

# Peer Clustering: A Hybrid Approach to Distributed Virtual Environments

Alvin Chen  
University of California, Los Angeles  
3285 Boelter Hall  
Los Angeles, CA 90095  
alchemy@cs.ucla.edu

Richard R. Muntz  
University of California, Los Angeles  
3277 Boelter Hall  
Los Angeles, CA 90095  
muntz@cs.ucla.edu

## ABSTRACT

This paper proposes a hybrid architecture for distributed virtual environments, utilizing servers alongside peer-to-peer components. Current research into peer-based systems seeks to alleviate resource constraints, but it largely ignores a number of difficult problems, from bootstrapping and persistence to user authentication and system security (i.e., cheat resistance). This work proposes a hybrid architecture that turns the massive scale of the system from a problem into an asset, while still providing the features essential to a distributed virtual environment. Peers work together to distribute the workload, allowing redundant peer clusters to overcome failures and detect unacceptable behavior. The goal is to reduce cost and significantly increase the size of the concurrent user base while providing equivalent levels of robustness, persistence, and security. Simulations show that the hybrid architecture can handle massive populations.

## Categories and Subject Descriptors

C.2.4 [Computer-Communication Networks]: Distributed Systems – Distributed applications; D.2.11 [Software Engineering]: Software Architectures – Domain-specific architectures; K.8.0 [Personal Computing]: General – Games

## General Terms

Design, Performance, Reliability

## Keywords

MMO, Peer-to-Peer, Hybrid Architecture

## 1. INTRODUCTION

Today, many computer applications present spatial environments to their users, from virtual desktops to fully animated worlds. As a familiar mental construct, spatiality serves as an effective means for organizing, presenting, and accessing computing resources. Currently, the entertainment industry

produces the mostly widely used virtual environments in the form of multiplayer online games. These games can range in scale from small peer-to-peer strategy simulations and peer-hosted shooting games to sprawling massively multiplayer online (MMO) games. While a number of other virtual environments exist, we focus on MMO systems due to their extreme scale and commercial impact, which respectively provide interesting problems and a vested interest in solving them.

In the typical MMO game, players take on the role of a character in a virtual world, often in a fantasy, science-fiction, or historical setting. They move through vast virtual spaces, engaging in activities appropriate to each game, from slaying dragons to planting fields. Most importantly, they can interact with each other, communicating, collaborating, or competing. Successful MMO games support user bases of tens or hundreds of thousands, while the most popular (e.g., World of Warcraft [3]) boast millions of players across the globe, each paying monthly subscription fees to participate. Many such games feature vast environments that can take game characters hours to traverse. However, even when sparsely populated, current MMO systems can only support a few thousand concurrent players. Each group of players effectively exists in separate copies of the game world, cut off from the vast majority of the total user population.

### 1.1 MMO Requirements

We can frame the problems facing MMO architectures in terms of the needs of MMO game players and game publishers. At any given time, a player will interact with a small number of users located nearby in the virtual environment, engaging in conversation, item exchange, cooperative action, or competition. This behavior is similar to that of earlier online games, played by a handful of friends over the Internet. Crowds of too many users can overwhelm an area, making it visually confusing and reducing graphical performance. However, while a player may only wish to interact with a handful at a time, he expects to be able to meet up with any of the users occupying the same virtual world. Additionally, he would like the game to perform well, and he needs a secure, persistent environment that will preserve the time and effort he puts into the game. An enduring game should also allow for the introduction of new content to continually expand the playing experience.

Game publishers have related desires for MMO systems. They share many of the players' concerns, since they need

to maintain a satisfied customer base. Additionally, they would like to accommodate as many players as possible, maintaining acceptable performance at a low cost. They need a secure system to allow paying customers a fair experience. The system should also resist failures that interrupt system uptime. Finally, the publishers may wish to maintain a certain level of control over the system for troubleshooting and updating of game software, with an eye toward new content and the sale of game expansions.

## 1.2 Client-Server vs. Peer-to-Peer

Currently, the client-server architecture provides the key features needed by today's MMO systems but at great cost. The advantages of a centralized server solution include a high level of control over the system, which makes authentication, persistence, and security possible. Similarly, a cluster of high-end commercial servers can bring a lot of computational power to bear, while a data center can aggregate a large amount of network traffic. Unfortunately, the strongly centralized nature of the client-server architecture results in a severe performance bottleneck. Despite the expense, the server cluster still has limits to its computational power, and network traffic concentrates at the data center's routers. The increasing costs put strict limits to the architecture's scalability, requiring the division of the MMO game into separate instantiations. With a few servers expected to handle so much work, a single failure can cripple the entire system. Additionally, to achieve even this level of performance, game publishers incur a heavy financial burden just to maintain the system infrastructure.

On the other hand, small-scale games often employ peer-to-peer architectures. The peer-to-peer framework exhibits a number of useful properties that have attracted recent research attention for MMO systems. Small-scale games will often share the burden of simulating an environment. Peer-to-peer systems allow new users to provide resources to support the additional load they place on the system, helping it scale upward. If one host should fail, the remaining computers can handle its duties, providing added robustness. Network traffic flows according to interest, sharing the load among the users involved. All of this comes at little to no cost for the game publisher. Personal computers can already handle low-latency shooter games of 32 or more players [18], so they should have no trouble supporting small-group interactions with the more modest requirements of MMO games.

