmäßig über das System zu verteilen [9,42]. Es gibt jedoch auch Verfahren mit anderen Zielen. Ta et al. wollen mit ihrem Verfahren z.B. die Interaktivität maximieren. Dabei gehen sie davon aus, dass die Server über mehrere Regionen der Welt verteilt sind. Sie versuchen deshalb die Zonen, so auf die Server zu verteilen, dass Dauer der Nachrichtenübermittlung zwischen Client und Server möglichst gering ist [38]. Lee et al. verfolgen das Ziel, möglichst wenig Server zu belasten, damit Datenzentren möglichst gering ausgelastet werden und auf diese Weise Strom gespart wird. In ihrem Vorschlag geht es deshalb vor allem darum die virtuellen Maschinen, in denen die Server laufen, effizient zu verteilen [23]. Li et al. versuchen mit ihrem Verfahren die Inkonsistenzen zwischen Clients und Server zu minimieren. Dazu wird betrachtet, wie lange es dauert bis ein Replikat auf einem Client mit dem Zustand auf dem Server konsistent ist. Die Zonen werden dabei so aufgeteilt, dass die Updatezeit der Clients möglichst minimiert werden. Die Server verwalten zwar alle Avatare, die sich in ihrer zugeordneten Region befinden, allerdings können Clients auch mit Servern verbunden sein, die nicht deren Avatar verwalten. In einem solchen Fall dient der Server mit dem der Client verbunden ist als Proxy und übermittelt alle Befehle des Clients an den verwaltenden Server bzw. sendet alle Updates vom Server an den Client. Durch Ausprobieren wird in diesem Verfahren die optimale Zuordnung der Clients sowie der Zonen zu den Servern ermittelt [24]. Zhang et al. betrachten bei der Repartitionierung neben der Last der Server auch die Kommunikationsverzögerung, die zwischen Client und Server entsteht. Sollte die Last eines Servers oberhalb eines Thresholds liegen, so werden die benachbarten Server kontaktiert. Bei der Bewertung eines geeigneten Kandidaten fließen neben der Last der Server auch Strafpunkte ein, die den erhöhten Kommunikationsaufwand beschreiben. Beim Übermitteln der Zonen werden zunächst nur Zellen als potentielle Kandidaten ausgewählt, bei denen die Integrität der verwalteten Zonen nicht zerstört wird. Diese werden nach einer Dissatisfaction Optimization Regel sortiert, welche den erhöhten Kommunikationsaufwand beschreibt, wenn die Zelle migriert wird. Anschließend werden so viele Zelle übertragen bis die Last des Servers unterhalb eines zuvor definierten Wertes ist [43].

2.2. Partitionierungsstrategien im kommerziellen Kontext

Im kommerziellen Bereich werden vor allem statische Partitionierungsmethoden eingesetzt. So geschieht dies z.B. in Second Life [32] sowie World of Warcraft [11]. Zudem kommt in diesem Bereich oft das sogenannte Sharding zum Einsatz [21,23]. Hierbei wird die komplette virtuelle Welt repliziert. Die maximale Auslastung kann nun über die Begrenzung der maximalen Teilnehmerzahl gesteuert werden. Ein großer Nachteil eines solchen Verfahrens ist die vollständige Trennung der Teilnehmer innerhalb der verschiedenen virtuellen Umgebungen, sodass diese nicht miteinander interagieren können. In EverQuest kommt zudem ein Just-in-Time System zum Einsatz, wenn Teilnehmer eine Dungeon betreten. Hierbei wird jedes Dungeon spontan initialisiert, wenn es benötigt wird. Dafür wird dem Dungeon dynamisch eine Maschine zugewiesen, um eine optimale Auslastung zu garantieren [21]. Statische Partitionierungen sind verhältnismäßig einfach umzusetzen, haben jedoch den Nachteil, dass einzelne Server schnell überbelastet werden können, während andere kaum ausgelastet werden [11]. Dies kann vor allem schnell passieren wenn ein sogenanntes Flocking auftritt. D.h. dass sich viele Avatare in ein bestimmtes Gebiet oder zu einem Punkt bewegen. Dies kann z.B. aufgrund eines bestimmten Ereignisses oder durch Absprache unter Teilnehmern stattfinden [7].

In Eve-Online findet eine dynamische Partitionierung statt. Hier wird die Spielwelt in einzelne Systeme aufgeteilt. Mithilfe eines Binären Baumes wird versucht die Last optimal aufzuteilen [42]. Das Basisverfahren beruht auf einer Halbierung der Last auf jeder Ebene des Baumes, bis es so viele Blätter wie Serverknoten gibt. Dies hat jedoch den Nachteil, dass diese Methode nur funktioniert, wenn die Anzahl der Serverknoten einer Zweierpotenz entspricht. Deshalb werden auch die Server mittels eines Binären Baumes aufgeteilt. Auf jeder Ebene wird auf dem einen Zweig die maximale Binäre Anzahl gewählt und auf der anderen Seite der Rest behalten für den dasselbe nochmal gemacht wird, bis nur noch Gruppen von Zweierpotenzen existieren. Gibt es z.B. 85 Server so werden die in Gruppen von 64 und 21 aufgeteilt. Da die 21 keine Zweierpotenz ist, wird diese wiederum in 16 und 5 aufgeteilt usw.. Auf diese Gruppen wird nun anteilig das oben genannte Verfahren angewendet. Damit soll auf jede Gruppe soviel Last aufgeteilt werden, wie ihr Verhältnis zur Gesamtserverzahl ist. Im Falle von 64 heißt dies, dass das Verfahren mit 75,3% der Last durchgeführt wird, weil 64 75,3% von 85 entspricht [42].

2.3. Dynamische Partionierungsstrategien

Im wissenschaftlichen Kontext werden vor allem dynamische Partitionierungsverfahren betrachtet. Dazu werden die

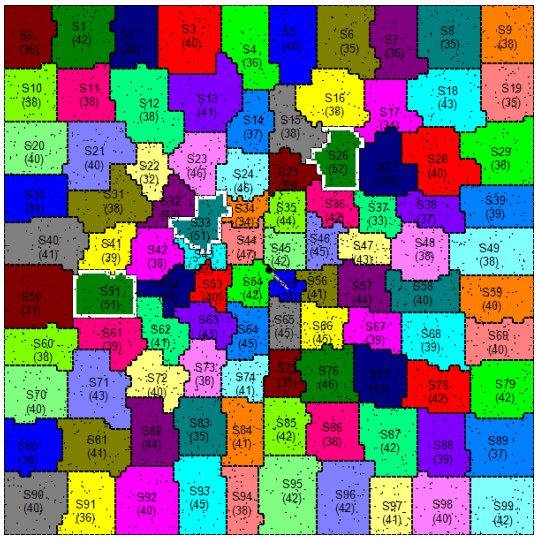


Abbildung 1: Mögliche Aufteilung einer virtuellen Welt auf Server bei einem dynamischen Partitionierungsverfahren [9]

virtuelle Welten meistens in viele kleine Zellen unterteilt, die dynamisch verschiedenen Servern zugeordnet werden können, so auch bei Ng et al.. Sie schlagen ein globales Verfahren vor, bei dem die Last an einen sogenannten Loading Collector (LoC) übermittelt wird. Sollte ein Server feststellen, dass er stark ausgelastet ist, wird der LoC kontaktiert um die Lastinformationen von den Servern der Nachbarregionen zu erhalten. Anschließend wird der Server mit der geringsten Last als Ziel definiert. Auf diesen soll nun die überschüssige Last übermittelt werden. Zur Lastübertragung werden verwaltete Zellen abgetreten. Hierbei werden so viele abgegeben, bis ein zuvor definierter Zielwert bei der Abgabe an Last überschritten wurde. Beim Übertragen der Zellen sollen die verwalteten Regionen der Server zusammenhängend bleiben. Deshalb werden nur Zellen abgetreten, die sich direkt an der Grenze befinden. Dabei werden zunächst die Zellen abgetreten, die sich mit ihrer Last am nächstem zu dem erreichenden Zielwert befinden [30]. De Vleeschauwer et al. haben mit Mikrozellen auch ein Ansatz, bei dem die virtuelle Welt in viele kleine Zellen unterteilt wird. Allerdings wird hier die Integrität von Regionen auf Servern nicht garantiert. In ihrer Arbeit werden mehrere Algorithmen verglichen. Dabei stellen sie fest, dass mittels linearer Programmierung eine optimale Lösung gefunden werden kann. Jedoch ist die Berechnung sehr teuer und kann schon bei 64 Mikrozellen auf 4 Computern mehrere Tage betragen [8]. Beim Locality Aware Partitioning ist für jeden Server ein Quality of Service (QoS) definiert. Dieser setzt sich zusammen aus der Belastung durch Clients und der Interserverkommunikation mit anderen Servern. Wird eine der Qualitätskriterien nicht mehr eingehalten, so wird eine Repartitionierung initialisiert. Sollte z.B. das zuvor definierte Aktualisierungsintervall, in dem die Clients Updates vom Server erhalten, für 90% der Clients überschritten sein, so müssen Zellen abgetreten werden. Dazu verwaltet jeder Server einen Lastgraph, der die Belastung der anderen Server beschreibt. Dieser wird regelmäßig mit den benachbarten Servern ausgetauscht, damit dieser aktualisiert werden kann. Zunächst wird versucht die Last an benachbarte Server abzutreten, damit verwaltete Regionen nach Möglichkeit zusammenhängend bleiben. Sollte sich jedoch kein geeigneter Kandidat finden bzw. kein benachbarter Server bereit sein Zellen anzunehmen, so können diese auch an andere Server abgetreten werden. Ist die Interserverkommunikation zu hoch, so kann dies an einer zu starken Verteilung der Zellen liegen. Ist dies der Fall, werden Regionen aggregiert um den Kommunikationsaufwand zu verringern [7].

