Project Proposal – Voxel Populi: A Decentralised Peer-to-Peer Voxel-Based World

Samuel J. Sully

25 October 2019

Project Supervisor: Prof. Jon Crowcroft
Director of Studies: Prof. Alan Mycroft
Project Overseers: Prof. Marcelo Fiore & Dr. Amanda Prorok

1 Introduction

The Massively Multiplayer Online game genre is popular in modern gaming, often involving thousands of concurrent players [1]. These games are typically implemented using a client-server model, requiring some form of 'sharding' [2] whereby players are separated into independent instances of the same world meaning that players can only interact with others connected to the same instance (shard) as them.

An alternative approach is to distribute the game world over a peer-to-peer network, where each peer (or group of peers) is responsible for managing a small part of the game world. This helps with load balancing, while ensuring that all players are – effectively – in the same virtual world. It also has the advantage of decentralising the game world, increasing its resilience to some failure modes.

This has the advantage that you are able to have a user make persistent changes to the game world, in a sharding implementation as the world is replicated it is hard to have a user make a meaningful change to the world, preventing Minecraft (or similar) like worlds which are fully customisable.

Additionally, when large scale MMOs cease to be profitable for the developers, who operate the servers, they will often shut down the servers [3] even if there remains a large active player base. By allowing individuals to setup their own game servers to help run the MMO as part of the P2P network, we ensure that the game can survive, at no cost to the developers, as long as there is a community dedicated to ensuring its survival. This is typically not possible for MMOs, only for games which have a client-server multiplayer system.

One issue with this approach is that, if a rogue server operator decides to behave maliciously, they could modify their server to – for example – give

themselves an unfair advantage when in territory managed by their server. To combat this one could include a consensus/voting system [4].

2 Starting Point

Creation of the distributed world will rely mostly on two Part IB course:

- Concurrent and Distributed Systems
- Computer Networking

as well as drawing on material from the Part II course Principles of Communication.

The Concurrent and Distributed Systems course will be useful as this project is an example of a distributed system. I will be making use of Remote Procedure Calls (RPCs) in my project which are covered in this course, one of my proposed extensions aims to include a consensus or voting system – these topics are covered extensively in the CDS course.

The Computer Networking course will be relevant as it covers Distributed Hash Tables (DHTs) which I intend to use as the fundamental basis for my project. Other concepts introduced in this course, and further developed in the Part II course Principles of Communications (such as routing) will also be relevant to my project.

Additionally, the Part IA course Introduction to Graphics and the Part IB course Further Graphics will be useful in the development of a simple client to connect to and interact with my distributed world, this client would likely be written using OpenGL which is covered extensively in these two courses. It will, however, be very primitive indeed as it will be only a prototype.

3 Substance and Structure

My core project consists of three main parts, they are:

- The DHT
- The 3D World
- The Graphical Client

I have researched a number of potential implementations for the DHT including Kademlia [5] and OpenDHT [6] (which has an open source implementation with convenient Python bindings). I plan to implement my own based on the Kademlia whitepaper [5], potentially with some minor extensions (i.e. extra RPCs, for example: I may need a `GENERATE` RPC to generate a world chunk). This will be used as the basis for my distributed world.

The 3D world will be a simple voxel-based one. With clients able to place and remove voxels from the world. The world will consist of a number of discrete 'chunks' of voxels. These chunks will re represented as a simple 3D array and will represent a contiguous $x * y * z$ compartment of the world (where z is the height of the world). This is the approach taken in games such as Minecraft [7]. The world will also allow 'entities' which are not part of the voxel array but are objects which can move freely in the 3D space (e.g. the player). Chunks will have a separate list to store any entities in that chunk along with their position. It might be worthwhile to explore usage of an kd-tree (or equivalent) for storage of entities but this would be a minor extension and is not required.

Usage of the DHT to aid in locating entities as well as chunks in the network will be necessary, as when a client connects to the world looking for a particular player to take control of, they will need a way to find which chunk that player is currently in. In the event that the player is in no chunk (i.e. either the chunks they were in have been lost or they are new), then a random chunk can be selected as a spawn location.

As this is a distributed system I will need some form of synchronisation between nodes; a logical ordering system such as a vector clock is unlikely to be appropriate as the system is operating in real time and the real time between events is relevant to the computation. So I intend to use NTP for synchronisation, as minor clock skew will not affect the system to a significant degree. I will be using an existing NTP library for this as it is likely to be more robust than any implementation I can produce.

Additionally, there will be a client able to connect to this distributed 3D world and to interact with it. This will be a simple game client which displays the world and allows basic interaction with the world (likely little more than placing and destroying voxels). This could be implemented in OpenGL [8] either directly in C++ or using a library such as LWJGL [9]. Alternatively, it could be written using an existing engine such as Unity.

