Manycraft: Scaling Minecraft to Millions

Raluca Diaconu[†*] Joaquín Keller[*] Mathieu Valero[*]

[*]Orange Labs,
Issy-les-Moulineaux, France
{joaquin.keller, mathieu.valero}@orange.com

[†]Laboratoire d'Informatique de Paris 6,
Université Pierre et Marie Curie, Paris, France
raluca.diaconu@lip6.fr

***Abstract*—Minecraft is a popular game with more than 20 million paying users and many more playing the free version. However, in multiplayer mode, only a few thousand users can play together. Our measurements show that, even reducing the landscape, i.e., the map, to a uniform flat land, a server cannot host significantly more users.**

For a common use case, when players cannot modify the map, we have designed and implemented Manycraft, an architecture to scale the number of users. Minecraft protocol messages are of three kinds: control, entity and map. In our approach, Kiwano, a distributed infrastructure for scaling virtual worlds, takes care of the entity related messages while the others are processed by a Minecraft server assigned to the player.

I. Introduction

Minecraft is a game where users build their own world. Players do not have specific goals to accomplish; they choose how to play the game. Minecraft has forgone realism for creativity and simplicity. This is probably the recipe of its huge popularity: estimations attribute Minecraft more than 20 million paying users and many more using the free editions.

While it is mostly a single user game, Minecraft has a multiplayer mode where users connect to a server in order to interact and communicate with each other within the same world. However, servers are strongly limited in the maximum number of simultaneously connected users: the most powerful ones reach only a few thousands.

Trying to understand where this limitation comes from, we conducted experiments with many simulated users. Our simulations show that bandwidth consumption, CPU load and memory usage significantly increase with the complexity of the world and the number of connected players.

Unexpectedly, with a minimalistic map, a uniform flat land for instance, and totally inactive players, all resource utilization remains high and the maximum number of players supported by a server does not increase significantly. Therefore, we can deduce that merely maintaining the presence of the players is a resource intensive activity.

One of the main interests of Minecraft is the creative mode where players build their own world driven by the pleasure of construction. Then, the creators have the desire to show their Lego-like universes on the Internet.

(a) Notre Dame de Paris

(b) WesterosCraft

WesterosCraft recreates the fantasy world of Game of Thrones. Notre Dame de Paris, the Empire State Building, the Enterprise Spaceship and countless real and imaginary places have been produced. Some can be visited but most are only visible in online videos.

Using Kiwano, an infrastructure for scaling virtual worlds, we have designed and implemented *Manycraft*, a solution to allow an unlimited number of users to simultaneously play inside such static Minecraft worlds.

We start by presenting related works aimed at scaling online games and virtual worlds. We present in Section III a description of the Minecraft game with an emphasis on the multiplayer mode. Since the source code of the game is not public, we designed and ran performance tests and the precise settings and results are related in Section IV. Our main contribution is the design and implementation of a novel cloud-based solution to allow an unlimited number of users to play together and interact in a Minecraft map. The solution, *Manycraft,* is described in Section V and is available at http://manycraft.net. Finally, we list the ongoing work to enhance and extend our solution to other problems.

II. Context and Related Work

When looking for academic papers related to Minecraft, unsurprisingly, we found none in the computer science area. Having initially a unique developer, its technical design is pragmatic and classical: it is simple, with few protocol messages, coarse pixelated graphics and trivial game physics.

This very same simplicity has attracted a lot of tinkering activity: users have reverse-engineered the protocol, decompiled the client and the server. Finally, modifying the software has become part of Minecraft game [1], and users give superpowers to their characters or invent new rules. Particularly, Bukkit [2] software is an add-on to the Minecraft multiplayer server to modify its standard behavior. This allows, for instance, protocol analysis and technical enhancements. This makes it possible to investigate and propose solutions for scalability in multiplayer mode.

