Exploring Server-Centric Scalability for Social VR

1st Sebastian Friston
Computer Science
University College London
London, UK
sebastian.friston@ucl.ac.uk

2nd Otto Olkkonen
Computer Science
University College London
London, UK
otto.olkkonen.20@alumni.ucl.ac.uk

3rd Ben Congdon
Computer Science
University College London
London, UK
ben.congdon.11@ucl.ac.uk

4th Anthony Steed
Computer Science
University College London
London, UK
a.steed@ucl.ac.uk

***Abstract*—Social Virtual Reality (SVR) systems are growing in both popularity and breadth. New systems are attempting to support a wider range of increasingly large social situations. A key characteristic of SVR systems is user capacity. Many systems are designed for small group discussion or team games, and support up to 20-30 users. Some support larger counts, but via special features coupled to a specific social situation. As SVR broaches medium to large gatherings (10s-1000s of people), capacity becomes a major challenge. Naïve implementations where each user communicates directly with every other will quickly overwhelm the resources of the system. The constraints on scalability have been known since the first SVR systems were created. However, while there are many systems out there, their scalability mechanisms are often application specific, and rarely openly discussed. In this paper we explore scalability in the open-source SVR Ubiq. We consider two partitioning schemes to increase relay-server capacity. We examine the current failure modes of the relay-server system, and demonstrate how these schemes improve capacity by a factor of 2-3x. We look at the interactions between the schemes and simulated user behaviour, to see what lessons can be gleaned for research into scalability for SVR.**

I. Introduction

An increasingly popular use of virtual reality is social virtual reality (SVR). SVRs have diverse implementations, but a common subset of features. These include representations of users as avatars and real-time communication. There are various successful commercial SVR applications, such as AltSpace [38] and RecRoom [48], as well as numerous research systems that implement custom features for scientific investigation, e.g. [32].

However, it is rare to see capacity openly discussed. Some systems, such as Horizon Venues [15], are built around large-scale social situations. Others, such as AltSpace, attempt support for medium-scale situations through special features [39]. Most systems without special features (e.g. RecRoom [48], Hubs [41]) have capacities of 20-30 users. These capacities are not explicitly enforced at the platform level, and seem not to arise from an intentional target, but rather appear as a consequence of other technical decisions. The number of collaborating users significantly affects the nature of an interaction [3], [24] and quality of experience [32], [13], [45]. The types of experience taking place between small groups [40] and large concert crowds [62] are very different. There is good reason then to consider user capacity in SVR as a primary technical requirement, which changes with the purpose of the application.

This work was partly funded by United Kingdom EPSRC project Graphics Pipelines for Next Generation Mixed Reality Systems (grant reference EP/T01346X/1) and EU Horizon 2020 project RISE (grant number 739578)

The biggest challenge in scaling user capacity is the number of messages that must be exchanged to keep increasing numbers of peers synchronised. Messages can be made more efficient, and different topologies can be used to minimise duplication. Ultimately though, if N peers can all 'see' each other, then an update from one needs to reach, some way or another, N-1 peers, and vice versa.

The bottlenecks of capacity in SVR have been known since the earliest days, and various schemes have been devised to help [53], [55]. For example, spatial partitioning schemes [30] organise the space or the users into small groups to reduce the total number of messages. Exploring and integrating spatial schemes is a challenge however for two reasons. First, the space of social interactions is not continuous [3]. The social cues exchanged, and the space over which this is done, change depending on the context and environment. This makes the goals of a partitioning scheme itself application specific. Second, the implementation of a scheme is necessarily quite low level, limiting re-usability and lessons learned between systems. As such, there are no yet accepted general purpose solutions to scalability. Further, systems that have functioning scalability schemes are typically commercial and proprietary, and so how these schemes are implemented and how they perform is not common knowledge.

In this paper we explore scalability in the open-source SVR system Ubiq [17]. Ubiq is designed for research and teaching. It is a logically peer-to-peer system based on application-layer multicasting, but with a simple server for rendezvous and relay over the Internet. Currently, each Ubiq peer in a session sees all other peers, regardless of the environment size, creating a bottleneck in the server fan-out. We explore increasing the capacity of a Ubiq session, and in doing so its potential as a platform for exploring scalability schemes in general. We profile the system to identify its primary bottleneck. We implement two spatial partitioning schemes from the existing literature, and evaluate them through stress testing with large numbers of bots. We examine how these schemes increase user capacity, but also the interactions between the schemes and different bot behaviours. Our aim is to take the first steps into exploring how Ubiq could be used to evaluate scalability schemes, and support a broader set of use cases that exceed the capacity of most existing applications.

II. Previous Works

A compelling use of immersive VR is to enable collaboration. The same subsystems designed to facilitate immersion, e.g. body tracking and spatialised audio, make VR amenable for sharing social cues by connecting these systems over a network. Some of the earliest VR systems had social demonstrations (e.g. Reality Built for Two from VPL Research [4]). There were many demonstrations in early academic systems [56], [37], [12], [11], [31], and the resurgence of consumer VR has led to development of many new commercial applications. Schulz's blog about SVR [52] lists over 150 applications or platforms. Over 250 systems are listed in the XR Collaboration directory [61].

The networking strategies behind SVR systems vary. Early networked VR systems, such as DIVE [8], MASSIVE [21] and Blue-C [42] already supported common SVR features. These include avatars, action-based interaction, message passing and spatially-mediated interaction. Different architectures have been proposed. A common strategy is the manipulation of a shared scene graph. This strategy is common because a scene-graph is a common building block of VR software. An important design decision is how the graph is shared. Scene-synchronisation often uses the concept of authority to assign ownership over branches. In client-server systems the server often owns the graph absolutely and message passing is asymmetric. In peer-to-peer architectures, nodes are synchronised with approximations of primitives such as mutexes [14] or message-pumps [51]. Grimstead et al's review (c.a. 2005) [22] breaks down how many systems of the time fit into the above categories of access control, architecture and synchronisation. The review showed a definite preference for client-server architectures. The authority model and architecture are not completely dependent however. Logical peer-to-peer systems may still use a STAR (peer-to-peer with hub-based distribution) or a client-server connection architecture. Further, the interaction between authority model and architecture determine the messaging protocol, and so the number of messages required to fully synchronise a system.

User capacity is limited by the rate at which a system can exchange messages. In a naive peer-to-peer implementation, message rates scale $O(N^2)$ with the number of peers (N) as peers update each other. Synchronisation methods impact scalability through the amount of data exchanged [60]. Input-mirroring or other forms of duplicate simulation (e.g. [57], [23], [18]) can reduce bandwidth requirements. However while these approaches make messages more efficient, the interactions scale the same way. Ultimately it is necessary to reduce the number of interactions, which can be done by partitioning users.

