Klient-Server Architekturen

Matthias Schmid
matthias.schmid@informatik.uni-ulm.de

27. Juni 2005

Inhaltsverzeichnis

1 Einleitung

Diese Ausarbeitung zum Proseminar Virtuelle Präsenzen soll zunächst beleuchten, welche Netzwerk-Architekturen sich für virtuelle Präsenzen eignen. Es wird betrachtet, welche Vor- und Nachteile die Klient-Server Architektur, die Thema dieses Vortrages ist, im Speziellen hat. Schließlich werden verschiedene Techniken erläutert, die zum Ziel haben, die Probleme der Klient-Server Architektur zu minimieren.

2 Anforderungen

Ego-Shooter stellen hohe Anforderungen an Netzwerk-Architekturen. Sie sind auf geringe Verzögerungen angewiesen (ideal wären weniger als 100 ms) und benötigen einen hohen Grad an Konsistenz.
Ruckelt das Spiel, oder steht der Gegner, der gerade eben abgeschossen wurde, in der nächsten Sekunde wieder auf, schränkt das den Spielspass ein.
Strategie- und Rollenspiele sind in der Hinsicht genügsamer, es stört z.B. nicht allzusehr, wenn das Geld für einen Gegenstand, der eben verkauft wurde, erst eine halbe Sekunde später wieder in Besitz des Spielers ist. Wünschenswert ist jedoch trotzdem eine Minimierung der geforderten Bandbreite, so dass das Netzwerk nicht unnötig belastet wird. Ebenfalls ist eine einfache Administration von Vorteil, d.h. dass z.B. einfach Patches eingespielt werden können. So können unnötige Kosten und unnötiger Aufwand weitgehend vermieden werden.
Ich denke virtuelle Präsenzen lassen sich zwischen diesen beiden Kategorien einordnen. Wünschenswert ist hier eine Echtzeitinteraktion mit anderen Teilnehmern der virtuellen Präsenz, sowie dass alle Beteiligten die gleiche Ansicht auf eine komplexe virtuelle Welt teilen.

3 Definitionen

Eine Klient-Server Architekur besteht im Wesentlichen aus zwei (oder mehreren) Rechnern, zum Einem dem Dienstleister (genannt Server), zum Anderen einem (oder mehreren) Dienstanforderern (genannt Client). Sie besteht im Allgemeinen aus 3 Komponenten: der Datenhaltung, der Applikation und der Präsentation.
Je nach dem, wie diese Komponenten auf den Server oder den Client aufgeteilt werden, entstehen verschiedenen Varianten: die zentrale Klient-Server Architektur und mehrstufige Klient-Server Architekturen.

4 Übersicht über verschiedene Architekturen

- Zentral (Klient-Server)
  Jeder Klient ist direkt mit dem Server verbunden.

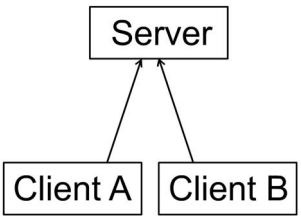

Abbildung 1: Klient-Server Architektur

Hier liegt die Applikation und die Datenhaltung auf dem Server, ausschließlich die Präsentation findet auf den Clients statt.

- Verteilt (Peer-to-Peer)
  Jeder Rechner (Peer) ist mit jedem anderen Peer verbunden.

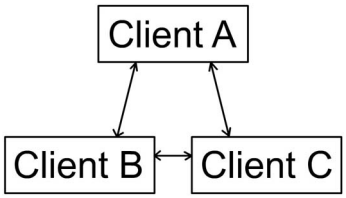

Abbildung 2: Peer to Peer

- Kompromiss: Mirrored-Server
  Jeder Klient ist mit einem Server verbunden und die Server wieder unter-

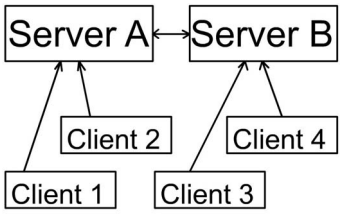

Abbildung 3: Mirrored-Server Architektur

einander als Peers. Hier sind Datenhaltung und Applikation auf mehrere Server verteilt, die Präsentation findet auf den Clients statt.

