Transforming Minecraft into a Research Platform

Herman A. Engelbrecht
MIH Media Lab
Stellenbosch University, Stellenbosch, South Africa
Email: hebrecht@ml.sun.ac.za

Gregor Schiele
DERI, National University of Ireland
Galway, Ireland
Email: gregor.schiele@deri.org

***Abstract*—Experimental evaluation of MMVE research is mostly based on simulation. To convince industry of the advantages of alternate MMVE architectures requires experimental results obtained from using a commercially deployed MMVE, such as Minecraft. In this paper we describe the transformation of Minecraft into a flexible research platform, that allows researchers to modify Minecraft on different layers and functional areas. As proof-of-concept we replace Minecraft's client/server architecture with a distributed server cluster. The Minecraft virtual environment is distributed amongst the server cluster, allowing for migration of users between nodes of the cluster. This is achieved without modifying the Minecraft client and with minimal modification of the server.**

I. Introduction

Commercial Massively Multi-user Virtual Environments (MMVEs) predominantly use Client/Server (C/S) and Client/Distributed Server (C/DS) architectures [1]. Although such architectures allow for the provisioning of sufficient resources for hosting an MMVE at a central point, they do not scale indefinitely with the number of simultaneous users. Knutsson sparked research into the field of peer-to-peer (P2P) architectures for MMVEs, by eliminating the need for a centralised server and distributing the workload amongst all peers that participate [2]. However, P2P architectures so far have not been adopted by industry, as a result of a number of unsolved challenges (such as availability, persistence and cheating) [3]. Evaluating the performance of MMVE techniques, using real-world experiments, requires the development of a custom MMVE from scratch, or the modification of an existing MMVE, as well as a large user base. Developing an MMVE is expensive and time-consuming, while modifying an existing MMVE implies access to the software code, which is difficult to obtain from commercial entities. Minecraft is a popular MMVE that allows for the modification (but not redistribution) of the client and server software. Server plugins can be developed for Minecraft and may be redistributed. In this paper we present Koekepan, a system that allows us to transform Minecraft into a platform for MMVE research. Such a platform allows MMVE researchers to leverage the large Minecraft community when performing experiments using a real MMVE, without the effort of developing an MMVE from scratch. To ensure compatibility with the existing Minecraft client, we require that our architecture not modify the client software, and that modifications to the Minecraft server code be kept to a minimum.

This paper is organised as follows: Section II discusses MMVE platforms that have been used for real-world experiments, followed by some Minecraft background in Section III. Section IV proposes and explains the flexibility of the Koekepan architecture. Sections V and VI detail our proof-of-concept research: replacing Minecraft's C/S architecture with a distributed server architecture and distributing the virtual environment (VE). Our progress with implementing the proof-of-concept is detailed in Section VII. We conclude and discuss future work in Section VIII.

II. Related Work

Various architectures have been proposed for MMVEs that all attempt to address scalability by partitioning and distributing either the VE or the users. A number of research platforms that use a real MMVE have been developed. Bharambe et. al developed Donnybrook, using the popular First-Person Shooter (FPS) Quake 3, that uses a P2P architecture and mainly focuses on limiting updates sent between peers, while not allowing modification of the VE [4]. P2P SecondLife replaced the C/S architecture of SecondLife (SL) with a P2P architecture based on Kad and investigated the consistency, persistence and scalability of P2P SL [5]. They found that users experienced latency as a result of peers joining the P2P architecture and avatars crossing borders between regions. Although the platform was based on a real MMVE, the experimental evaluation was performed using emulated clients using object and avatar traces collected from SL. OpenSimulator [6] is an open source MMVE framework that is most related to our work. The framework is designed to be easily extensible, can be used to simulate virtual environments similar to SL, and can be accessed with the regular SL clients. OpenSimulator is built on the premise that the VE is partitioned into regions using zoning [7]. Our Minecraft research platform aims to allow researchers more flexibility in configuring the MMVE network architecture and distributing the VE, and we plan to publish Koekepan as open source software.