Classical partitioning solutions as zoning, sharding and instancing [3] allow many users to connect but do not solve the problem of scaling to an unlimited number of simultaneous users in a unique contiguous space.

Other solutions propose to improve scalability using p2p architectures. The distributed scene graph in [4] is used to scale the complex 3D landscape of virtual worlds such as Second Life. However, Minecraft has simple graphics and therefore does not have the same requirements for 3D scene computation. Colyseus [5] and Donnybrook [6] are middlewares to develop multiplayer games build on top of Mercury, a p2p overlay of range-queriable DHTs. They have been tested for Quake III with up to 900 simultaneous players. However, DHTs have been proved [7] to fail under frequent player movements. All in all, these solutions scale at most up to several hundreds, a performance already overran by existing Minecraft servers [8].

Separating the static elements of the scene from the avatar movements is a more promising approach taken by many research teams [9], [10], [11], [12]. However, despite good theoretical or simulated results, these solutions have failed to be widely deployed. The reason may lie in their p2p architecture that is somehow impractical: adoption by users is slowed by the extra steps needed to install the software; developing and maintaining code for multiple versions of several platforms is costly and cumbersome; overall performance is impacted by the slowest peers, etc.

Recently, some of us released Kiwano [13], a cloud based solution to scale virtual worlds by handling the avatars movements. Kiwano offers a streaming API where avatars report theirs positions and get notified of the events in the neighborhood; the static elements are provided by other means. The typical example is HybridEarth [14] which relies on Kiwano for the avatars while the landscape is StreetView provided and hosted by Google.

III. Minecraft

A Minecraft *map* is made of simple cubic *blocks*. All blocks have the same size and only vary in their type (wood, coal ore, stone,...). Players are represented by two block high *entities*. They can choose to play the game in creative, adventure or PVP (Player Versus Player) mode. PVP mode is centred on player interaction often with bellicose gaming experiences. In adventure mode, the purpose of the game is defined by world's creator, and players mostly interact with other players and dynamic objects in the décor.

In creative mode, where the purpose is to build things, players add and remove blocks. As it can be played alone, this mode is very popular. But currently, in order to present their creations, players either host a server on their own computer and allow others to connect, or record a video and share it on the internet. The first solution is cumbersome and costly: the user needs to maintain a server for a few friends that might visit, while the second one does not offer those interested the possibility to visit the world.

Interestingly, Minecraft provides different mechanisms to limit or even forbid players to modify blocks. Moreover, many popular gaming scenarios tend to emphasize player to player interactions. For instance, in role playing or racing games, the map is not meant to be modifiable.

Inside the game, each player is represented by an *entity*. Around its position, each has a square-shaped *awareness area* providing the visible map and *neighbors* to be transmitted. There are also *game entities* called *mobs* affected by the environment the same way as players, but no data about the map or neighborhood is transmitted.

In multiplayer mode all players run a Minecraft client. To play together they connect to the same server.

A. Minecraft server

A Minecraft server hosts the world and delivers the content to the connected players. Server's time unit is the tick, which lasts for 50ms. Within each tick, the server updates world physics, mobs' behavior, and notifies each player about changes that occurred in her awareness area. The elements constituting the map evolve. Dynamics such as water flowing, plants growing or night and day alternation, are computed when they are part of someone's field of vision. Also, mobs are simulated and they react to the environment by moving, attacking, etc.

B. Minecraft client

After connection, the player is spawned at a position chosen by the server. During the game, the client sends to the server player actions, namely position updates, block destruction, etc., and is notified back about events occurring in the awareness area.

C. Minecraft protocol

With each new version Minecraft evolves and so does the protocol [15]. The first release of the game at the end of 2011 had 19 message types and this number grew up to 83 at the end of 2013. Meanwhile, the game added up new types of entities, new skins, different control for small and large movements, prediction, and so on. Albeit the significant evolution, the protocol messages can be divided into the following categories: map, entity and control.