In dieser Ausarbeitung wird die Klient-Server Architektur sowie deren Variationen behandelt. Zur Peer-to-Peer Architektur sei auf den gleichnamigen Vortrag verwiesen, der ebenfalls in diesem Seminar gehalten wurde.

4.1 Die zentrale Klient-Server Architektur

Ablauf Jeder Client für sich übergibt dem Server Kommandos, die der Benutzer eingegeben hat (z.B. einen Tastendruck, um die Spielfigur zu bewegen).
Der Server ist nun alleine verantwortlich den aktuellen Spielzustand zu berechnen. Dazu berechnet er den neuen Spielzustand auf Basis des alten unter Berücksichtigung der eingegangenen Kommandos. Er sendet anschließend Updates an alle Clients. Die Clients wiederum stellen nun den aktuellen Spielzustand dar. Der Server wartet nun wieder auf Kommandos von den Clients.
Zusätzlich ist der Server zuständig für die Verwaltung von Spielen: Auf ihm können Spiele gestartet und beendet werden. Sowie ist er dafür verantwortlich, Verbindungen von Klienten zu akzeptieren.

Vor- und Nachteile Die Vorteile sind, dass es praktisch keine Konsistenzprobleme gibt, da alles vom Server berechnet wird. Zudem ist diese Architektur einfach zu administrieren, da sich alle Teilnehmer auf einem Server aufhalten.
Leider wird der einzelne Server auch zum Nachteil. Fällt er aus, kann auf das ganze System nicht mehr zugegriffen werden. Außerdem kann die Bandbreite zum Server schnell zum Flaschenhals werden.

4.2 Mehrstufige Architekturen

Die bisher vorgestellte Architektur ist als zweistufig (two-tiered) zu bezeichnen.

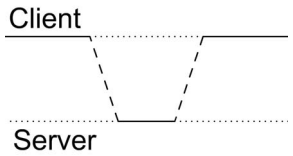

Abbildung 4: Zweistufige Architektur

Ablauf Der Client fordert Daten von Server an und wartet bis er ein Ergebnis zurückerhält.

Erweiterung Ebenfalls denkbar sind nun mehrstufige Architekturen (multi-tiered). Hier werden die verschiedenen Schichten auf verschiedene Server aufgeteilt. Zum Beispiel ist eine Aufteilung in einen Applikations-Server und einen Datenhaltungs-Server denkbar. Fordert der Client nun Daten an, muss der Applikations-Server zunächst den Datenhaltungs-Server kontaktieren.

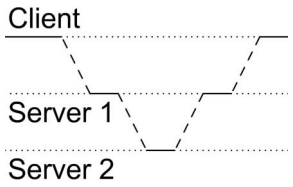

Abbildung 5: Eine Dreistufige Architektur (three-tiered)

Vorteil Mehrstufige Architekturen skalieren wesentlich besser als zweistufige.

4.3 Die Mirrored-Server Architektur

Die Mirrored-Server Architektur ist eine Kombination aus zentralem Klient-Server und P2P. Jeder Client kommuniziert mit einem einzelnen dieser Server. Untereinander kommunizieren die Server z.B. per Multicast, d.h. eine Nachricht wird gleichzeitig und zuverlässig an alle anderen Server gesendet.

Vor- und Nachteile Die Vorteile, die daraus resultieren, liegen auf der Hand: Fällt ein Server aus, so kann trotzdem auf das System zugegriffen werden (die Redundanz wird erhöht). Leider entstehen auch Nachteile, so ist die Konsistenz nicht mehr gegeben, da diese Architektur mit mehreren parallelen Simulationszuständen umgehen muss. Daher ist eine Synchronisierung notwendig. Ebenso werden mehrere Server und ein Netzwerk, welches diese verbindet, benötigt.