The most popular partitioning approaches are spatial partitionings, where users are grouped by their location. The idea is that only necessary interactions, e.g. updates about the parts of the world the user can see, are communicated [7]. In one implementation the world is partitioned into cells of fixed size and position. Only users within the same cell interact. Area of Interest (AOI) methods interact with multiple cells at a time. There are a many approaches to AOIs. One is to have a fixed radius [36]. Steed and Angus [54] took advantage of the nature of dense environments to compute visible peers exactly. In contrast, Backhaus and Krause [2] ignore the structure of the environment and used only user location to build a graph of nearest neighbours.

As the number of these partitions increases, it is necessary to introduce load balancing to prevent a single server or peer from being overloaded with the overhead of managing the partitions. This affects the topology of the network, and is also dependent on its capabilities. For example, AOIs have been implemented directly at the network layer using multicast groups [36], [59], however this is not possible on the public internet. Second Life [34] allocates resources by $256m^2$ cells called *regions*, but this can leave some regions overloaded if their computational requirements are more intense. Chertov and Fahmy [10] resized and reallocated cells dynamically to balance users between resources. Lake et al [30] and Allard et al [1] proposed splitting up the simulation at the task level. Resources can be allocated by an arbitration server or through methods such as Distributed Hash Tables [26].

At higher levels, hierarchical methods are used to keep the world persistent across multiple groups [19]. Iimura et al [26] proposed federations of servers, where changes can be made by local authoritative nodes but propagated upwards so all players see a consistent world. Aggregation of this type is often seen in audio. Audio is highly challenging as each user would ideally receive a unique mix of streams for accurate spatialisation, presenting an $O(N^2)$ problem again, but with severe latency requirements. Methods vary between mixing all streams into one at a server, to creating N unique streams at a server, to forwarding all streams in a peer-to-peer fashion. By arranging mixers into a tree, arbitrarily large numbers of peers could exchange audio at a cost of spatialisation error, and increasing latency with distance through the tree [5], [6], [47], [43].

The messages exchanged by an SVR depend on the cues it is designed to transmit, and the necessary cues depend on the type of social interaction. The space of social interactions is not uniform or continuous however. For example, interactions can be distinguished between focused and unfocused [20], and within *focused* by how the loci is shared. Common-focused interactions have a non-reciprocal focus of attention, such as a lecture, while jointly-focused interactions, such as a conversation, are more mutual [28]. Sociologists have identified a number of dimensions on which to categorise interaction. Importantly, what types of interaction are available depend on the interaction between these measures [3]. For example, one measure is group-size. In a large crowd, not all users will interact reciprocally with the same level of attention, as users would not be able to share attention between such a large number of people. The type of interaction and group configuration affects what cues are transferred [29], [24]. A joint-focused conversation would entail a rich set of social cues between all participants. In public unfocused-interaction such as a queue, individuals must be aware of each others' locations to avoid colliding, but would not need to speak. (If they did speak, that would become a different type of

interaction.) Interactions can be locally scoped as well; small groups could form within an audience, for example. Further, while direct reciprocity between all audience members may not occur, audience members do influence each other indirectly [24], [16], [25].

For SVR, this means that the modality and quality of cues are highly application - or interaction - specific. Loss of spatialisation or increased latency due to mixing, for example, may be inconsequential to some situations but detrimental to others. While there is a spatial component to this, it is not sufficiently straightforward to design scalability schemes on its own. This ambiguity extends to all cues, even those which may seem straightforward. For example, in very large crowds seen from a distance, a custom skin for each avatar may be important, or not, depending on whether large sets of those avatars were wearing, e.g. uniforms. The taxonomy of social interactions goes some way towards predicting this, but has not, as far as we know, yet been sufficiently quantified to design objectively optimal scalability systems. As a result, different cues often use different approaches. For example, it may be expected that the VOIP system would have a different interest set than the avatar system. Additionally, different systems will be tuned for specific applications. They may also not be truly scalable but use illusions to achieve a particular effect. For example, virtual venues may simulate the majority of attendees locally, knowing the user would have no way to attempt an interaction with them.

Commercial systems vary in how they use these techniques to scale. AltSpace and RecRoom for example have individual room count limits of $\sim$40, but FrontRow allows AltSpace to broadcast activity to multiple rooms to support events with 100's of viewers [39]. Photon recommends 16-64 players, but relies on the developer to enforce fair use rather than setting a limit [46]. Roblox recently added experimental support for 700 users per room [50]. Voice chat in Roblox is still experimental however, and only supported in rooms with up to 30 users [49]. Virbela has supported $\sim$1000 person conferences. Users experience two types of audio: spatialised, but only from immediate neighbours, and a flat stream broadcast from the presenters [58]. Improbable connects servers that each handle a region of the world (spatial partition), theoretically scaling infinitely with compute resources [27]. The problem of scaling is the same as that in many large online games. However, VR is particularly challenging due to its requirements of low-level interaction and high responsiveness, required to maintain believability and presence. Therefore not all techniques employed by games are applicable. For example, EVE Online can scale in time to avoid latency spikes [9], a technique not easily applicable to immersive systems.

As the number of users increases, the number of interaction messages in an SVR increases geometrically, exhausting the resources of the systems on which the SVR is built. In the first case, users must be grouped to reduce the number of interactions, and in the second case load-balancing and aggregation used to distribute resources between these groups. The implementation of these techniques is currently very application specific, and a lot of the knowledge tied up in proprietary systems. In this paper we explore the subject of scalability in Ubiq. We choose this platform because we intend to use it for future SVR research at scale, but also because it is distinct in that the connection architecture can be separated from the message passing, which is logically peer-to-peer. Our aim is to improve Ubiq's capacity so it can be used to answer more questions about social scale in SVR. Also though, to understand its potential for testing scalability techniques, by evaluating not just the techniques themselves, but what we can learn about them and how they interact with the system used to evaluate them.

III. Ubiq

We evaluate two partitioning schemes by extending the Ubiq platform [17]. Messaging in Ubiq is logically peer-to-peer, with the authority scheme decided per object. For example, Avatars send authoritative updates, while physical objects could use force-integration. Messages are exchanged using application-layer multicasting: messages are addressed to an object, and received by all objects with the same identity. Fan-out is performed by the messaging layer. For the rest of this paper we use the term multicasting to mean application-layer multicasting.

To operate over the public internet Ubiq uses a central server for rendezvous and matchmaking. Users use pre-shared secrets to join *rooms*. A *room* is effectively a list of peers that should exchange messages. The server forwards messages between all peers in a room. This implements the server-based application-layer multicast. For voice, direct peer-to-peer connections are established using the server to exchange signalling messages for bootstrapping. Other messages (which require fan-out) go through the server. Message exchange is illustrated in Figure 1. Ubiq is designed to be heavily peer-centric. Ubiq can function without a server if peers can establish direct connections between themselves. However the rooms system assumes a central server. The server can support multiple rooms. Application behaviour is determined entirely by what is running on the peers. Peers can run the same Unity scene, but join different rooms, creating multiple instances or *shards* of an application. In Ubiq we consider a shard to be a room with users using a particular application. A server can support multiple shards for an application, and multiple applications. The server performs no simulation. It doesn't have the concept of a message loop, but reacts asynchronously to incoming messages, dispatching copies to all members of the source peers' room.