Da bei einer unglücklichen Bewegung der Avatare, die Last, mit der Zeit, in Richtung einer zweiten stark belasteten Region bewegt werden kann, wenn nur lokale Informationen betrachtet werden, schlagen Lau und Rynson ein hybrides Verfahren vor. Hier werden die Lastinformationen eines Servers mit globalen Informationen angereichert. Sollte ein Server überlastet sein, tritt es einzelne Zellen an benachbarte Server ab. Dabei werden neben den Belastungsinformationen der benachbarten Server auch die globalen Informationen betrachtet. Hierbei wird nach Servercluster gesucht, die eine geringe Auslastung haben und die Last nach Möglichkeit in diese Richtung bewegt [22]. Das Verfahren von Deng et al. basiert auf Hitzediffusion. Die Auslastung der Server wird in diesem Verfahren in Form von Hitze dargestellt. Verwaltet werden die Server mithilfe eines Graphen, in dem benachbarte Server mittels einer Kante verbunden sind. Das Gewicht der Kante beschreibt die Lastdifferenz zwischen den Servern. Diese wird von den lokalen Servern genutzt, um zu ermitteln, wie viele Zellen an benachbarte Server übertragen werden können. Berechnet wird die Lastdifferenz von einem zentralen Server mittels Hitzediffusionsgleichung, um so den optimalen Fluss zu erhalten, der die Last gleichmäßig verteilt [9, 11]. Eine mögliche Partitionierung, die durch dieses Verfahren entsteht, ist in Abb. 1 zu sehen. Durch die Round Trip Time zum zentralen Server unterscheidet sich jedoch die Last der einzelnen Server, von der Last, die bei der Lastberechnung vorgelegen hat. Deshalb wird lokal zusätzlich betrachtet, wie viele Clients seit dem letzten Übermitteln der Lastinformationen den Server gewechselt haben. Mit dem ermittelten Wert wird die Menge der zu übermittelnden Last angeglichen [10]. Kazem et al. verwalten die virtuelle Umgebung nicht mit quadratischen Zellen, sondern durch Hexagone. Sie sind der Meinung, dass eine dynamische Repartitionierung mittels Zellen zu aufwendig sei. Deshalb definieren sie in jedem

Hexagon feste Bereiche, die im Falle einer Überlastung an benachbarte Server abgetreten werden [19].

Eine Partitionierung der virtuellen Welt muss nicht mittels Zellen erfolgen, die eine fest definierte Form haben. Die Form der Partition kann sich auch dynamisch über die Avatare gestalten, die verwaltet werden. So schlagen Lui und Chan ein Verfahren vor, in dem die Verteilung der Avatare über einen Graphen geschieht. Die Berechnung, welche Avatare migriert werden sollen, findet in zwei Schritten statt. Im Layer Partitioning wird zunächst ein Graph über die Avatare erstellt. Die Knoten repräsentieren hierbei die Avatare und eine Kante wird gebildet, wenn sich diese im AoI voneinander befinden. Sollte ein Avatar in keinem AoI eines anderen Avatars sein, so wird eine Kante zwischen ihm und dem nächsten Avatar auf demselben Server gebildet. Der Algorithmus testet nun wie sich die Kostenfunktion ändert, wenn Avatare, die sich am Rand eines Servers befinden, auf andere Server umverteilt werden. Anschließend findet das Communication Refinement Partitioning statt. In diesem Schritt wird versucht, die Kommunikation zwischen den Servern zu verringern. Da dieses Verfahren bei großen Mengen an Avataren aufwendig ist, wird zudem der Parallel Partitioning Algorithmus vorgestellt. Hier wird die virtuelle Welt in mehrere große Zellen unterteilt und innerhalb dieser Zellen das Layer Partitioning und Communication Refinement Partitioning durchgeführt [26]. Aufbauend auf der Kostenfunktion von Lui und Chan haben Morillo et al. ein Verfahren auf Basis des Ameisenalgorithmus entwickelt. Zunächst werden dazu alle Avatare gesucht, die sich an der Grenze zu einem anderen Server befinden. Damit sind alle Avatare gemeint, die in ihrem AoI einen Avatar haben, der sich nicht auf dem selben Server befindet. Mithilfe des Ameisenalgorithmus werden die Avatare nun über mehrere Iterationen umverteilt, um die Kostenfunktion zu optimieren [27].

2.4. Lastverteilung ohne Partitionierung

Es gibt auch Verfahren, die vollständig auf eine Partitionierung der virtuellen Welt verzichten. In diesen Verfahren werden die Avatare nicht aufgrund ihrer Lokalität innerhalb der virtuellen Welt verwaltet, sondern über eine freie Verteilung, die vom Algorithmus abhängig ist. Dies sorgt für eine losere Kopplung zwischen Servern und Avataren. Partitionierung hat vor allem das Problem, dass sämtliche Avatare einer Zone migriert werden müssen, wenn es eine Repartitionierung gibt [19]. Morillo et al. nennen ihr Verfahren *Objektorientiertes Management*. Jeder Avatar wird einem Server, unabhängig von seiner räumlichen Position, zugewiesen. Ziel ist es die Last der Server unter 100% zu halten. Dazu startet eine Umverteilung, wenn die Auslastung eines Servers bei 99% liegt. Die zu migrierenden Avatare werden nach ihrem Präsenzfaktor sortiert und umverteilt. Der Präsenzfaktor beschreibt, in wie vielen AoIs von anderen Avataren der Avatar auftaucht. Dabei wird jeder Avatar auf den aktuell am geringsten ausgelasteten Server migriert, bis die Last des Servers mindestens auf 90% verringert wurde [28]. Lu et al. empfinden die meisten Lastverteilungsverfahren als zu komplex. Sie schlagen deshalb vor, dass Standardverfahren, wie Round Robin, zur Lastverteilung genutzt werden sollen. Dazu arbeiten sie mit Sticky Sessions. Die Verteilung erfolgt über einen zentralen NAT, der die Anfragen an die Server weiterleitet [25]. Elfizar et al. gehen sogar einen Schritt weiter. Hier wird jedes Objekt einzeln simuliert und für jedes Objekt ein eigener Prozess gestartet. Es wird argumentiert, dass dies deutlich skalierbarer sei, da für jeden Avatar nur ein neuer Prozess gestartet werden muss. In Tests konnte gezeigt werden, dass der Ansatz Komponenten zu verteilen effizienter ist als Regionen zu verteilen. Mit dem Ansatz konnte eine höhere Framerate und auch eine geringere CPU-Auslastung als mit einem Partitionierungsverfahren erreicht werden [12].

2.5. Nutzerverhalten in DVEs

Damit verteilte virtuelle Systeme simuliert werden können, muss bekannt sein, wie sich Teilnehmer innerhalb der virtuellen Welten verhalten. Dazu gibt es eine Reihe von Untersuchungen. Die meisten von ihnen beschäftigen sich mit dem Verhalten in Online-Spielen. Neben möglichen Verhaltensmustern wird in vielen Quellen auch analysiert wie lange einzelne Sessions dauern, wie oft Spieler neu ins Spiel einsteigen und wie viele Spieler aktiv sind [36]. Auch wenn die spezifischen Verhaltensmuster stark von der konkreten Anwendung abhängig sind, so geben die Studien eine gute Idee, wie sich Teilnehmer allgemein verhalten. Denn das Teilnehmerverhalten ist wichtig, um Lösungen zur Skalierbarkeit ermitteln zu können und so Serverüberlastungen verhindern zu können. Grundsätzlich gibt es vier Arten das Verhalten der Teilnehmer zu ermitteln [36]:

- Netzwerk Traces
- Polling vom Spielclient mittels Spielemechaniken
- Logdateien vom Spielebetreiber
- Messungen auf Spielercomputern

Da nur sehr wenige Spielebetreiber ihre Daten öffentlich zur Verfügung stellen ist bei Messungen innerhalb von kommerziellen Spieletiteln das Polling im Spielclient sehr beliebt. Spiele wie z.B. World of Warcraft stellen Funktionen zur Verfügung, um zu ermitteln, welche Spieler sich gerade wo befinden. Diese können ausgenutzt werden um das Verhalten von Spielern zu ermitteln [23, 31, 36].

In manchen Studien wurden eigene Spiele entwickelt, um

Behavior	Total %	Behavior	Total %
Circle-Strafe	15.9%	Hide-Snipe	0.9%
Scatter	15.7%	Snipe 50m	4.0%
Snipe 100m	6.6%	Snipe 150m	2.7%
Snipe 200m	1.7%	Snipe 300m	0.9%
In-Building Snipe	4.6%	Chase-Kill	15.9%
Reverse-Attack	3.5%	Wander	10.2%
Shoot into Building	1.5%	Inactive	7.7%
Parallel Strafe	1.5%	Sinusoid	2.1%
Stand & Attack	4.6%		

Abbildung 2: Ergebnis der Auswertung von Spielerverhalten nach Sing et al. [35]

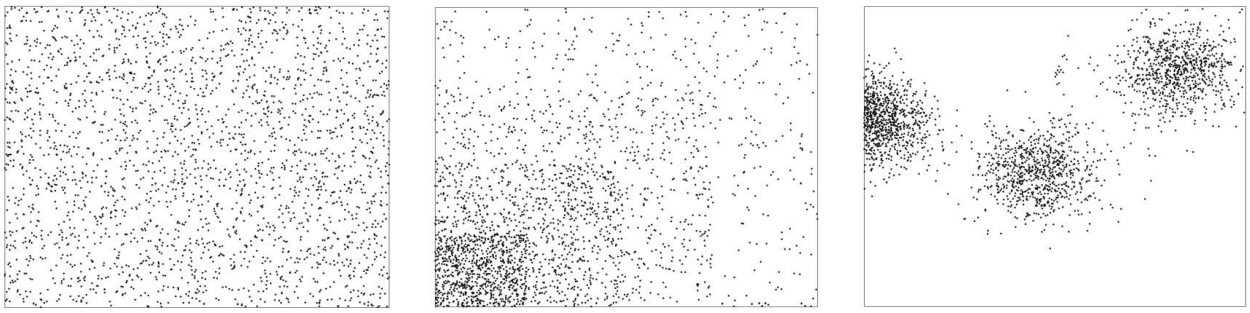
das Verhalten der Spieler zu messen. So konnte unter anderem Greenhalgh in mehreren Studien feststellen, dass sich die Teilnehmer in ihrer virtuellen Simulation im Schnitt nur zu 19,6% der Zeit aktiv bewegen. Es variiert jedoch zwischen 7,5% und 54,6% bei den einzelnen Teilnehmern [14]. Auch Sing et al. haben ein selbst bereitgestelltes Spiel genutzt, um das Nutzerverhalten zu analysieren. Dabei haben sie 19 verschiedene Spielerverhalten herausgearbeitet und ihre Häufigkeit prozentual aufgeführt [35] (siehe Abb. 2).