III. Background

A. System Model

Our system model consists of a number of users that want to participate in an MMVE, the server infrastructure hosting the MMVE, the network infrastructure that allows for communication between the users and the MMVE, and the MMVE software. The MMVE software is divided into client software and server software. The number of users is unknown beforehand and changes dynamically as users join and leave the MMVE. Each user has a machine that executes the client software. The server software is executed by the server infrastructure that, together with the user machines, are connected to the network infrastructure. Users are represented by avatars

in the MMVE and the avatars can perform various activities in the MMVE, such as moving around, interacting and moving objects, as well as interacting with server-controlled entities called non-player entities (NPEs). The state of the MMVE is distributed on the server infrastructure although the client software has a local copy of a portion of the MMVE (the virtual environment in the immediate vicinity of the user's avatar). When a user initiates an activity, an event is generated and sent to the server software. The server software validates the event, generates state updates and disseminates these state updates to all clients affected by the change in the MMVE. Communication between the server and client is limited to state updates regarding entities in the avatar's *area of interest* (AoI). An important consideration is that for MMVEs, the latency between event generation and the client applying the received state update should be as little as possible.

B. Minecraft Background

Minecraft is a sandbox construction game [8]. The game involves players placing and breaking various types of blocks in a three-dimensional environment that consists completely of blocks. The player controls an avatar that can break and place blocks on multi-player servers and single-player worlds across multiple game modes. The Minecraft VE is divided into chunks, which are the smallest data structure used to store and distribute the landscape of the VE. Chunks are columns with a length and width of 16 blocks and a height of 256 blocks. Minecraft differs from most commercial MMOGs since the whole VE is modifiable by the users. It uses a C/S architecture that currently cannot utilise a server cluster, although it can use multiple cores [8]. The communication protocol of Minecraft is well-known and documented [9]. The Minecraft server uses internally consistent, unique serials for all virtual entities. To ensure that the state of the client and server is kept consistent, events and updates sent between them use the entity serials as generated by the server. Minecraft is essentially a discrete event simulator that processes events and updates the MMVE state every server tick. It attempts to maintain a fixed rate of 20 server ticks/second, but the rate may decrease if the server becomes overloaded. A Minecraft server maintains virtual server time i.t.o. the ticks since the VE was initially created.

C. Minecraft Server Clone: Bukkit

The Minecraft software is written in Java, which has enabled Minecraft enthusiasts to decompile the software, and has led to the establishment of a large "modding" community. A number of open source server clones have been created that are mostly compatible with the native Minecraft client [8]. One of the more popular and mature server clones is Bukkit [10]. The Bukkit API allows for the development of separate third-party server plugins. However, the focus is on modifying and extending the VE by introducing event-based software hooks into the native Minecraft server code that allow plugins to be notified of events processed by the Minecraft server (such as users connecting/disconnecting, entities being created/destroyed, blocks being placed/broken).

IV. Koekepan Architecture

Fig. 1 illustrates the architecture of *Koekepan*, our proposed system used to transform Minecraft into a research

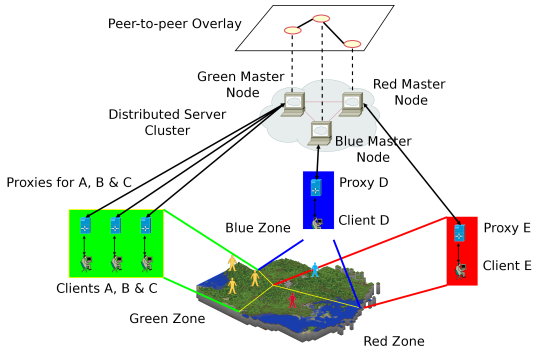

Fig. 1. Network architecture

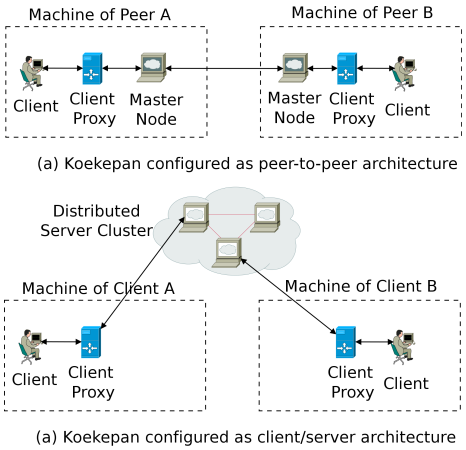

Fig. 2. Koekepan architecture configurations

platform. Our architecture is similar to other proposed distributed architectures [7], [11], [12], and consists of four software components: the Minecraft server clone (Bukkit), the unmodified Minecraft client, a client proxy and Augeo, a server plugin for Bukkit. In Fig. 1 Bukkit and Augeo are both executed on each node of the distributed server cluster, while the client and proxy are executed on separate machines. The components of Koekepan allow for the modification of Minecraft on different layers and functional areas. The proxy gives full control of the networking layer. For example, it allows for re-routing of network traffic between the server and client, for rewriting of packets, and for bridging of the Minecraft protocol. On the application layer, the server plugin gives a convenient abstraction if a researcher wants to replace a specific functional area of the MMVE, such as the migration strategy, the AoI management, or the partitioning of the VE. The aim of Koekepan is to provide a more generic framework and support functions so that the researcher need focus only on the functional area being replaced.