Unfortunately, peer-to-peer architectures feature key drawbacks that preclude them from mainstream use in MMO systems. Most importantly, they have no mechanism to guarantee persistence of game state. When users disconnect from a peer-to-peer system, they take their resources with them, including any data they may have managed. Since users can come and go, new users also need some way of finding their way into the system, but they cannot depend on finding any specific participant. With no central authority, publishers have difficulty securing the environment and updating software. Finally, though peers can provide more resources to the system, each machine has strict limits to its computational power. Without a straightforward way to share the load, a high concentration of activity can easily overwhelm any one peer's resources. Table 1 compares various strengths and weaknesses of the two architectures.

Despite many attractive properties, the limitations of peer-to-peer architectures have left game publishers with client-server architectures as the only alternative for their MMO systems.

## 1.3 Hybrid Approach

We propose a system architecture that utilizes elements of both client-server and peer-to-peer design to support a virtual environment, providing greater scalability than existing architectures at a lower cost to the publisher. User machines will support common small-group interactions, but powerful servers will still handle certain crucial elements, such as user authentication, game persistence, and high-density user interaction. Peer-to-peer communication and peer services cost the game publishers nothing. By distributing computation and, most importantly, network traffic amongst the users, game publishers can support larger populations at much lower cost, reducing the price of releasing and maintaining an MMO game. If peers can provide the bulk of MMO functionality, this can even significantly lower the financial barrier to entry in MMO game development, resulting in a more diverse and dynamic market. With position updates making up about 70% of MMO traffic [11] in some current systems, even modest peer management of transient data could have a major impact on MMO requirements.

Tying user machines more directly into the game system requires solutions to a number of problems. First of all, the system must integrate new peers with the existing participants. The system will need to distribute the workload among the available peers, and they will need some way to find appropriate sources and destinations for their data. They must handle constant user turnover, redistributing work and avoiding data loss while resisting attempts at cheating by unscrupulous users with unprecedented access to game functionality. The hybrid architecture needs to provide this functionality at a lower cost to the game publishers, and the game system may well have to provide some sort of incentive to the users to make their system resources available.

## 2. ARCHITECTURAL OVERVIEW

Our architecture moves the most common services to the peer network, freeing up commercial servers to handle book-keeping and focused computation, while also leveraging the availability of peer resources. The architecture rests on a partitioning of the virtual environment into regions small enough for a typical peer to manage. A hash-based overlay network divides responsibilities for these regions among the peers, balancing the demands placed on each *region manager* to limit the incidence of overloading, which requires server attention. Additionally, the overlay allows peers to withdraw gracefully from the system. We organize peers into *peer clusters* that can handle unexpected disconnections and discourage cheating within the MMO game by policing each other's behavior. Because the publisher's game servers are only involved in controlled services like account authentication, along with handling high-activity game regions, the peer services can significantly extend the capabilities of the server infrastructure. The architecture can also allow for a heterogeneous set of peers, providing benefits with incremental deployment.

	Client-Server	Peer-to-Peer
Cost	expensive leasing	<i>customer-provided/free</i>
Total Computational Power	limited, pricy	<i>highly scalable</i>
Local Computational Power	<i>powerful server cluster</i>	limited, heterogeneous personal computers
Network Load	concentrated, metered	<i>distributed, customer-provided</i>
Persistence	<i>powerful central databases</i>	unreliable storage, no guaranteed availability
Robustness	<i>managed</i> , central point of failure	unreliable but possibly <i>isolated failures</i>
Control/Maintenance	<i>centralized authority, staffed data center</i>	little to none after release
Security	<i>centralized, securable infrastructure, data logs</i> ; exposed client	limited; fully exposed

Table 1: Client-Server vs. Peer-to-Peer

## 2.1 Environment Partitioning

A user can log into the system in the standard manner, contacting a central login server for authentication. The user machine also receives a randomly generated identifier. His machine uses this identifier to join a hash-based overlay that allows other peers to locate it. This identifier also determines the requests managed by the new peer. The current prototype uses a Pastry-based routing mechanism [15]. Pastry assigns 128-bit identifiers to each of the peers in its overlay routing network. Each peer maintains a subset of other peer addresses, chosen to maintain a geometric distribution of identifiers within the 128-bit identifier space. Given a total of  $N$  peers, Pastry allows a peer to locate another peer using  $O(\log N)$  hops through the overlay network. Each peer also maintains lists of  $n$  peers with identifiers numerically nearest its own, one list containing  $n/2$  increasingly larger identifiers and one containing  $n/2$  smaller ones. These lists resist the concurrent failure of  $n/2$  consecutive peers within the overlay network, thus providing a robust means of locating peers within the system without depending on a central authority. Since peers receive random identifiers, even localized network failures occur independently in the identifier space. Large network problems will result in scattered failures that the adjacency lists can patch, making overlay failure less likely.