Pittman et al. haben das Spielerverhalten innerhalb von World of Warcraft untersucht. Es sollte vor allem herausgefunden werden, wie sich Spieler innerhalb der Spielwelt verteilen und bewegen. Das Ziel dieser Studie war es ein möglichst akkurates Spielermodell zu skizzieren, damit Spieler realistisch simuliert werden können. Dazu wurden fünf Wochen Aufzeichnungen, mithilfe der Freundesliste und des *who* Befehls, den World of Warcraft zur Verfügung stellt, gemacht und ausgewertet. Dadurch konnte festgestellt werden, wie viele und welche Spieler sich zu welchem Zeitpunkt in welcher Zone der Spielwelt befanden. Dabei konnte ermittelt werden, dass die Verteilung der Spieler sehr ungleichmäßig ist. Während etwa 5% der Zonen von durchschnittlich mehr als 40 Spielern besucht wurde, waren es bei 75% der Zonen weniger als 10. Das deutet daraufhin, dass es in der Spielwelt wenige Hot Spots gibt, die von einem Großteil der Spieler angesteuert werden. Es konnte auch gezeigt werden, dass knapp 40% der Spieler sich während einer Session in nur einer Zone aufgehalten haben. Knapp 50% der Spieler haben sich zudem in 6 oder weniger Zonen innerhalb einer Session aufgehalten. 90% der Sessions haben hierbei 3 Stunden oder weniger gedauert, 50% sogar weniger als 10 Minuten. Der große Anteil an kurzen Sessions wurde damit erklärt, dass viele Spieler nachschauen ob Freunde spielen und das Spiel verlassen, wenn dem nicht so ist. Zudem gab es deutliche Schwankungen bei der Anzahl der Spieler. Die Hochzeiten waren dabei vor allem an den Abendstunden und am Wochenende. Die Spielerzahl lag abends und am Wochenende ab der Mittagszeit durchschnittlich bei über 1200 Spielern pro Server. Wohingegen die Spielerzahlen, zwischen 2 und 10 Uhr unter der Woche, im Schnitt unterhalb von 600 lagen [31]. Dies konnte auch von Lee et al. festgestellt werden. Auch sie haben den *who* Befehlt genutzt um über 273 Tage zu ermitteln, wie viele Spieler sich in einem Realm aufhalten. Dabei wurde auch festgestellt, dass die Anzahl der Spieler abends höher liegt als tagsüber [23].

2.6. Simulieren vom DVEs

Eine Vielzahl von Verfahren zur Simulation von verteilten virtuellen Systemen existieren bereits. Eine paar von diesen werden im Folgenden vorgestellt. Dabei wird im Folgenden der Fokus vor allem auf die Bewegung von Avataren gesetzt. Grundsätzlich können die zu untersuchenden Probleme von verteilten virtuellen Systemen in sechs Dimensionen eingeteilt werden: Korrektheit, Fehlertoleranz, Skalierbarkeit, Zeitintensität, Konsistenz und Overheadkosten durch die Verteilung [40]. Bevor eine Simulation gestartet werden kann, müssen die Avatare initial über die Spielwelt verteilt werden. Dabei ist es üblich, die Spieler gleichmäßig, gedrängt um einen Punkt oder in mehrere Gruppen aufgeteilt über die Spielwelt zu verteilen [3,26–28,33,36] (siehe Abb. 3).

Sing et al. haben auf Basis ihrer Untersuchung (Abb. 2) ein Simulationsmodell entwickelt, welches jede der aufgeführten Aktionen zufällig ausführt. Hierbei sollte vor allem ein Modell entwickelt werden, um die Skalierbarkeit von verteilten virtuellen Systemen ermitteln zu können. Hierzu wurden die Metriken in die zwei Kategorien Quality of Virtuality (QoV) und Engine Performance unterteilt. Bei der QoV wird überprüft wie korrekt Objekte bei den Clients ankommen und wie konsistent die Datensätze zwischen den Clients sind. Dadurch kann indirekt auch die User-Experience gemessen werden. Bei der Engine Performance geht es um die Leistungsfähigkeit des Spielservers. Dazu wird überprüft, wie lange eine Iteration innerhalb der Engine dauert [35]. Ng et al. haben zum testen ihres Verfahrens ein Changing Circular Moving Pattern eingesetzt (CCP). Hierbei ändert jeder simulierte Avatar alle vier Schritte seine Bewegungsrichtung um zehn Grad [30]. Tveit et al. wollen ein möglichst realistisches Spielerverhalten simulieren und setzen dabei auf einen agentenbasierten Ansatz. Dazu wurden verschiedene Spielertypen modelliert, die echte Spielertypen widerspiegeln sollen. So gibt es Achiever, Socializer, Killer und Explorer. Da es sich um ein agentenbasierten Ansatz handelt, hat jeder Agent auch die Phasen Sense, Reason und React. In der Reason-Phase, in der der Agent entscheidet, welche Aktion als nächstes ausgeführt wird, kann mit drei verschiedenen Verfahren gearbeitet werden. Eine davon ist greedy und aggressiv. In diesem wird versucht das nächste Monster zu töten. Ein weiteres sind Markov Ketten. Hier wird abhängig von den vorherigen Entscheidungen eine



(a) (b) (c)

Abbildung 3: Übliche Verteilung von Avataren: gleichmäßig (a), gedrängt (b) und gruppiert (c). [26]

Entscheidung über eine Markov Matrix getroffen wird. Das letzte arbeitet mit einem hierarchischen Plan. In diesem wird vor der Simulation definiert wie sich ein Spieler in bestimmten Situationen verhalten soll [39]. Rueda et al. haben ein Verfahren zur Simulation von Peer-to-Peer DVEs entwickelt. Ziel ihres Verfahrens ist es, Inkonsistenzen innerhalb des verteilten virtuellen Systems aufzudecken. Dazu berechnen sie die Average System Response. Hierbei handelt es sich um die Zeit, die benötigt wird, bis Änderungen bei verschiedenen Clients ankommen. Um diese berechnen zu können, sendet der Client bei Update-Nachrichten einen Zeitstempel mit. Nachbarn, die von der Änderung benachrichtigt werden, bestätigen diese entsprechend. In ihrem System werden verschiedene Bewegungsmuster bereits unterstützt [33]. Das Changing Circular Pattern [30], Hot Points All, bei dem es Hot Points gibt, wo alle Avatare sich mit der Zeit hinbewegen, und Hot Points Near, bei dem sich nur Avatare in bestimmten Radien von einem Hot Point in dessen Richtung bewegen [33].

In manchen Fällen kommen auch Verfahren zum Einsatz, die ursprünglich zum Testen mobiler Netze gedacht waren. Ein solches Verfahren ist Random Waypoint [11, 18, 24]. Ursprünglich wurde es von Johnson und Maltz entwickelt [18] und bekam die Bezeichnung Random Waypoint erstmals durch Broch et al. [5]. Bei diesem Verfahren erhält jeder Avatar ein zufälliges Ziel, auf das er sich zubewegt. Wird dieses Ziel erreicht, bewegt sich der Avatar auf ein neues zufälliges Ziel zu. Li und Cai haben ihr Verfahren gegen Random Way Point, Reference Group Mobility Model [16] und Pursue Mobility Model [34] getestet. Im Pursue Mobility Model versuchen mehrere Entitäten ein einzelnes Ziel zu fangen [34]. Im Falle von Li und Cai ist dieses Ziel ein Avatar, der sich nach dem Random Waypoint Schema bewegt. Dazu werden die Avatare zunächst gruppiert. Innerhalb jeder Gruppe ist ein Avatar das Ziel, welches die anderen Avatare einer Gruppe versuchen zu erreichen [24].

3. ARCHITEKTUR

Die Architektur von Timadorus kann in fünf Hauptkomponenten eingeteilt werden. Den Spieleclients, den Spieleservern, dem Auth-Server, QuP sowie dem Client Assignment Manager (CAM) (Abb.4). Für die Lastverteilung ist der CAM zuständig. Dieser weist Avatare bestimmten Spielserver zu, und weist diese ggf. auch einem neuen Server zu, wenn es erforderlich ist. Dabei wird die Eigenschaft des QuP ausgenutzt, dass mehrere Spielserver Elemente derselben Region verwalten können. Bei QuP handelt es sich um das Persistenzsystem von Timadorus. Dieses verwaltet den Zustand der Spielwelt und informiert die Spielserver, wenn sich dieser geändert hat. In zukünftigen Versionen des CAM soll dieser auch das Interessenmanagement des QuP stärker nutzen, um noch weiter optimierte Lösungen ermitteln zu können. Aufgrund dieser engen Kopplung soll im Folgenden nicht nur der Client Assignment Manager näher betrachtet werden, sondern auch das QuP-System.

3.1. QuP

Beim QuP handelt es sich um ein verteiltes Persistenzsystem. Dieses verwaltet den Zustand der gesamten Spielwelt und ist dabei als Publish-Subscribe System umgesetzt worden. Benötigt ein Spielserver bestimmte Elemente der Spielwelt, so kann dieser die entsprechenden Elemente abonnieren und wird benachrichtigt, sobald diese sich verändern. Sollte ein Spielserver ein Objekt manipulieren, so wird dies auch dem QuP mitgeteilt. Dieser entscheidet zudem letztendlich, ob die Manipulation erfolgreich ist und bestätigt dies auch. Sollten mehrere Spielserver ein Objekte zeitgleich manipulieren, so kann es zu einer Kollision kommen und die Änderung wird verweigert. Es gibt auch die Möglichkeit, Regionen der Spielwelt zu abonnieren (Spatial-Publish-Subscribe). Der Spielserver abonniert in einem solchen Fall alle Elemente die sich innerhalb eines bestimmten räumlichen Bereiches befinden und wird informiert, wenn es Änderungen gibt. Dies gilt auch für Elemente, die nach dem Abonnieren in

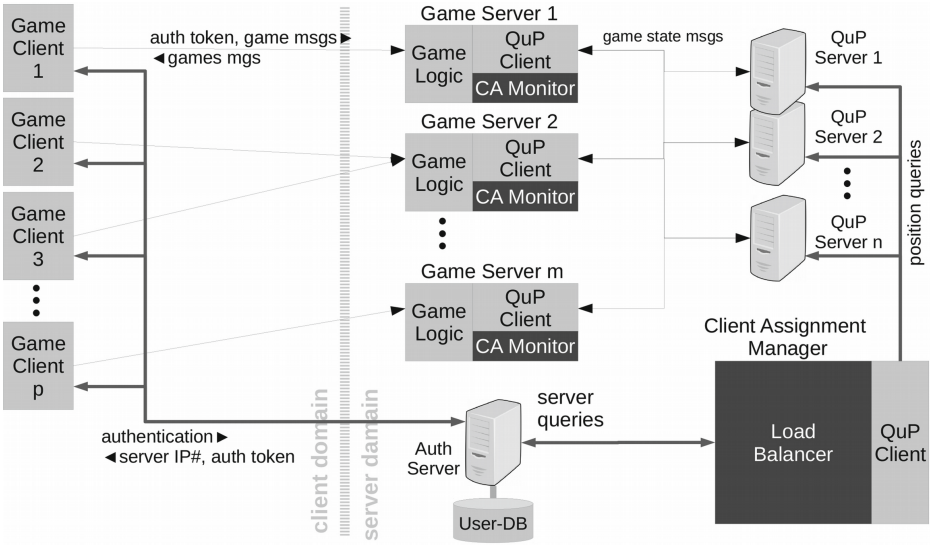


Abbildung 4: Architekturvorschlag für Timadorus [3]

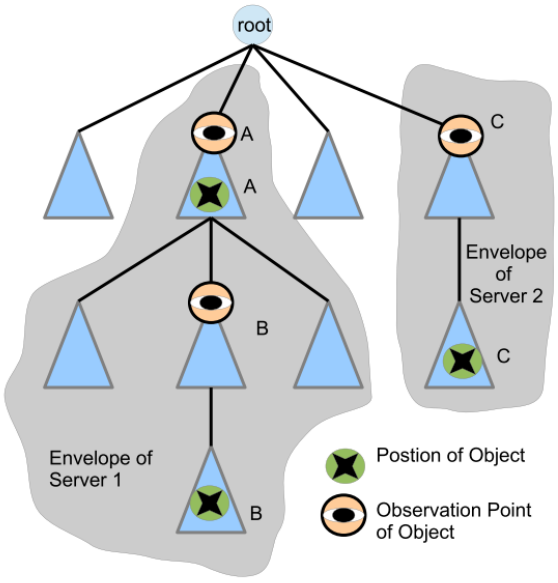


Abbildung 5: Quadtree in QuP [4]

diesen Bereich gelangen. Sollte ein Element den Bereich verlassen, so wird auch das Abonnement aufgehoben. Um dies zu machen registriert der Spielserver beim QuP einen Observer. Die Elemente der Spielwelt werden dabei in einem Octree verwaltet (Abb. 5 (vereinfachte Darstellung als Quadtree)). Der Observer befindet sich nun auf einem Knoten innerhalb dieses Octrees. Alles was sich auf oder unterhalb dieses Knoten befindet ist der abonnierte Bereich. Durch die Verwendung des Octrees wird die Spielwelt somit auf dreidimensionaler Ebene verwaltet. Genauer gesagt handelt es sich um einen normalisierten Würfel. Positionen sind somit normalisiert, sodass sie sich immer zwischen 0 und 1 auf jeder Achse befinden [4]. Eine Registrierung eines Observers geschieht üblicherweise, wenn Avatare einem Spielserver zugewiesen werden. In einem solchen Fall wird über den Sichtbereich des Avatars ein Observer registriert, welcher den AoI von diesem repräsentiert.