5 Kommunikation

Die Nachrichten, die der Client an den Server sendet, nennen sich *Kommandos*. Sie bestehen hauptsächlich aus den Eingaben des Anwenders, wie z.B. ein Befehl, die Spielfigur zu bewegen. Netzwerk-Pakete, die Kommandos enthalten, werden normalerweise unzuverlässig gesendet. Deshalb enthält jedes Paket die letzten 3 Kommandos, um verloren gegangene Pakete zu kompensieren. Nachrichten vom Server an den Client nennen sich *Statusinformationen*. Sie geben z.B. Auskunft über die aktuelle Position des Spielers oder seine Lebensenergie.

6 Synchronisierungsmechanismen

6.1 Konservative Algorithmen

Es existieren einfache Algorithmen, z.B. Lockstep, die sich um die Synchronisierung kümmern. Hierbei wird einfach die Simulationszeit angehalten, bis alle Clients fertig sind mit ihren Berechnungen. Keinem Client wird erlaubt seine Simulationszeit fortzusetzen, bevor nicht alle anderen bestätigt haben, dass sie genauso weit sind. Dieser Algorithmus wird als konservativ bezeichnet. Leider ist er nicht anwendbar für realistisch wirkende virtuelle Präsenzen, da so große Verzögerungen entstehen würden, dass eine Echtzeitinteraktion nicht möglich wäre.

6.2 Optimistische Algorithmen

Es ist besser, wenn Ereignisse optimistisch berechnet werden. So kann ein wesentlich besseres Antwortverhalten erreicht werden. Durch einen flüssigen Ablauf der Simulation sind diese Algorithmen für virtuelle Präsenzen besser geeignet. Durch die optimistischen Berechnungen können jedoch leider auch Fehler auftreten. Diese müssen erkannt und korrigiert werden. Es werden zwei Beispiele vorgestellt, die Time Warp Synchronisierung und die Trailing State Synchronisierung.

6.2.1 Time Warp Synchronisierung

Idee Vom aktuellen Simulationszustand werden in bestimmten Abständen Schnappschüsse (Snapshots) gespeichert. Geht nun ein Ereignis verspätet ein, wird zu einem älteren Zustand gewechselt (Rollback), damit dieses Ereignis berücksichtigt werden kann. Um nun wieder auf den aktuellen Stand zu kommen, müssen alle Ereignisse, die danach eintrafen, noch einmal ausgeführt werden.

Nachteil Die Snapshots haben einen hohen Speicher- und Rechenzeit-Aufwand

6.2.2 Trailing State Synchronisierung

Idee Ereignisse werden hier grundsätzlich verzögert ausgeführt, dadurch können sie in gewissem Rahmen umsortiert werden, falls sie in der falschen Reihenfolge eintreffen. Um feststellen zu können, welches die richtige Reihenfolge ist, müssen sie mit Zeitstempeln versehen sein.
Die Idee, die hinter TSS steckt, ist, dass mehrere Kopien der Simulation (Zustände) parallel berechnet werden. Der führende Zustand ist der optimistischste: er wartet nur kurz, bis er Ereignisse ausführt. Dies ist der Zustand, der auf Grund seiner schnellen Reaktionszeit auch den Clients mitgeteilt wird. Zusätzlich werden nun

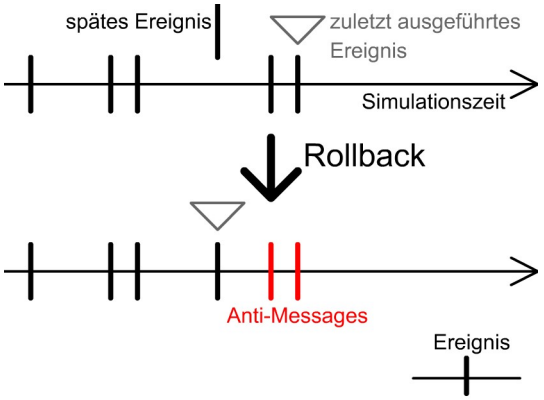

Abbildung 6: Time Warp Synchronisierung

weniger optimistische, dafür aber konsistentere Zustände geführt (zurückhängende Zustände, Trailing States). Sie warten einfach länger, bevor sie Ereignisse ausführen. Je größer die Verzögerung ist, desto unwahrscheinlicher wird es, dass noch Kommandos fehlen und umso besser wird damit die Konsistenz sein.