To complement the overlay network, the game partitions its virtual environment into many smaller regions, each manageable by a typical peer system. Each game region then receives a random identifier as well. Each peer manages those regions that are numerically nearest its identifier. Requests and queries intended for a given region can thus use the region's identifier to route through the overlay network, finding the peer responsible for managing the given area; we call this peer the *region manager*. Note that the random assignment of the regional identifiers dissociates peer management from regional locality. That is, as the peer population increases, it becomes extremely unlikely that a given peer will manage geographically adjacent regions. This helps keep flash crowds from overwhelming a given peer, since it will not be managing nearby regions that are likely experiencing crowding as well. This also distributes management responsibilities throughout the virtual environment, acting as an automatic division of labor. Similarly, the fixed nature of the peer identifier means that the peer's client component can move a player's game character from region to region without affecting the management duties of the server component.

## 2.2 Regional Initialization

When a user decides to connect to one of his game characters, the login server will provide his game client with the identifier of the region where his character will appear. The game client can then perform an overlay lookup to find the region manager for the given game region. If the region manager is not already servicing the region, this contact will cause it to start loading the region and initializing its neighbors. This helps hide the overlay lookups within the standard startup delay of an MMO game.

The region manager for a newly activated region will load the data associated with the region and will initialize the neighboring regions. Regional loading may involve retrieving game geometry, setting up computer agents within the region (e.g., non-player characters or computer-controlled enemies), etc. The region manager will also use the overlay network to contact the region managers for adjacent regions. It will store the network addresses of these peers, so they can communicate directly and avoid the delay of future overlay lookups.

The neighboring regions will enter an initialized state, loading the same types of game data as an active region, in preparation for the movement of characters into their regions, as well as to display remote events (e.g., computer-controlled agents visible in an adjacent region). However, these neighbors will not become fully active until a character actually enters their region. Only fully active regions will perform overlay lookups and initialize their neighboring regions, thus propagating the initialized state. Regions must initialize their neighbors, since players will often be able to perceive events in adjacent areas, but preemptive initialization also provides other benefits. Most importantly, it hides the delay involved in making overlay lookups for neighboring regions. The network addresses allow the peers to communicate directly with one another, delivering events relevant to their locale. The active region can also deliver the network addresses of these peers to its clients, thus speeding up the handoff process when a character moves from one region to another. When the character moves to the new region, it will also find a fully initialized area, ready to service any client requests.

## 3. REDUNDANT PEER CLUSTERS

With an abundance of participants, the hybrid architecture can allocate regional responsibilities to groups of peers, which we call *peer clusters*. A region's peer cluster consists of its region manager and a small number of *backup region*

*managers*. These backups can prepare themselves to take up management duties in case there are problems with the region manager. As they process game data, they can also verify the results generated by the region manager. Since each peer can serve data to multiple clients, new participants provide redundant resources that peer clusters can harness to improve the robustness of the system.

### 3.1 Peer Withdrawal

During the game system's regular operation, many peers will eventually disconnect from the system, especially as players decide to end a specific game session. By electing to log off of the game system, the player's peer can initiate a graceful withdrawal. If it manages an initialized or active region, it must pass its responsibilities on to another peer. At the very least, a region manager should identify another peer capable of handling its responsibilities and inform its clients of the handoff. The nature of the Pastry routing scheme provides us with a natural choice at this point. When the current region manager leaves the system, clients entering its old region will contact the peer with the numerically closest identifier still left in the system. Logically, the old region manager should pass its responsibilities to the peer second closest to a region's identifier. Fortunately, Pastry peers maintain an adjacency list of exactly those peers that are nearest in identifier, so one can easily determine the appropriate recipient and contact it directly.

During the disconnection process, the exiting peer can transfer all necessary data to the appropriate recipient, updating clients and neighbors with the address of the new region manager. Depending on the number of objects and events the peer needs to transfer, this may take a few moments, during which client service may suffer. Here we rely on peer redundancy to smooth out the process. We can define the *peer cluster* for a region to be the peers with identifiers nearest that of the region; naturally, this includes the region manager, but the other peers become *backup region managers*. While the region manager maintains official control of a given region, resolving timing issues, etc., a backup region manager can maintain an identical simulation of the region's state, thus allowing eventual peer withdrawals to happen transparently. While this utilizes another peer's resources, peer redundancy does not impact the game publisher's data centers at all, as new communication and computation all occur on peer systems. Additionally, backups can process events received from the region manager more lazily, aggregating events and processing them as their systems have idle time, since they are not directly responsible for client interaction. Already, peer-to-peer-based computer games have successfully used redundant simulation to provide a number of game features. For instance, the popular Age of Empires, a real-time strategy game that engages up to eight online opponents, runs its game simulation on every single peer involved in a given game session [2]. In fact, redundant peer clustering also increases system robustness and security.

### 3.2 Peer Failure

The establishment of one or more backup region managers provides the system with robustness even to the unexpected loss of a peer. If clients lose their connections to a given region manager, they can immediately contact the backup region manager, delivering events as usual. Similarly, if the

backup detects the failure of the region manager, it can directly contact the clients and neighboring regions. For most purposes, the peers can depend on soft state messages from a region manager to notice peer failure. Since the backup retains all previous state and clients store their pending events, the system can easily continue game processing, hiding peer failure from the players.