As a proof-of-concept, we replace the single server with a distributed server cluster, and distribute the hosting of Minecraft amongst the server cluster nodes. To effectively balance the load of the distributed server, migration of client connections between server nodes must be possible, therefore clients connect via proxies. Although we specifically focus on creating a distributed server architecture, the Koekepan architecture can also be configured as P2P or hybrid architectures. Fig. 2(a) shows that, by executing the client, proxy and Augeo-

enabled Minecraft server on the same machine, a P2P MMVE architecture can be emulated. Fig. 2(b) shows how a distributed server architecture can be emulated, by executing the client and Augeo-enabled Minecraft server on different machines. Koekepan's flexibility means researchers using it to perform real-world experiments, are not limited to a specific MMVE architecture. Augeo and the proxy also allow researchers to log activities and messages on all the different layers of the architecture, which is useful for evaluations.

Two aspects need addressing when modifying the Minecraft C/S architecture: (1) on the network layer: a distributed server cluster needs to be created and managed and (2) on the application layer: the VE must be partitioned and distributed. Augeo is responsible for both of these aspects while the client proxy re-routes server/client communication. We first discuss the creation of the distributed server cluster and then explain the distribution of the VE.

V. Managing the Distributed Server Cluster

A. Server network topology

We have designed Augeo for use with on-demand computing. To allow for server nodes to be added/removed from the multi cluster, we propose that the server cluster use a self-organising P2P overlay. Nodes can be added to the cluster, as long as they can join the peer-to-peer overlay.

Joining the server network overlay: One of the nodes, designated the boot node, is used as the entry point for the rest of the servers to join the P2P overlay. The IP address of the boot node is known beforehand by all servers that join the overlay. The boot node assists new nodes joining the cluster in discovering their neighbouring nodes (thus joining the overlay). Knowledge of its neighbours allow a node to establish direct communication with any other node that is part of the P2P overlay.

Establishing inter-node communication: Once a node has joined the overlay, it can establish a direct socket connection with any other node in the overlay. Using direct socket connections reduces unnecessary communication latency. Successfully established connections are used for bidirectional communication by sending overlay packets. We encapsulate the Minecraft protocol with an overlay protocol so that additional control packets can be exchanged. As shown in Fig. 3, each overlay packet contains the packet identification number, the total packet length, the payload data representing the encapsulated Minecraft packet, as well as a time stamp and sequence number. Since transmitted packets may experience varying transmission latency, the time stamp and sequence number is required for processing received packets in the correct order.

Leaving the server overlay: A node can gracefully disconnect from the cluster by notifying its neighbours of its intention to leave and by closing all established direct socket connections.

B. Client/Server network topology

We introduce a proxy between the client and the distributed server so that the client is unaware of the migration of the avatar between server nodes. The proxy prevents the communication with the client from being interrupted, when the client connection is transferred between server nodes. The responsibilities of the proxy include: (1) establishing connections with clients, (2) establishing connections with the server nodes (3) handling user authentication on behalf of the client (4) re-routing client packets to the correct server node and (5) handling migration of the client connection between server nodes.

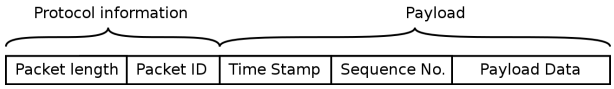

Fig. 3. Overlay packet structure

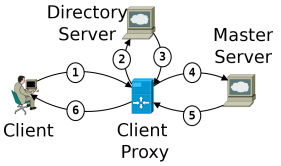

Fig. 4. Login Procedure

VI. Distributing the Virtual Environment

In this section we discuss a number of aspects related to distributing the MMVE amongst server nodes. These aspects include: (1) partitioning the VE, (2) interest management, (3) distributing the virtual entities, (4) state consistency, (5) handling the joining and leaving of users, and (6) object migration.

