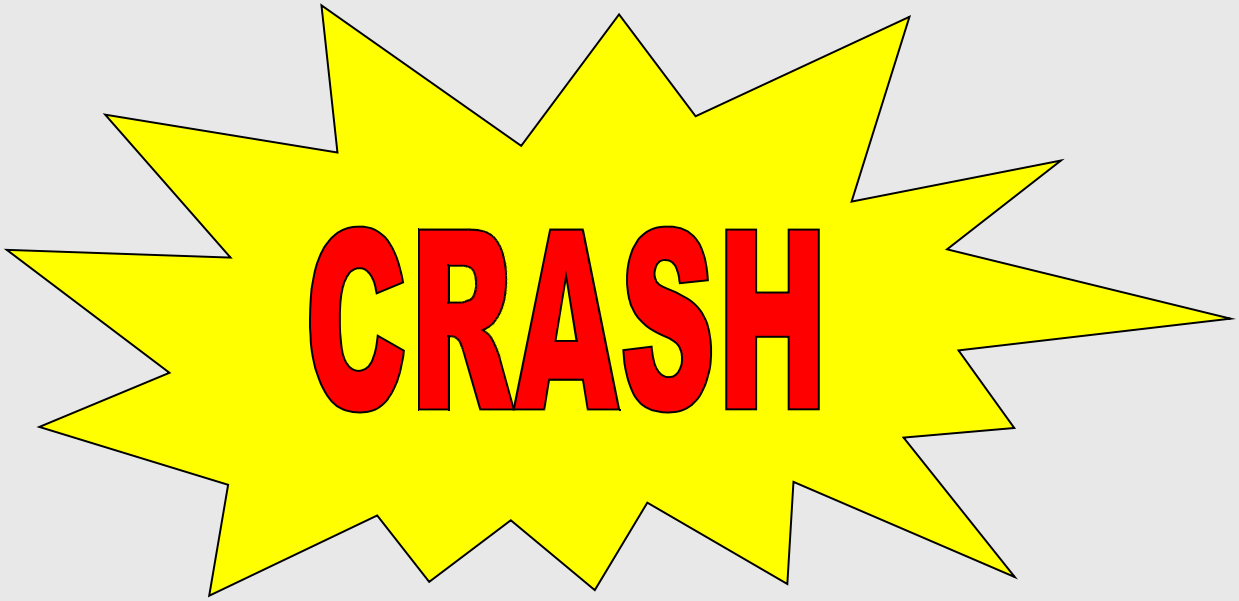
Persistence in Massively Multiplayer Online Games

Kaiwen Zhang Bettina Kemme Alexandre Denault

McGill


Flower4031 (4031)
A rapidly growing weed.
Player Inventory
Capacity : 12 / 16 Weight: 0.300 / 8.0
Chat Window
Flower4032 Grabbed.
Dropped Flower4032.
Flower4037 Grabbed.
Dropped Flower4037.
Flower4037 Grabbed.
Dropped Flower4037.
Flower4029 Grabbed.
Dropped Flower4029.
Flower4028 Grabbed.
Dropped Flower4028.
FPS: 62 - Counts: Mesh(80) Vert(3176)

Mammoth




Player Inventory
Chat Window
FPS: 62 - Counts: Mesh(274) Vert(1032

Mammoth

Motivation

- Common game world
- Importance of game progression
- Longevity of MMORPGs
- Necessity of world state recovery

Persistence

- Recovery of the world state
- Shutdown situation and saving/loading
- Real-time game monitoring
- Storing world state on stable storage
- Ideally: exact recovery

Content

- Goals of persistence
- Consistency categories
- Storing player position
- Implementation details
- Throughput analysis
  - Exact solution
  - Approximate solutions
- Conclusion

Goals

- Consistent data requirement:
  - Plausible state
- Efficiency:
  - Minimal overhead perceived by clients
- Scalability:
  - Avoid bottleneck
  - Target in the range of thousands players.