A single backup can handle the failure of a region manager, leaving only a vulnerable window equal to the time required to establish a backup for the new manager. The central game servers can also provide a last-ditch backup, giving a coarse-grained consistency based on persistent data in the game databases and the default state for a region. Fortunately, game designers can tune the reliability of their system by adjusting the size of peer clusters. Region managers can simply multicast updates to a set of backups, instantly providing multiple levels of redundancy for unexpected peer failures. Of course, this comes at the cost of additional processing on these backup peers. Fortunately, since players tend to form small, sparsely distributed groups, many peers can take up redundant processing, depending on the robustness and performance goals of the game designers.

### 3.3 Peer Overloading

When a region manager's service begins to degrade due to overpopulation, it can initiate a transfer of responsibilities, this time to the central game servers. Since consumer machines have a limited amount of processing power available, as well as restricted network capacity, they will only gracefully service a limited number of players. Older hardware can handle latency-critical shooter games with 32 or more players [18], suggesting that modern systems running MMO games, with their significantly relaxed performance requirements, should manage many dozens of players without problems. Still, most MMO games allow hundreds of players to congregate within a limited region, even if this happens very rarely. As a region manager approaches overflow, it can transfer its responsibilities to the game's high-powered commercial servers. This involves the same state transfer that occurs during a graceful peer withdrawal. The region's neighbors can then forward new clients entering the overpopulated region to the game servers, which can handle much higher player densities. When the crowd disperses, the game servers can transfer responsibility back to the designated region manager.

Additionally, certain social hubs, such as in-game starting areas or city centers, may regularly entertain dense player populations. The overflow system allows the game designers to easily set up regions permanently maintained by the central game servers, anticipating the heavy workload placed on such regions. In fact, each region can presumably have a tunable or adaptive threshold that would trigger an overflow transition based on available peer resources, compartmentalization of the region, and other game-specific factors. Given the right metrics, the designers can even take advantage of improving trends in user hardware or adapt to heterogeneity within the peer system.

### 3.4 Peer Review

The availability of backup managers concurrently running the simulation provides an ideal security check for the re-

gion manager. Like the multiple simulations run in small-scale games like Age of Empires [2], the redundant peers can guarantee that the peer manager does not unfairly reward or punish clients in its region, since each peer should produce the same simulation results. The region manager can only engage in minor protocol-level cheating, perhaps by reordering or delaying the packets it receives, unless it can collude with the backup managers. Since peer identifiers are allocated randomly, opportunities for collusion would be extremely rare, especially as the number of backups increases. In fact, if a client system runs a copy of the simulation, it can also verify the region manager’s decisions. Peer clusters essentially reproduce the small-scale peer-to-peer environment within the larger peer-to-peer system.

If clients remain anonymous, known only by their temporary identifiers, unscrupulous peer managers would have even less incentive to cheat, except to blindly punish all foreign clients. To further discourage cheating, the system may force peers to recuse themselves from managing a region occupied by their own client. Peer review gives designers a tunable means of providing security similar to that of a client-server system, while still allowing the use of traditional security measures (e.g., logging of game statistics).

## 4. ANALYSIS AND EVALUATION

### 4.1 Robustness Through Peer Clustering

We can take a quick look at the impact of peer clustering on the robustness of a peer-supported MMO system. Let  $T$  be the average time to failure for a peer machine. Let  $t$  be the time required to transfer region state to create a new backup region manager in case of a failure. Therefore, the probability of a failure occurring during a state transfer is  $\frac{t}{T}$ . Given a peer cluster with  $k$  backups, the probability of all backups failing during a state transfer is  $(\frac{t}{T})^k$ , and we get  $\frac{1}{T} \cdot (\frac{t}{T})^k$  failures over time. Inverting, we see that backups extend the average time to failure to  $T \cdot (\frac{T}{t})^k$ .

Dynamic region state should consist of a limited amount of data, such as the location of mobile agents and temporary modifiers like decreases in their health, data that regularly reaches players in fractions of a second. Let us pretend that a particularly active region may take five seconds to transfer its state, even though this would be unacceptable latency in a real game. Given a somewhat stable user machine and a reliable broadband connection, a peer might have an average time to failure of eight hours. With only a single backup, we have an average time to failure of over five years. Given a horribly unreliable peer, say one with a connection that completely fails every hour, a single backup gives an average time to failure of about 30 days, leading to a coarse-grained server recovery about once a month, but a second backup jumps the average time to failure up to about 59 years, even with a system full of poor network connections. Just a couple of backups can greatly increase the robustness of the system.

### 4.2 Lower Bound on Overflow

Analysis can help guide the design of the system as well. For a uniformly random distribution of players, we can equate the system with the classic balls-into-bins problem. In this well-studied scenario, one randomly tosses  $m$  balls into  $n$

bins, possibly leaving some bins empty while others end up containing multiple balls. In our case, we are distributing  $m$  players into  $n$  regions. While player distribution in a real MMO game may be far from uniformly random, such an analysis provides a lower bound on the clustering one can expect. In fact, even the random walk used in our simulations leads to non-uniform distributions due to the finite boundaries on our game world.