A. Virtual Environment Partitioning

We use zoning [7] for dividing the VE into master zones. The master zone represents the authoritative version of all player and non-player entities located within that region of the VE. The master zone is hosted by one of the nodes of the server cluster, referred to as the master node of that zone. This allows for an arbitrarily large VE to be hosted by adding more server nodes to the cluster. The architecture has been designed so that dynamic load balancing of the server nodes could be performed by dynamically altering the geometry of the zones. If a node becomes overloaded, as a result of too many users in its master zone, the size of the zone could be reduced and the users migrated to nodes hosting the neighbouring zones in the VE, but this has not been implemented yet.

B. Interest Management

The user should be unaware of the distributed server architecture, thus avatars crossing the virtual border between master zones should not be apparent to users. This can be done by mirroring entities that are outside the master zone, but are within the client's area of interest [13], [14]. Another method is to use spatial partitioning [5] and have the proxy make simultaneous connections to neighbouring master nodes. These techniques require that a master node be aware of virtual entities close to its border, and thus exchange state between neighbouring master nodes. We separate the Area of Interest Management (AoIM) into server AoIM and client AoIM.

Server Interest Management: We use a spatial publish/subscribe mechanism for server AoIM. When a new server node joins the distributed server and direct socket connections between neighbouring nodes have been successfully established, each of the neighbouring master nodes sends information regarding their master zone to the new server node. The new server node responds by subscribing to the neighbouring master node. A master node sends state updates to all server nodes that have subscribed to its master zone. In our current implementation state updates regarding all player entities (PEs), non-player entities (NPEs) and environment objects (EOs) are sent to subscribed server nodes. To keep master entities and mirrored entities separate on a server node, we make use of separate managers for the master zone and each of the subscribed replica zones.

Client Interest Management: Since we are primarily concerned with replacing the C/S architecture, the existing client interest management implemented by the Minecraft server software is re-used. Only updates to the environment, within the client avatar's render distance, are sent to the client. The server also only sends updates of virtual entities that are within a circle centred on the location of the avatar.

C. Virtual Entities Distribution and Persistence

As mentioned in the previous section, all virtual entities in the same master zone are assigned to the master node hosting that region. Persistence of the MMVE is achieved by the master node storing all information regarding the EOs and entities. Virtual entities must be able to migrate between different server nodes, but the Minecraft C/S architecture was not designed for a distributed server architecture. When two server nodes exchange state regarding virtual entities, both could be using the same serial to refer to different virtual entities. Since communication between server nodes is handled by Augeo, we introduce globally unique, authoritative serials for all virtual entities. When a new virtual entity is created, a 16-bit serial is generated using Java's random number generator. This 16-bit serial is discarded (and a new serial is generated) if it collides with any authoritative serial existing on the master node or any of its neighbouring server nodes. This ensures that all server nodes use entity serials that are unique for entities in the VE and should allow for the proxy to connect to neighbouring zones simultaneously. It is also necessary to ensure that state updates sent to the client use the global authoritative serials. Client state updates are handled by the native Minecraft server code, which uses the internal serials for entities. Since the client connection can also migrate between server nodes, it is necessary to intercept and rewrite state updates, so that all entity serials sent to the client are replaced by the globally unique entity serial. Events received from the client will refer the globally unique entity serial, and these serials also need to be replaced with the native Minecraft server's serial (since the native Minecraft server is unaware of the globally unique serials). The entity serial rewriting is implemented using ProtocolLib, a popular plugin for Bukkit. Introducing globally unique entity serials implies that clients can now be migrated between server nodes, without having to interrupt the client connection, or retransmitting the entity information in the client avatar's AoI. When a new server node first subscribes to a zone, the master node of that zone sends the EO state followed by the state of NPEs and PEs within that master zone. Until a neighbouring node unsubscribes from a master zone, the master node will continue sending state updates to the subscribed server node.