3.2. Client Assignment Manager

Der CAM ist für die Verteilung der Avatare auf den Spielservern zuständig und übernimmt somit die Lastverteilung im System. Dieser besteht aus zwei Komponenten. Dem CA Monitor sowie dem Load Balancer. Der CA Monitor hat die Aufgabe die Zustände der Spielserver zu überwachen und an den Load Balancer zu übermitteln. Darunter fallen die Auslastung des Systems sowie Informationen für das Lastverteilungsverfahren. Zudem können zusätzliche Informationen zur Neuberechnung übermittelt werden. Im Falle des Heat Map-Verfahrens wird z.B. übermittelt welche Felder sich verändert haben. Der Load Balancer aggregiert diese Informationen und macht auf Basis dieses Wissens Neuberechnungen über die Zuweisung der Avatare. Der Client Assignment Manager bietet zudem die Möglichkeit, das verschiedene Verfahren konfiguriert werden können [2]. Neu ist seit dieser Arbeit, dass auch Informationen zur Neuberechnung übermittelt werden. Zuvor wurden nur Informationen übermittelt, die für die initiale Zuweisung benötigt werden. Dies ermöglicht eine flexiblere Gestaltung von Verfahren zur Neuberechnung mittels verfahrensspezifischen Informationen vom Client.

Der CA Monitor ist wiederum in mehrere Module unterteilt

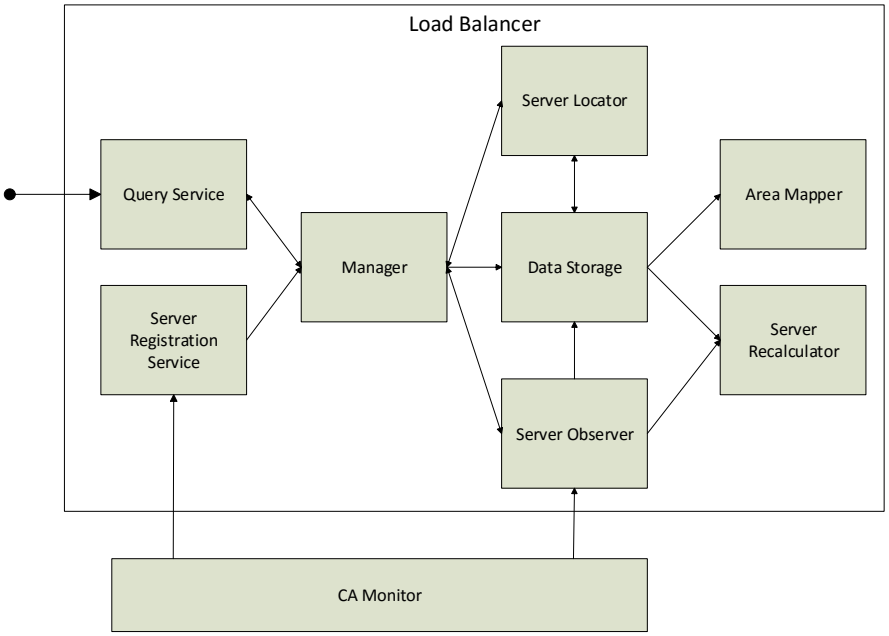


Abbildung 7: Interner Aufbau vom Load Balancer

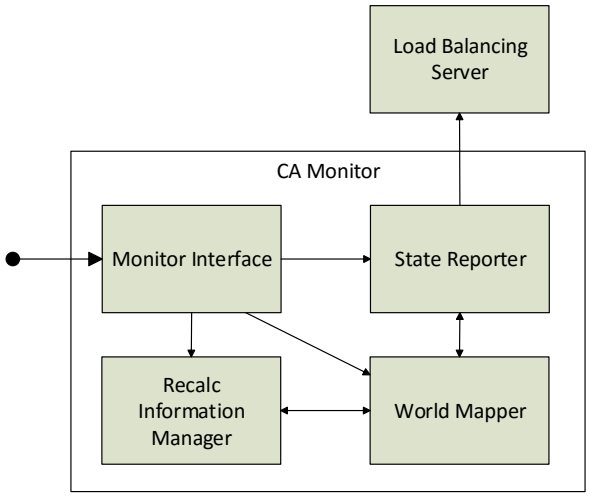


Abbildung 6: Interner Aufbau vom CA Monitor

(Abb. 6). Über das Monitor Interface ist der Spielserver in der Lage mit dem Monitor zu kommunizieren, wenn sich z.B. ein neuer Avatar angemeldet hat. Der State Reporter ist die Komponente, die mit dem Load Balancer kommuniziert. Beim Recalc Information Manager sowie dem World Mapper handelt es sich um Komponenten, die den Zustand des CA Monitors verwalten. Wie diese konkret verwaltet werden, hängt von dem konfigurierten Verfahren ab. Dadurch ist es möglich, verschiedene Verfahren unter verschiedenen Konfigurationen zu vergleichen. So merkt sich der World Mapper beim Heat Map Verfahren welche Felder sich seit der letzten Übermittlung von Zustandsinformationen an den Load Balancer geändert haben [2]. Der Recalc Information Manager ist im Zuge dieser Arbeit entstanden. Dieser hat die Aufgabe alle Informationen zu sammeln, die für eine Neuberechnung der Zuweisung der Avatare relevant sind. Die Informationen, die an den Load Balancer übermittelt werden, werden mit diesen Daten angereichert.

Auch der Load Balancer ist in mehrere Module unterteilt. Über den Query Service kann ein Server für einen neuen Avatar ermittelt werden. Der CA Monitor kommuniziert mit dem Load Balancer über den Server Registration Service sowie dem ihm zugewiesenen Server Observer. Der Server Registration Service wird genutzt, damit sich ein Spielserver beim Load Balancer an- und abmelden kann. Über den Server Observer übermittelt der CA Monitor den Zustand des Spielservers. Dafür wird für jeden neu registrierten Spielserver ein eigener Observer gestartet. Dieser alarmiert zudem das System, wenn über einen zu langen Zeitraum keine Zustandsmitteilung eintrifft. Das System geht in einem solchen Fall davon aus, dass der Server nicht mehr verfügbar ist. Im Data Storage werden die Zustände der Spielserver gespeichert. Hinter dem Data Storage befindet sich der Area Mapper. Dieser ist speziell dafür gedacht die Verteilung der Avatare zu verwalten. Die genaue Implementierung ist auch hier von dem genutzten Verfahren abhängig, welches konfi-

gurierbar ist. Im Falle des Heat Map Verfahrens ist dies z.B. das Raster, in dem die Daten der Heat Map gespeichert sind. Auch die Implementierung vom Server Locator und Server Recalculator sind von der Konfiguration abhängig. Beim Server Locator handelt es sich um das Modul, welches den Server berechnet auf den ein Avatar zugewiesen wird. Eine mögliche Konfiguration für den Server Locator ist das Heat Map Verfahren, welches im Abschnitt 4 vorgestellt wird [2]. Der Server Recalculator ist im Rahmen dieser Arbeit entstanden und ist für die Neuberechnung von Servern zur Laufzeit zuständig. Ziel ist es, dass Avatare neuen Spielservern zugewiesen werden, wenn die Aufteilung nicht mehr als ideal betrachtet wird. Aktuell gibt es eine periodische Neuberechnung, die für jeden Avatar in festen Abständen eine Neuberechnung macht. In zukünftigen Arbeiten soll jedoch auch ein intelligenteres Verfahren entstehen. Dieses soll potentielle Kandidaten zum Serverwechsel ermitteln und nur bei diesen eine Neuberechnung durchführen.

4. DAS HEAT MAP VERFAHREN

Das Heat Map Verfahren ist der Algorithmus, der für die Lastverteilung zuständig ist. Ziel des Verfahrens ist es eine möglichst optimale Lösung auf Basis von zwei Anforderungen zu finden. Der Gruppierung der Avatare sowie der Auslastung des Servers. Avatare die sich nah beieinander befinden sollen nach Möglichkeit dem selben Server zugewiesen werden. Da sich diese potentiell viele Objekte in ihren AoIs teilen, kann auf diese Weise Speicher gespart werden. Zudem manipulieren diese Avatare potentiell dieselben Objekte in der Spielwelt. Ein Zusammenlegen dieser würde somit das Risiko von Kollisionen auf dem QuP verringern. Ist der Spielserver stark ausgelastet sollen Avatare jedoch einem alternativen Server zugewiesen werden. Dies ermöglicht es flexibel auf starke Lastsituationen zu reagieren und in solchen Fällen von der Gruppierung abzuweichen. Dies spiegelt sich auch in der Kostenfunktion wider:

$$cost(a, s) = w0 * -g(a, s) + w1 * cpu - w2 * memory \quad (1)$$

Diese Definiert die Kosten wenn ein Avatar a dem Server s zugewiesen wird. cpu ist die CPU-Auslastung und $memory$ der noch vorhandene Speicher des Spielservers. g ist die Qualität der Gruppierung laut dem Heat Map Algorithmus. Mit $w0$, $w1$ und $w2$ lassen sich diese Faktoren gewichten. [3]

