Klient-Server Architekturen

Matthias Schmid

Universität Ulm

1. Juni 2005

Einleitung

- Welche Anforderungen stellt eine virtuelle Präsenz an eine Netzwerk-Architektur?
- Welchen Beschränkungen unterliegt sie?
- Wie lassen sich diese umgehen?

Anforderungen

- Ego-Shooter stellen hohe Anforderungen
  - geringe Verzögerungen (ca. 100ms)
  - hoher Grad an Konsistenz
- Strategie- und Rollenspiele genügsamer
- jedoch wünschenswert:
  - Bandbreite minimieren
  - Einfache Administration
- Virtuelle Präsenzen dazwischen anzusiedeln?

Übersicht Architekturen

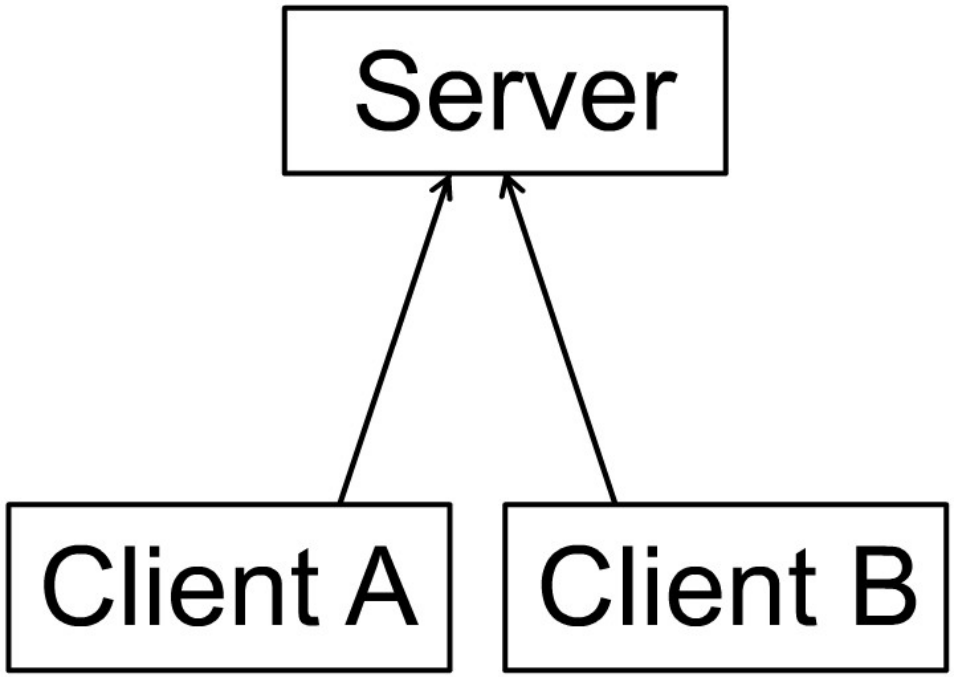

Abbildung: Klient-Server Architektur

Übersicht Architekturen (2)