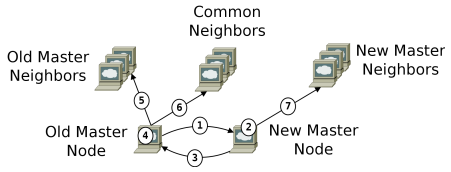

Fig. 5. Migration of non-player entities

D. State Consistency and Event Ordering

Each server node has a number of replica zones equal to the number of master zones to which it has subscribed. Each replica zone has a separate replica manager for that zone. The role of the manager is to receive and apply all state updates relevant to that replica zone. Since each received update contains a time stamp and a sequence number, it is possible to delay processing of received updates, to ensure sequential state consistency between the replica and master zone. Since this can lead to high latencies, we adopt a more optimistic approach [15]. Each replica manager maintains a queue of received updates and an update processing thread that is configured to execute every fixed number of server ticks. All updates generated during the same virtual server time tick has a unique update sequence number starting at zero. The processing thread orders and processes all updates in the queue based on the update's time stamp and sequence number. All updates with identical time stamp are processed (starting with the lowest sequence number) before processing updates with later time stamps. When a master zone generates updates, the sequence number is reset to zero whenever the virtual server time increments. Decreasing the execution frequency of the processing thread increases the probability that the updates are processed in the correct order, at the cost of increasing the latency between generation of updates in the master node and execution of updates in the replica zone. Since each replica zone has its own update queue and processing thread, the state of the replica zones is independently kept 'consistent' with the state of their respective master zones. Using time stamps for event ordering implies that the virtual server time of all the server nodes is synchronised. In our current implementation the overlay boot node has been configured to periodically transmit server time updates to all server nodes in the distributed cluster, thus synchronising the virtual server time.

E. Joining and leaving of users

The previous sections explained how a distributed server cluster is constructed and how state is distributed and maintained amongst the server nodes of the cluster. In this section we discuss the procedures followed when users join and leave the MMVE. Since every server node executes a copy of the Minecraft server, any of the nodes is capable of authenticating users that attempt to join the MMVE. This requires that clients be informed of the list of server nodes to be used when joining the MMVE. We have decided to use a single point of entry to the MMVE by using a directory server for user authentication. Since the IP of the boot node is known beforehand by any

machine joining the distributed server cluster, the boot node also acts as directory server. The directory server maintains a record of all master zones and their associated server nodes. It also maintains a separate database of user account information, specifically the information required to log into a server as well as the last known position of a user's avatar in the virtual VE. The master zone record and the user account database allow the directory server to redirect a connecting client connection to the server node hosting the master zone in which the user's avatar will appear when joining the MMVE. Fig. 4 illustrates the joining procedure.

Join Procedure: (1) The client application first establishes a connection to a proxy server and supplies the necessary authentication information (2) The proxy connects to the directory server (essentially logging into the Minecraft server hosted by the boot node) (3) The directory server retrieves the location of the user's avatar as well as the IP of the master node that contains the avatar's location. The directory server redirects the proxy by sending it the master node IP and closing the proxy connection. (4) The proxy connects and attempts to log into the master node. (5) The master server accepts (or rejects) the user login and sends an appropriate response to the proxy. (6) The proxy sends the master nodes response to the client and if the login was accepted, simply forwards all further traffic between the master node and client.

Leave Procedure: A user gracefully leaves the MMVE when a client sends a disconnect event to its master node (via its proxy). The last known location of the user must be recorded by the directory server to allow a user to rejoin the MMVE at the same location where his avatar left. Since a direct socket connection might not exist between the directory server and the master node, the master node sends the user information as an overlay message.

F. Object Migration

Using zoning for partitioning the VE implies that user avatars, client connections and virtual entities are handled by the same server node. This simplifies interaction in the VE (within the master zone) and improves the latency experienced by users. However, virtual entities entering and leaving master zones must be migrated between server nodes. NPE and PE migration is handled in a similar manner, but PE migration has the added complexity of also migrating the client connection. In our current implementation virtual entities are migrated when they cross the border between master zones. This creates a migration artefact visible to the user as a result of the finite time taken to log into the new master node. In the future we plan on initiating the migration process when virtual entities are close to the border, to avoid creating visible migration artefacts. We first explain the NPE migration procedure before discussing the more complex PE migration.