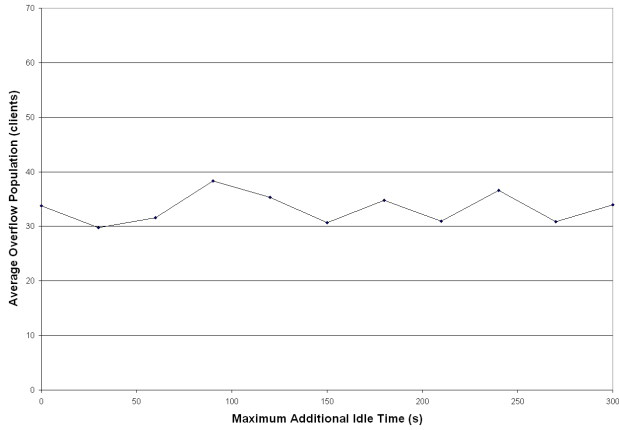
[14] presents a tight bound on the expected number of balls (players) in the most occupied bin (region), depending on how large or small  $m$  is compared to  $n$ . With  $m$  equal to  $n$ , a population density of 1 client per region, the expected number of clients in the most occupied region grows very slowly, at about  $\frac{\ln n}{\ln \ln n}$ . At high population densities, the highest occupancy should remain very close to the population density of  $\frac{m}{n}$ ; the region occupancies tend to even out. To see major gains, a real implementation of the hybrid architecture must keep player density below peer capacity. However, a great deal of room exists between this lower bound and the worst case of complete overflow, where the game servers must manage all the players; this worst case basically reduces to the client-server scenario.

### 4.3 Simulation Results

For the baseline simulation, we use a 20-by-20 grid of hexagonal regions. Players spend a minimum of 30 seconds traveling in each region. Players may spend up to another 180 seconds dealing with terrain or encounters in the area. They have a 40% chance of remaining in their current region; otherwise, they enter a randomly chosen neighboring region. It would only take ten minutes for a determined and unhindered client to cross the entire grid, making the base environment orders of magnitude smaller than the typical game world. Given the modern population cap of about 2000 clients in each game instantiation, we randomly distribute all 2000 across the tiny world, presenting an extreme population density of five clients per region. We give each peer the capacity to manage up to 16 clients across all of their assigned regions (a very conservative estimate) before having to hand off regions to the central servers. As the simulation results will show, lower population densities, though more realistic, simply do not stress the system at this level of peer capacity. As a measure of system load not handled by peer machines, we look at the number of clients that overflow to the game servers. All data points arise from the averaging of ten simulation runs, each lasting two hours of simulated time.

Figure 1 shows the first experiments varying the speed of the clients as they move through regions. While the basic amount of time to traverse a region remains 30 seconds, the experiment varies the maximum amount of additional time spent from nothing to five minutes. Note that clients spend a uniformly random amount of this additional time, so clients will still sometimes move through a region completely unhindered, even in the slowest case.

Figure 1 shows that variations in speed, and thus volatility of the client distribution, have no discernible impact on the average overflow population that the central servers must handle. Of the 2000 clients, only about 30 to 40 end up overflowing their regions’ managers at a time, showing that



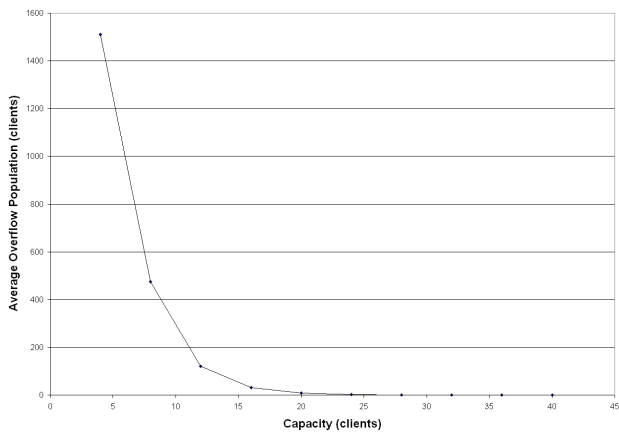
**Figure 1: Speed variation**

the peer systems can handle almost the entire load placed by wandering clients, even at this exaggerated population density.

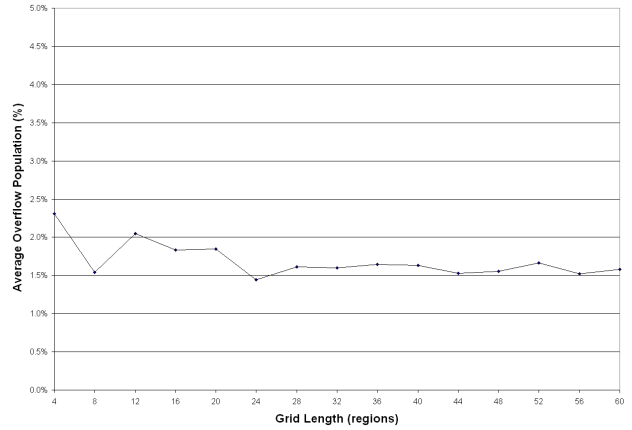
The next set of simulations used the baseline simulation model but varied the capacity of the peers from 4 clients to 40. Since high-end user machines can handle over 50 clients in highly constrained, latency-intolerant shooter games, peer systems should reasonably handle at least that amount for the slower pace of a typical MMO environment.