Fehlerkorrektur Um feststellen zu können, ob der führende Zustand einen Fehler gemacht hat, wird er fortwährend mit den zurückhängenden Zuständen verglichen. Treten Unterschiede auf, findet ein Rollback statt:
Der führende Zustand wird über von dem konsistenteren überschrieben. Alle Ereignisse, die der alte führende Zustand bereits ausgeführt hat, müssen nun von den neuen führenden Zustand noch einmal ausgeführt werden.

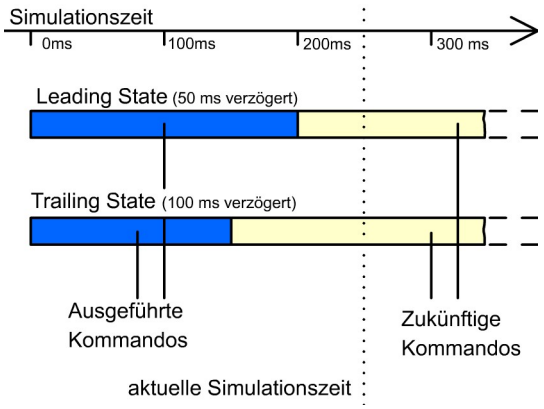

Abbildung 7: Trailing State Synchronisierung

Erläuterung Zukünftige Kommandos werden für jeden Zustand sortiert in einer Liste gehalten. Die Hauptschleife des Programms läuft nun durch jeden Zustand. Wartende Kommandos in der Liste, deren Zeitstempel hinter der Ausführungszeit liegt, werden ausgeführt.

Wieviele Zustände sind sinnvoll? Es stellt sich die Frage, wie viele parallele Zustände der Simulation sinnvoll sind. Zu viele Zustände verursachen einen unnötigen Overhead. Werden zu wenige Zustände verwendet, müssen die Abstände zwischen den Zuständen vergrößert werden, was wiederum zu großen Rollbacks führt. Allgemein ist eine große Verzögerung für den letzten Zustand nötig, damit er in der Lage ist, auch sehr spät eintreffende Ereignisse zu berücksichtigen. Zusammenfassend kann man sagen, dass die Anzahl und die jeweiligen Verzögerungen der Zustände dynamisch an die Netzwerk-Gegebenheiten angepasst werden sollte.

Vorteile Es ist kein Kompromiss zwischen Antwortverhalten und Konsistenz nötig, da TSS beides zufriedenstellend vereint. Im Vergleich zu Time Warp ist wenig Speicher nötig, da eine Kopie nur den Speicherplatz eines Snapshots benötigt. Als weiterer Vorteil können mehrere Inkonsistenzen in einem Rollback korrigiert werden. Da mehrere Zustände parallel berechnet werden, können Mehrprozessor-Systeme gut ausgenutzt werden.

7 Optimierungen

Es werden zwei Techniken vorgestellt, um das Antwortverhalten zu verbessern.

7.1 Dead Reckoning

Motivation Clients können durch Bandbreitenbeschränkungen nicht genügend Updates vom Server erhalten, um Bewegungen flüssig zu halten.

Beispiel Ein 56k Modem kann pro Sekunde 7KB übertragen. Das entspricht bei 30 Frames/Sekunde nur 233 Bytes/Frame. Damit ist die Größe der Updates stark eingeschränkt. Der Ausweg ist weniger Updates zu senden. Nun muss eine Abschätzungsmethode auf der Client-Seite die Zeit zwischen den Updates überbrücken.

Idee des Dead-Reckoning Der Server sendet nur nötige Updates. Alle Clients verwenden eine vereinbarte Abschätzung, um eine Szene anzuzeigen, bis ein neues Update eintrifft. Wenn alle Clients die gleiche Abschätzungsmethode verwen-

den, wird so auch Konsistenz erreicht.