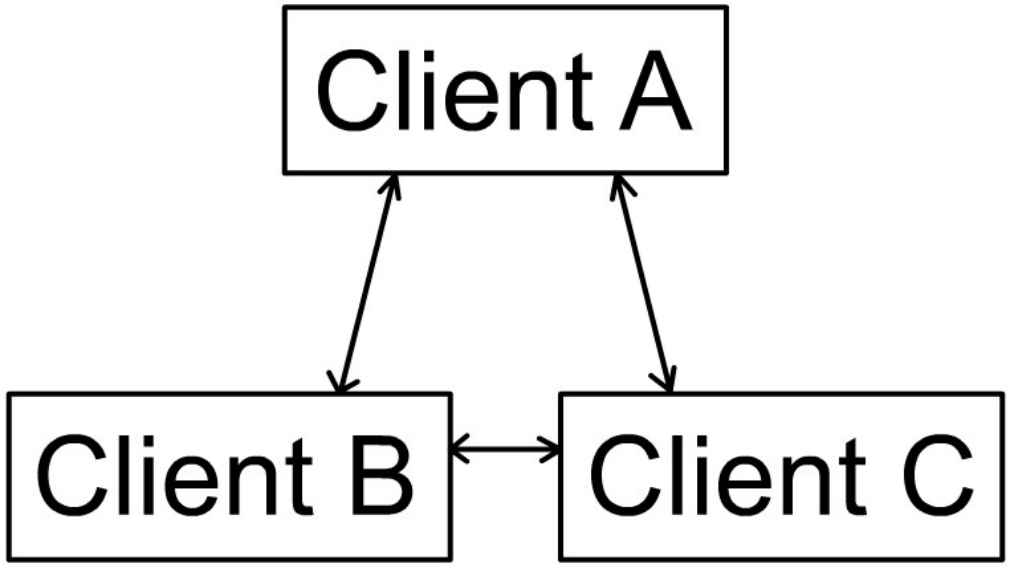

Abbildung: Peer to Peer

Übersicht Architekturen (3)

Klient-Server Architekturen:

- Vorteile
  - Relativ einfach zu implementieren
  - Firmen behalten Kontrolle über ihre Anwendung
  - Clients können sich einfach verbinden
- Nachteile
  - Anfällig gegenüber Verzögerungen
  - Server wird oft zum Flaschenhals

Klient-Server Architekturen

- Bezeichnung:
  - Dienstleister (genannt Server)
  - Dienstanforderer (genannt Client)
- 3 Komponenten:
  - Präsentation (GUI)
  - Applikation
  - Datenhaltung
- Verschiedene Varianten:
  - Zentrale Klient-Server Architektur
  - Mirrored-Server Architektur

Zentrale Klient-Server Architektur

- Server berechnet den ganzen Simulationszustand
- Client ist dafür zuständig, den Simulationszustand anzuzeigen

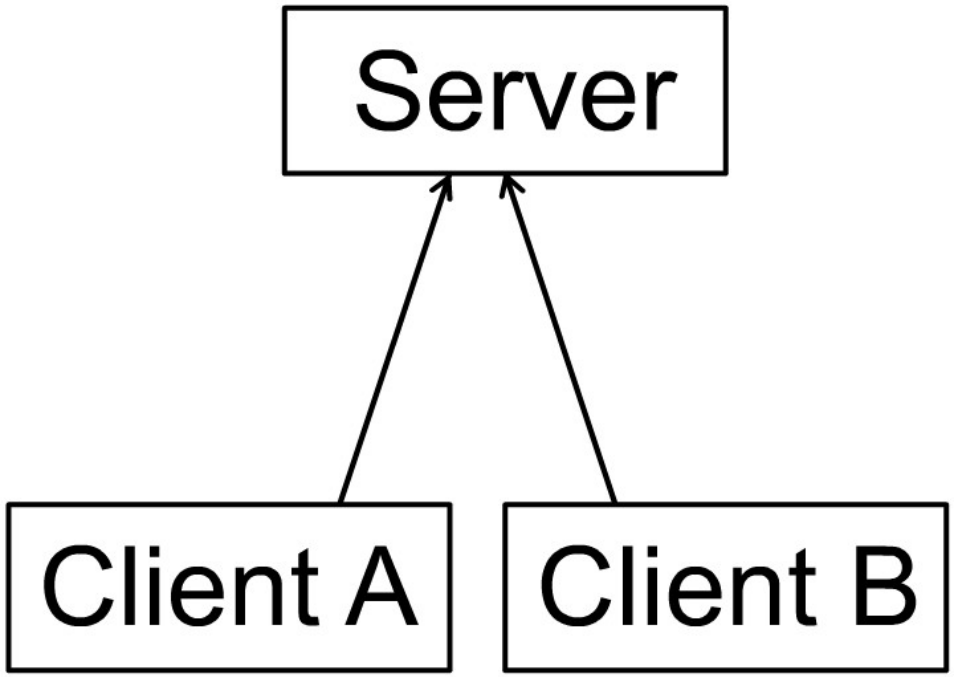

Abbildung: Klient-Server Architektur

Zentrale Klient-Server Architektur (2)

Ablauf:

- Client übergibt Server Kommandos (Figur bewegen)
- Server berechnet neuen Simulationszustand
- Server sendet Updates an alle Clients
- Clients stellen aktuellen Zustand dar
- Server wartet auf Kommandos von Clients

Zentrale Klient-Server Architektur (3)

- Vorteile
  - Automatisch konsistent
  - Leicht zu administrieren
- Nachteile
  - Single Point of Failure
  - Server wird zum Flaschenhals

Mirrored-Server Architektur

- Kombination aus Klient-Server und P2P
- Clients verbinden sich zu nächstgelegenem Server
- Server untereinander verbunden (P2P)

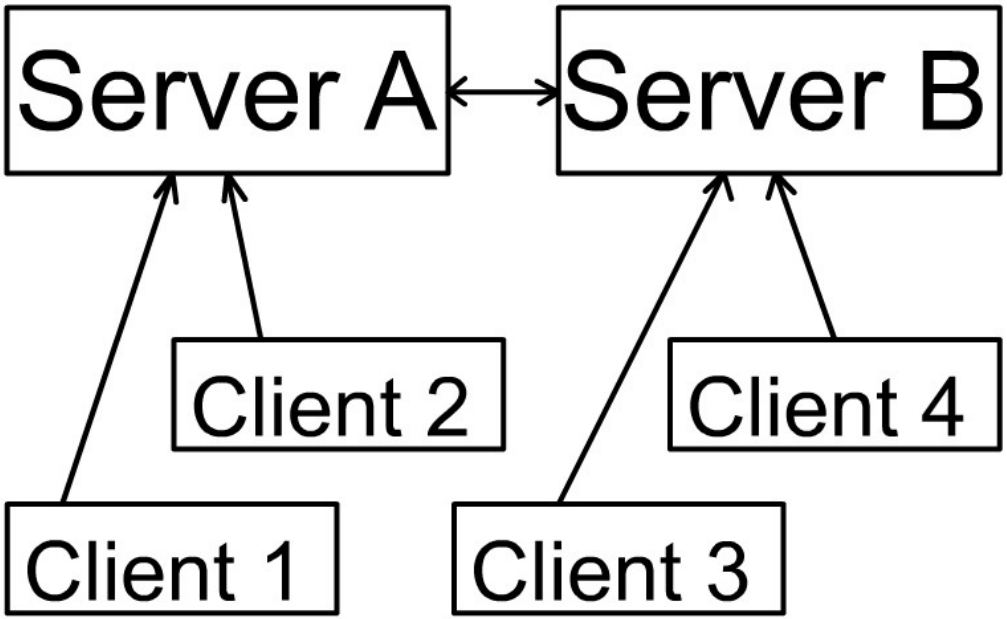

Abbildung: Mirrored-Server

Mirrored-Server Architektur (2)

- Vorteile
  - Bessere Antwortzeiten
  - Redundanz bei Serverausfall
- Nachteile
  - Konsistenz nicht mehr gegeben, Synchronisierung notwendig
  - Mehr Hardware nötig, teurer

Kommunikation

- Nachrichten von Client an Server: Kommandos
  - Eingaben des Anwenders
  - Pakete normalerweise unzuverlässig gesendet
- Nachrichten vom Server an Client: Statusinformationen
  - z.B. Position der Spielfigur

Kommunikation (2)

gängiges Konzept: Remote Procedure Call (RPC)

- Einzelne Nachrichten zu senden ist aufwendig
- Idee: Client ruft Prozedur auf dem Server auf
- Vorteil: Programmierer muss sich nicht um Nachrichtenübermittlung kümmern
- Nachteil: Aufrufender und aufgerufener Prozess i.A. in verschiedenen Adressräumen

Optimierungen

Techniken, um Antwortverhalten zu verbessern

- Area of Interest Filtering
- Dead Reckoning

Area of Interest Filtering

- Server sendet nur relevante Informationen an Clients
- Filtern z.B. nach ortsabhängigen Gruppen (Sichtfeld des Spielers)

