Peer-to-Peer Support of Massively Multiplayer Games

Bjorn Knutsson
Honghui Lu
Wei Xu
Byan Hopkins

Presented by: Darcy Lien
Mark McElhinney

Outline

- Introduction
- Overview of Massively Multiplayer Games
- Current Multiplayer Infrastructure
- Peer-to-Peer Infrastructure
- Distributed Game Design
- Distributed Game on A P2P Overlay
- Implementation Details
- Experimental Results

Introduction

- Propose the use of peer-to-peer (P2P) overlays to support massively multi-player games (MMGs) on the Internet.
- Exploits the fact that players in MMGs display locality of interest
  - Can form self-organizing groups
- Implemented a simple game called SimMud to experiment with
  - Up to 4000 players

Problems to Consider

- Every massively multiplayer online game must consider:
  - **Performance** – updates must be propagated under time constraints, limited bandwidth an issue
  - **Availability** – whether the “model” or game state be accessed at all times
  - **Security** – prevention of account thefts, cheating etc (Clinton and Terry will discuss this)

Security

- P2P design prevents account theft by centralizing account management at the server
- Separate issue from the basic performance and availability of P2P gaming.

Overview of Massively Multiplayer Games

- Allows thousands of players to share a single game world.
- Typically MMGs are role-playing games(RPG) or real-time strategy (RTS/RPG).
  - Ex. EverQuest, Utlima, World of Warcraft, Sims Online

Overview of Massively Multiplayer Games

- Game States
  - Immutable landscape information (the Terrain)
  - Characters controlled by players (PCs)
  - Mutable objects such as food, tools, weapons
  - Mutable landscape (e.g. breakable windows)
  - Non-player characters (NPCs)

Overview of Massively Multiplayer Games

- Players are allowed three types of actions
  - Position change
  - Player-object interaction
  - Player-Player interaction
- Resulting world is huge
  - Divided into regions connected with each other
  - Generally each region is hosted by its own server

Current Multiplayer Infrastructure

- Traditionally supported by a client-server architecture
- Scalability is achieved by deploying server clusters
- Players connect to centralized server
- Server responsible for maintaining and distributing game state
  - Also handles account management

Current Multiplayer Infrastructure

- A typical single machine server can support 2000 to 6000 concurrent clients
- A cluster solution supports up to 32,000 players.


World of Lineage
ACROV
<Nailz> merry x-mass all!
<Baron> BOOOOOO
Littlebug: lol
ANUSBREATH: fucked
CHERGABANG: 시발
<General%%> damn
Windy1437
TheWorld: event end?????
Breeze: lol
GameMaster: r/hsiw white rabbit
FlyZ: zelgo or dai
GameMaster:
<General%> more
FlyZ: white rabbit hehee
<NewB> lemme out???

Peer-to-Peer Infrastructure

- P2P overlays provide the functionality of a scalable distributed hash table
  - Reliably mapping object keys to a unique live network node
- Built SimMud on top of the Pastry P2P overlay and Scribe Application level Multicast Infrastructure

Peer-to-Peer Infrastructure - Pastry

- Objects and Nodes mapped to random, uniformly distributed Ids
  - Objects mapped to node with numerically closest ID
  - Mapping procedure uses binary tree to index the hash table
  - Message routed to destination node in

    $Log_{2^b}N$ b = height of tree, N = # of nodes

    steps

Distributed Game Design

- Distribute the transient game state P2P
- Persistent information (payment info, character experience) handled by central server
- Delegate bandwidth and game state management to clients, while maintaining control over persistent game state

Distributed Game Design

- Partition the game world
  - Interest management determines partitioning
  - Players in same region form interest group of that region