4.1. Die Heat Map

Die Heat Map ist die Datenstruktur, in der die Verteilung der Avatare abgelegt wird. Wie im QuP (Abschnitt 3.1) wird die Spielwelt in einem dreidimensionalen Würfel verwaltet. Dieser ist in ein Raster unterteilt. Dabei definiert die Präzision P wie viele Zellen pro Achse vorliegen. Innerhalb der Heat Map wird verwaltet wie viele AoIs von Avataren sich in dem Bereich befinden. Abbildung 8a zeigt dies an einem Beispiel für einen Server. Der Sichtbereich eines jeden Avatars überlappt mit mehreren Zellen in dem Gitter. Für jedes AoI eines Avatars, welches in die Zelle reicht, wird der Wert um eins erhöht. Je mehr AoIs sich in einer Zelle befinden, desto wärmer ist diese. Verwaltet wird das für jeden Server einzeln. Dies geschieht mithilfe einer kombinierten Heat Map (Abb. 8b). Hier wird in jeder Zelle, für jeden Server eingetragen, wie der Wert der Zelle ist. [3]

4.2. Berechnung der Gruppierungsqualität

Zur Berechnung der Gruppierung wird jede Zelle betrachtet, mit der sich das AoI eines Avatars überschneidet. Für jeden Server wird nun die Summe aus den Werten berechnet, die sich in diesen Zellen befindet. Das Ergebnis entspricht dem Gruppierungsfaktor. Sollte es mit einem Spielserver keine überschneidenden Zellen geben, so wird die Manhattan Distanz zum nächsten Feld berechnet, in dem sich das AoI eines Avatars auf dem Server befindet. In diesem Fall ist der Gruppierungsfaktor $g = -Distanz$. Es konnte in einer vorherigen Arbeit bereits gezeigt werden, dass durch diesen direkten Zugriff auf das Raster, die Berechnung eines Spielservers in konstanter Zeit erfolgen kann. [3]

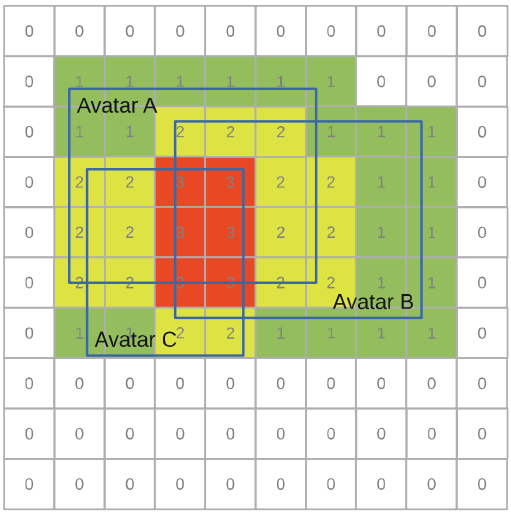
Ein Sonderfall, der noch betrachtet werden muss, sind Server denen kein Avatar zugewiesen ist. In einem solchen Fall ist eine Berechnung nach dem obigen Schema nicht möglich. Deshalb wird von einer idealisierten Verteilung ausgegangen. Dazu wird zunächst berechnet wie viele Felder ein Server in einer perfekt verteilten Spielwelt verwalten sollte:

$$IdealFieldCount = g0\frac{P^3}{ServerCount} + g1\frac{UsedFields}{ServerCount} \quad (2)$$

Hierbei wird mittels $g0$ und $g1$ zwischen zwei Idealen gewichtet. Zum einen, dass die komplette Spielwelt gleichmäßig über die Spielserver verteilt ist und zum anderen, dass die belegten Felder gleichmäßig über die Spielserver verteilt sind. Mithilfe der durchschnittlich genutzten Felder pro Server, wird dieser Wert in Relation zur tatsächlichen Verteilung gestellt:

$$Distance = \frac{\sqrt[3]{IdealFieldCount} - \sqrt[3]{AvgFieldCount}}{2} \quad (3)$$

Dieser Wert entspricht dem Distanzwert für nicht belegte Spielserver. Wie zuvor gilt auch hier $g = -Distance$. [3] Für alle berechneten Gruppierungsfaktoren g in diesem Abschnitt gilt, dass diese anschließend mit $(\frac{1}{P})^3$ multipliziert werden, damit die berechnete Zahl unabhängig von der eingestellten Präzision ist.



(a)

[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
[ ]	[{a,1}]	[{a,1}]	[{a,1}]	[{a,1}]	[{a,1}]	[{a,1}]	[ ]	[ ]	[ ]
[ ]	[{a,1}]	[{a,1}]	[{a,2}]	[{a,2}]	[{a,2}]	[{a,1}]	[{a,1}]	[{a,1}]	[ ]
[ ]	[{a,2}]	[{a,2}, {b,1}]	[{a,3}, {b,1}]	[{a,3}, {b,1}]	[{a,3}, {b,1}]	[{a,2}, {b,1}]	[{a,1}, {b,1}]	[{a,1}, {b,1}]	[{b,1}]
[ ]	[{a,2}]	[{a,2}, {b,1}]	[{a,3}, {b,2}]	[{a,3}, {b,2}]	[{a,3}, {b,2}]	[{a,2}, {b,2}]	[{a,1}, {b,2}]	[{a,1}, {b,1}]	[{b,1}]
[ ]	[{a,2}]	[{a,2}, {b,1}]	[{a,3}, {b,2}]	[{a,3}, {b,3}]	[{a,3}, {b,3}]	[{a,2}, {b,3}]	[{a,1}, {b,2}]	[{a,1}, {b,1}]	[{b,1}]
[ ]	[{a,1}]	[{a,1}, {b,1}]	[{a,2}, {b,2}]	[{a,2}, {b,3}]	[{a,1}, {b,3}]	[{a,1}, {b,3}]	[{a,1}, {b,2}]	[{a,1}, {b,1}]	[{b,1}]
[ ]	[ ]	[{b,1}]	[{b,2}]	[{b,3}]	[{b,3}]	[{b,3}]	[{b,2}]	[{b,1}]	[{b,1}]
[ ]	[ ]	[{b,1}]	[{b,2}]	[{b,2}]	[{b,2}]	[{b,2}]	[{b,2}]	[{b,1}]	[{b,1}]
[ ]	[ ]	[{b,1}]	[{b,1}]	[{b,1}]	[{b,1}]	[{b,1}]	[{b,1}]	[{b,1}]	[{b,1}]

(b)

Abbildung 8: Die Heat Map für einen Server mit den verschiedenen AoIs der Avatare (a) und die kombinierte Heat Map zur Darstellung der Heat Maps verschiedener Server (b) [3]

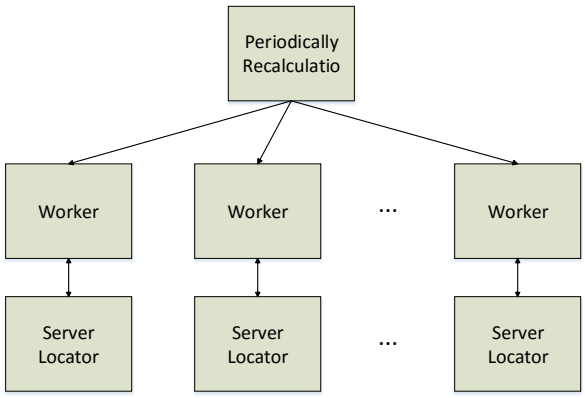


Abbildung 9: Aufbau des periodischen Recalculators

4.3. Reassignment

Beim Reassignment werden die Avatare einem neuen Server zugeordnet, wenn die Verteilung aus Sicht des Load Balancers nicht mehr optimal ist. Damit dies möglich ist, muss auf Seiten des Load Balancers eine regelmäßige Neuberechnung stattfinden. Bereits in einer vorherigen Arbeit wurde beschrieben, dass aktuell die Neuberechnung für jeden Avatar periodisch erfolgt [3]. Im Zuge dieser Arbeit wurde jedoch eine dynamischere Architektur entworfen, die es ermöglicht verschiedene Strategien zu konfigurieren.

Die Neuberechnung findet im Server Recalculator statt. In der aktuellen Version geschieht dies für jeden Avatar periodisch. Dazu wird für jeden Avatar ein eigener Workerprozess gestartet, der in zuvor definierten Zeitintervallen den konfigurierten Server Locator eine Serverberechnung für den Avatar machen lässt (Abb. 9). Stellt der Worker fest, dass sich der berechnete Server von dem zugewiesenen Server unterscheidet, so wird ein Migrations-Prozess gestartet.

5. DIE TESTUMGEBUNG

Die Testumgebung wurde mithilfe des Common Test Frameworks von Erlang realisiert. Dabei startet ein Master-Knoten alle nötigen Komponenten im Netzwerk als Slaves. Dies ermöglicht es die Testumgebung auf allen Umgebungen zu starten sobald ein Erlang-Deamon auf den beteiligten Rechnern installiert ist. Während der Tests werden regelmäßig Daten der beteiligten Server aufgezeichnet. Dabei können zu folgende Aspekten Daten erhoben werden:

- Initiale Zuweisung der Avatare
- Position der Avatare auf den Spielservern
- Avatarmigratiom
- Auslastung der Server

Bei der initialen Zuweisung wird aufgezeichnet, welcher Avatar zu welchem Zeitpunkt, welchen Server zugewiesen bekommen hat, nachdem dieser sich initial zur Spielwelt angemeldet hat. Hier wird auch betrachtet wie lange diese Berechnung gedauert hat. Die Position der Avatare auf den Spielservern wird mittels regelmäßiger Abfrage der Spielserever ermittelt. Hierbei wird zu jedem Spielserver

aufgezeichnet welche Avatare sich auf welcher Position auf dem Spielserver befinden. Zur Beobachtung der Avatarmigration wird jede erfolgreiche Migration eines Avatars festgehalten. Hierbei wird aufgezeichnet welchem Server ein Avatar zu welchem Zeitpunkt zugewiesen wurde. Da es sich bei allen Servern um Slaves des Testsystems handelt, kann die Auslastung über einfache RPC-Aufrufe ermittelt werden. Hierbei soll vor allem die CPU-Auslastung sowie die Speicherauslastung beobachtet werden.