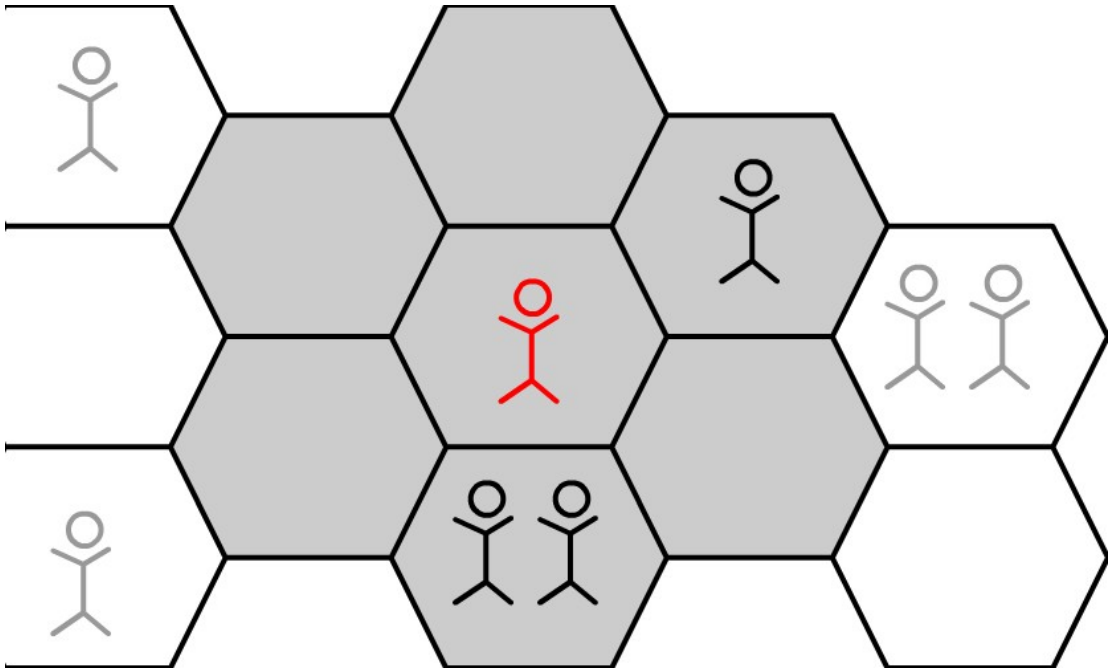

Abbildung: Räumliche Partitionierung

Dead Reckoning

- Bandbreite oft beschränkt
- Anzahl Updates einschränken

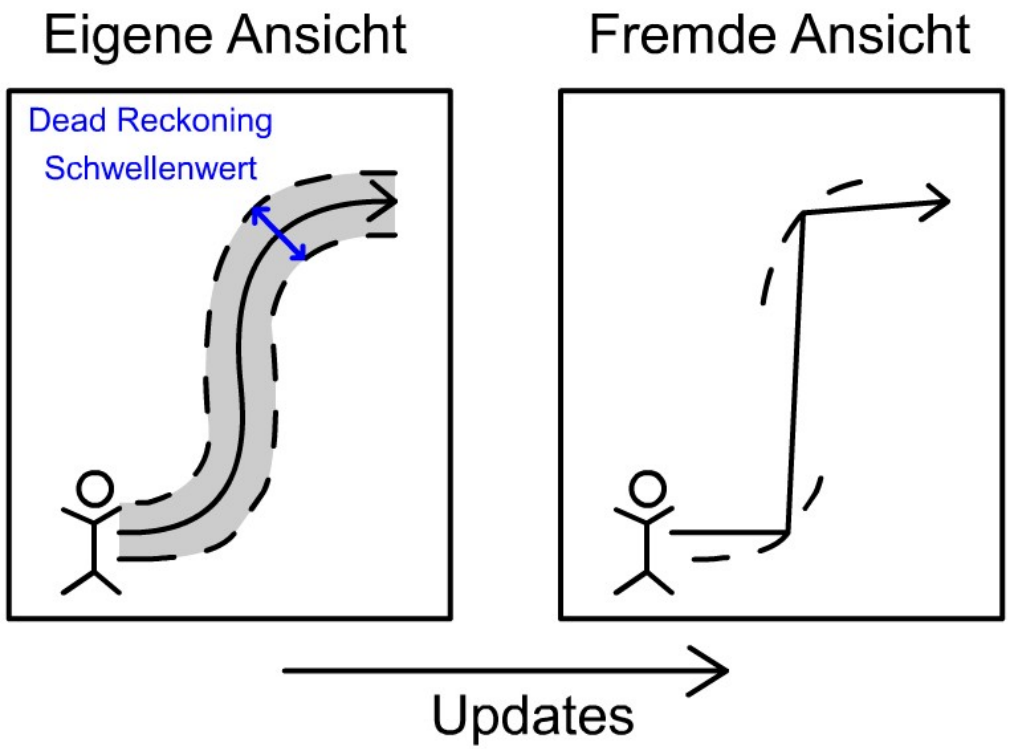

Abbildung: Pfad auf eigenem und fremden Client

Dead Reckoning (2)

- Clients müssen Zeit zwischen Updates überbrücken
- Dead Reckoning: Pfad wird anhand alter Position und Geschwindigkeit geschätzt