Consistency

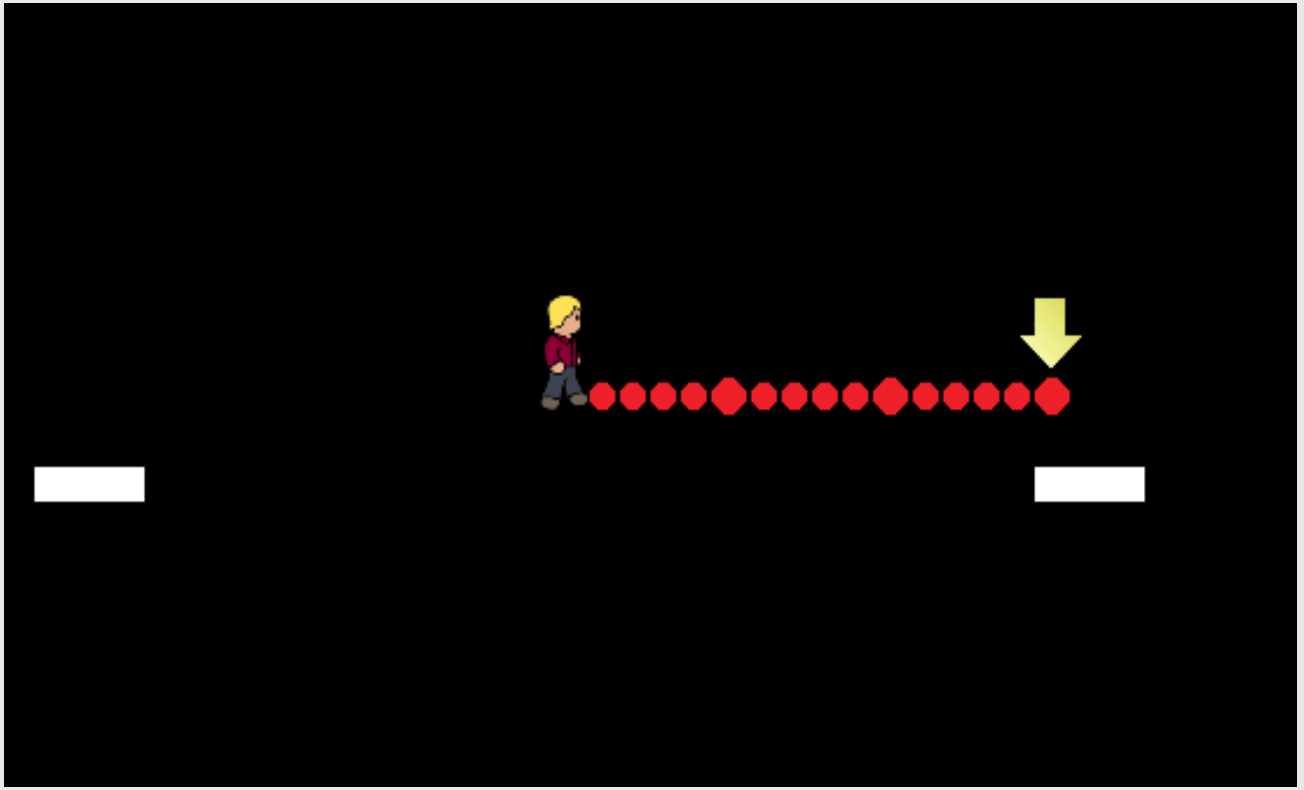
- Exact saving: too expensive
- Consistency categories:
  - Immutable/inferred/minor (walls, names, …)
  - Mutable, low (position)
  - Mutable, exact (item states, trades…)

Storing player position

- Vast worlds
- Exact position not necessary
- Movement: a series of small position changes (“steps”)
- Majority of events in a game are position updates

Strategies

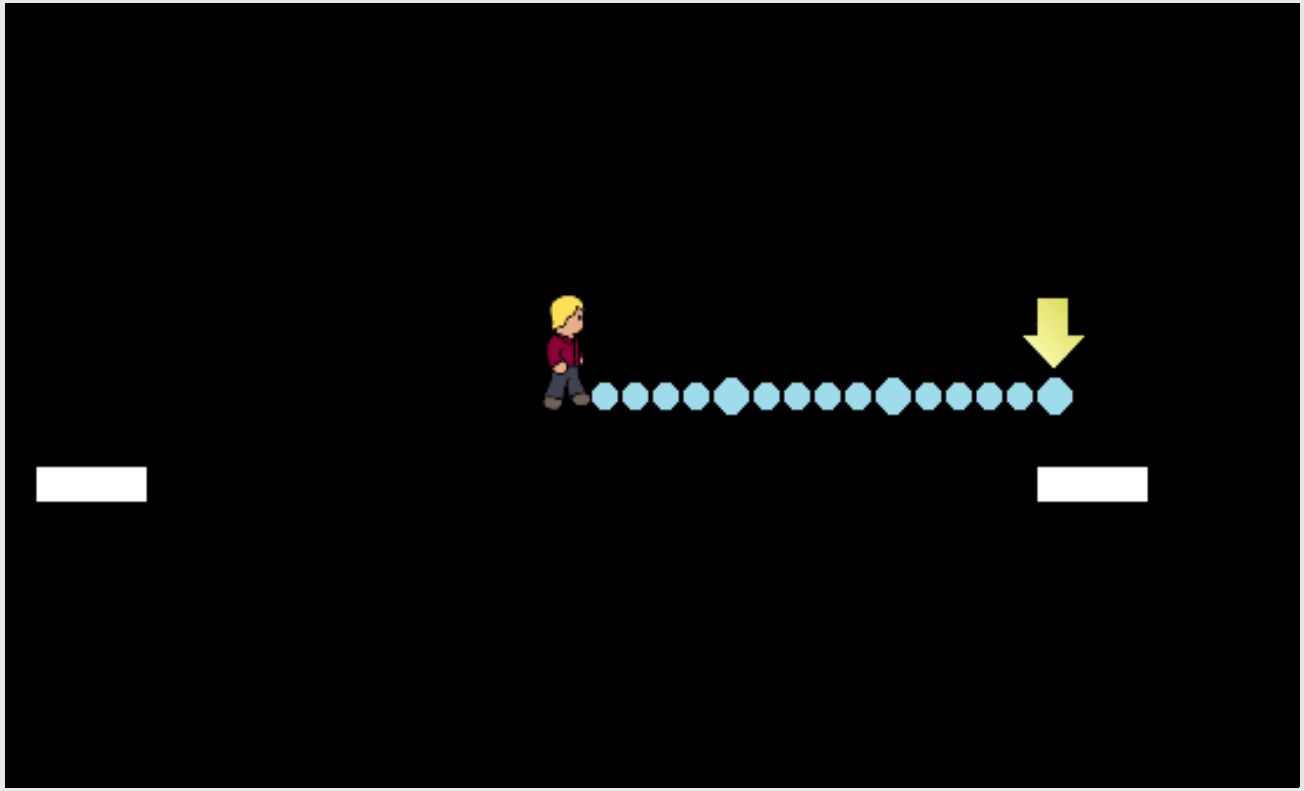
- Naïve solution
  - Exact
  - Expensive
  - Scalability issues



