Chapter 3: Distributed Game Architectures
Overview

• architectures for distributed games
• distributed action handling
  • Fat-Client vs. Thin-Client
  • problems of centralized and decentralized computation
  • problems with local time stamps
• spatial movement and dead reckoning
  • update strategies
  • movement models
  • error correction
• network protocols and games
  • typical network load of games
  • TCP and games
  • UDP and games
MMOG Architectures

Client-Server:
- provider hosts game in a computing center
- game client and server run different software
- centralized solution for:
  - account-management
  - partitioning of the game world
  - monitoring
  - persistence

Multi-Server:
- several servers
- redundant data storage
- network distance between client and server is generally shorter
- dynamic solutions:
  - replication
  - proxy-Server

Peer-to-Peer:
- no explicit Servers
- data exchange between adjacent peers
- every peer is hosting part of the game world
- dynamic partition of the game world
Detailed Client Server Architecture

- hosted in a computing center
- several game servers share game state
  - zones shards/realms, instances
  - strict division of zones
  - seamless distribution (communication between servers)
- authentication / account management service
- action- and response-multiplexer (proxy) may ease the load on game servers by taking over particular functionalities
Distributing the Game Core

design choices:

• What kind of peers do exist?
• What are the peers exchanging? (actions, object states, user input, …)
• Who is authorized to read which part of the game state and who is also entitled to write?
• How is the load redistributed among existing peers?
• How is time between peers synchronized?
Protocol Content

• Object attributes: (Action Result Protocol)
  • protocol sets the current parameter value of a game entity
    (set player „Facemelt0r“ HP to 96)
  • protocol sends relative changes
    (reduce player „Facemelt0r“ HP by 100)

• Actions: (Action Request Protocol)
  • contains only player input without direct impact on game state
  • protocol only transfers user input
    => results must be calculated on the server
    (try to hit Player „Facemelt0r“ with „Uppercut“)
Thin Client Solution

- server holds the complete game state and is solely authorized to change it
- clients receive a part of the game state upon login
- server transmits game state changes to clients
- client transmits actions it wants to execute to the server (Action Requests)
- server collects all incoming action requests
- actions are handled in order of arrival and results are transmitted to the affected clients
Thin Client Solution

Advantages:
• game state is centrally managed
  • consistent game state for calculating action results
  • no conflicts from several contradictory game states
  • persistence system is able to save consistent game states
• low potential for cheating/ action handling only on the server

Disadvantages:
• maximum server load because all action handling is done on the server
• potential for high latencies due to round trip times (actions need to be transmitted to the server and back to take effect)
• processing power at client side is largely unused (clients only display the game state and transmit user inputs to the server)
Fat Client Solutions

- every client has its own objects which only can be edited by itself
- server manages chronological sequence with time stamps and transmits changes to the other clients
- local game states may vary, due to transmission delay
- chronological sequence may be inconsistent because local changes may be applied before global changes with an earlier time stamp arrive
Conflicts during decentralized computing

- local changes need time to be transmitted within the network
- actions are calculated for and executed on local game states
  \[\Rightarrow\] changes that predate the action may not be taken into account
- simple solutions:
  - client is not allowed to change local data without server acknowledgment
  - using object protocols the server may send an update of the current game entity state.

*example for network effects on the game state:*

<table>
<thead>
<tr>
<th>Server</th>
<th>Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>A heals S for 50 hp</td>
<td>050/200</td>
</tr>
<tr>
<td></td>
<td>050/200</td>
</tr>
<tr>
<td>050/200</td>
<td>dead</td>
</tr>
<tr>
<td>100/200</td>
<td>dead</td>
</tr>
<tr>
<td>020/200</td>
<td>dead</td>
</tr>
<tr>
<td></td>
<td>C hits S for 80 hp</td>
</tr>
</tbody>
</table>
Solution Approach

reset local actions

- client has 2 game states:
  - optimistic GS (contains local changes)
  - pessimistic GS (contains actions transmitted by the server)
- on mismatch: reset the optimistic GS to the state of the pessimistic GS

A heals S for 50 hp

C hits S for 80 hp

colonel

conflict
Local Time

up until now: one server handles processing sequence
• impossible for P2P games and multi server architecture
  => sequence is inconclusive after arrival at server
  => organization by local time stamps on creation
• during processing both, own and foreign changes may appear in incorrect sequence
• in case of inconsistencies game entities must be synchronized

<table>
<thead>
<tr>
<th>Peer1</th>
<th>Peer 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>050/200</td>
<td>050/200</td>
</tr>
<tr>
<td>100/200</td>
<td>dead</td>
</tr>
<tr>
<td>020/200</td>
<td>dead</td>
</tr>
</tbody>
</table>

time axis

A heals S for 50 hp

C hits S for 80 hp
problem is caused by the lack of knowledge about previous actions