Realisierung Es wird versucht, die Bewegung einer Figur wird mit Hilfe ihrer letzten Position, ihrer Richtung und ihrer Geschwindigkeit vorherzusagen. Diese Methode wurde früher in der Schifffahrt zur Navigation eingesetzt (dort wurde sie heute allerdings vorwiegend durch GPS abgelöst).
Tritt nun ein Fehler auf, d.h. entspricht die Position nach einem Update vom Server nicht der abgeschätzten, muss sie korrigiert werden. Leider springen die Figuren ohne weitere Anpassung nun. Eine Anpassung kann jedoch z.B. mit linearer Konvergenz oder mit Cubic Splines stattfinden.

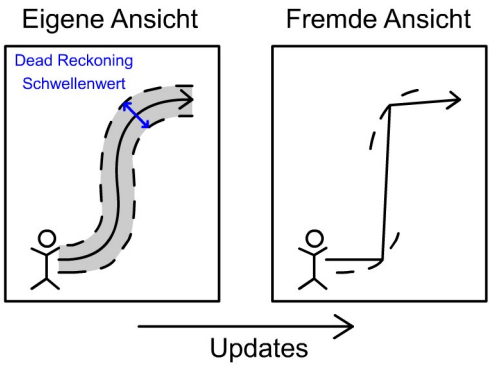

Abbildung 8: Vergleich Pfad auf eigenem und fremden Client

Lineare Konvergenz Wird eine neue Position empfangen, wird vom alten auf den neuen Pfad linear übergeleitet. Leider wirkt dies etwas unnatürlich.

Cubic Splines Cubic Splines bieten eine einfache Möglichkeit, eine Kurve über mehrere Punkte zu legen. Dazu werden Punkte des alten und des neuen Pfades verwendet.

Noch besser sieht das Ganze aus, wenn zusätzlich noch Anfangs- und Endgeschwindigkeiten berücksichtigt werden.

Vor- und Nachteile Dead-Reckoning skaliert gut, da es auf der Seite der Clients stattfindet. Zudem ist eine geringere Bandbreite notwendig. Leider wird durch die Abschätzung die Genauigkeit der Positionen der Figuren eingeschränkt, die Konsistenz ist ebenfalls etwas schwächer. Hektische Bewegungen einer Figur sind

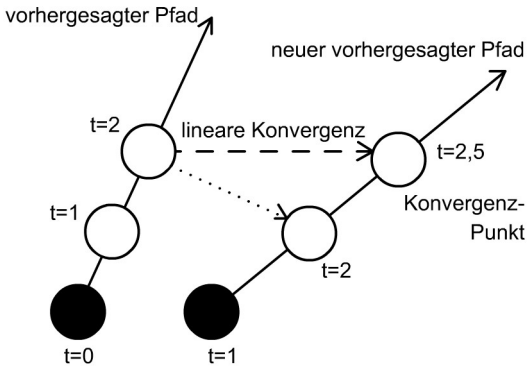

Abbildung 9: Anpassung durch lineare Konvergenz

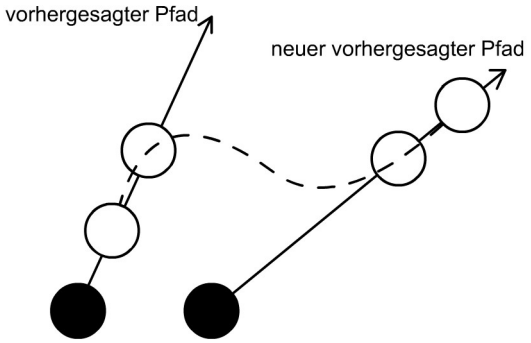

Abbildung 10: Anpassung durch Cubic Splines

ebenfalls problematisch, da sich die Fehler durch die Abschätzung aufschaukeln können.

7.2 Area of Interest Filtering

Idee Beim Area of Interest Filtering filtert der Server die Informationen, die an die Clients gesendet werden, dynamisch. Er sendet nur für den jeweiligen Client relevante Daten.