Mammoth

Strategies

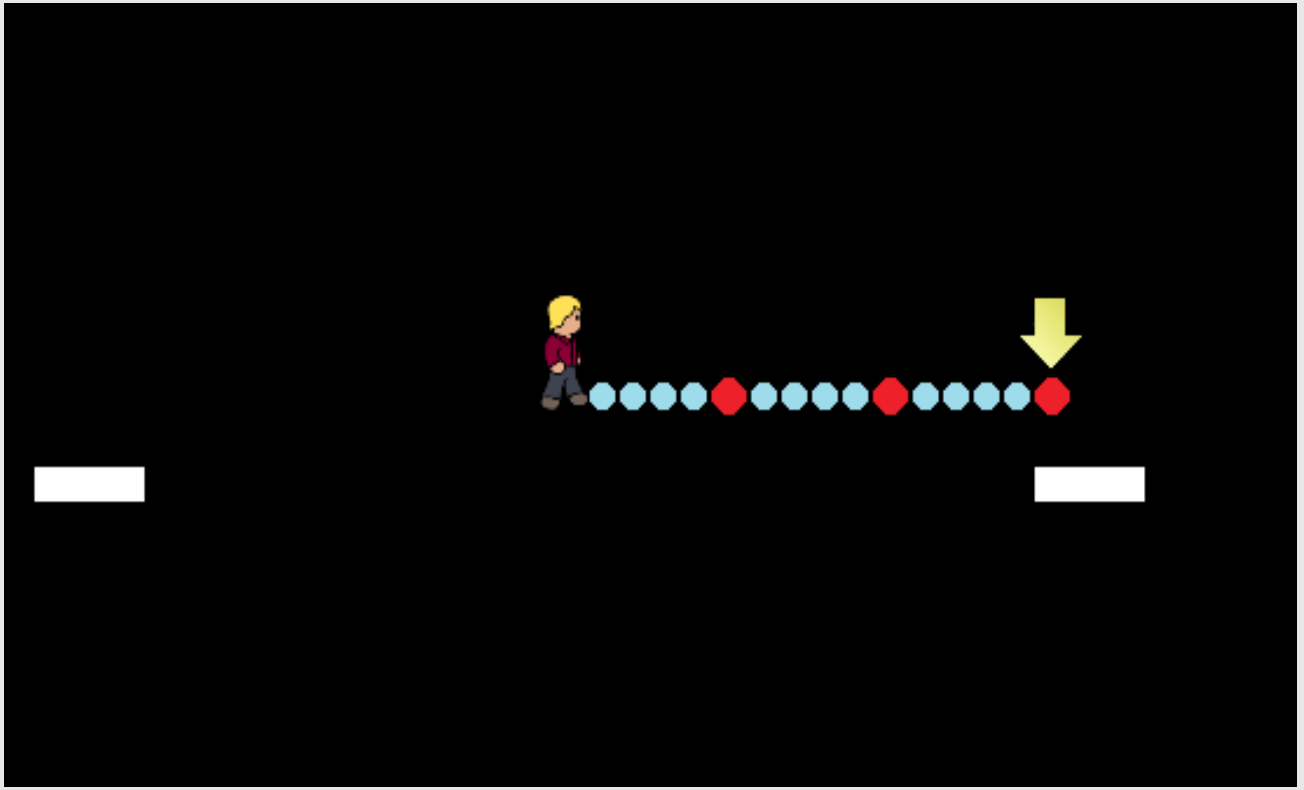
- Snapshot
  - Low overhead
  - Timestamp
  - Spikes
  - Inefficient