Map Map related messages are generated when a player changes position or when blocks in the awareness area are changed. The server only sends the differences with respect to the last configuration.

Entities When a player or a mob moves, the new position is send by the server to every player that should be aware of it. Entity related messages also bear chat, actions or other entity attributes.

Control During each tick, the server sends keep alive messages to clients. There are also cyclic events, such as day-night alternation, that generate messages at regular time intervals.

IV. Evaluating Minecraft Scalability

Looking at online Minecraft multiplayer servers [8], we can see that the maximum number of simultaneous players is in the few thousands. However, it is not clear what impedes to go further. Our hypothesis is that the mere presence of players is enough to explain this limit, and having a map reduced to the minimum does not push the limit by orders of magnitude.

A. Experimental Setup

To estimate the load generated by the players' presence we reduce both environment complexity and in game interactions. Firstly, the world contains no mobs, only blocks and players. Hence the server has no animal nor monster behaviors to compute. Secondly, the map is *Superflat* [16] and made of a single layer of rocky blocks. As a consequence, we basically eliminate world's physic as no plants are growing and no water is flowing. Thirdly, our simulated players don't chat, nor mine, nor craft. Fourthly, we set the view distance to its minimum value. This last point is a well-known hint to reduce server's load [17].

We ran our experiments on four identical machines hosted in the same datacenter. Each machine had an 8-core Intel Xeon at 2.8GHz, 24 gigabytes of RAM, ran Ubuntu 12.04, Oracle Java 7, and are connected together with gigabit-ethernet. The average latency between machines is around 0.8ms. One of the machines ran a CraftBukkit [2] server. A CraftBukkit server is a Minecraft server modded to accept Bukkit plug-ins. The other machines ran several simulated players using text-only Minecraft clients, with no graphical rendering.

Players' movements followed the model of avatars movement in virtual worlds described in [18]: a player is either slowly random walking or moving fast in a given direction. Players move at the same speed as regular (i.e., human driven) Minecraft players and similarly update their positions 20 times per second.

B. Measurements

In our experiment we add players to the server one by one, waiting 5 seconds between two insertions. This delay avoids players to have their connection denied by the server due to an instant heap of connections. After slowly inserting a batch of one hundred we wait several minutes for the system to stabilize before taking the measure. This ensures that the initial cost of entering the world does not impact the measurements for resource utilization.

a) Throughput: In Figure 1 the x-axis is the number of simulated players connected while the y-axis is the server throughput in megabytes per second. These two curves clearly show that the bandwidth consumption grows linearly with the number of players.

Each Minecraft client sends ∼3.5Kbytes/s and receives ∼3Kbytes/s. For a server connected to the backbone through gigabit ethernet –a common setting for servers in datacenters– we can assume a full duplex with maximum throughput of 500 Mbits/s. This indicates that bandwidth should start to be a limiting factor for approximatively 20K players.

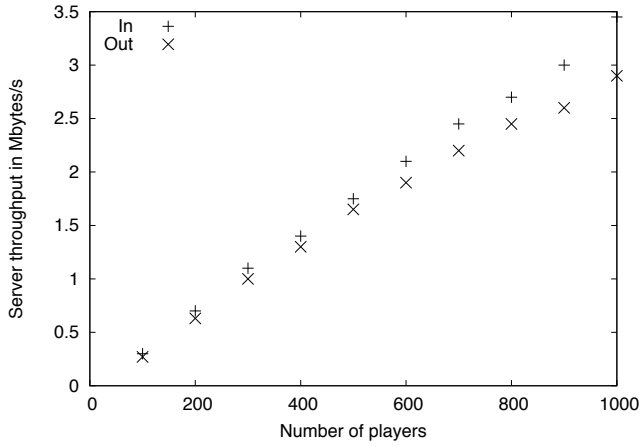

Fig. 1: Throughput

With a more complex, modifiable and interactive map the throughput per user should be greater and the maximum number of player significantly reduced.