IV. Spatial Partitioning Schemes

A. Procedural Spatial Partition

The first scheme we investigate is an asymmetrical fixed spatial area-of-interest. The world is divided across its ground plane into $20m$ diameter hexagonal cells (Figure 2). As players move through the world they become members of the one cell they occupy, but can observe N surrounding cells. Observers see all peers that are members of the observed cells. This is typical

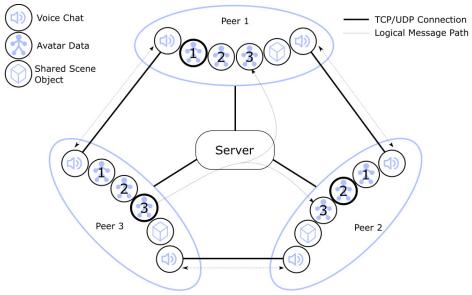

Fig. 1. Diagram of logical data flow in a typical Ubiq application. Multicast avatar data fans-out through a central server, while dedicated connections are established by the voice chat subsystem for each dyad.

behaviour in grid-based systems, such as NPSNET [35], for example. The partition itself is procedurally-generated based on the given fixed cell size. The scheme relies on player position only. Being procedural, it is accessible to all users simply through specifying a cell size and possibly an origin transform. The asymmetry works with the multicasting to match expected interactions: peers that overlap in observation will be aware of each other and establish VOIP channels, while those with one-way visibility can be seen, but will not expend resources on audio. Players move smoothly through the world, with others appearing or disappearing at the periphery.

To support this scheme, the server must have the concepts of a *member* and *observer*. As a room is effectively a list of peers, changing membership or observership will change the interest set. The interest set is all the members in the immediate room, and all the members in observed rooms. The set is maintained by the client. The server sends messages to add or remove peers from this set, as the observed rooms, or the peers within them, change. Rooms have very little memory overhead, and no CPU overhead unless they have members. They are created on demand when a member or observer requests them, and destroyed when the member and observer counts drop to zero.

Changes in membership or observership should be batched per client. This is because in a naive sequential implementation, a peer leaving a room and immediately observing it would see the other peers in that room disappear and then immediately reappear as each operation is actioned.

Each peer controls which room it is a member of, and which others it observes, based on its position. It does this by mapping locations to unique identifiers. The player position is considered to be the transform that defines the VR viewpoint. The server identifies rooms by RFC4122 V5 UUIDs [33]. V5 UUIDs are deterministic, allowing a local identifier to define a UUID within a namespace. Each shard or instance of the world has its own UUID which acts as the namespace. As the player moves, their occupying cell is determined, and its coordinates form the local identifier. Together these define a unique procedural room ID.

The partitioning of space is defined per-environment (that is, per-scene in Unity). The definition is a set of constants that define the procedural function and its parameters (for example, the size of a cell, and radius of the AOI). For our hexagonal partition we use an axial coordinate system [44], with the AOI radius specified in cells. Peers observe a ring of one cell in addition to their own. This methodology does not preclude manual partitioning. Many other partitioning schemes could be used, so long as they map a location to a GUID deterministically.

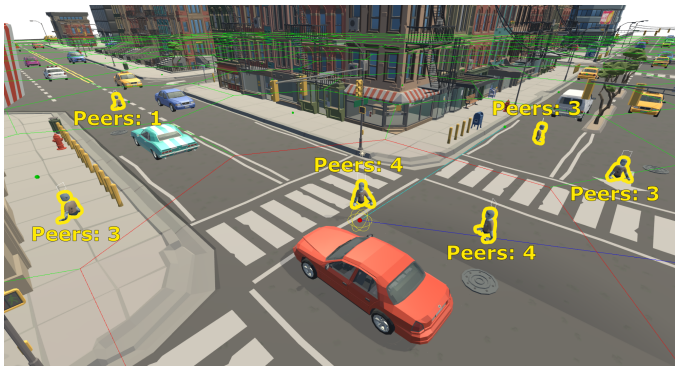

Fig. 2. Peers in a procedural (infinite) hex partition, imposed on a virtual city. The counts for each Peer are the number of Peers in their interest set. These include local and observed Peers in their cell and neighbouring cells, demonstrating the behaviour of the AOI partition.

B. K-Nearest Neighbour

We also consider a k-nearest neighbour (knn) partition. This defines the interest set based on relative player positions. The scheme is implemented on the server. All peers within each shard or instance have their own interest set, from which they receive messages. Interest sets may be asymmetrical. The server maintains the interest set for each peer. In our experiments, the server updates the sets at $5Hz$ and k is set to 20. Peers are notified of changes only when peers go in and out of scope.

This scheme requires that each player report its position in order to calculate the relative distances, compared to the spatial partition where each peer derived its own cells. The server must maintain two sets for each connected peer: the peers it observes, and those that observe it. The observed-by list determines where messages from that Peer are forwarded, which is not necessarily the same as the observed set. The $5Hz$ update rate limits how often peers can be added or removed, additionally, removing a peer is gated by a $500ms$ timeout. This prevents peers being added or removed at high frequency if they move around the limit of the K^{th} nearest distance.

Interest sets are calculated within a shard. In our prototype, the knn computations are calculated using brute force. This was to keep our implementation simple as at the peer counts considered we did not expect knn computation time to affect the results. If a server started to support 100's of peers or more in a shard, an acceleration structure would likely be required. This would be invisible to the client however.

V. Evaluation

We evaluate the partitionings by examining how they affect server capacity. The evaluation is performed for a single server. Performance is measured by latency, with the framerate

monitored to ensure clients do not become overloaded and skew the results. Latency is the time taken for two peers to exchange a message via the server. Framerate is the update rate of a peer's main loop. We look for changes in latency to detect when the server begins to become overloaded.

A. Bots

To get large user counts for the evaluation we created bots. A bot peer is a process that contains the same components as a user-facing application, but with their APIs driven procedurally, rather than via a user interface. The bots interact with the world in the same way as real users, and generate the same traffic. They move autonomously through the environment with use of a nav-mesh. Bots use the same avatars as real users, and can play and receive pre-recorded audio through synthetic audio devices. Bots and real users can join the same room and exchange RTC data. A single process can host multiple bots, but each bot is a separate peer, with its own connection to the server and state.

As we suspect user behaviour will significantly affect scalability schemes, we implement two types of Bot locomotive behaviour: Randomwalk and Boid. Randomwalk bots pick a location at random from their navigable area using Unity's *NavMesh* functionality, move to it at $3m/s$, and repeat indefinitely. A Randomwalk bot visits an average of 12 ± 2 rooms every 60 seconds. Boid bots implement flocking behaviour [63] based on the positions of the other Peers they can see. The experiments used a $200m^2$ area with no obstacles to avoid confounding environmental effects.

In addition to exchanging Avatar pose data, Bots can send messages of arbitrary size to each other each frame to emulate additional application specific traffic. We make both bot implementations available, along with the scalability scheme implementations.