Mammoth

Strategies

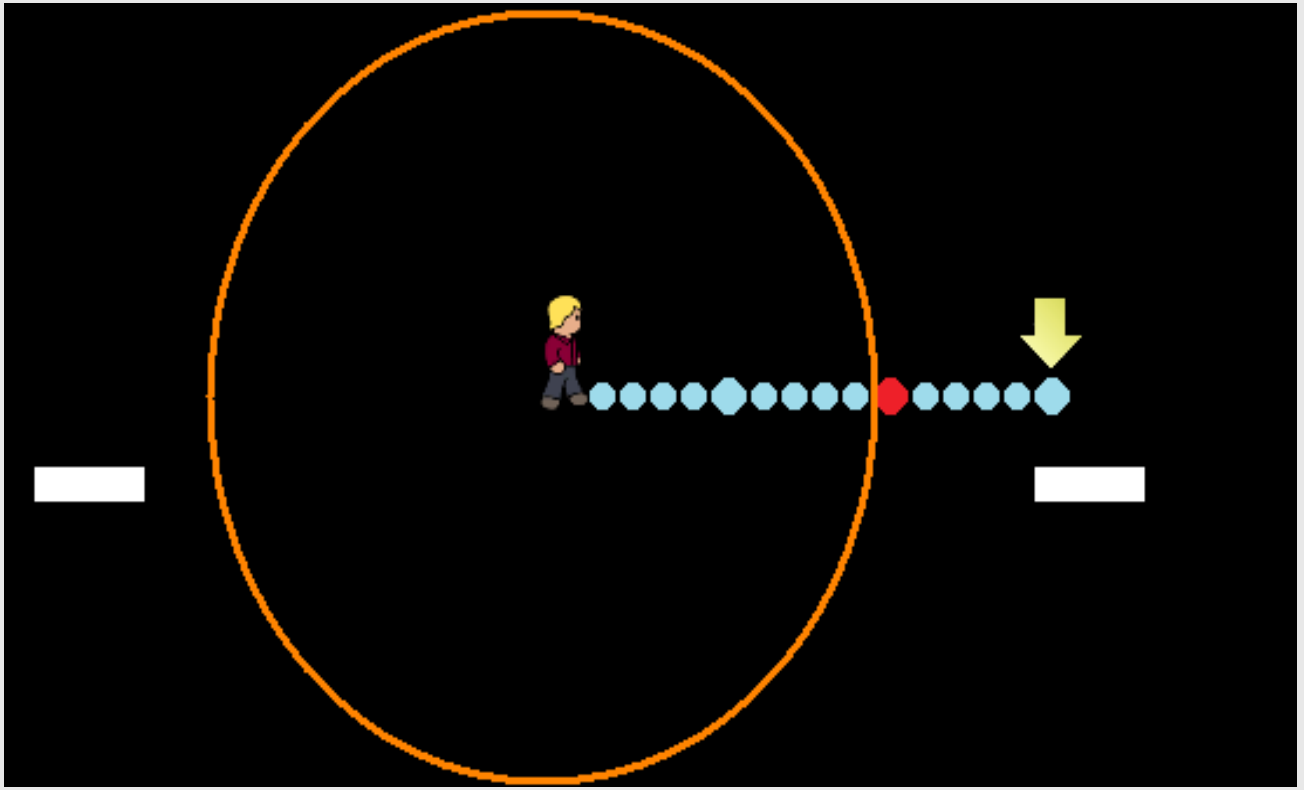
- Fractional storage
  - Considers activity
  - Unnecessary updates