Figure 2 depicts the average overflow population decreasing exponentially as peer capacity increases. In fact, even when peer capacity sits at 4 clients per peer, peers can handle about one quarter (500) of the clients for the servers, whenever a few happen to spread out. The overflow population drops rapidly as capacity approaches the expected maximum occupancy. Even in our high-density environment, a conservative peer capacity handles most of the system load, and less conservative estimates reduce overflow to nothing.

Given these encouraging results, our next experiments varied the scale of the environment. The simulations main-

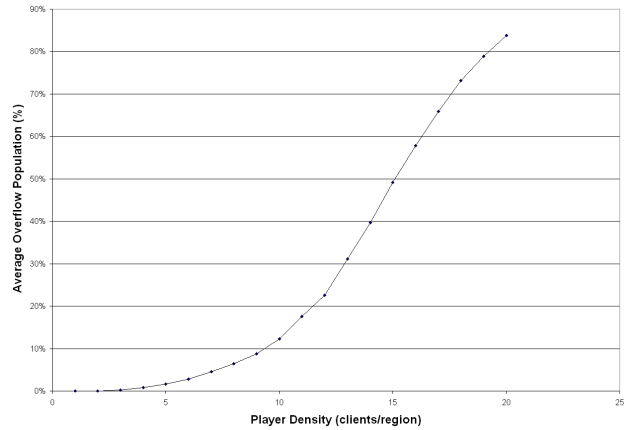


**Figure 2: Peer capacity variation**



**Figure 3: Simulation scale variation**

tained the high population density of 5 clients per region but expanded the grid size. The grid length varied from 4 to 60 hexagonal regions, resulting in 16 to 3600 regions in world area. Figure 3 shows that the average overflow population remains a nearly constant percentage of the total population, even with 18,000 clients moving through the world, as long as the total population density remains the same. On average, there are only about 284 overflowed clients out of 18,000. Extrapolating this trend, even a high-density, low-capacity environment would need over 125,000 simultaneous users for the central servers to regularly handle 2000 clients themselves.



**Figure 4: Player density**

To investigate the effects of changing population density, the next simulations maintained a constant world size while varying total density from 1 to 20 clients per region, corresponding to total populations of 400 to 8000 clients. Figure 4 shows that the average overflow population percentage increases gradually until it approaches the peer capacity of 16 clients. At about 11 clients per region and higher, the overflow population quickly becomes a large portion of the total population. Clearly, peers cannot handle average loads much higher than their capacities.

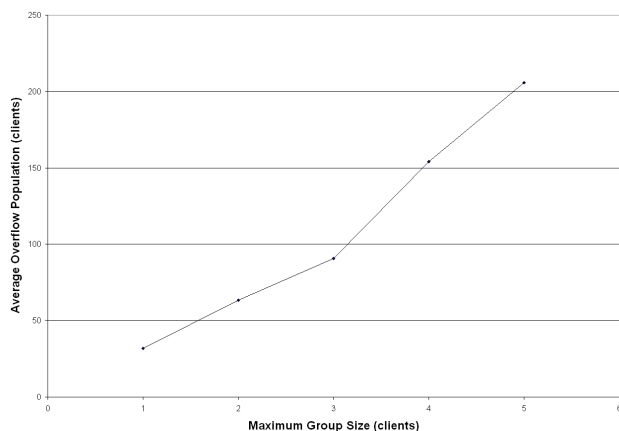


Figure 5: Group size distribution

Finally, most modern MMO systems allow players to form groups or teams. This allows groups of friends to interact with the virtual environment together. Such groups will travel together through the environment, leading to coordinated movement. Figure 5 shows simulations in which clients can form groups that move together from region to region. The simulations use a simple uniform distribution of group sizes. That is, if the maximum group size is 2, then about one half of the groups will be of size 1, and one half will be of size 2. Note that many more clients will end up in multi-client groups based on this distribution (e.g., two-thirds of the population with a maximum group size of 2). This should result in much coarser-grained behavior among the clients, since each group’s move will have a greater impact on region populations.

Figure 5 shows a nearly linear increase in the average overflow population as the maximum group size grows. Rather than 2000 clients moving about individually, the “maximum group size 5” simulation should see about 667 groups, divided equally among 1, 2, 3, 4, and 5-client groups. While a more realistic model would probably see many more individuals and small groups wandering the world, Figure 5 demonstrates that coordinated movement can have a significant effect on the overflow population. While the relationship between peer capacity and total population density may describe standard performance, willful clients can certainly overflow peers if they work together.

## 5. RELATED WORK

Though client-server systems run modern MMO games, a number of modern researchers have moved away from the traditional architecture. Many have tried to give their systems some of the benefits offered by peer-to-peer systems. Mirrored servers form a first step in this direction from the basic client-server architecture [6]. Clients communicate with nearby mirror servers as they would with traditional servers. However, these mirror servers communicate with each other across a low-latency, multicast-enabled private network, using replication and peer-to-peer protocols. In essence, mirrored servers are a distributed version of the server clusters normally located in a single data center, now designed to reduce latency and jitter over large distances,

though clearly at significant expense. In a similar vein, [16] details the use of grid technology to allocate server resources to various MMO games, creating more localized instances as demand rises. [1] and [12] describe hardware-oriented solutions that distribute network appliances, called booster boxes and proxies, respectively, to receive, filter, and aggregate client requests for a server. While not fully replicating servers as in a mirrored-server architecture, their servers do delegate tasks to the network appliances, giving them knowledge of game state, as well as arbitration authority. In return, they hope to gain the benefits of lower latency and reduced server load.