B. Apparatus

We ran our experiment on a set of AWS (Amazon Web Services) t2.medium VMs. These VMs had 2 CPUs and 4GB memory. All VMs were in the same region. Bots were run on headless Unity processes. Each Unity process can host an arbitrary number of bots. Within a process, Bots are entirely independent - sharing a process only serves to reduce memory overhead. We found each VM could host five processes, each hosting two Bots, before encountering frame-rate drops. We had one server VM and ten bot VMs, making a total of 100 bots available per trial.

C. Protocol

After starting a new instance of the server, the Bot processes were commanded to add a new bot every 1.5 seconds in a round robin manner until 100 bots had been instantiated in total. The Bots were then destroyed all at once. The system was then left to return to a stable condition to ensure all logs had been written before the processes were destroyed. The data collected consisted of server throughput measurements, process-wide fps measurements, and the positions and ping times measured from each bot. The experiments were run automatically across a parallel communication system that was not affected by the Bot Peer traffic.

D. Capacity

We first try to identify the failure mode of the system. We create a room and add Randomwalk bots until the latency begins to increase.

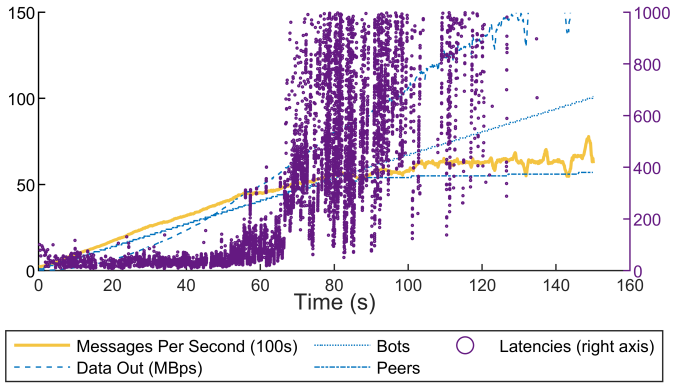

Fig. 3. Profile of the baseline capacity test. Each peer sent updates at 60 Hz with 500 byte padding.

Figure 3 shows the profile of this test in our baseline condition. This shows how the server fails. The messages per second and data out (left-axis) are measured at the server. The number of bots (same left-axis) is the total number of bots in the system at that time. The number of peers (same left-axis) is the number of bots that have joined the single room or shard. The latency scatter plot (right-axis, in milliseconds) shows all the latency measurements between all bots. In Figure 3 we see data out and messages per second increase approximately linearly with the number of bots at the start. The latency is stable, at less than $100ms$. At 55 seconds the latency begins to fluctuate, and by 70 seconds consistently reaches unacceptable levels above $400ms$. At the same time (70-80 seconds), the number of peers plateaus and deviates from the total number of bots in the system. This deviation occurs at the same time the message processing rate stops increasing. We can infer from this that at a certain load, the message processing rate limit is reached and messages begin to queue. So long as the traffic remains constant, the backlog will grow, until the server fails completely. However, performance degrades to unusable levels (latencies in the seconds) long before this happens.

To find out more about this limit, we change traffic patterns by varying the client update rate (10, 60, 50, 1000 Hz), and padding messages with arbitrary data (0, 500, 1500 Bytes). Table I shows metrics about server performance at the point the latency reaches $500ms$. **Peers** shows the capacity of the server at this time, for the different combinations of Update Rate and Padding (message size). There is an inverse relationship with Update Rate, and a slightly less obvious relationship with MPS. Row five corresponds to the baseline shown in Figure V-D.

There is no single acceptable latency in an SVR system as tolerable latency depends on the modality. All profiles begin

with latencies < 100 *ms* and all were tested to failure. Therefore 500*ms* is an arbitrary figure that reliably indicates that the server is beginning to fail. Table I therefore shows the maximum capacity of the server under different use cases (update rates, message size). Bots are the number of live processes, and Peers are the number of successfully connected bots. When Peers lags Bots, it means the server is becoming unresponsive. Update Rate is an independent variable. Gain is the bytes out per input message. It varies with the padding and approximates the 'total workload' of the server. In & Out combined show the overall bandwidth for the server. The dependent variable of interest is the number of Peers, as this shows the capacity right before failure. The bottom row shows the Pearson Correlation Coefficient of the each column with Peers.

Update Rate is the strongest predictor (of capacity) of all. Gain and Padding are weak predictors suggesting that it is the number of messages, and not their size, which is the limiting factor. *MPS* is the total number of messages processed per second. As shown by Figure 3, this limit determines the capacity of the server. However, we can see that limit it is not constant between trials. The variation suggests something dependent on the peer count, possibly the fan-out, affects how efficiently the server runs.

Bots	**Peers**	Update Rate	Padding	MPS	In (MB/s)	Out (MB/s)	Gain
48	48	10	0	1059	0.07	2.98	2952
48	48	10	500	1677	0.38	18.18	11370
48	48	10	1500	1416	0.82	39.42	29194
39	39	60	0	3752	0.18	6.92	1934
36	**36**	**60**	**500**	**3255**	**0.89**	**32.99**	**10629**
39	29	60	1500	4445	3.07	119.53	28200
20	19	500	0	9138	0.45	8.99	1031
24	20	500	500	9110	2.52	60.45	6958
16	11	500	1500	10131	7.61	129.34	13388
19	19	1000	0	9425	0.45	8.99	1000
20	17	1000	500	9369	2.58	54.04	6048
18	8	1000	1500	1760	1.12	25.87	15413
r		0.85	0.22	0.71	0.52	0.37	0.17

TABLE I
SERVER PERFORMANCE METRICS AT THE POINT PEER TO PEER LATENCIES EXCEEDED AN AVERAGE OF 500*ms*. **PEERS** IS THE SERVER CAPACITY AT THIS TIME (HIGHER IS BETTER). ROW *r* IS THE PEARSON CORRELATION COEFFICIENT OF EACH METRIC WITH *Peers*.

1) Hex Grid Partition: The first scheme evaluated was the Hex grid partition. Figure 4 shows a profile of the simulation with Randomwalk Bots. We can see the messages per second and peer-to-peer latencies remain low until over 100 bots were created. This demonstrates the scheme does allow the server to scale. However, only in certain conditions. This plot also shows how the room count increases with the number of bots. The server connects to 100 bots but the spatial distribution means peer-to-peer traffic remains low.

Figure 5 shows the same simulation but using Boids Bots. As these bots flock to the same location the server capacity is reached far earlier, being only minimally higher than in our baseline tests. Figure 9 shows an example of the distribution after one minute for two Hex trials. We see that almost all Boids are gathered within one cell, whereas a number of Randomwalk bots are beyond the limits of the plot. The differences are

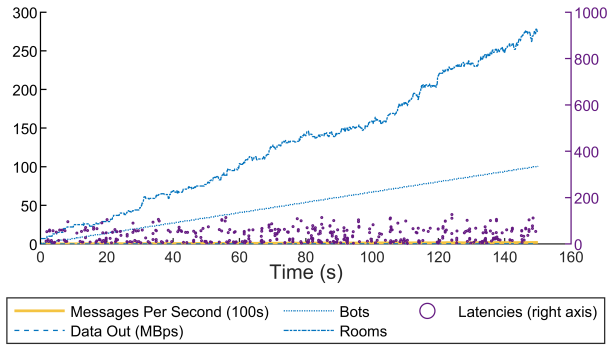

Fig. 4. Profile of the Hex Randomwalk trial. The latency remains low while the room count increases as the Bots spread out.