Non-Player Entity (NPE) migration: When a movement event is generated by a NPE crossing the boundary of a master zone, the identity of the server node to which the NPE must migrate is determined. This information is easily determined since each master node stores information regarding all its neighbouring server nodes. Migration involves transferring the authoritative copy of the NPE from the old master node to the new master node and informing the neighbours of both

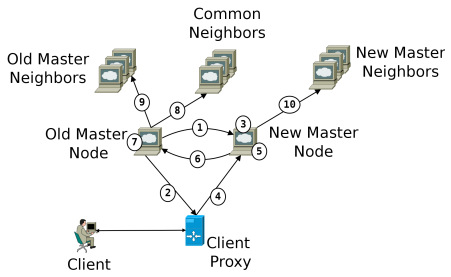

Fig. 6. Migration of player entities

the old and new master node. Fig. 5 illustrates the NPE migration procedure as follows: (1) The old master node sends a migrate message to the new master node identifying the NPE to be migrated (using the globally unique serial) as well as any additional data regarding the migrating NPE (such as its inventory, new virtual location). (2) The new master node creates an authoritative copy of the migrating NPE, adding it to its master zone. From this point all neighbouring server subscribed to this master zone will receive subsequent updates regarding the migrated NPE. (3) The new master node sends an acknowledgment to the old master node. (4) The old master server adds the state of the migrated NPE to the replica zone corresponding to the new master node. (5) The old master node sends a message to all server nodes that are neighbours of both the old and new master node to move the state of the migrated NPE from the old master's replica zone to the new master's replica zone. This is done so that neighbour server nodes will correctly receive and process updates from the new master node. (6) The old master node sends a message to all its neighbours that are not also neighbours of the new master node, informing them that the migrated entity has left its master zone. (7) The new master node sends a message to all its neighbours that are not also neighbours of the old master node, indicating that a new NPE entity has entered its master zone.

Player Entity (PE) migration: PE migration is very similar to NPE migration, with the following differences: the client sends a movement event (via the proxy) that is processed by the master node. If the user's avatar crosses the master zone boundary, PE migration is initiated. The migration of the client connection uses a similar mechanism as is used when the directory server redirects the proxy connection as described in Section VI-E. The procedure, illustrated in Fig. 6, is as follows: (1) The master node sends a migrate message to the new master node identifying the PE to be migrated (using the globally unique serial) as well as any additional player data needed to handle the expected login (such as the new avatar location). (2) The old master sends the IP of the new master node to the client's proxy. (3) The new master node stores the received player information and prepares for the proxy to establish a connection. (4) The proxy logs into the new master node. (5) The new master node creates an avatar for the client at the location received from the old master node and adds the new PE to the master zone. (6) The new master node sends an acknowledgment to the old master node. (7) The old master node adds the state of the migrated PE to the replica zone corresponding to the new master node. (8) The old master node sends a message to all server nodes that are

neighbours of both the old and new master node to move the state of the migrated PE from the old master's replica zone to the new master's replica zone. (9) The old master node sends a message to all its neighbours that are not also neighbours of the new master node, stating that the migrated entity has left its zone. (10) The new master node sends a message to all its neighbours that are not also neighbours of the old master node, indicating that a new PE entity has entered its zone.

VII. Progress of proof-of-concept implementation

A preliminary evaluation of the Koekepan architecture has been performed by manually partitioning the Minecraft VE into rectangular zones. Each of the master zones is assigned to a EC2 instance that represents a server node in the distributed server cluster. One of the EC2 instances is designated as the boot node and directory server. The only information initially shared by all server nodes, is the IP address of the boot node. The server nodes successfully self-organise into a distributed server cluster, subscribe to each other's master zones, and continue disseminating state updates to subscribed neighbouring servers, while their virtual server time is kept synchronised by the boot node. Users can join by connecting to the directory server via a proxy. If the directory server has no stored account information regarding a new login, the Minecraft server creates a completely new account and the user is logged into the Minecraft server hosted by the boot node. If an existing user logs into the directory server, the user account information is retrieved and the client connection is correctly redirected to the server node hosting the master zone containing the location where the user last left Minecraft. Users are able to modify the VE by placing and destroying blocks, and the state of the VE in the master zones is successfully replicated in the subscribed replica zones. Both player and non-player migration have been successfully demonstrated. It is possible to add new server nodes (in effect increasing the size of the VE) to the server cluster *while clients are connected to Minecraft*. This is conceptually similar to OverSim's ability to add regions to the VE by using the HyperGrid to allow users to migrate between different simulators [6].