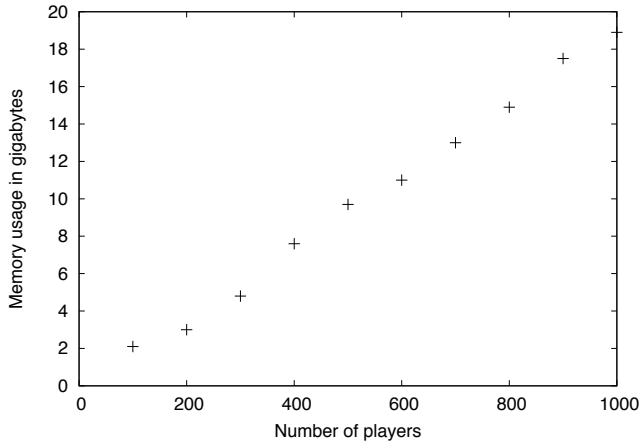

Fig. 2: Memory usage

b) Memory Usage: In figure 2 the x-axis is the number of players while the y-axis is the amount of RAM used by the server in gigabytes. Given that the server runs on a Java virtual machine, the slight deviations might come from the garbage collector behavior. At this scale, the memory usage of the server can be approximated to grow linearly with the number of connected Minecraft clients. A server that can allocate 24 gigabytes to the server reaches its RAM limitation for approximatively 1500 connected players.

c) CPU Load: To evaluate the impact of entity dynamics we performed the simulations using moving entities but also still players, i.e., they connect but remain inactive. Figure 3 shows the results of the two measurements: the x-axis is the number of players while y-axis is the aggregate percentage of the CPU load of all cores: a percentage above

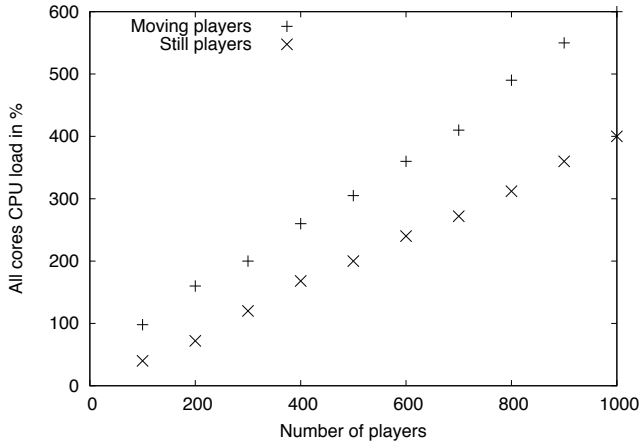

Fig. 3: CPU load

100% indicates that more than one core is used by the Minecraft server. Given that we used an 8-core server, the maximum load it can stand is 800%.

We were not sure that our model for avatar movement would reflect the actual behavior of Minecraft players. But the comparison of the CPU load, all the avatars moving vs. motionless avatars, shows, unexpectedly, that the mobility patterns have little influence on the CPU load of the server.

Both measurements show that the CPU load can be approximated to grow linearly with the number of players in the range supported by one server. Therefore, with a very simple map, the maximum load supported by our 8-core CPU should be just above 1000 players.

C. Conclusion

One of the prominent outcomes of this experiment is that the map complexity might lower the maximum number of simultaneous connected players but is not the major cause of this limitation. The other lesson is that there is not a unique bottleneck; when the number of players grows all resources become exhausted at once.

Finally, the only way to go far beyond ten thousand players –the actual observed limit– together in a Minecraft map is to distribute the load generated by the entities amongst many machines.

V. Manycraft

Manycraft is meant to allow an unlimited number of players to interact in a static Minecraft map. To join the system the player runs a Manycraft node on her own computer.