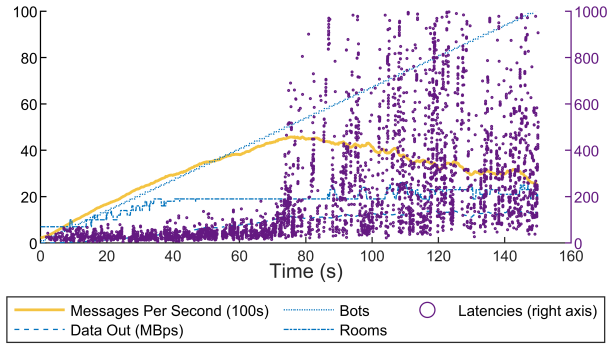

Fig. 5. Profile of the Hex Boids trial. This looks almost identical to the Baseline as the Boids flock to the center of the environment.

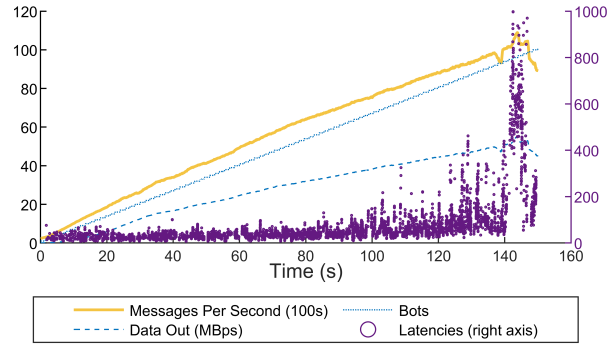

Fig. 6. Profile of the Knn Randomwalk trial. This demonstrates a much higher capacity than the Hex partition despite the increased computational server load.

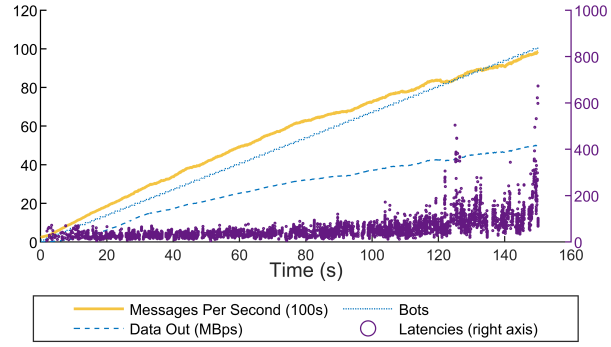

Fig. 7. Profile of the Knn Boids trial. This again is markedly different to the Hex trials, as the capacity is less effected by bot behaviour.

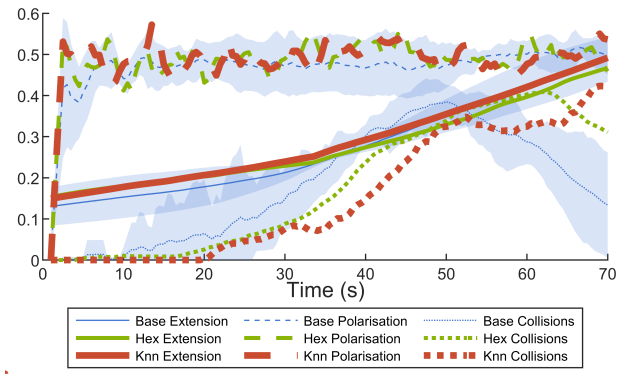

Fig. 8. Comparing objective measures of the Boids flocks when in the Baseline, Hex or Knn conditions. Values are normalised: polarisation to angle, extension to 200 m and collisions to the flock size.

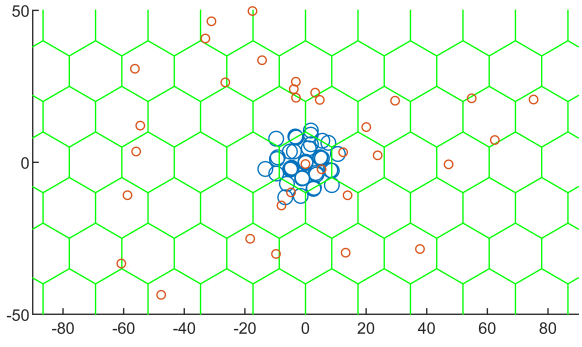

Fig. 9. The position of the Bots after 60 seconds, for both the Boids (large markers) and Randomwalk (small markers) Hex Trials. The Boids bots cluster around the center cell.

informative, as they provide evidence the limit is in the fan-out stage, and the server process itself can handle more than 100 connections so long as they are in sufficiently small groups.

We see bot behaviour affect server capacity, but another question is whether this dependency is two-way. Figure 8 shows three measures of flock behaviour: *extension* is the average deviation from the center of the flock, *polarisation* is how closely aligned the orientation of the boids are, and *collisions* are the proportion of Boids in a collision state at a given time [63]. For the Baseline condition, the plot shows the mean and 99% confidence intervals across 7 identical trials. The Hex grid and Knn show one trial each. We can see the Hex grid does not affect behaviour for the most part. This is not surprising as the server shows similar performance in both cases, due to the flocking behaviour keeping all Bots predominantly within a handful of cells (rooms).

2) K-Nearest Neighbour: The second scheme evaluated was the Knn partition. Figure 6 shows the profile of the Randomwalk trial and Figure 7 for the Boids. We can see a notable increase in capacity as the latencies only begin to change towards the end of the trials. This is about 70-80 bots or double the Baseline. We can also see that the threshold is lower, and MPS higher, for the Randomwalk compared to Boids. This is likely because the flocking behaviour results in more consistent interest sets over time. The differences however are small, especially compared with the MPS of the Baseline condition.

Considering behaviour, we can see the Knn flock exceeds the expected range of the measures more often than the Hex flock. Importantly though, while the *extension* measure is a little higher than the mean (and hence the collisions measure is lower), both remain within the confidence interval.

VI. Discussion

Our ultimate motivation is to improve the capacity of Ubiq in order to perform experiments with large groups in SVR. Based on our preliminary tests we hypothesised that the system bottleneck was the server fan-out. In this respect both schemes were successful as they allowed the server to handle more peers than during the baseline tests. They also confirmed that it is the fan-out workload (as opposed to say, connection count overhead) that is the limiting factor.