- Methode früher in der Schifffahrt zur Navigation eingesetzt
- Entspricht Position nach Update nicht der abgeschätzten, muss Pfad angepasst werden
- Anpassung durch lineare Konvergenz oder Cubic Splines

Dead Reckoning (3)

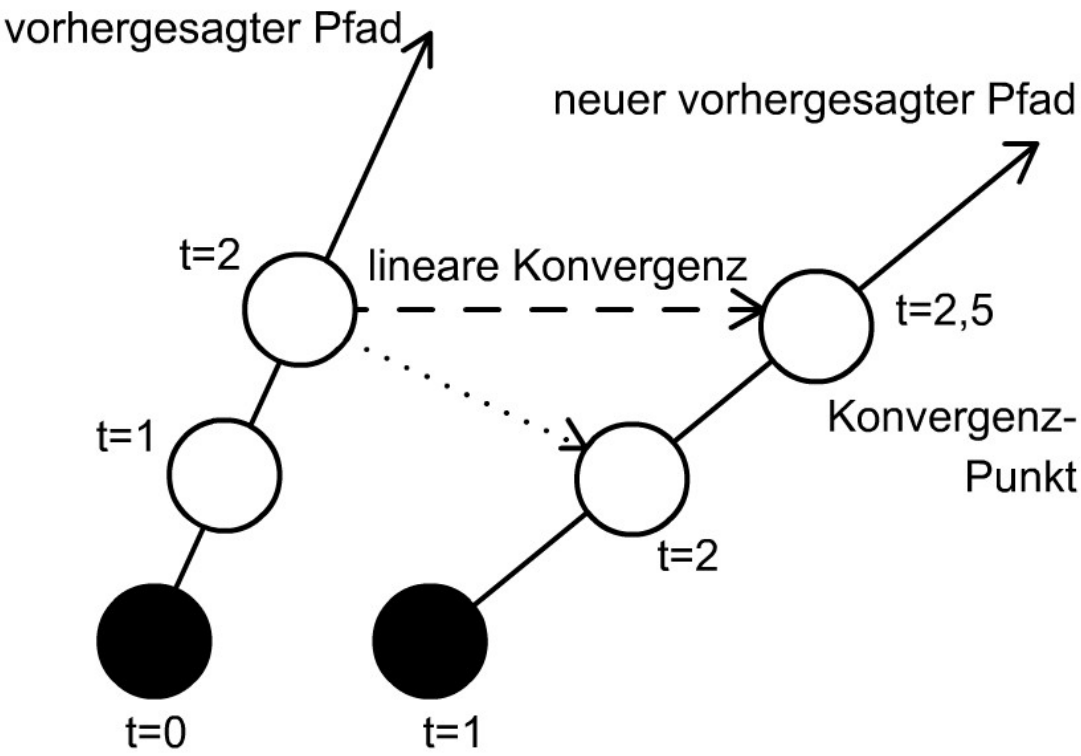

Abbildung: Anpassung durch lineare Konvergenz

Dead Reckoning (4)