**solution**: local lag mechanism

- processing updates is delayed to allow for other actions to arrive in time
- if this time frame is exceeded, conflict detection and reset are necessary

<table>
<thead>
<tr>
<th>Time Axis</th>
<th>Peer 1</th>
<th>Peer 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>020/200</td>
<td>020/200</td>
<td>020/200</td>
</tr>
<tr>
<td>100/200</td>
<td>100/200</td>
<td>100/200</td>
</tr>
<tr>
<td>050/200</td>
<td>050/200</td>
<td>050/200</td>
</tr>
<tr>
<td>A heals S for 50 hp</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

C hits S for 80 hp
Application in Games

Games can combine several approaches by processing actions differently.

### Server side processing
- accuracy is important
- response time less important
- chronological order is important

### Client side processing
- response time is crucial
- synchronization and sequence are less important

- damage and healing
- item pick up

- object positions and movement
- animations and other display effects

**Conclusion:**
- generally speaking, there is a trade off between latency (here: response time) and consistency of the game world. (c.f. CAP theorem)
- another issue is reducing remote updates to reduce the needed bandwidth.
Movement Information

movement-updates play a special role in distributed virtual environments

- fluid movement
  - positions may change several times per second (24-60 FPS)
  - calculation should be closely tied to rendering
  - handling movement and other actions in the same way might disturb fluid rendering of the gaming world
- precise positions are mostly irrelevant for game play:
  - due to the fast update rates, the loss of several position updates is often negligible

Consequences:

- movement for real-time games is predominantly calculated locally on the client
- parts of objects trajectories are not transferred to other peers to save bandwidth
- Movement is extrapolated locally and positions are synchronized only at certain points times

⇒ Dead Reckoning: simulating movement between two updates to allow for fluid movement with limited bandwidth
Dead Reckoning

dead reckoning components for games:

• **Update-Strategy** on the server side (owner of changed) game entity: controls when is position information transmitted to other peers and with what frequency (influences bandwidth and error rate on the client)

• **Movement model** on remote peer:
  Comprises all methods to extrapolate fluid movement based on the transmitted positions (influences error rate and perception of movement on the client)

• **Error correction** on remote peer:
  Allows for converging extrapolated local position and exact positions received from the owner of the game entity (influences perception on the client)

=> There is a trade-off between
  • bandwidth and error rate
  • perception quality and processing time
Update-Strategies for Dead Reckoning

• regular updates:
  • send updates in regular intervals

• event based updates:
  • send updates on changing direction or movement type

• distance-based-updates:
  • precise positions are more important the closer an object is
  • the closer an object is to a critical range (e.g., weapon range)
  • transmits regular updates, but with different rates, depending on distance.

  every 30 ticks
  every 15 ticks
  every 3 ticks
Movement model for Dead Reckoning

Point in time: \( t_i \) Position: \( p(t_i) = (x_i, y_i) \) Average speed: \( v(t_i) \) acceleration: \( a(t_i) \)

Linear movement with constant speed:

\[
p(t_1 + t_\Delta) = p(t_1) + \frac{p(t_1) - p(t_0)}{\|p(t_1) - p(t_0)\|} t_\Delta \cdot \frac{\|p(t_1) - p(t_0)\|}{(t_1 - t_0)} = p(t_1) + t_\Delta \cdot \frac{p(t_1) - p(t_0)}{(t_1 - t_0)}
\]

Linear movement with constant acceleration:

\[
p(t_i + t_\Delta) = \frac{1}{2} a(t_i) t_\Delta^2 + v(t_i) t_\Delta + p(t_i)
\]

\[
a(t_i) = \frac{\Delta v}{\Delta t} \approx \frac{v(t_i) - v(t_{i-1})}{t_i - t_{i-1}}
\]

\[
v(t_i) = \frac{\Delta p}{\Delta t} \approx \frac{p(t_i) - p(t_{i-1})}{t_i - t_{i-1}}
\]
Error Correction for Dead Reckoning

**problem**: prediction and update do not correspond.

- object on a remote peer is overwritten by an update for high error rates:
  - objects jump
  - objects disappear and reappear elsewhere

- both positions are merged with an accelerated fluid movement:
  - e.g. cubic polynomials: Bezier, B-Splines, Hermite
  - must allow for a certain correction time $\Delta t$
Hermite graphs for polynomial smoothing

four base polynomials:
• \( h_1(x) = 2x^3 - 3x^2 + 1 \)
• \( h_2(x) = -2x^3 + 3x^2 \)
• \( h_3(x) = x^3 - 2x^2 + x \)
• \( h_4(x) = x^3 - x^2 \)

connecting points \( p \) and \( p' + d' \) using the following linear combination

\[
p(x) = p(t) h_1(x) + p'(t+ \Delta t)h_2(x) + d(t) h_3(x) + d'(t) h_4(x) \quad (0 \leq x \leq 1)
\]