On the other hand, a lot of recent work uses large-scale peer-to-peer networks to support virtual environments. In these systems, networks of peers maintain the entire state of the environment. These approaches break down into nearest-neighbor networks and regional controllers. Unfortunately, both approaches still experience all the problems common to pure peer-to-peer schemes, such as lack of persistence. Most researchers have deferred discussions of the problems fundamental to peer-to-peer architectures, particularly security.

Nearest-neighbor approaches, like [9], [8], and [7], have each peer keep track of the peers nearest them in virtual space. [9]’s peers ensure that they remain within a convex hull formed by their neighbors, and they use a greedy routing algorithm for non-contiguous motion, such as when a new client joins the system. [7] has a peer construct a Voronoi diagram of its neighbors, thus obtaining enclosing sets of neighbors and boundary sets that keep track of the peer’s area of interest. Nearest-neighbor peer schemes require careful maintenance of neighbor sets to maintain consistency, and they suffer greatly in the presence of crowding. They also have very localized observational horizons that impact fundamental difficulties, like bootstrapping.

On the other hand, regional controller schemes elevate certain peers in responsibility. [17] designates certain peers as coordinators, and these coordinators control local regions defined by a Voronoi diagram over all coordinators. These regions change in size and shape as the coordinators move through the virtual environment, but peers can now simply treat local coordinators as servers for arbitrating local interactions. Local coordinators have direct reasons to cheat in their own favor, while coordinator density and distribution can still lead to problems.

Finally, a few researchers describe truly hybrid architectures. [5] provides the earliest outline of a system using a centralized server for user authentication and other sensitive but low-computation activities. [10] provides a more peer-focused approach, relegating its server to optional status for sensitive, persistent user data, like payment information. Like the architecture described here, [10] distributes responsibilities by matching random identifiers to peer systems, using Scribe [4] multicast (built on Pastry) for messaging. However, they assign fixed identifiers to all game objects, as well as game regions. One peer may arbitrate interactions within a given region, but state changes will need to be propagated to the many peers responsible for the region’s objects. The peers must also replicate all of this distributed

game state to help handle occasional node failures. While [10] uses a more peer-oriented, fine-grained approach, it has many similarities with our hybrid architecture due to the shared characteristic of randomized partitioning of responsibility. Our hybrid approach and redundant peer clusters can help address open problems facing their system, such as cheat resistance and overlay latency.

## 6. DISCUSSION

System-scale research needs to have realistic avenues for incremental deployment. For a hybrid system architecture that expects users to contribute system resources, one must clearly delineate the steps taken to encourage participation and ensure a viable system. A publisher seeking to develop a system using this hybrid architecture may have to scale up to full deployment but should still see benefits in the meantime.

### 6.1 Heterogeneous Peers

Client systems will likely vary greatly in their computational capacity, and their network resources may even fluctuate. The architecture predicates itself on the participation of peers capable of supporting the interactions of at least a few clients in a given region. While peers can abdicate responsibilities to the central game servers when they are overwhelmed, it might be best to restrict management duties to peers that can contribute some minimum level of resources to the system. Peers unable to meet these minimum requirements can still participate as clients, but they will no longer burden the system with frequent disconnections and state transfers. The system can also adjust overflow thresholds as peer systems improve. Fortunately, since a standard peer should be able to support many clients, full participation by even a small fraction of user systems can enable the entire system.

### 6.2 Peering Incentives

Users may be reluctant to contribute system and network resources to a commercial system, despite the inherent benefits. The system may even want to encourage certain users to opt out of participation, as in the case of the substandard peers described above. To attract participants, the publishers may want to offer some form of incentive for users to participate in peer management. Ideally, users would contribute resources even when not actively using the system, since their machines could always provide additional resources and stability. While publishers could simply offer users a fraction of the monetary savings provided by their systems' participation (e.g., reduced network traffic costs), this would probably amount to negligible amounts when divided across all participants, and it would cut into profitability. They could more easily offer in-game incentives, which cost them nothing to provide. Preferably, such incentives should not overly impact gameplay. For example, rewarding in-game wealth would encourage cheating and discourage players with substandard machines. Prestige rewards, such as unique decorative items, graphical embellishments, and character titles, could suitably reward participants without influencing gameplay. Such an incentive system could reward users for just making their systems available, or it could reward users for actual time spent managing a region for other clients.

## 6.3 Console Platforms

A large number of deployment problems disappear if one considers commercial game consoles as the deployment platform. Modern game consoles, like Microsoft's Xbox 360 [13], feature processing and networking capabilities comparable to the most advanced personal computers. They provide a homogeneous working environment with identical system resources, though network resources may still differ. Their hardware features also provide a relatively secure computing environment, much less prone to successful cheating and featuring strong disincentives for doing so (e.g., lifetime banning from the online service). Game consoles provide a nearly ideal deployment scenario for the hybrid architecture.

## 7. CONCLUSION

Our hybrid architecture holds great promise for supporting truly massive distributed virtual environments. Using this approach, existing servers can support more concurrent users by converting players from system load into system resources. However, as seen from the simulations involving grouped players, coordinated movement can still adversely impact server performance.