VIII. Conclusion and Future Work

In this paper we described our approach to transform Minecraft into a platform for MMVE research. Our proof-of-concept system replaces the current C/S architecture with a distributed server architecture by adding P2P functionality to the Minecraft server clone (Bukkit) with the server plugin (Augeo), and inserting a proxy between the client and server. This is done without modifying the existing Minecraft client, and only required adding an additional software hook, for notifying plugins of NPE movement events, to the existing Bukkit server. Koekepan is not limited to a C/DS architecture, but can also be configured as a pure P2P or hybrid architecture. The components of Koekepan allow for the modification of Minecraft on different layers and functional areas and also allow researchers to log activities and messages on all the different layers of the architecture, which is useful for evaluations.

Future work includes investigating dynamic load balancing of the distributed server cluster. We are currently investigating the use of Voronoi diagrams and Quad-trees for dynamically altering the size of the master zones. Changes to the geometry of the master zones will result in the migration of environment object and virtual entities to server nodes with a lower load. We currently use Pastry for our P2P overlay, since it is mature library implementation for Java (FreePastry [16]) and offers the functionality for establishing direct socket connections (by using FreePastry's application sockets). Pastry uses a one-dimensional, circular key-space for key-based routing of messages. Currently there is no relation between the master zone a server node is hosting (i.e. the location of a server node in the VE) to its location in the overlay. It would be advantageous if server nodes hosting neighbouring master zones are also neighbours in the peer-to-peer overlay. We are investigating replacing Pastry with a P2P overlay that uses a two-dimensional key-space. We are also considering Hilbert curves, to preserve locality when mapping server locations in the two-dimensional VE to Pastry's 128-bit circular key-space.

References

[1] K.-c. Kim, I. Yeom, and J. Lee, "HYMS: A Hybrid MMOG Server Architecture," *Ieice Transactions On Information And Systems*, vol. E87-D, no. 12, pp. 2706–2713, 2004.

[2] B. Knutsson, H. Lu, W. Xu, and B. Hopkins, "Peer-to-peer support for massively multiplayer games," in *IEEE INFOCOM 2004*, vol. 1. IEEE, 2004, pp. 96–107.

[3] A. Yahyavi and B. Kemme, "Peer-to-Peer Architectures for Massively Multiplayer Online Games : A Survey," *ACM Computing Surveys*, vol. 46, no. 1, 2013.

[4] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang, "Donnybrook: Enabling Large-Scale, High-Speed, Peer-to-Peer Games," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, 2008, pp. 389–400.

[5] S. Kumar, J. Chhugani, C. Kim, D. Kim, A. Nguyen, P. Dubey, C. Bienia, and Y. Kim, "Second Life and the New Generation of Virtual Worlds," *Computer*, vol. 41, no. 9, pp. 46–53, Sep. 2008.

[6] "OpenSimulator." [Online]. Available: http://opensimulator.org

[7] W. Cai, P. Xavier, S. J. Turner, and B.-S. Lee, "A scalable architecture for supporting interactive games on the internet," in *Proc. of the 16th workshop on Parallel and distributed simulation*. IEEE Computer Society, 2002, pp. 60–67.

[8] "Minecraft.net." [Online]. Available: https://minecraft.net

[9] "Minecraft Coalition Wiki." [Online]. Available: http://wiki.vg/Protocol

[10] "Bukkit: Minecraft Server Mod." [Online]. Available: http://bukkit.org

[11] M. Assiotis and V. Tzanov, "A distributed architecture for MMORPG," *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games NetGames 06*, p. 4, 2006.

[12] S. Rieche, K. Wehrle, M. Fouquet, H. Niedermayer, L. Petrak, and G. Carle, "Peer-to-Peer-Based Infrastructure Support for Massively Multiplayer Online Games," *2007 4th IEEE Consumer Communications and Networking Conference*, pp. 763–767, 2007.

[13] E. Cronin, A. R. Kurc, B. Filstrup, and S. Jamin, "An Efficient Synchronization Mechanism for Mirrored Game Architectures," *Multimedia Tools and Applications*, vol. 23, no. 1, pp. 7–30, 2004.

[14] 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*. New York, New York, USA: ACM Press, 2002, pp. 25–28.

[15] G. Schiele, R. Suselbeck, A. Wacker, T. Triebel, and C. Becker, "Consistency Management for Peer-to-Peer-based Massively Multiuser Virtual Environments," in *Proc. of 1st Intl. Workshop on Massively Multiuser Virtual Environments (MMVE 08)*, 2008, pp. 14–18.

[16] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," *Design*, vol. 11, no. November 2001, pp. 329–350, 2001.