5.1. Aufbau

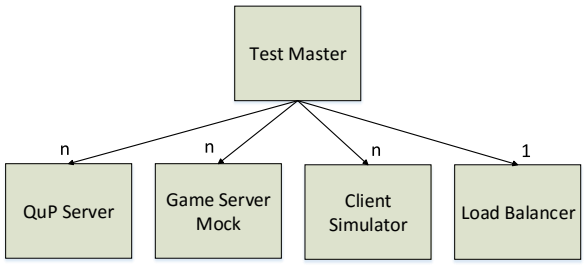


Abbildung 10: Das Testsystem als Master-Slave-System

Die Testumgebung ist als Master-Slave-System umgesetzt worden. Hierbei gibt es einen Test-Master von dem aus alle benötigten Server als Slaves gestartet werden. Dieser Master definiert wie die Testumgebung auszusehen hat und verwaltet sämtliche Statistiken, die von den Komponenten erfasst werden. Eine Reihe von Systemen der Testumgebung können mehrfach gestartet werden, da sie in Clustern arbeiten können. Zu diesen gehören der QuP-Server, der Spielserver-Mock sowie der Client Simulator. Vom Load Balancer wird in jedem Fall nur eine Instanz gestartet. (Abb. 10)

Beim Spielserver-Mock (Abb. 11) sowie dem Client-Simulator

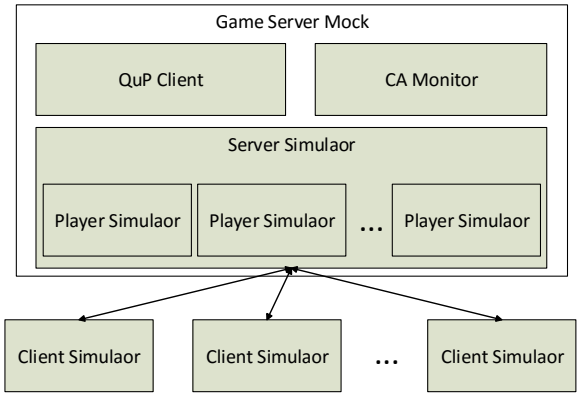


Abbildung 11: Aufbau des Spielserver-Mocks

handelt es sich um Komponenten die extra für die Testumgebung implementiert wurden. Der Spielserver-Mock soll eine Spielserverinstanz darstellen und dessen Verhalten simulieren. Entsprechend sind der QuP-Client und CA Monitor zwei interne Komponenten, mit denen benötigte Objekte bezogen werden bzw. die Lastinformationen an den Load Balancer übermittelt werden. Innerhalb der Spielserversimulation wird für jeden dem Server zugewiesenen Avatar ein Player-Simulator-Prozess gestartet. Dieser simuliert die Bewegung der Spieler. Das genaue Pattern ist hierbei dynamisch konfigurierbar. Je nachdem welche Konfiguration zum Start eines Avatars übergeben wird, können unterschiedliche Implementierungen gestartet werden. So können in einer Simulation auch verschiedene Arten von Avataren gestartet werden. In den aktuell vorliegenden Implementierungen wird die Bewegung auf dem Server gesteuert. Eine clientseitige Kontrolle wäre jedoch durch die Implementierung einer weiteren Variante möglich. Die Client Simulatoren verwalten die Anbindung der Avatare. D.h. für jeden im System vorliegenden Avatar existiert ein Client Simulator, der diesen an einen Spielserver anbindet. Initial wird der Client mit dem durch den Load Balancer berechneten Spielserver gestartet, dieser wird durch den Client Simulator kontaktiert und der Avatar dem Spielserver zugewiesen. In diesem Schritt teilt dieser dem Spielserver auch mit, um welche Art von Avatar es sich handelt. Erhält der Spielserver vom Load Balancer die Nachricht, dass ein Avatar den Spielserver wechseln soll, so teilt der Spielserver dies dem Client Simulator mit und informiert diesen auf welchen Server er wechseln soll. Anschließend meldet sich der Client beim neuen Server an. Für zukünftige Entwicklungen kann es auch denkbar sein mehr Verantwortung an die Clients zu übertragen, sodass diese die Bewegung der Avatare steuern. Auf diese Weise könnten auch zusätzlich Netzwerkeigenschaften, wie Verzögerungen, betrachtet werden, wie es in [43] geschehen ist.

5.2. Initialisieren der Spielwelt

Bevor die Simulation gestartet wird, wird diese mittels des World Creators erzeugt. Das genaue Muster nach dem eine Spielwelt erzeugt werden soll ist wiederum konfigurierbar. Sollte eine weitere Form, z.B. durch Auslesen einer Datei, benötigt werden, so muss diese nur in einem neuen Modul implementiert werden und kann mithilfe der Konfiguration angestoßen werden. Aktuell liegt ein Modell vor, in dem die Avatare auf Basis einer vorgegebenen Verteilung in der Spielwelt verteil werden. Hierbei gibt es die Muster gleichmäßig, gedrängt und in Cluster. Bei diesen handelt es sich um die Standardmuster, wie sie in vielen Tests eingesetzt werden [26–28, 33, 36]. Die genaue Aufteilung ist konfigurierbar. So kann bestimmt werden wie viele Avatare im gedrängten Szenario in welchem Bereich gedrängt sind. Bei der Aufteilung in Cluster kann definiert werden wie

viele Cluster es gibt, wie groß diese sein sollen und wie viele Avatare sich in diesen befinden sollen.

Der Avatar Spawner startet die Client Simulatoren. Dazu kontaktiert dieser zunächst den Load Balancer und übergibt den initialen Spielserver dem Client Simulator. Der aktuelle Avatar Spawner startet die Avatare in fest definierbaren Intervallen. Dazu wird definiert wie viele Avatare sofort gestartet werden sollen. Alle weiteren Avatare werden in dem definierten Intervall gestartet. Die Art und Weise wie sich Avatare bewegen sollen kann dabei übergeben werden. Aktuell gibt es ein Muster, in dem sich die Avatare gerade in Richtung eines Ziels bewegen und eines in dem sich die Avatare in einem zufälligen Muster bewegen.

Durch die dynamische Implementierung können neue Verteilungsmuster durch implementieren neuer Module realisiert werden. Denkbar wären so realistischere Verteilungen, wie sie z.B. Pittman et al. ermittelt haben. Dort wurde festgestellt, dass sich Spieler nur in wenigen Bereichen ballen. Sie konnten ermitteln, dass die Spieler in den Zonen von World of Warcraft Exponentialverteilt sind [31].

5.3. Avatare bewegen

Die Bewegung der Avatare wird innerhalb der Spielserver-Mocks simuliert. Wie schon in Abschnitt 5.1 beschrieben ist das genaue Bewegungsmuster konfigurierbar. Durch Implementieren eines neuen Moduls ist es möglich neue Bewegungsmuster hinzuzufügen. Aktuell stehen zwei Muster zur Verfügung. Eines in dem sich die Avatare gerade auf ein Ziel zu bewegen und eines in dem sich die Avatare nach einem zufälligen Muster bewegen.

Bei den gerade laufenden Avataren wird jedem zur Initialisierung aus einer Menge von statisch definierten Zielen ein zufälliges ausgesucht, auf dass dieser sich zu bewegen soll. Zudem wird definiert mit welcher Geschwindigkeit sich die Avatare bewegen sollen. Das zufällige Bewegungsmuster ist eine Mischung aus Random Way Point (RWP) [18] und Hot Points All (HPA) [33]. Auch hier wird definiert in welcher Geschwindigkeit sich die Avatare bewegen sollen. Wie bei RWP können sich Avatare auf ein zufälliges Ziel zu bewegen. Hierzu wird jedem Avatar eine Menge zufälliger Ziele zugewiesen, auf welche sich die Avatare in zufälliger Reihenfolge zu bewegen. Innerhalb der Menge der Ziele sind jedoch auch eine bestimmte Menge von Zielen enthalten, die allen Avataren zugewiesen werden und sogenannte Hot Points darstellen sollen. Im Gegensatz zu Random Way Point bewegen sich die Avatare nicht zwingend bis zum angepeilten Ziel, sondern es wird ein zufälliges Zeitfenster ermittelt, für das sich ein Avatar auf ein Ziel zubewegt. Läuft dieses Zeitfenster ab oder erreicht ein Avatar das Ziel vor Ende der Zeit, wählt sich der Avatar, aus der zugewiesenen Menge an Zielen, ein zufälliges neues aus. Ziel des zweiten Bewegungsmusters war es Mängel der geraden Bewegung auszugleichen und ein realistischeres Bewegungsverhalten abzubilden.

5.4. Testfälle

Um zu zeigen, dass die Testumgebung funktionsfähig ist, werden an dieser Stelle zwei Tests vorgestellt. Beide haben ein ähnliches Setting. Der einzige Unterschied ist die Anzahl der Avatare und wie viele verzögert gestartet werden. Für beide Tests standen jeweils drei virtuelle Maschinen zur Verfügung. Bei diesen handelt es sich um Ubuntu 16.04 Maschinen, mit einem Intel Xeon Prozessor mit 2,4 GHz und 4 GB an RAM-Speicher. Das System wurde mit einem QuP-Server Knoten und einem Load Balancer Knoten gestartet. Zudem wurden drei Spielserver-Mocks initialisiert. Hierbei lagen zwei Spielserver-Mocks auf der selben virtuellen Maschine. Der dritte teilt sich eine Maschine mit dem Load Balancer. Der Qup-Server liegt zusammen mit den Spielclientsimulator-Prozessen auf dem dritten Server. In beiden Szenarien wurde das zufällige Bewegungsmuster mit drei gemeinsamen Zielen für alle Avatare eingesetzt. Die Avatare wurden dazu initial gleichmäßig über die virtuelle Welt verteilt. Zudem haben sie sich in einem Abstand von 350 ms um eine Strecke von 0,0001 bewegt. Die Tests dauerten jeweils 30 Minuten. Damit hat jeder Avatar am Ende des Tests eine Strecke von ca. 0,5 zurückgelegt.

Der erste Test erfolgte mit 2500 Avataren. Diese wurden alle zeitgleich zu Beginn des Tests gestartet. In Abb. 12 ist zu sehen wie sich die Verteilung der Avatare im Laufe der Simulation verändert hat. Hierbei repräsentiert jeder Punkt einen Avatar und die Farbe in der diese eingefärbt sind, den Server, dem die Avatare zugewiesen sind. Der Würfel in dem diese sich Bewegen ist die Simulationsumgebung. Es ist zu erkennen, dass das Verfahren in der Lage ist Avatare, die sich in räumlicher nähe befinden, demselben Server zuzuweisen. Auch ist deutlich sichtbar, dass sich die Verteilung der Avatare über die Spielserver mit der Zeit verändert. So sind deutlich mehr Avatare Spielserver 2 zu Minute 9 zugewiesen als zu Minute 0. Auch ist sichtbar, dass im Laufe der Zeit immer weniger Avatare Spielserver 1 zugewiesen sind, bis ab Minute 18 kein Avatar diesem Server mehr zugewiesen ist. Das Gruppieren von Avataren die sich in räumlicher Nähe befinden wird zudem besser je geballter Avatare beieinander stehen. Zusätzlich wurde analysiert wie viele Serverwechsel die Avatare getätigt haben (siehe Tabelle 1). Betrachtet man wie viele Avatare während des Tests den Server gewechselt haben, so sind dies 1353 und somit 54,1% der Avatare. Wird der Durchschnitt betrachtet, wie oft ein Avatar den Server gewechselt hat, so liegt dieser bei 0,966. Werden nur die Avatare betrachtet, die auch tatsächlich einen Wechsel vollzogen haben, so liegt dieser