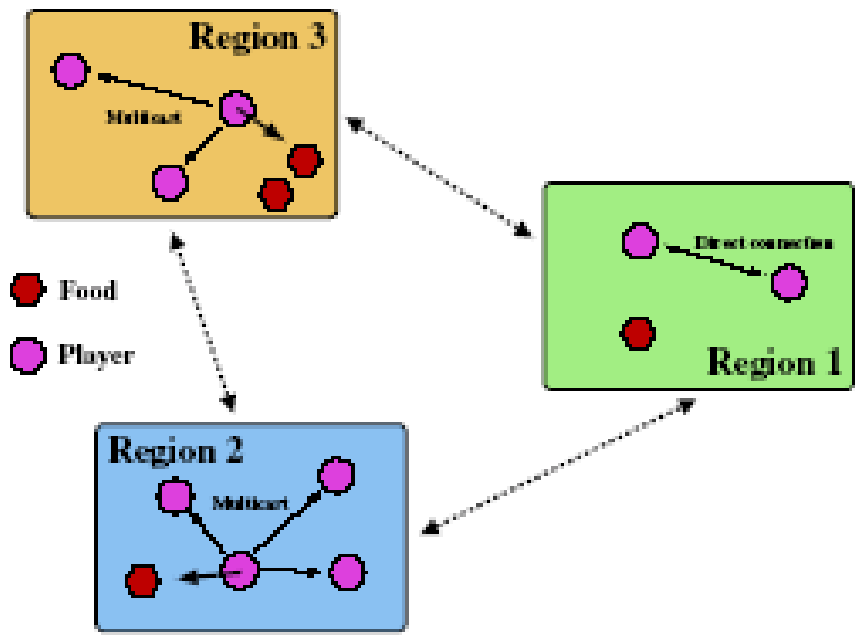

Fig. 1. Game Design

Distributed Game Design

- Game State Consistency
  - Split into *Player state*, *Object state*, *Map*
  - *Player state* – Position is multicast at a fixed interval to all other players in the region
  - *Object state* – Each object is assigned a coordinator that maintains and distributes its state (Coordinators discussed later)
  - *Map* – Distributed with the client software, updated using a software update mechanism

Distributed Game on P2P Overlay

- Based on Pastry and Scribe
- Mapping game states to peers
  - Each region is assigned an ID
  - Live node with closest ID serves as the coordinator for that region
  - Random mapping means coordinator not likely to be member of the region

Distributed Game on P2P Overlay

- Fault-tolerance problem
  - Clients may crash or disconnect at random
  - Must have replicas of games states
  - We assume:
    - Node failures are independent
    - Failure frequency is relatively low
    - Messages will be routed to the correct node

Distributed Game on P2P Overlay

- Shared state replication
  - Failures are detected using regular game events
  - Dynamically replicate the coordinator once a failure is detected
    - At least one replica is kept under all circumstances to prevent losses.
    - To cope with higher frequency failures it can be extended to multiple replicas
  - Failure and replication use P2P communication to route messages to the closest Node ID

Implementation Details

Uses SimMud on top of FreePastry (java)

- Map and Objects
  - Each region is described by a 2D array of terrain information
  - An object array tracks one kind of mutable object, food, modeled by the game
  - A player object handles the player's current position and other states
  - Players can perform moving, eating, and fighting

Implementation Details

- Inter-Player Interaction
  - Inter-player interactions are implemented with direct UDP messages.
  - All actions are executed on all participating parties with the same input algorithm and the results are exchanged for comparison
  - The coordinator is used as an arbiter where event ordering is important

Implementation Details

- Object Updates
  - Initialization are implemented trivially by sending the object and value to the coordinator
  - For read-write updates, the update is only valid if the current actual values matches the client's cached value
    - Requests for read-writes are keep in a queue for the coordinate to process the requests in order.

Experiment Results

- Implemented entirely in Java
- Used Pastry's network emulator to simulate various number of players
- Players eat and fight every 20 seconds
- Remain in a region for 40 seconds
- Position updates every 150 millisec by multicast

Experiment Results

Number of Nodes		1000	1000	4000	4000	1000	1000	1000
Number of Regions		100	100	400	400	25	25	100
Message Aggregation		Yes	No	Yes	No	Yes	No	Yes
Failure (node(s) per sec)		0	0	0	0	0	0	1
Total Messages	Average	24.12	82.17	26.98	82.12	26.58	283.13	24.52
	Max	227.62	115.57	216.14	125.18	383.11	341.71	221.38
Average Application Messages		14.76	81.84	14.73	81.7	17.46	282.81	15.09
Average Position Updates		13.34	80.33	13.34	80.18	13.34	278.89	13.31
Object Update Messages	Average	0.85	0.84	0.93	0.95	2.31	2.28	1.07
	Max	8.24	7.82	11.31	10.75	13.92	14.42	13.48

TABLE I

BREAKDOWN OF MESSAGE RATE BY FUNCTIONALITIES.

Experiment Results

- Position updates take up the majority of the messages
- Region changes take most bandwidth
- Message rate of object updates higher then player-player updates

Conclusions

- Measurements with up to 4000 players show that SimMud scales with the number of players
- The average message delay of 150ms can be easily tolerated by MMGs
- The bandwidth requirement on a peer is 7.2KB/sec on avg. and peaks to 22.34KB/sec
- Can sustain a practical failure rate for up to 20 hrs, exceeding the game interval for refreshing games