Mammoth

Strategies

- Distance based
  - No unnecessary updates
  - Other factors



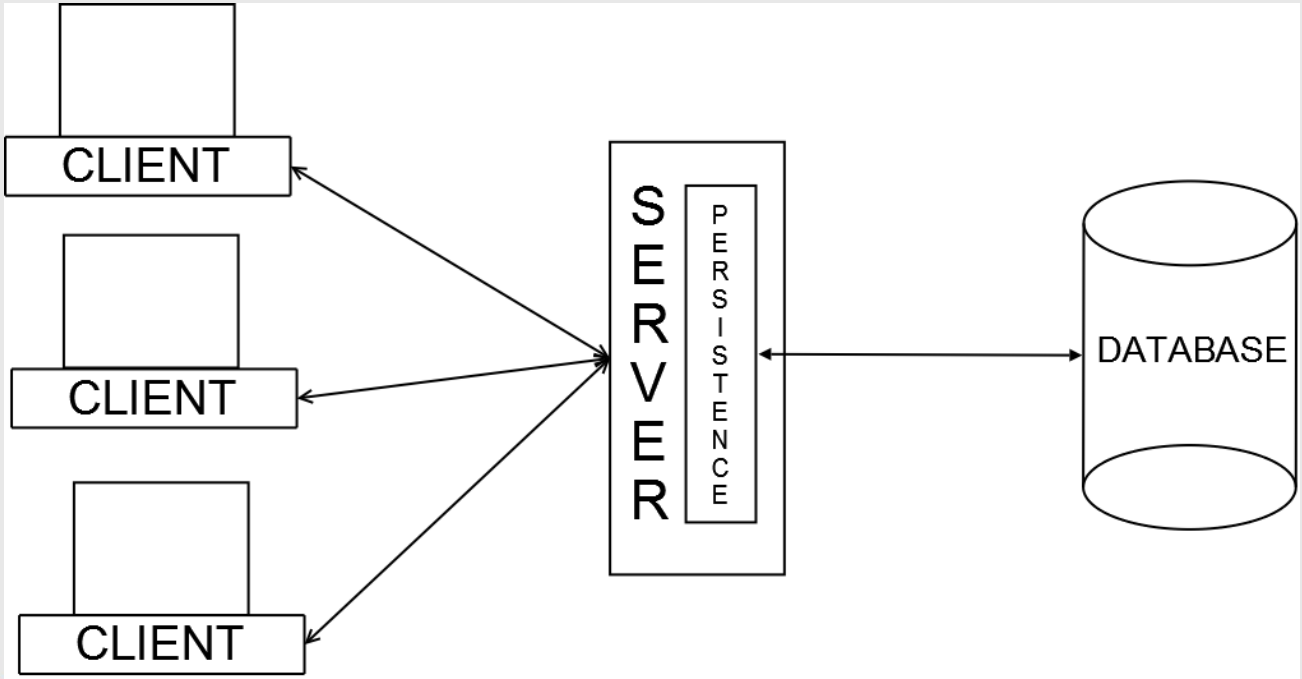
Mammoth

Implementation setup

- Mammoth, MMORPG framework
- Untuned MySQL backend
- Server, Database on separate machines.
- Load generated by automated artificial clients.