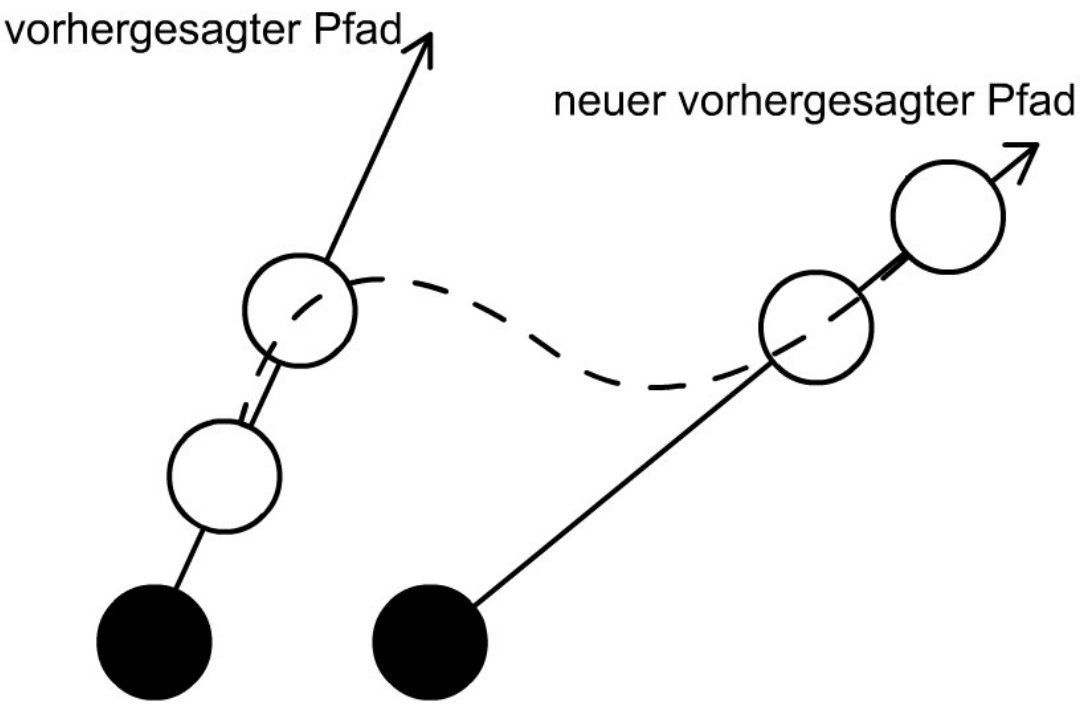

Abbildung: Anpassung durch Cubic Splines

Dead Reckoning (5)

- Vorteile
  - Skaliert gut
  - Geringere Bandbreite notwendig
- Nachteile
  - Schwächere Konsistenz
  - Nur bedingte Genauigkeit
  - Hektische Bewegungen problematisch

Synchronisierung

- Konservative Algorithmen, z.B. Lockstep
  - Einfach
  - Simulationszeit wird angehalten, bis alle Clients fertig sind mit Berechnungen
  - Nicht anwendbar für realistisch wirkende virtuelle Präsenzen
- Optimistische Algorithmen
  - Ereignisse werden optimistisch berechnet
  - Falls der Algorithmus Fehler macht, korrigiert er sie hinterher
  - Beispiel:
    - Trailing State Synchronisation

Trailing State Synchronisation

- Ereignisse werden verzögert ausgeführt
- Es werden mehrere Kopien (Zustände) der Simulation gehalten
- Jede Kopie an einer anderen Simulationszeit
- Wie kann zu altem Zustand gewechselt werden? (Rollback)

Trailing State Synchronisation (2)