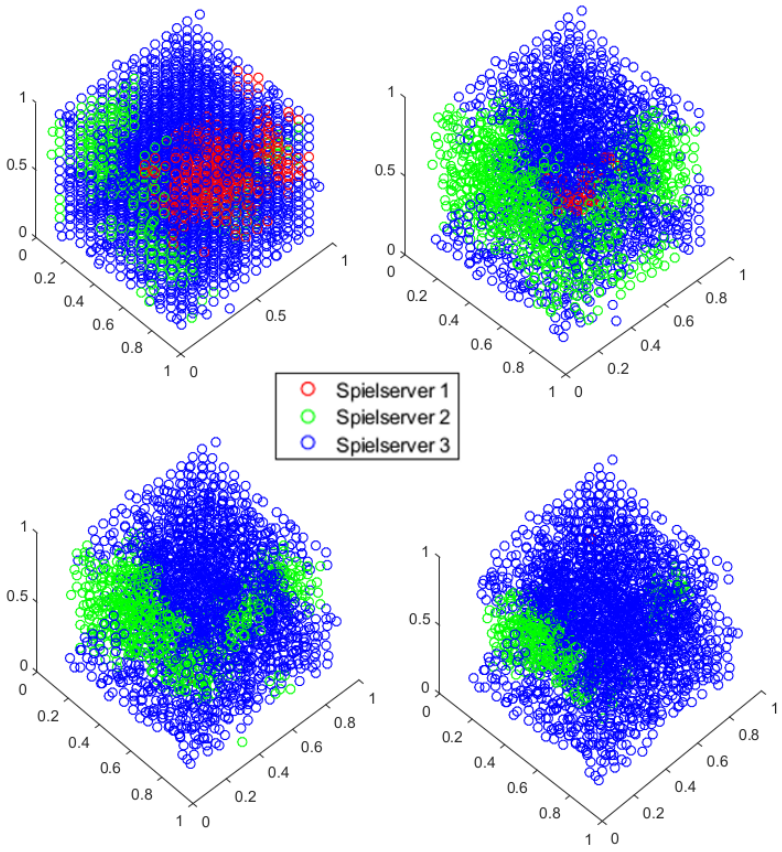


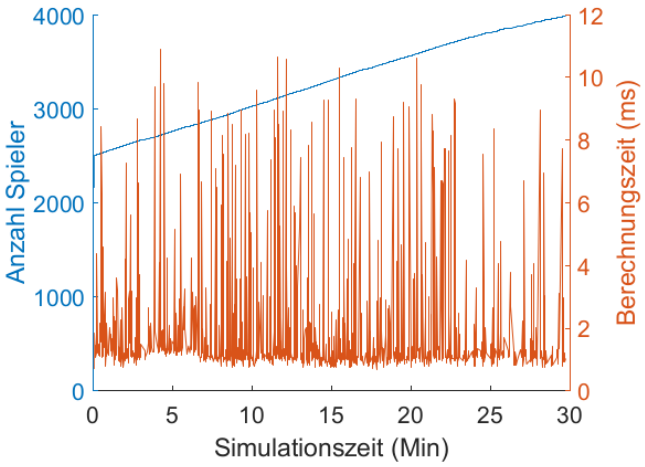
Abbildung 12: Zuweisung der Avatare zu: o.l.: Minute 0, o.r.: Minute 9, u.l.: Minute 18, u.r.: Minute 27

Tabelle 1: Wechsel der Avatare zwischen den Servern

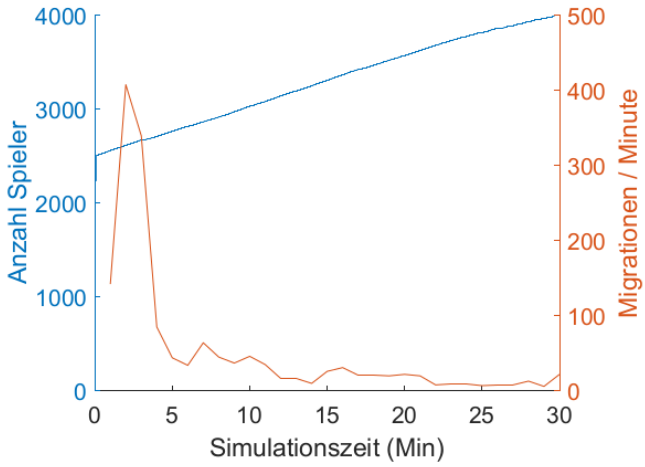
	Durchschnitt alle	Durchschnitt Avatare die gewechselt haben	Min	Max
Wechsel	0,966	1,7849	0	12
Anzahl Zugewiesene Server	1,602	2,1123	1	3

sogar bei 1,7849 und damit fast doppelt so hoch (1,84 mal mehr) wie im Gesamtdurchschnitt. Allerdings ist auch die Menge der Wechsel sehr unterschiedlich, während 45,9% der Avatare nicht einmal den Server gewechselt haben liegt der Höchstwert bei 12 mal. Dies entspricht alle 2,5 Minuten. Insgesamt war jeder Avatar im Schnitt 1,602 Spielservern zugewiesen. Werden nur die Avatare betrachtet, die auch Spielserver gewechselt haben, so liegt der Durchschnitt bei 2,1123.

Im zweiten Test wurde evaluiert wie sich eine steigende Anzahl an Avataren auf die Eigenschaften des Systems auswirkt. Dazu wurden in dem Test insgesamt 4000 Avatare eingesetzt, wobei 2500 sofort gestartet sind und die Restlichen 1500 nach Start des Tests in Abständen von einer Sekunde gestartet wurden. Zunächst wird betrachtet wie stark die zusätzliche Belastung die Berechnungszeit von Spielservern beeinflusst. In Abb. 13a ist die Berechnungszeit für neue Server zu den Anzahl der Spieler dargestellt. Zur besseren Übersicht handelt es sich hierbei um die 95er Perzentile. Es ist zu sehen, dass auch eine erhöhte Spielerzahl die Berechnungszeit nicht negativ beeinflusst. Sie hat einen Durchschnittswert von 1,147 ms und bleibt konstant um diesen Wert herum. In Abb. 13b wird dargestellt, wie sich die Anzahl der Serverwechsel mit steigender Anzahl an Avataren verändert. Dazu wird zu jeder Minute betrachtet wie viele Serverwechsel es in der letzten Minute gab. Da mehr Avatare vorliegen könnte erwartet werden, dass auch die Anzahl der Wechsel steigt, weil mehr Prozesse Neuberechnungen machen. Tatsächlich ist dies nicht der Fall. Am Anfang des Tests steigt die Anzahl der Wechsel stark bis auf 407 an. Danach fällt sie jedoch und bewegt sich ab Minute



(a) Berechnungszeit (95er Perzentile)



(b) Migrationen / Minute

Abbildung 13: Ergebnisse des zweiten Tests

8 durchgehend unterhalb von 50. Gegen Ende bewegt sich die Zahl der Wechsel pro Minute sogar unterhalb von 20. Damit ist der Trend fallend und nicht, wie intuitiv erwartet, steigend.

Es konnte gezeigt werden, dass durch die Tests mehrere Aspekte darstellbar sind. Näher zu untersuchen ist, warum die Anzahl der Migrationen bei mehr Avataren stärker abfällt. Eine mögliche Begründung könnte sein, dass das Verfahren mit der Zeit dazu neigt weniger Server zu beanspruchen und es sich zu Beginn vor allem um Wechsel von dem Server handelt der gegen Ende nur mit wenigen oder gar keinem Avatar belastet wird. In allen Fällen werden die Server noch mit unrealistisch hohen Zahlen an Avataren belastet. Werden andere Quellen betrachtet so liegt die Auslastung pro Server normalerweise bei um die 100 Spieler [21]. Durch den Einsatz von mehr Servern soll dies in Zukunft noch realistischer getestet werden.

6. FAZIT UND AUSBLICK

In dieser Arbeit konnte ein Überblick über bisherige Verfahren zur Verteilung von Avataren in verteilten virtuellen Systemen gegeben werden und wie diese getestet werden. Das bisherige System wurde um ein dynamisches Modul erweitert, welches in der Lage ist verschiedene Neuberechnungsstrategien zur Migration von Avataren durchzuführen. Außerdem wurde das Protokoll zwischen CA Monitor und Load Balancer erweitert, um den Austausch von Informationen zur Neuberechnung zu ermöglichen. Zudem wurde eine Testumgebung umgesetzt, die es ermöglicht das Verfahren flexibel zu testen und die Umgebung leicht zu erweitern, in dem neue Module implementiert werden.

Auf dieser Basis ist es möglich, die Testumgebung für zukünftige Gegebenheiten anzupassen. So könnte auch das Bewegungsmuster verfeinert werden, sodass sich Avatare noch realistischer bewegen. Ein weiterer Aspekt der in Zukunft noch betrachtet werden kann ist die Verzögerung durch Kommunikation zwischen den Clients und den Spielservern. Dabei wäre möglich sich an Zhang et al. zu orientieren [43].

Als nächste Schritte soll die Neuberechnung zur Serverzuweisung optimiert werden. Ziel soll es sein ein intelligenteres Verfahren zu haben, dass eine Neuberechnung gezielter auf potentielle Kandidaten ausführt. Zudem soll stärker darauf geachtet werden, dass die Anzahl der Wechsel optimiert wird und möglichst gering gehalten wird. Zudem soll überprüft werden inwiefern die Performance der Spielserver durch die Umverteilung gesteigert werden kann.

7. REFERENCES

[1] Ali, A. A. F. ; Ismail, A. A. S. A. ; Bade, A. : An Overview of Networking Infrastructure for Massively Multiplayer Online Games. In: *Comp.Utm.My* (2009), Nr. October 2008, 619–628. http://comp.utm.my/pars/files/2013/04/An-Overview-of-Networking-Infrastructure-for-Massively-Multiplayer-Online-Games.pdf

[2] Allers, S. : *Lastverteilung in einer clusterbasierten ver-*

teilten virtuellen Umgebung, HAW Hamburg, Bachelor of Science, 2015

[3] Behnke, L. ; Allers, S. ; Wang, Q. ; Grecos, C. ; Luck, K. von: Avatar Density Based Client Assignment. In: *Entertainment Computing – ICEC 2016* Bd. 1, 2016. – ISBN 9783319461007, S. 137–148

[4] Behnke, L. ; Grecos, C. ; Luck, K. von: QuP : Graceful Degradation in State Propagation for DVEs. (2014)

[5] Broch, J. ; Maltz, D. A. ; Johnson, D. B.: A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In: *International Conference on Mobile Computing and Networking* (1998), 1–13. http://dx.doi.org/10.1145/288235.288256. – DOI 10.1145/288235.288256. – ISBN 158113035X

[6] Carlini, E. ; Ricci, L. ; Coppola, M. : Flexible load distribution for hybrid distributed virtual environments. (2013). http://dx.doi.org/10.1016/j.future.2012.09.004. – DOI 10.1016/j.future.2012.09.004