Future work should focus on more realistic movement models for the clients. While movement will differ greatly from game to game (e.g., walking peasants vs. soldiers in fighter jets), most environments will feature sparser areas with localized hotspots of activity. While current MMO environments feature extremely low-density populations scattered over massive areas, it remains theoretically possible for all clients to purposely converge upon a single region and overload it. Environmental constraints and gameplay pressures must discourage them from doing so, but a real-world system should expect the presence of occasional crowds.

The basic system for random region distribution does a good job of balancing client load, but a real MMO game might want explicit control over identifier assignment. For instance, the game servers might assign identifiers to purposely balance uneven distributions left by peer withdrawals. Similarly, nearby regions should have very different identifiers, making it less likely that crowding will affect peers in the same peer clusters. We may also want to further investigate system overhead, like state transfers resulting from peer overloading. With the variety of features involved, a full implementation has many algorithms from which to choose.

Both analysis and system simulation suggest that a hybrid architecture can greatly improve the efficiency of a massively scaled virtual environment. With incremental deployment techniques to help maintain a stable peer system, the hybrid architecture should provide access to the ever-expanding resources of the system's users, letting standard server clusters handle vast numbers of simultaneous users at a lower cost. The problematic large user base can now provide redundancy that makes the hybrid MMO system more robust and securable, while the server elements provide features like environmental persistence and user accountability generally missing from purely peer-to-peer approaches. While more study remains, the hybrid architecture serves as a viable, realistic solution to the problems presented by modern MMO systems.

## 8. REFERENCES

- [1] D. Bauer, S. Rooney, and P. Scotton. Network infrastructure for massively distributed games. In *Proceedings of the 1st Workshop on Network and System Support for Games (NetGames '02)*, pages 36–43, New York, NY, 2002. ACM Press.
- [2] P. Bettner and M. Terrano. 1500 archers on a 28.8: Network programming in age of empires and beyond. In *Game Developer's Conference (GDC 2001)*, March 2001.
- [3] Blizzard Entertainment. World of warcraft reaches more than one million customers in europe. Web page, January 19 2006. <http://www.blizzard.com/press/060119.shtml>.
- [4] M. Castro, M. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In *Proceedings of the 22nd Conference of the IEEE Communications Society (Infocom 2003)*, Washington, D.C., April 2003. IEEE Computer Society.
- [5] F. Cecin, R. Jannone, C. Geyer, M. Martins, and J. Barbosa. Freemng: A hybrid peer-to-peer and client-server model for massively multiplayer games. In *Proceedings of ACM SIGCOMM 2004 Workshops on NetGames '04: Network and System Support for Games*, page 172, New York, NY, 2004. ACM Press.
- [6] E. Cronin, A. Kurc, B. Filstrup, and S. Jamin. An efficient synchronization mechanism for mirrored game architectures. In *Multimedia Tools and Applications*, volume 23, pages 7–30, London, UK, May 2004. Springer Science+Business Media B.V.
- [7] S.-Y. Hu and G.-M. Liao. Scalable peer-to-peer networked virtual environment. In *Proceedings of ACM SIGCOMM 2004 Workshops on NetGames '04: Network and System Support for Games*, pages 129–133, New York, NY, 2004. ACM Press.
- [8] Y. Kawahara, H. Morikawa, and T. Aoyama. A peer-to-peer message exchange scheme for large-scale networked virtual environments. *Telecommunication Systems*, 25(3-4):353–370, March-April 2004.
- [9] J. Keller and G. Simon. Solipsis: A massively multi-participant virtual world. In *Proceedings of the 2003 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '03)*, 2003.
- [10] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In *Proceedings of the 23rd Conference of the IEEE Communications Society (Infocom 2004)*, Washington, D.C., 2004. IEEE Computer Society.
- [11] J. Lee. Considerations for movement and physics in mmp games. In *Massively Multiplayer Game Development*, pages 275–289, Hingham, MA, 2003. Charles River Media, Inc.
- [12] M. Mauve, S. Fischer, and J. Widmer. A generic proxy system for networked computer games. In *Proceedings of the 1st Workshop on Network and System Support for Games (NetGames '02)*, pages 25–28, New York, NY, 2002. ACM Press.
- [13] Microsoft Corporation. Xbox.com home. Web page, 2006. <http://www.xbox.com/en-US/>.
- [14] M. Raab and A. Steger. “balls into bins” – a simple and tight analysis. In *Proceedings of the Second International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM'98)*, pages 159–170, London, UK, 1998. Springer-Verlag.
- [15] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–350, London, UK, 2001. Springer-Verlag.
- [16] D. Saha, S. Sahu, and A. Shaikh. A service platform for on-line games. In *Proceedings of the 2nd Workshop on Network and System Support for Games (NetGames '03)*, pages 180–184, New York, NY, 2003. ACM Press.
- [17] A. Tumbde and S. Venugopalan. A voronoi partitioning approach to support massively multiplayer online games. CS 740 Project, The University of Wisconsin, Madison, 2004.
- [18] Valve Corporation. Counter-strike.net - the official web site. Web page, 2004. <http://www.counter-strike.net>.