• position: \( p(t) \) via Dead Reckoning
• movement direction: \( d(t) \) via Dead Reckoning
• target: \( p'(t) + d'(t) \) via server-update
  where \( p'(t+\Delta t) = p'(t) + d'(t) \) is the position at the point \( t+\Delta t \) in time
• \( \Delta t \): Time for corrections (compensation by faster speed)
Thoughts on Client-Server Communication

important factors influencing the communication

- **Latency**: time until the system reacts
  - round trip time (RTT)
  - package size
  - system load aside from the network
- **Bandwidth**: How large is the transferred volume?
- **Burstiness**: How is the data volume distributed over time?
- **Connection-oriented/package oriented protocols**
  - connection oriented: Routing happens once
  - package oriented: Routing happens for every package
- **Security**: Is loss of data possible?
Requirements of computer games

- small package sizes
- little bandwidth is used
- latency by genres:
  - RTS-games: <1000 ms
  - RPG: < 500 ms
  - FPS: < 100 ms
  (Estimated latency for observing a serious impairment of the gaming experience.)

<table>
<thead>
<tr>
<th>application/platform</th>
<th>avg.</th>
<th>payload size (bytes)</th>
<th>avg. bandwidth requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>min</td>
<td>max</td>
</tr>
<tr>
<td>Anarchy Online (PC)‡</td>
<td>98</td>
<td>8</td>
<td>1333</td>
</tr>
<tr>
<td>World of Warcraft (PC)</td>
<td>26</td>
<td>6</td>
<td>1228</td>
</tr>
<tr>
<td>Counter Strike (PC)</td>
<td>36</td>
<td>25</td>
<td>1342</td>
</tr>
<tr>
<td>Halo 3</td>
<td>247</td>
<td>32</td>
<td>1264</td>
</tr>
<tr>
<td>Gears of War (XBOX 360)</td>
<td>66</td>
<td>32</td>
<td>705</td>
</tr>
<tr>
<td>Tony Hawk’s Project 8 (XBOX 360)</td>
<td>90</td>
<td>32</td>
<td>576</td>
</tr>
<tr>
<td>Test Unlimited (XBOX 360)</td>
<td>80</td>
<td>34</td>
<td>104</td>
</tr>
</tbody>
</table>

Protocols and Communication solutions

TCP/IP:

- safe protocol: by re-transmission
- flow control and congestion control
- optimized for bandwidth usage and data transfer
  (sending big packages to reduce the transmitted TCP-Headers)

Disadvantages:

- packages may arrive with significant delay (re-transmission)
  
  => increased latency
  
  => package may not be needed anymore since newer information has already been transmitted

- optimizing bandwidth artificially increases latency
  
  - waiting for payload for underfilled packages
  
  - confirmation packages confirm several packages or are embedded within returning traffic

- optimization by tuning and turning features off.
Protocols and Communication solutions

UDP

- minimal datagram service
- no explicit connection to the remote station
- unsafe transmission, correct sequence is not guaranteed
- no congestion control mechanisms

Advantages:
- no retransmission of lost packages
  => outdated information will not be resend
- less header overhead

Use:
- middle ware solutions which implement missing service features in the higher layer protocols:
  - maintain update sequence
  - security for certain update operations (e.g. picking up items, ...)

24
Conclusion network protocols

- TCP/IP is still the most used protocol since routers and infrastructure are usually designed for TCP/IP

- UDP offers a light weight solution for just-in-time services (voice, movement, …)

- secure services are still mandatory for most games and must be implemented in the application layer of the protocol

- studies on other protocols (e.g. SCTP), show no significant increase in performance

- majority of games uses TCP for communication
Learning Goal

• Client-Server and P2P architecture in games
• distribution of action handling:
  • global processing
  • local processing with centralized chronology
  • local processing with local chronology

• Dead Reckoning
  • update-strategies
  • movement models
  • error correction

• requirements for network protocols for games
  • TCP and games
  • UDP, middle ware and games
List of references

- N. Gupta, A. Demers, J. Gehrke, P. Unterbrunner, W. White
  Scalability for virtual worlds

- Jens Müller, Andreas Gössling, Sergei Gorlatch
  On correctness of scalable multi-server state replication in online games
  In Network and System Support for Games (NETGAMES'06), 2006.

- Jouni Smed, Timo Kaukoranta, Harri Hakonen
  A Review on Networking and Multiplayer Computer Games
  In IN MULTIPLAYER COMPUTER GAMES, PROC. INT. CONF. ON APPLICATION AND

- S. Harcsik, A. Petlund, C. Griwodz, P. Halvorsen
  Latency evaluation of networking mechanisms for game traffic