[7] Chen, J. ; Wu, B. ; Delap, M. ; Knutsson, B. ; Lu, H. ; Amza, C. : Locality aware dynamic load management for massively multiplayer games. In: *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming - PPoPP '05* (2005), 289. http://dx.doi.org/10.1145/1065944.1065982. – DOI 10.1145/1065944.1065982. ISBN 1595930809

[8] De Vleeschauwer, B. ; Van Den Bossche, B. ; Verdickt, T. ; De Turck, F. ; Dhoedt, B. ; Demeester, P. : Dynamic microcell assignment for massively multiplayer online gaming. In: *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games - NetGames '05* (2005), 1–7. http://dx.doi.org/10.1145/1103599.1103611. – DOI 10.1145/1103599.1103611. ISBN 1595931562

[9] Deng, Y. ; Lau, R. W. H.: Heat diffusion based dynamic load balancing for distributed virtual environments. In: *Proceedings of the 17th ACM Symposium on Virtual Reality Software and Technology* 1 (2010), Nr. 212, 203–210. http://dx.doi.org/http://doi.acm.org/10.1145/1889863.1889910. – DOI http://doi.acm.org/10.1145/1889863.1889910. ISBN 978–1–4503–0441–2

[10] Deng, Y. ; Lau, R. W. H.: On delay adjustment for dynamic load balancing in distributed virtual environments. In: *IEEE Transactions on Visualization and Computer Graphics* 18 (2012), Nr. 4, S. 529–537. http://dx.doi.org/10.1109/TVCG.2012.52. – DOI 10.1109/TVCG.2012.52. – ISSN 10772626

[11] Deng, Y. ; Lau, R. W. H.: Dynamic load balancing in distributed virtual environments using heat diffusion. In: *ACM Transactions on Multimedia Computing, Communications, and Applications* 10 (2014), feb, Nr. 2, 1–19. http://dx.doi.org/10.1145/2499906. – DOI 10.1145/2499906. – ISSN 15516857

[12] Elfizar ; Baba, M. S. ; Herawan, T. : Object-Based Viewpoint For Large-Scale Distributed Virtual Environment. 28 (2015), Nr. 4, S. 301–317

[13] Falkenauer, E. : A New Representation and Operators for Genetic Algorithms Applied to Grouping Problems. In: *Evolutionary Computation* 2 (1994), jun, Nr. 2, 123–144. http://dx.doi.org/10.1162/evco.1994.2.2.123. – DOI 10.1162/evco.1994.2.2.123. – ISSN 1063–6560

[14] Greenhalgh, C. : Analysing movement and world transi- tions in virtual reality tele-conferencing. (1997), S. 313–328

[15] Hindman, B. ; Konwinski, A. ; Zaharia, M. ; Ghodsi, A. ; Joseph, A. D. ; Katz, R. H. ; Shenker, S. ; Stoica, I. : Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In: *NSDI* 11 (2011)

[16] Hong, X. ; Gerla, M. ; Pei, G. ; Chiang, C.-C. : A group mobility model for ad hoc wireless networks. In: *Proceedings of the 2nd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems - MSWiM '99* (1999), 53–60. http://dx.doi.org/10.1145/313237.313248. – DOI 10.1145/313237.313248. ISBN 1581131739

[17] Iimura, T. ; Hazeyama, H. ; Kadobayashi, Y. : Zoned Federation of Game Servers: a Peer-to-peer Approach to Scalable Multi-player Online Games. In: *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games - NetGames '04* (2004), 116–120. http://dx.doi.org/10.1145/1016540.1016549. – DOI 10.1145/1016540.1016549. ISBN 158113942X

[18] Johnson, D. B. ; Maltz, D. A.: Dynamic Source Routing in Ad Hoc Wireless Networks. In: *Mobile Computing* 353 (1996), 153–181. http://dx.doi.org/10.1007/b102605. – DOI 10.1007/b102605. ISBN 978–0–7923–9697–0

[19] Kazem, I. ; Ahmed, D. T. ; Shirmohammadi, S. : A visibility-driven approach to managing interest

in distributed simulations with dynamic load balancing. In: *Proceedings - IEEE International Symposium on Distributed Simulation and Real-Time Applications, DS-RT* (2007), S. 31–38. http://dx.doi.org/10.1109/DS-RT.2007.11. – DOI 10.1109/DS–RT.2007.11. – ISBN 0769530117

[20] Knutsson, B. ; Hopkins, B. ; Lu, H. ; Xu, W. : Peer-to-peer support for massively multiplayer games. In: *Ieee Infocom* 1 (2004), Nr. C, 96–107. http://dx.doi.org/10.1109/INFCOM.2004.1354485. – DOI 10.1109/INFCOM.2004.1354485. – ISBN 0–7803–8355–9

[21] Kushner, D. : Engineering everquest. In: *IEEE SPECTRUM* 42 (2005), Nr. 7, S. 28

[22] Lau, R. W.: Hybrid load balancing for online games. In: *Proceedings of the international conference on Multimedia - MM '10* (2010), 1231. http://dx.doi.org/10.1145/1873951.1874194. – DOI 10.1145/1873951.1874194. ISBN 9781605589336

[23] Lee, Y. T. ; Chen, K. T.: Is server consolidation beneficial to MMORPG? A case study of world of warcraft. In: *Proceedings - 2010 IEEE 3rd International Conference on Cloud Computing, CLOUD 2010* (2010), S. 435–442. http://dx.doi.org/10.1109/CLOUD.2010.57. – DOI 10.1109/CLOUD.2010.57. ISBN 9780769541303

[24] Li, Y. ; Cai, W. : Consistency-Aware Zone Mapping and Client Assignment in Multi-Server Distributed Virtual Environments. 26 (2015), Nr. 6, S. 1570–1579

[25] Lu, F. ; Parkin, S. ; Morgan, G. : Load balancing for massively multiplayer online games. In: *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games - NetGames '06* (2006), 1. http://dx.doi.org/10.1145/1230040.1230064. – DOI 10.1145/1230040.1230064. ISBN 1595935894

[26] Lui, J. C. S. ; Chan, M. F.: An efficient partitioning algorithm for distributed virtual environment systems. In: *IEEE Transactions on Parallel and Distributed Systems* 13 (2002), Nr. 3, 193–211. http://dx.doi.org/10.1109/71.993202. – DOI 10.1109/71.993202. – ISSN 10459219

[27] Morillo, P. ; Fernández, M. ; Orduña, J. : An ACS-based partitioning method for distributed virtual environment systems. In: *Proceedings International Parallel and Distributed Processing Symposium* 00 (2003), Nr. C. http://dx.doi.org/10.1109/IPDPS.2003.1213283. – DOI 10.1109/IPDPS.2003.1213283. – ISBN 0–7695–1926–1

[28] Morillo, P. ; Orduña, J. M. ; Fernández, M. ; Duato, J. : An Adaptive Load Balancing Technique For Distributed Virtual Environment Systems. In: *Proceedings of the 15th IASTED international PDCS-03* (2003), S. 256–261

[29] Neumann, C. ; Prigent, N. ; Varvello, M. ; Suh, K. : Challenges in peer-to-peer gaming. In: *ACM SIGCOMM Computer Communication Review* 37 (2007), Nr. 1, S. 79. http://dx.doi.org/10.1145/1198255.1198269. – DOI 10.1145/1198255.1198269. – ISBN 0146–4833

[30] Ng, B. ; Si, A. ; Lau, R. W. ; Li, F. W.: A multiserver architecture for distributed virtual walkthrough. In: *Proceedings of the ACM symposium on Virtual reality software and technology - VRST '02* (2002), 163. http://dx.doi.org/10.1145/585767.585768. – DOI 10.1145/585767.585768. ISBN 1581135300

[31] Pittman, D. ; GauthierDickey, C. : A measurement study of virtual populations in massively multiplayer online games. In: *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games - NetGames '07* (2007), 25–30. http://dx.doi.org/10.1145/1326257.1326262. – DOI 10.1145/1326257.1326262. ISBN 9780980446005

[32] Rosedale, P. ; Ondrejka, C. : *Enabling Player-Created Online Worlds with Grid Computing and Streaming*. http://www.gamasutra.com/resource{_}guide/20030916/rosedale{_}pfv.htm. Version: 2003, Abruf: 2015-08-10

[33] Rueda, S. ; Morillo, P. ; Orduña, J. M. ; Valencia, U. D.: A Peer-To-Peer Platform for Simulating Distributed Virtual Environments. (2007). ISBN 9781424418909

[34] Sánchez, M. ; Manzoni, P. : ANEJOS: A Java based simulator for ad hoc networks. In: *Future Generation Computer Systems* 17 (2001), Nr. 5, S. 573–583. http://dx.doi.org/10.1016/S0167-739X(00)00040-6. – DOI 10.1016/S0167–739X(00)00040–6. – ISSN 0167739X

[35] Singh, H. L. ; Gračanin, D. ; Matković, K. : A Load Simulation and Metrics Framework for Distributed Virtual Reality. (2008), S. 287–288. ISBN 9781424419715

[36] Suznjevic, M. ; Matijasevic, M. : Player behavior and traffic characterization for MMORPGs : a survey. (2013), S. 199–220. http://dx.doi.org/10.1007/s00530-012-0270-4. – DOI 10.1007/s00530–012–0270–4

[37] Ta, D. N. B. ; Zhou, S. : A network-centric approach to enhancing the interactivity of large-scale distributed virtual environments. In: *Computer Communications* 29 (2006), Nr. 17, S. 3553–3566. http://dx.doi.org/10.1016/j.comcom.2006.05.015. – DOI 10.1016/j.comcom.2006.05.015. – ISSN 01403664

[38] Ta, D. N. B. ; Zhou, S. ; Cai, W. ; Tang, X. ; Ayani, R. : Efficient zone mapping algorithms for distributed virtual environments. In: *Proceedings - Workshop on Principles of Advanced and Distributed Simulation, PADS* (2009), S. 137–144. http://dx.doi.org/10.1109/PADS.2009.10. – DOI 10.1109/PADS.2009.10. ISBN 9780769537139

[39] Tveit, A. ; Rein, Ø. ; Iversen, J. V. ; Matskin, M. : Scalable Agent-Based Simulation of Players in Massively Multiplayer Online Games. In: *Eighth Scandinavian Conference on Artificial Intelligence: SCAI*, 2003

[40] Valadares, A. ; Gabrielova, E. ; Lopes, C. V.: On Designing and Testing Distributed Virtual Environments. (2015)

[41] Waters, R. C. ; Barrus, J. W.: The rise of shared virtual environments. In: *Spectrum, IEEE* 34 (1997), Nr. 3, S. 20–25. http://dx.doi.org/10.1109/6.576004. – DOI 10.1109/6.576004

[42] X, C. P.: *Building a Balanced Universe - EVE Community*. http://community.eveonline.com/news/dev-blogs/building-a-balanced-universe/. Version: 2013, Abruf: 2017-01-15

[43] Zhang, W. ; Zhou, H. : A Dynamic Mapping Method to Keep Region Connectedness in DVE Systems. (2016), S. 0–3