The Manycraft node is a Minecraft server "modded" with Bukkit in order to divert the entity messages to Kiwano which, in return, notifies about the events occurring in the player's neighborhood. The local Minecraft server has the static map preloaded and its main duty is to deliver the map to a unique player. The load generated by the avatars has been shifted entirely to Kiwano.

```
{
   method:"update",
   urlid:"http://manycraft.net/u6nMnT",
   iid:"john.smith",
   lng:52.5, lat:15.3, angle:40, alt:40,
   appdata:{
      manycraft: {
         sneaking:false,
         pitch:0.147
      }
   }
}
```

Fig. 4: Update command

```
{
   method:"updates",
   urlid:"http://manycraft.net/u6nMnT",
   iid:"john.smith",
   new:
   [
      {
         urlid:"http://manycraft.net/w28QPu",
         iid:"alice.smith",
         lng:52.5, lat:15.3, angle:40, alt:40,
         appdata:{
            manycraft:{
               sneaking:true,
               pitch:3.2
            }
         }
      }
   ]
   old:
   [
      ("http://manycraft.net/w28QPu",
       "charlie.doe"),
   ]
}
```

Fig. 5: Neighborhood update notification

```
{
   method:"msg",
   urlid:"http://manycraft.net/u6nMnT",
   iid:"john.smith",
   recipients:
   [
      ("http://manycraft.net/8Qw2RJ",
       "alice.smith"),
   ],
   msg:"",
   appdata:{
      manycraft:{animation:"swing_arm"}
   }
}
```

Fig. 6: Message notification

A. Kiwano

Kiwano [13] is a scalable distributed infrastructure for virtual worlds, designed to support an unlimited number of moving objects updating their position at arbitrary high frequencies. In Kiwano the set of moving entities is distributed onto many servers, each taking care of a group of entities based on their geographical proximity. The positions are indexed using a distributed Delaunay triangulation for an efficient neighborhood access.

The same indexing structure is used to partition the space into zones following entity density. As entities move, these zones dynamically change their shape. To free the application layer from the distributed internals, Kiwano provides an API [19] to developers.

When connecting, a client is assigned a proxy, the entry point to Kiwano for the entire session. Each proxy is connected to a subset of all entities and, for each of them, maintains the communication with the corresponding zone and neighborhood. And, in order to scale, Kiwano spawns as many zone servers and proxies as needed.

Kiwano clients send three types of commands: *update* entity state, *message to all* and *private message*. On the other side, they receive four types of notifications: the full list of *neighbors*, neighborhood *updates*, *message* from another client, and *remove* neighboring entities. All Kiwano messages are encoded in json, common now in most programming languages.

Kiwano has been designed independently of any particular system and messages have an *appdata* field to carry data specific to the virtual world to scale.