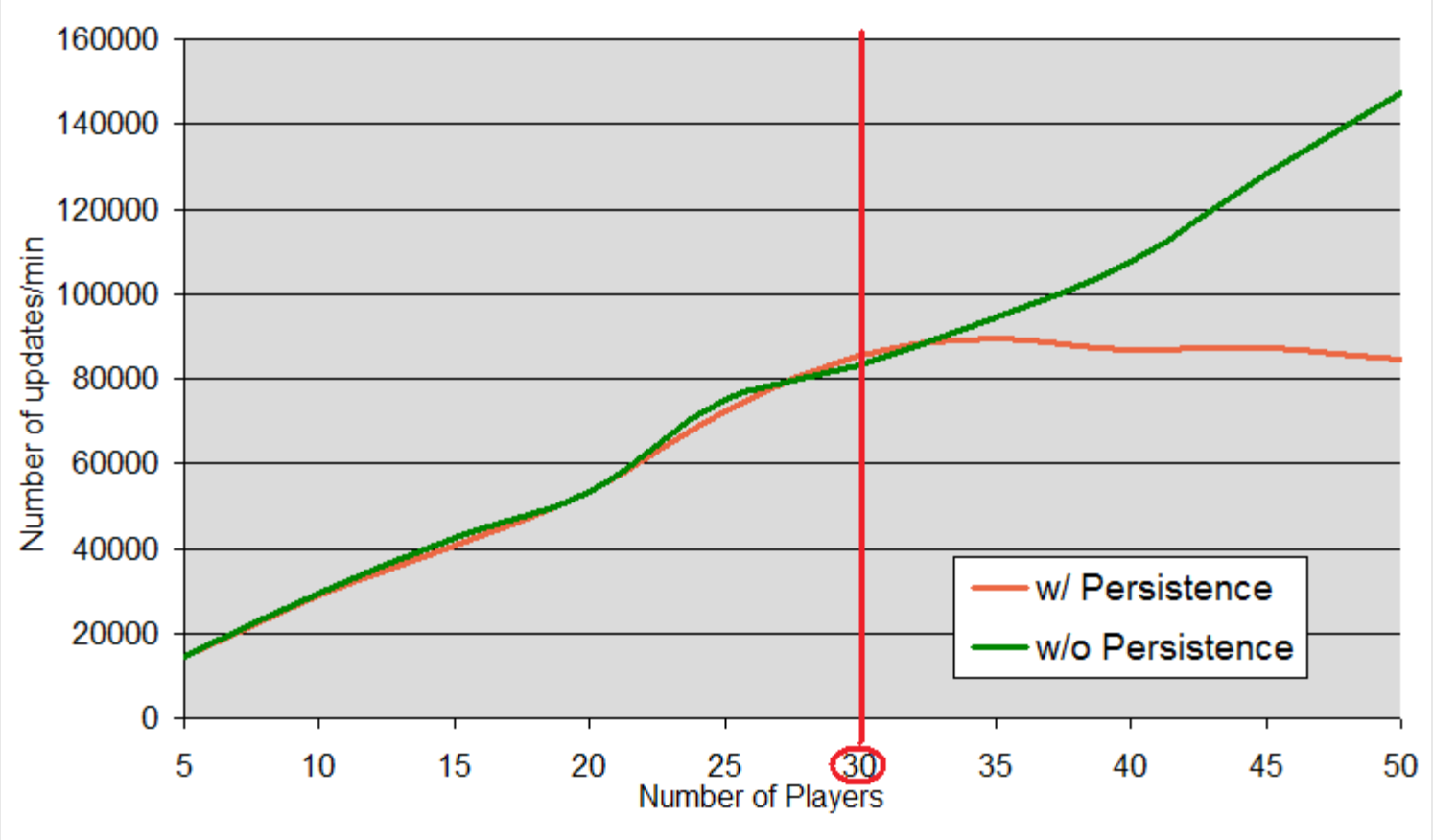
Implementation Structure

- Client/Server Game architecture
- Single server
- Persistence layer
- Relational database



Mammoth

Throughput Analysis (Naïve)



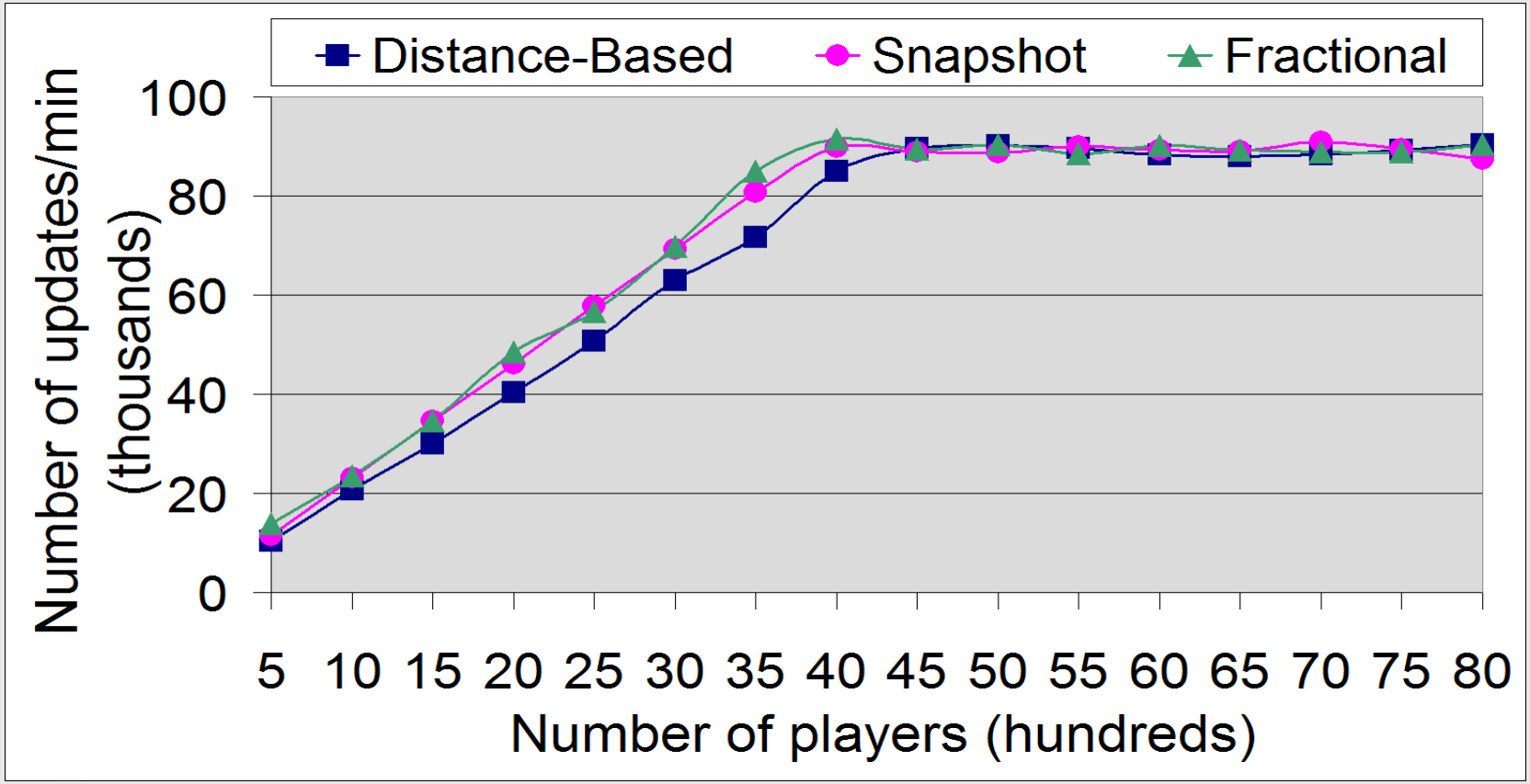
- Players are moving constantly.