A. Performance

Our baseline evaluation showed a single server could support approximately 40 peers. Using the Hex grid with a radius of 10m, this increased to 100 Randomwalk peers (our maximum Bot count) with no sign of degrading. The Knn scheme increased capacity to 70-80 peers before latency began to increase. The dominant factor in the message processing limit appears to be message overhead (that is, the number of copies), rather than size. This is suggested by the weakness of size as a performance predictor. By creating interest sets, the fan-out factor is reduced increasing the overall message rate of the server, and thus user capacity.

The k of 20 is not arbitrary, but is roughly the limit on peer-to-peer voice channels we've observed empirically on a number of platforms, including Mozilla Hubs, RecRoom and Ubiq (Roblox has a similar hard limit of 30). This figure likely comes from the CPU overhead of WebRTC, the library/standard that powers the voice chat of many SVR applications.

B. Behaviour

However, both schemes were affected by Peer behaviour. The Knn scheme was far less susceptible. This is because the interest sets have a constant size regardless of the total Peer count. To see the effect of the *scheme* on Peer behaviour, we also instrument the Bots, measuring characteristics of the flock over time.

These results showed minimal deviations from the expected Baseline behaviour. This was not surprising for the Hex conditions due to the *extension* being roughly equivalent to one cell. It was surprising for the Knn condition however. With a k of 20, at the end of each trial each Bot would see only 1/5th of the flock, yet behaved almost identically to when it could see all of them. As interest sets are asymmetrical what we are observing are bots influencing each other indirectly. Figure 10 shows the centroids of 100 randomly selected subgroups of the Boids Knn trial at 60 seconds, along with the *extension* plotted as a circle around them. What we see from this is that

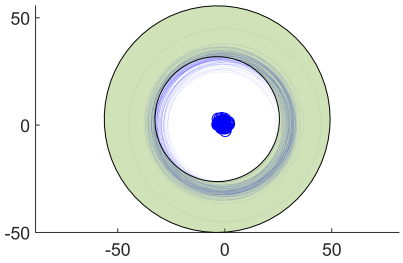

Fig. 10. The extension measure of the randomly selected subgroups in the Boids Knn Trials, compared to the 99% CIs from the Boids Baseline.

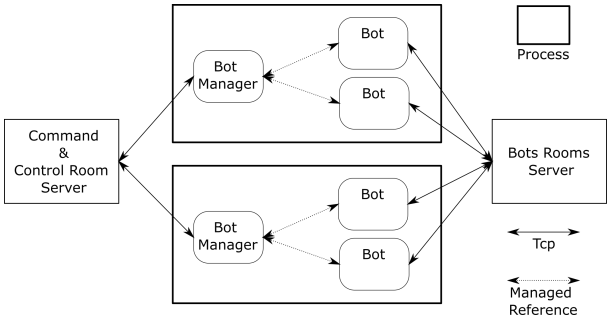

Fig. 11. Connection architecture of the experiment, showing two separate Ubiq Peer networks communicating via references within common processes.

each subgroup has similar characteristics as each other, and the larger flock, irrespective of the group composition. The largest deviations are seen in the polarisation measure.

It is interesting to see indirect effects in the Knn scheme, and verify neither scheme undermines bot behaviour. The lessons from the measures however are limited in that it is not easy to consider them objectively in the same context as real user behaviour. We are still left to question whether the three metrics are not particularly sensitive, or the schemes are particularly robust to this behaviour. If we were to change the cell size, k, or the maximum speed of the Bots, we may see different flock behaviour emerge. In future work however we feel it would be more worthwhile to consider a larger range of virtual user behaviour, such as crowd simulations or data-driven behaviour from real user captures.

C. Integration

Both schemes use only player position, independent of the environment. The fixed partition scheme performs its work on the clients. The knn scheme requires more computational effort from the server. We performed our investigation in Ubiq, which is highly client-centric. We had to modify both the client and server. To support observing cells we had to modify the messaging schema and the server behaviour to support membership and observation. To support the Knn scheme, we subclassed the type that handled within-room fan-out and modified this to maintain the observed and observed-by sets. We used a general purpose dictionary feature of Ubiq that involved sending deltas to update the user position, so no schema changes were required. On the client side we created new Unity Components to drive the Rooms API. The schemes had different levels of invasive changes. We feel Ubiq performed moderately well as a platform for exploring scalability schemes. The Knn support could be applied without upstream changes, however cell observation could not and required forking.

To manage the Bots we use Ubiq as well, with an entirely parallel Peer network (Figure 11). Each Unity process can host one or more Ubiq Peers. In the Bot processes, one of these Peers had a Component to manage the bots within the process. The other Peers were Bots themselves. The two types of Peer connected to different servers over different connections, but have a form of communication through direct references in a shared memory space. We found this to work very well, and the ability of Ubiq to function as the control structure and subject of the experiment at the same time is an unusual capability.

D. Generalisability and Limitations

A challenge of social VR is that systems implementation cannot be decoupled from the use case. Technical decisions derive from the requirements to share certain information with a certain QoS, and these depend on what cues are exchanged in what social contexts [3]. Therefore there is no objectively optimal scalability scheme(s) for all systems. The schemes considered here are specific to the research and teaching use cases of Ubiq, with associated constraints. For example, we cannot take advantage of environmental occluders as done in Steed and Angus [54]. Our results are meant to be considered alongside similar experiments on scalability schemes for SVRs.

Our results should generalise to any system that uses server fan-out, but not to those using peer-to-peer unicasting. Our experiments are message-content agnostic. In a real system, cues such as audio or avatar transforms will likely use aggregation schemes. Our figures do not directly indicate the user capacity in this case, but are applicable to the message capacity of the system outside the aggregation.

The biggest limitation in our experiments is the behaviour of the bots. By using two types of locomotion we have shown how schemes can be dependent on user behaviour. Neither type of bot reflects real user behaviour however but we are not aware of a general purpose simulator either. We expect that such a simulation would be very difficult to create, due to the dependence of user behaviour on environment and context. Accordingly, we expect as work in this area becomes more advanced, it will become more narrow in its focus and more application-specific. This suggests a need for flexible and accessible simulation platforms such as we have used here.

VII. Conclusion

In this paper we investigate scalability schemes in the open-source SVR Ubiq. Social VR covers a broad range of systems that share a common subset of functionality. In naive many-to-many arrangements, resources scale geometrically with user count, quickly exhausting the limits of real systems. Accordingly, scalability schemes are used to improve

capacity. However these are rarely discussed publicly, and their application specific nature makes it difficult to generalise the results we do have.

Our ultimate aim is to support collective-scale social experiences, including large crowds. The first step is to move from small to medium gatherings. This involves creating interest sets that are manageable by Ubiq's client and relay server, which can eventually be organised into a hierarchy for scaling at higher levels.