Anwendung Gefiltert werden kann z.B. nach ortsabhängigen Gruppen (Sichtfeld des Spielers). So muss die Landschaft um das Haus, in dem sich der Spieler aufhält, nicht mitgesendet werden, da er sie sowieso nicht sehen kann.

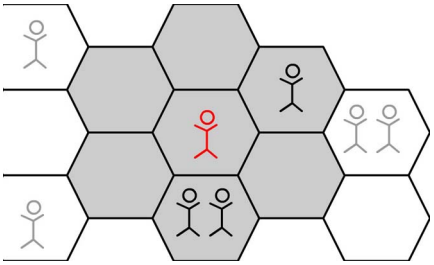

Abbildung 11: Räumliche Partitionierung

Ebenso möglich ist ein Filtern des Tons. Töne, die außerhalb eines bestimmten Radius um den Spieler erklingen, wird dieser sowieso nicht hören, und brauchen damit auch nicht gesendet werden.

Vorteil Mit Area of Interest Filtering wird eine starke Reduzierung des Nachrichtenaufkommens erreicht.

8 Fazit

Klient-Server-Architekturen sind aufgrund ihrer Eigenschaften gut geeignet für virtuelle Präsenzen. Die virtuelle Welt wird auf dem Server gehalten und die Clients verschaffen sich mit einer Verbindung zum Server einen Zugang zu ihr. Da feststeht welches der Server ist, ist es einfach für Clients sich zu verbinden (im Gegensatz zu P2P).
Bei der *zentralen Klient-Server Architektur* gibt es weder Konsistenz- noch Synchronisierungsprobleme. Anfällig ist sie jedoch gegen Verzögerungen im Netzwerk sowie gegen Beschränkungen bei der Anbindung des Servers. Verzögerungen können jedoch gering gehalten werden, wenn mehrere voneinander unabhängige Simulationen auf verschiedenen Servern ausgeführt werden. Durch eine geographische Verteilung der Server ist es so wahrscheinlicher, dass immer ein Server in der Nähe des jeweiligen Clients steht. Bei der *Mirrored-Server Architektur* gewinnt das System an Redundanz, d.h. es wird unanfällig gegenüber Problemen mit einem einzelnen Server. Leider müssen hier nun mehrere parallele Simulationszustände verwaltet werden, das System ist nicht mehr automatisch synchron. Hierzu können jedoch die vorgestellten Algorithmen eingesetzt werden.
Eine Firma, die eine virtuelle Welt anbietet, hat mit der Klient-Server Architektur folgende Vorteile:
Da sie die Kontrolle über ihre virtuelle Welt behalten, ist es möglich, persistente virtuelle Welten zu schaffen. Es ist einfach Patches zu verteilen, in dem der Client einfach gezwungen wird, ihn zu installieren, bevor er sich verbinden darf.

Zudem lassen sich illegale Kopien der Software reduzieren, indem der Anwender gezwungen wird, sich anzumelden. Leider können die Spieler hiermit aber auch überwacht werden.
Ein Nachteil für eine Firma ist, dass sie auf jeden Fall eigene Server bereitstellen und warten muss.

9 Literaturverzeichnis

- Objektverteilung in virtuelle Präsenzen
  M. Schöttner
  http://www-vs.informatik.uni-ulm.de/teach/ws04/vp/VPS_WS0405_06_Objektverteilung.pdf

- Distributed Systems: Principles and Paradigms
  Andrew S. Tanenbaum; Maarten van Steen
  Prentice Hall 2002

- An Efficient Synchronization Mechanism for Mirrored Game Architectures
  Eric Cronin; Burton Filstrup; Anthony R. Kurc; Sugih Jamin
  http://warriors.eecs.umich.edu/games/papers/netgames02-tss.pdf

- A Distributed Multiplayer Game Server System
  Eric Cronin; Burton Filstrup; Anthony Kurc
  http://warriors.eecs.umich.edu/games/papers/quakefinal.pdf

- Server-Oriented Multiplayer Games
  Eric Fesenmaier
  http://www.cs.bris.ac.uk/home/dg1767/doc/servergames.ppt