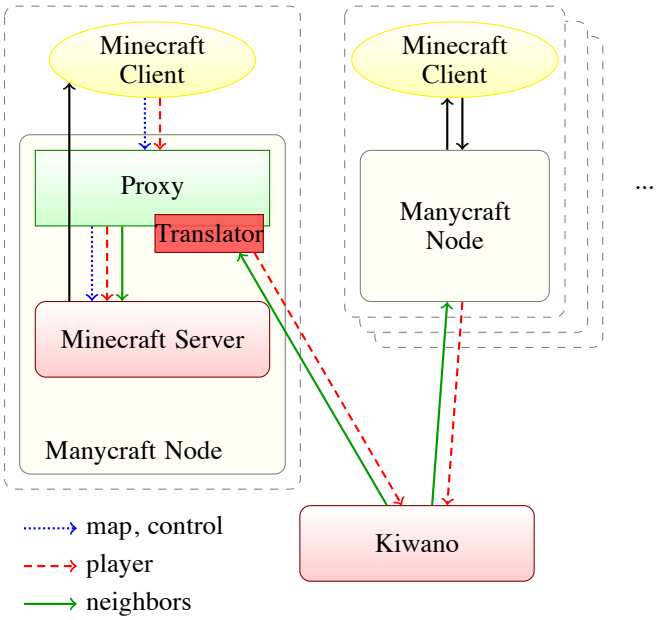

Fig. 7: Manycraft Architecture

B. Manycraft Node

To join a Manycraft world, the player downloads the node preloaded with the corresponding Minecraft static map. The node is composed of a regular Minecraft server and a proxy. Once the node is running on her machine, she connects through the proxy her regular Minecraft client to the local Minecraft server embedded in the node. The local server receives the packets from the client and serves the blocks of the map as the player moves. The proxy also sends, after translation, player movements and actions to Kiwano.

Kiwano notifies back about the movements and actions occurring at nodes of the neighboring players. These notifications are translated and processed by the internal Minecraft server which generates the packets for the client. Remote players movements and actions are then rendered by the client.

As players are in the same map, see each other and interact: they are together in the world.

C. Bridging Minecraft over Kiwano

The essential task of a Manycraft Node is to route the relevant Minecraft traffic from and to Kiwano. The proxy inspects the messages issued by the client. All the messages, including player movements, are forwarded without modification to the Minecraft server. On position updates the Minecraft server sends the surrounding blocks. Messages concerning player actions are translated and sent to Kiwano which sends back notifications about the neighbors. These messages are translated into Minecraft messages so that the neighbors entities can be placed on the map as mobs. This way, the server attached to a player has all the information corresponding to the awareness area.

Minecraft player action	**Kiwano command**
join	connect to Kiwano
quit	disconnect from Kiwano
kicked	disconnect from Kiwano
move	update lng, lat, alt
look	update angle
toggle sneaking	update appdata
toggle flying	update appdata
toggle sprinting	update appdata
animation	message to all appdata
chat/say	message to all
chat/tell	private message

TABLE I: Minecraft actions translation

Minecraft actions to Kiwano commands: Each entity related message issued by the client is intercepted by the proxy and translated to a Kiwano command. The Minecraft account name is translated into a Kiwano object id, see Section V-A. Player's state modification generates a Kiwano update command, player's coordinates and yaw are respectively mapped on lat, alt, lng and angle fields. Minecraft specific description of the avatar is sent in the appdata field with appid 'manycraft'.

Chat and player actions are translated into Kiwano message commands.

Kiwano notification	Minecraft handling
update	create or update entities
remove	remove corresponding entities
message	issue corresponding action (animate an entity's arm, display message, etc.)

TABLE II: Kiwano notifications translation

Kiwano notifications handling: Kiwano notifications inform a node about actions and state changes of remote players in the neighborhood. Remote players are represented in the Minecraft server as mobs. Kiwano notifications trigger movements and actions of these mobs. Upon notification of entity arrivals/departures in player's awareness area the mobs are created/removed accordingly in the Minecraft server.

D. Manycraft Scalability

The Minecraft server embedded in the node supports only the load of one client. The traffic with Kiwano is a subset of the normal Minecraft traffic and remains low. Finally, the translation and routing processes of the proxy do not generate any significant load.

Compared to a regular Minecraft client, the CPU load of the user's computer is only increased by the low activity of the embedded server. The memory depends on the size of the preloaded map but remains constant and low in our tests. Because in average the number of neighbors provided by Kiwano is constant, resources used by the proxy remain constant. The external communication of a Manycraft node happens only with Kiwano and the throughput is similar to a Minecraft client, minus the incoming map description messages. All in all, compared to classical setting —a Minecraft client connected to a remote multiplayer server,— the load of the user's machine is lower regarding bandwidth and slightly higher regarding CPU and memory usage.

Manycraft is a distributed system composed of nodes connected through Kiwano infrastructure.

VI. Conclusion

Manycraft is a solution to allow an unlimited number of players to simultaneously interact in the same static Minecraft map. It relies on separation of the load generated by the entities from the map and control. We forward entity related messages to Kiwano, a general architecture to scale virtual worlds, and keep on player's machine only the relevant subset of neighbors.