In this paper we explored two schemes: a fixed spatial partition and knn partition. Each scheme is procedural, supporting infinitely vast worlds. Both schemes are conceptually simple and have only one or two tuneable parameters. Both schemes successfully improve server capacity - with caveats. The knn scheme doubles capacity from 40 to 80 peers. The hexagonal grid offers an improvement to over 100, but only when the peers are spatially well distributed. While both schemes are effective, they differ in how they interact with other aspects of the SVR, and user behaviour. The knn scheme requires the server to actively maintain interest sets. Maintaining the hex grid groups is distributed between the peers. The server is more passive in this scheme to the extent it could be replaced with network-level multicasting. The partition configuration depends on the expected social situation however, and requires a partition to be set for each scene, whereas the server based knn is transparent to the user and may better generalise to different situations.

Ubiq allowed both schemes to be implemented with minimal difficulty, while also functioning as the command and control system for the experiment. Surprisingly, it was the more server-heavy Knn scheme that was easiest to implement. This is because constraining the workload to the server meant the peer-server communication could be simpler, compared to peer-maintained area-of-interests.

All the extensions to Ubiq described in this paper will be made open source as part of Ubiq in a separate branch. As well as the code availability we host example Ubiq relay servers for testing (see the Ubiq documentation at https://github.com/UCL-VR/ubiq). Further, as all the code is available researchers can extend the system and implement a broader variety of mechanisms. While Ubiq already supports multiple independent servers, it would be interesting to explore inter-server mechanisms. We believe that the developments to Ubiq presented herein offer a useful tool for the community to explore SVRs in more depth.

REFERENCES

[1] J. Allard, V. Gouranton, L. Lecointre, S. Limet, E. Melin, B. Raffin, and S. Robert. FlowVR: A Middleware for Large Scale Virtual Reality Applications. In *Parallel Processing. 10th International Euro-Par Conference*, pp. 497–505. Springer Berlin Heidelberg, 2004. doi: 10.1007/978-3-540-27866-5_65

[2] H. Backhaus and S. Krause. Voronoi-based adaptive scalable transfer revisited. *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games - NetGames '07*, pp. 49–54, 2007. doi: 10.1145/1326257.1326266

[3] C. Bassetti. Social Interaction in Temporary Gatherings. In *Group and Crowd Behavior for Computer Vision*, pp. 15–28. Elsevier, 1 ed., 2017. doi: 10.1016/B978-0-12-809276-7.00003-5

[4] C. Blanchard, S. Burgess, Y. Harvill, J. Lanier, A. Lasko, M. Oberman, and M. Teitel. Reality Built for Two: A Virtual Reality Tool. In *Proceedings of the 1990 symposium on Interactive 3D graphics*, I3D '90, pp. 35–36. Association for Computing Machinery, New York, NY, USA, Feb. 1990.

[5] P. Boustead and F. Safaei. Comparison of delivery architectures for immersive audio in crowded networked games. pp. 22–27, 2004. doi: 10.1145/1005847.1005854

[6] P. Boustead, F. Safaei, and M. Dowlatshahi. DICE: Internet delivery of immersive voice communication for crowded virtual spaces. *Proceedings - IEEE Virtual Reality*, 2005:35–41, 2005. doi: 10.1109/vr.2005.29

[7] E. Buyukkaya, M. Abdallah, and G. Simon. A survey of peer-to-peer overlay approaches for networked virtual environments. *Peer-to-Peer Networking and Applications*, 8(2):276–300, 2013. doi: 10.1007/s12083-013-0231-5

[8] C. Carlsson and O. Hagsand. DIVE A Multi-User Virtual Reality System. pp. 394–400, 1993. doi: 10.1109/VRAIS.1993.380753

[9] CCP. Introducing time dilation (tidi). https://www.eveonline.com/news/view/introducing-time-dilation-tidi, 2011.

[10] R. Chertov and S. Fahmy. Optimistic load balancing in a distributed virtual environment. In *Proceedings of the 2006 international workshop on Network and operating systems support for digital audio and video - NOSSDAV '06*. ACM Press, New York, New York, USA, 2006. doi: 10.1145/1378191.1378208

[11] E. F. Churchill and D. Snowdon. Collaborative Virtual Environments: An Introductory Review of Issues and Systems. *Virtual Reality*, 3(1):3–15, Mar. 1998.

[12] B. Damer. *Avatars!; Exploring and Building Virtual Worlds on the Internet*. Peachpit Press, 1997.

[13] P. Dickinson, K. Gerling, K. Hicks, J. Murray, J. Shearer, and J. Greenwood. Virtual reality crowd simulation: effects of agent density on user experience and behaviour. *Virtual Reality*, 23(1):19–32, 2019. doi: 10.1007/s10055-018-0365-0

[14] F. Drolet, M. Mokhtari, F. Bernier, and D. Laurendeau. A Software Architecture for Sharing Distributed Virtual Worlds. In *Proceedings of the 2009 IEEE Virtual Reality Conference*, pp. 271–272. IEEE, 3 2009. doi: 10.1109/VR.2009.4811050

[15] Facebook. Horizon Venues, 2022.

[16] I. Farkas, D. Helbing, and T. Vicsek. Mexican waves in an excitable medium. *Nature*, 419(6903):131–132, sep 2002. doi: 10.1038/419131a

[17] S. Friston, B. Congdon, D. Swapp, L. Izzouzi, K. Brandstätter, D. Archer, O. Olkkonen, F. J. Thiel, and A. Steed. *UBiQ: A system to build flexible social virtual reality experiences*, vol. 1. Association for Computing Machinery, 2021. doi: 10.1145/3489849.3489871

[18] S. J. Friston, E. Griffith, D. Swapp, S. Julier, I. C. Irondi, F. Jjunju, R. Ward, A. Marshall, and A. Steed. Consensus Based Networking of Distributed Virtual Environments. *IEEE Transactions on Visualization and Computer Graphics*, 1:1–1, 2021. doi: 10.1109/TVCG.2021.3052580

[19] T. Funkhouser. Network topologies for scalable multi-user virtual environments. 1996.

[20] E. Goffman. *Behavior in Public Places: Notes on the Social Organization of Gatherings*. Free Press of Glencoe, 1964.

[21] C. Greenhalgh and S. Benford. MASSIVE. *ACM Transactions on Computer-Human Interaction*, 2(3):239–261, 9 1995. doi: 10.1145/210079.210088

[22] I. J. Grimstead, D. W. Walker, and N. J. Avis. Collaborative Visualization: A Review and Taxonomy. pp. 61–69, 2005. doi: 10.1109/DISTRA.2005.12

[23] G. Gutmann and A. Konagaya. Predictive Simulation: Using Regression and Artificial Neural Networks to Negate Latency in Networked Interactive Virtual Reality. *arXiv*, oct 2019.

[24] A. P. Hare. Group Size. *American Behavioral Scientist*, 24(5):695–708, 1981. doi: 10.1177/000276428102400507

[25] J. E. Hocking. Sports and Spectators: Intra-audience Effects. *Journal of Communication*, 32(1):100–108, mar 1982. doi: 10.1111/j.1460-2466.1982.tb00481.x

[26] T. Iimura, H. Hazeyama, and Y. Kadobayashi. Zoned federation of game servers: A peer-to-peer approach to scalable multi-player online games. In *Proceedings of the ACM SIGCOMM Workshop on Network and System Support for Games, NetGames'04*, pp. 116–120, 2004. doi: 10.1145/1016540.1016549

[27] IMS. Networking for high-scale multiplayer gameplay. https://ims.improbable.io/products/spatialos, 2021.