Mammoth

Approximation strategies

- Target: 4000 players
- Capacity: 90000 messages per minute
- Worst-case locality, worst-case activity
- Parameters
  - Snapshot: every 2.67 seconds
  - Fractional: 0.75% of updates
  - Distance: threshold of 2.7
  - Error bound: roughly half a screen for all

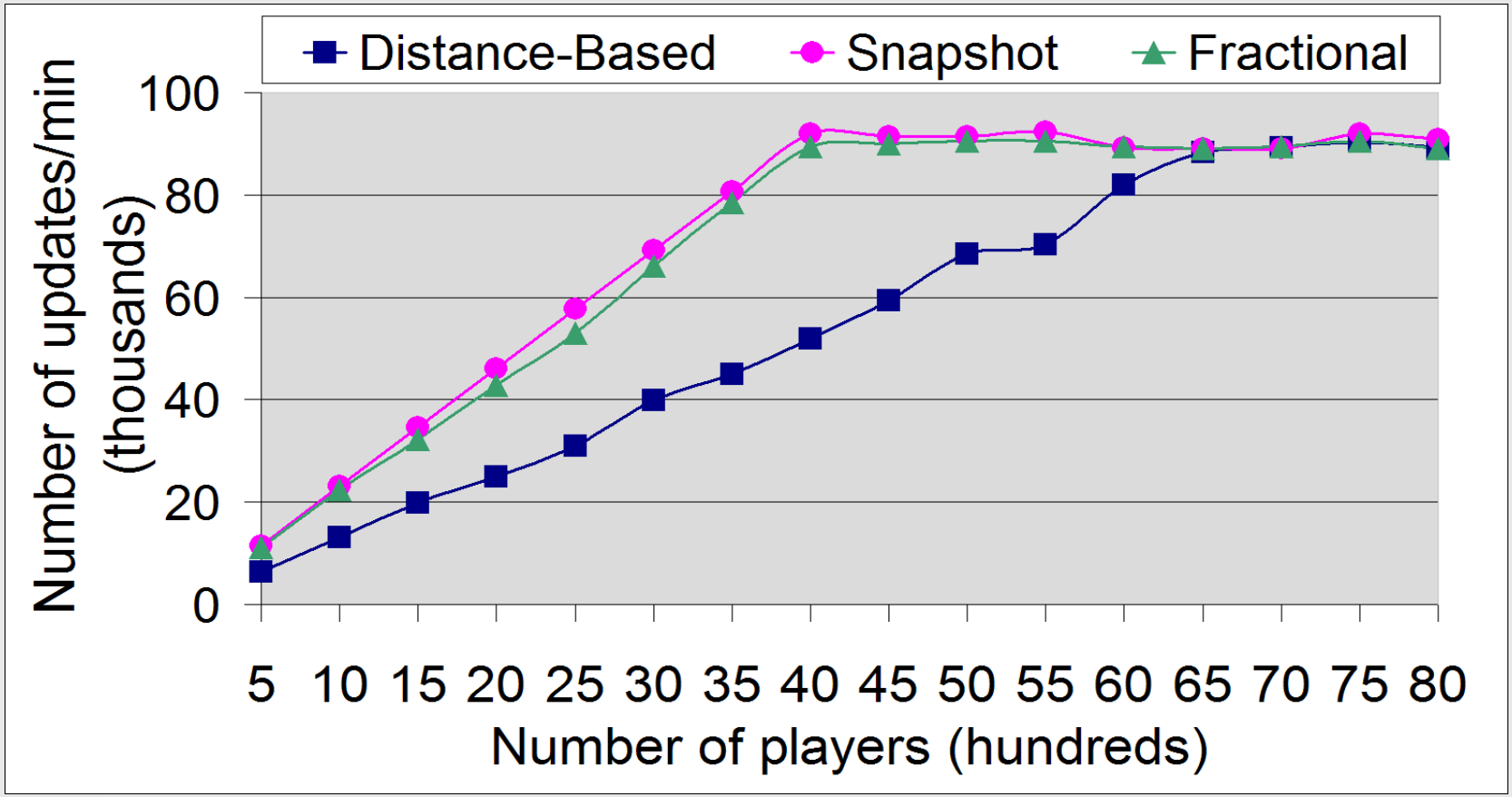
Worst-case scenario



- Players moving constantly in straight lines

Mammoth

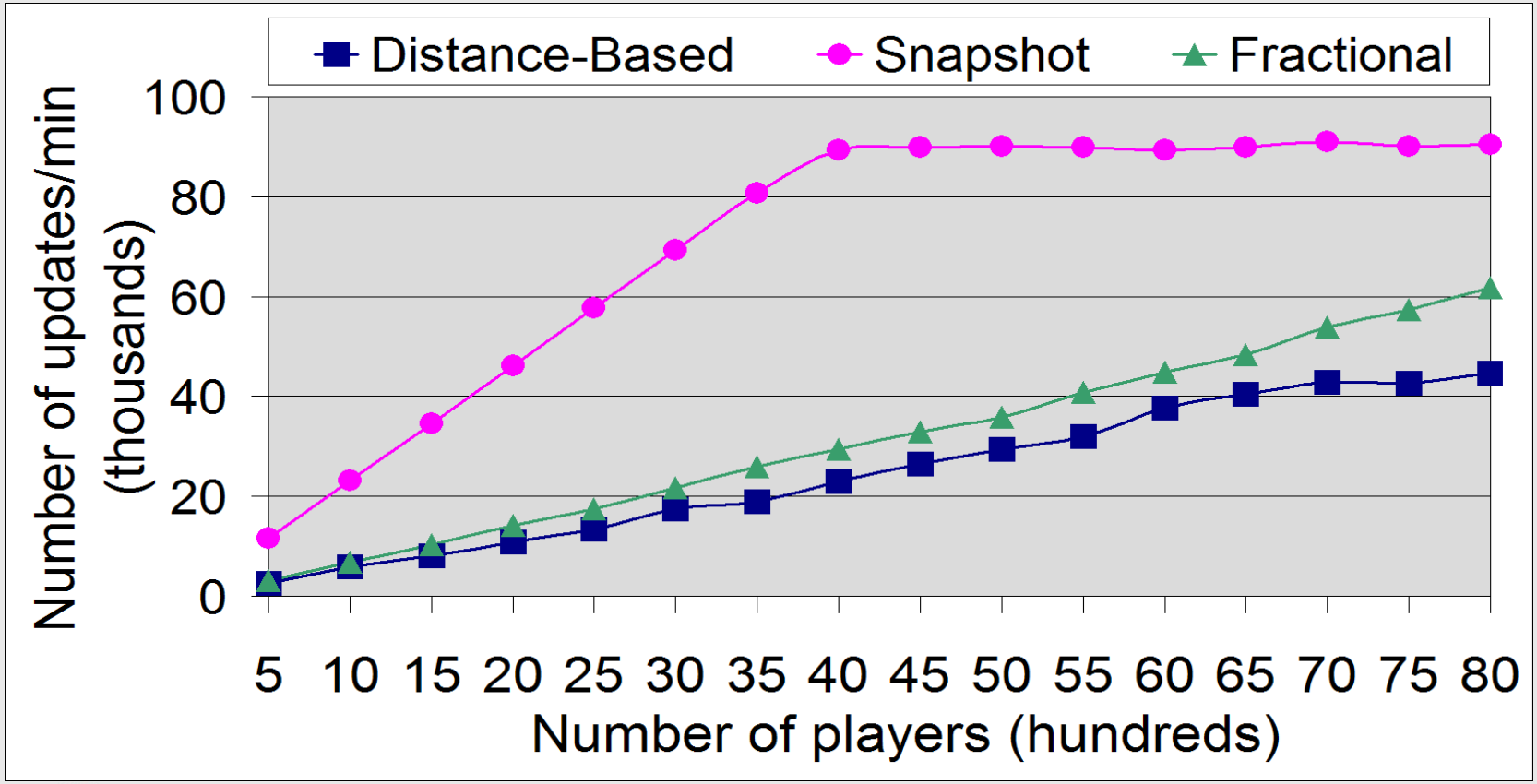
Worst-case activity scenario



- Players moving constantly and turning

Mammoth

Average case scenario



- Players moving and stopping to pick up items

Mammoth

Conclusion

- Game-aware persistence
- What? Consistency categories
- How? Strategies
- Inexpensive compared to generic solutions
- MAMMOTH: http://mammoth.cs.mcgill.ca/

Future Work

- Explore other approximate solutions and compare their overhead:
  - Adaptive/hybrid solutions
  - Dead reckoning
  - Zone-based
- Explore issues of exact storing schemes:
  - Transactions, especially distributed
  - Possible optimization (sequential logging)
- Distributed persistence
  - Fault-tolerance