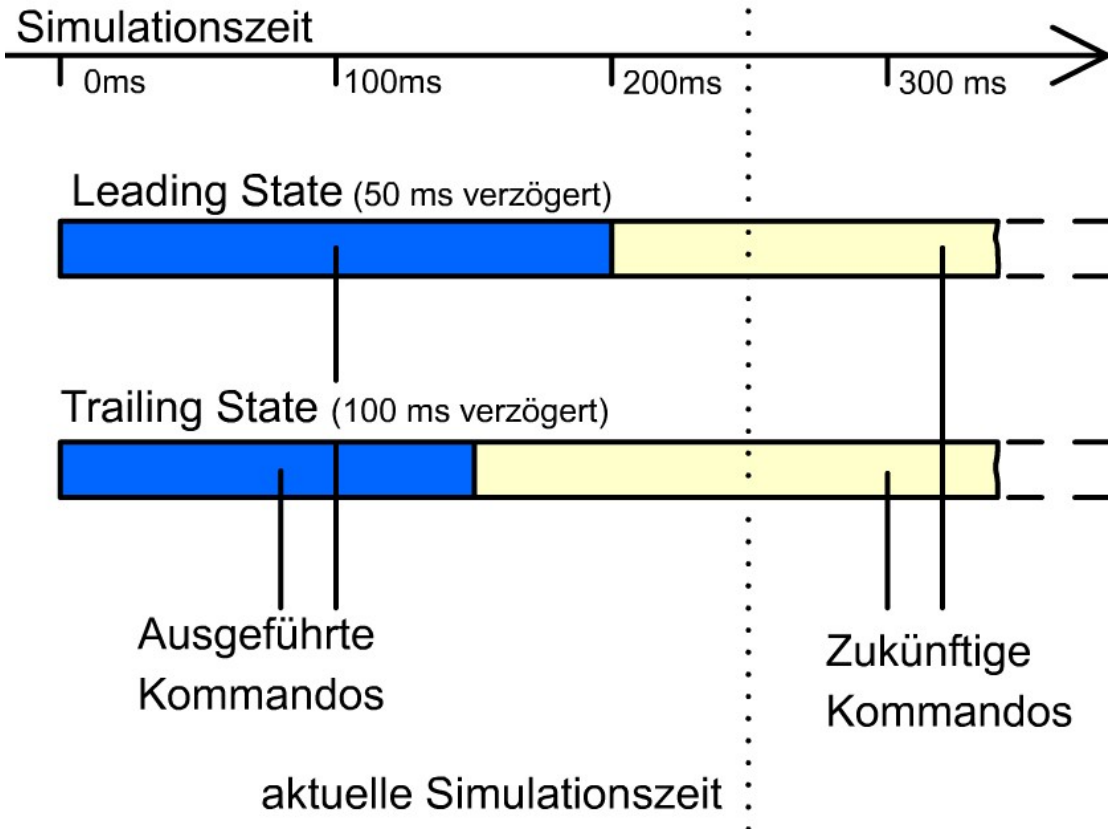

Abbildung: Trailing State Synchronisation

Trailing State Synchronisation (3)

- Führender Zustand wird sofort oder mit geringer Verzögerung ausgeführt
- Der erste zurückhängende Zustand (Trailing State) wartet länger, bevor er Ereignisse ausführt
- Dadurch unwahrscheinlicher, dass Inkonsistenzen auftreten
- Mehrere zurückhängende Zustände bilden Kette
- Um Inkonsistenzen zu entdecken vergleicht sich jeder Zustand mit dem vorhergehenden
- Tritt Inkonsistenz auf, wird vom führenden auf den zurückhängenden Zustand gewechselt
- Alle Ereignisse bis dahin werden noch einmal ausgeführt

Trailing State Synchronisation (4)

Vorteile

- Wenig Speicher nötig, da nur wenige Kopien notwendig
- Es können mehrere Inkonsistenzen in einem Rollback korrigiert werden
- Kann Mehrprozessor-Systeme können ausgenutzt werden

Fazit

- Klient-Server Architekturen skalieren gut
- Sie sind einfach zu implementieren
- Einfach zu administrieren/kontrollieren
- Wenig Konsistenz-/Synchronisationsprobleme
- Problematisch ist Bandbreite auf der Server-Seite
- Mirrored-Server interessant, wenn Verzögerung verkleinert und Redundanz erhöht werden soll

Ende

Noch Fragen?