[28] A. Kendon. Goffman's approach to face-to-face interaction. In *Goffman: exploring the interaction order*, pp. 14–40. Cambridge: Polity Press, 1988.

[29] A. Kendon. Spacing and Orientation in Co-present Interaction. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5967:1–15, 2010.

[30] D. Lake, M. Bowman, and H. Liu. Distributed Scene Graph to Enable Thousands of Interacting Users in a Virtual Environment. In *9th Annual Workshop on Network and Systems Support for Games*, pp. 1–6. IEEE, 11 2010. doi: 10.1109/NETGAMES.2010.5679669

[31] M. E. Latoschik, C. Fröhlich, and A. Wendler. Scene Synchronization in Close Coupled World Representations using SCIVE. *The International Journal of Virtual Reality*, 5(3):47–52, 2006.

[32] M. E. Latoschik, F. Kern, J. P. Stauffert, A. Bartl, M. Botsch, and J. L. Lugrin. Not Alone Here?! Scalability and User Experience of Embodied Ambient Crowds in Distributed Social Virtual Reality. *IEEE Transactions on Visualization and Computer Graphics*, 25(5):2134–2144, 2019. doi: 10.1109/TVCG.2019.2899250

[33] P. J. Leach, R. Salz, and M. H. Mealling. A Universally Unique IDentifier (UUID) URN Namespace. RFC 4122, July 2005. doi: 10.17487/RFC4122

[34] Linden Lab. SecondLife. https://secondlife.com/, 2021.

[35] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz. NPSNET:A Network Software Architecture for LargeScale Virtual Environments. *Presence: Teleoperators and Virtual Environments*, 3(4):265–287, 1994. doi: 10.1162/pres.1994.3.4.265

[36] M. R. Macedonia, M. J. Zyda, D. R. Pratt, D. P. Brutzman, and P. T. Barham. Exploiting reality with multicast groups. *IEEE Computer Graphics and Applications*, 15(5):38–45, 1995. doi: 10.1109/38.403826

[37] G. Mantovani. Virtual Reality as a Communication Environment: Consensual Hallucination, Fiction, and Possible Selves. *Human Relations*, 48(6):669–683, 1995.

[38] Microsoft. AltspaceVR. https://altvr.com, 2021.

[39] Microsoft. Scaling your audiences with frontrow feature. https://docs.microsoft.com/en-us/windows/mixed-reality/altspace-vr/faqs/scaling-audiences, 2021.

[40] F. Moustafa and A. Steed. A longitudinal study of small group interaction in social virtual reality. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology, VRST*, 2018. doi: 10.1145/3281505.3281527

[41] Mozilla Foundation. Mozilla Hubs, 2021.

[42] M. Naef, E. Lamboray, O. Staadt, and M. Gross. The Blue-C Distributed Scene Graph. In *Proceedings of the 2003 IEEE Virtual Reality Conference*, pp. 275–276. IEEE Comput. Soc, 2003. doi: 10.1109/VR.2003.1191157

[43] C. D. Nguyen, F. Safaei, and P. Boustead. A distributed proxy system for provisioning immersive audio communication to massively multi-player games. In *Proceedings of the ACM SIGCOMM Workshop on Network and System Support for Games, NetGames'04*, p. 166, 2004. doi: 10.1145/1016540.1016561

[44] A. Patel. Hexagonal Grids, 2020.

[45] N. Pelechano, C. Stocker, J. Allbeck, and N. Badler. Being a part of the crowd: Towards validating VR crowds using presence. *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 1:135–141, 2008.

[46] Photon. Load balancing. https://doc.photonengine.com/zh-CN/realtime/current/troubleshooting/faq, 2021.

[47] M. Radenkovic, C. Greenhalgh, and S. Benford. Deployment issues for multi-user audio support in CVEs. *ACM Symposium on Virtual Reality Software and Technology, Proceedings, VRST*, pp. 179–185, 2002. doi: 10.1145/585767.585770

[48] Rec Room Inc. Rec Room, 2021.

[49] Roblox. Spatial voice. https://developer.roblox.com/en-us/articles/spatial-voice, 2021.

[50] Roblox Wiki. Mega place. https://roblox.fandom.com/wiki/MEGA_Place, 2021.

[51] M. Roth, G. Voss, and D. Reiners. Multi-Threading and Clustering for Scene Graph Systems. *Computers and Graphics (Pergamon)*, 28(1):63–66, 2004. doi: 10.1016/j.cag.2003.10.004

[52] R. Schulz. Comprehensive List of Social VR Platforms and Virtual Worlds, 2020.

[53] S. Singhal and M. Zyda. *Networked Virtual Environments: Design and Implementation*. Addison Wesley, Reading, MA, Aug. 1999.

[54] A. Steed and C. Angus. Supporting scalable peer to peer virtual environments using frontier sets. In *IEEE Proceedings. VR 2005. Virtual Reality, 2005.*, vol. 2005, pp. 27–34, 2005. doi: 10.1109/VR.2005.1492750

[55] A. Steed and M. F. Oliveira. *Networked Graphics: Building Networked Games and Virtual Environments*. Elsevier, Oct. 2009. Google-Books-ID: 76C_quJqVXcC.

[56] V. E. Stone. Social Interaction and Social Development in Virtual Environments. *Presence: Teleoperators and Virtual Environments*, 2(2):153–161, 1993.

[57] M. Terrano and P. Bettner. 1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond, 2001.

[58] Virbela. With scalability, close to 1k attend a virtual auditorium conference with more possible. https://www.virbela.com/blog/916-avatars-in-a-virtual-world, 2019.

[59] Z. Wang, X. Jiang, and J. Shi. HIVE: a highly scalable framework for DVE. *IEEE Virtual Reality 2004*, pp. 261–262, 2004. doi: 10.1109/VR.2004.1310099

[60] R. Wolff, D. J. Roberts, and O. Otto. A Study of Event Traffic During the Shared Manipulation of Objects Within a Collaborative Virtual Environment. *Presence: Teleoperators and Virtual Environments*, 13(3):251–262, jun 2004. doi: 10.1162/1054746041422280

[61] XR Ignite. Interactive Directory of XR Collaboration Platforms, 2021.

[62] H. Yakura and M. Goto. Enhancing Participation Experience in VR Live Concerts by Improving Motions of Virtual Audience Avatars. In *Proceedings - 2020 IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2020*, pp. 555–565, 2020. doi: 10.1109/ISMAR50242.2020.00083

[63] J. L. Zapotecatl, A. Muoz-Melndez, and C. Gershenson. Performance Metrics of Collective Coordinated Motion in Flocks. In *Proceedings of the Artificial Life Conference 2016*, pp. 322–329. MIT Press, Cancun, Mexico, 2016. doi: 10.7551/978-0-262-33936-0-ch054