The creativity of the Minecraft crowds produce vast and rich landscapes. However, due to Minecraft server scalability limitations, even the most popular remain, with few thousands simultaneous players, mostly empty.

With Manycraft a map can be populated by an unlimited number of players opening new perspectives in massive social interactions such as concerts, fairs, demonstrations, battles, etc. The first version, available at http://manycraft.net, supports static maps. Ongoing works focus on map dynamics and on extending players interactions.

Future work may also include the migration of the Manycraft node to datacenters, relieving the user of cumbersome installs. This first experiment with Minecraft suggests that similar approach could be successfully applied to Second Life.

References

[1] S. Parkin, "The Secret to a Video-Game Phenomenon," *MIT Technology Review*, vol. 116, no. 4, June 2013.

[2] "Bukkit, a free, opensource, extension of minecraft multiplayer server," http://bukkit.org.

[3] N. Gupta, A. Demers, J. Gehrke, P. Unterbrunner, and W. White, "Scalability for Virtual Worlds," in *IEEE International Conference on Data Engineering (ICDE)*, 2009.

[4] D. Lake, M. Bowman, and H. Liu, "Distributed scene graph to enable thousands of interacting users in a virtual environment," in *NetGames*, 2010.

[5] A. Bharambe, J. Pang, and S. Seshan, "Colyseus: a distributed architecture for online multiplayer games," in *Networked Systems Design & Implementation (NSDI)*, 2006.

[6] 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 *SIGCOMM 2008 conference on Data communication*, 2008.

[7] M. Varvello, C. Diout, and E. Biersack, "P2p second life: experimental validation using kad," in *International Conference on Computer Communications (INFOCOM)*, 2009.

[8] "A directory of minecraft servers," http://minecraftservers.org/. Retrieved August 10, 2013.

[9] O. Beaumont, A.-M. Kermarrec, and E. Riviere, "Peer to peer multidimensional overlays: Approximating complex structures," in *International Conference On Principles Of Distributed Systems (OPODIS)*, 2007.

[10] O. Beaumont, A.-M. Kermarrec, L. Marchal, and E. Riviere, "Voronet: A scalable object network based on voronoi tessellations," in *International Parallel & Distributed Processing Symposium (IPDPS)*, 2007.

[11] S.-Y. Hu, J.-F. Chen, and T.-H. Chen, "VON: a Scalable Peer-to-Peer Network for Virtual Environments," *IEEE Network*, vol. 20, no. 4, Jul. 2006.

[12] J. Keller and G. Simon, "Toward a peer-to-peer shared virtual reality," in *International Conference on Distributed Computing Systems (ICDCS)*, 2002.

[13] R. Diaconu and J. Keller, "Kiwano: A scalable distributed infrastructure for virtual worlds," in *High Performance Computing & Simulation (HPCS)*, 2013.

[14] "Hybrid earth: Mixed reality at planet scale," http://blog.hybridearth.net/.

[15] "Minecraft current stable protocol," http://mc.kev009.com/Protocol. Retrieved August 20, 2013.

[16] "Minecraft flatlands," http://minecraftwiki.net/wiki/Superflat. Retrieved August 20, 2013.

[17] "Improving your minecraft servers performance," http://sk89q.com/2013/03/improving-your-minecraft-servers-performance. Retrieved August 20, 2013.

[18] S. Legtchenko, S. Monnet, and G. Thomas, "Blue banana: resilience to avatar mobility in distributed mmogs," in *Dependable Systems & Networks (DSN)*, 2010.

[19] "Kiwano api," http://kiwano.li/.

[20] "Manycraft project," http://manycraft.net/.

[21] Top-Cat, "Npclib: a library to spawn npcs in craftbukkit," https://github.com/Top-Cat/NPCLib.