Finally, I will need to write a very simple test agent which connects to the server and performs some example gameplay activity, such as constructing a basic structure. This will be needed to test the performance of the system in the evaluation phase. This will be a paired down version of the above client so will require little work.

4 Evaluation and Success Criteria

4.1 Methodology

This project's success will be gauged entirely quantitatively, no human participants will be required.
My project will be evaluated in the following ways:

- My DHT implementation will be compared to the Kademlia specification [5]. The RPCs set out in the Kademlia paper will be analysed to ensure they function as specified, for example, the number of nodes contacted during a lookup should be $\mathcal{O}(\log n)$ where n is the number of nodes in the system.

- I will containerise the peer-to-peer server and deploy it on a small to medium scale. I will then test it by connecting a number of my test agents to the server to see how it performs. I will be testing both the network performance and the performance of the individual machines. These figures will be used as a baseline for the next section.

- I will then attempt to estimate how well the system will perform at scale by extrapolating from the baseline acquired in the previous section. I will use some higher level reasoning to compare the performance of my system with large contemporary systems such as the games World of Warcraft [12], EVE Online [13] and Runescape [14].

- I will analyse my implementation's resilience to a number of attacks/failures. Potential examples include: Byzantine Faults, Sybil Attacks, Impersonation Attacks, etc.

4.2 Success Criteria

1. My DHT must adhere to the Kademlia specification. It is possible I will need to make some changes to fit the specification better to my needs and this is acceptable.

2. The peer-to-peer node program must join the network, bootstrapping via some known node, and then will be able to participate in hosting the game world as it becomes part of the DHT.

3. It must be possible to interact with the world using a simple 3D graphical client, which is able to place and remove voxels from the world. These changes must persist.

4. The system must handle player moving between separate chunks (and thus, separate peers) seamlessly, with no loading screen.

5. There must be a simple test agent which connects to and interacts with the world in some notional way to emulate the behaviour of a human user. This is for the purposes of my above quantitative evaluation methodology.

5 Possible Extensions

Below are a number of potential extensions to my project. This list is not exhaustive and I may choose to implement others as well as or instead of those listed below.

- **Weak Consensus/Voting System:** as mentioned in my introduction I propose the inclusion of a weak consensus or voting system to aid in the detection of rogue peers and, similarly, rogue clients. This would work by replicating each chunk over n nodes where $n >= 3$ and having each of the n peers review the activities of the others to ensure that they are acting correctly. If a large enough nodes were able to agree that a node is acting maliciously, they could blacklist it from the network.
- **3D Terrain Generation:** rather than having a bland, flat world, it would be nice to have some variable terrain. This could easily be achieved using Perlin noise [10], a type of gradient noise commonly used in video games for realistic terrain generation.
- **Further World Features:** including other gameplay, for example a basic mob [11], would enable me to more accurately test the performance of the system at a large scale.

6 Timetable

A brief outline of my plan of work is below.

1. **24th October to 13th November:** Study and implement the Kademlia DHT to specification [5].
   Deliverable: DHT which can be demonstrated with a toy system and tested with appropriate unit tests, must conform (with appropriate deviation) to the Kademlia specification.
2. **14th November to 4th December:** Server-side 3D world implementation. This will involve setting up the state representation for the 3D world; making a server able to load a chunk from disk and then to complete the necessary processing to update the game state while a player is using to this chunk (processing will include, physics calculations such as collisions, handling interaction with the voxels and – if implemented – processing mobs). It will also need to be possible for a user to connect to the server and load a chunk which is stored in that node. The protocol for interaction between client and server will need to be established here.
   Deliverable: 3D world server, this will be the software run by a single peer in the network and will be built on top of the DHT.

3. **5th December to 22nd December:** Implement 3D graphical client. This will be a simple 3D client used to interact with the world, written, most likely, in OpenGL. The client will connect to the DHT, locate the relevant chunk(s) to be rendered, load these by connecting to the relevant server(s) and provide an interactable display of the world. It will make use of the above protocol for communication between client and server.
   Deliverable: Graphical client program, able to join the P2P game by bootstrapping via a known node.

4. **2nd January to 10th:** Implement test agent. This should be a fairly easy task as it will be a paired down version of the client developed earlier in the vacation. This is to be used in the evaluation stage and will have no graphical interface by default (though one may be pertinent for testing purposes).
   Deliverable: Test agent program, to be used to test the system at a small to medium scale.

5. **11th January to 15th January:** Begin evaluation testing of system on a small scale using the test agent developed in the previous section.
   Deliverable: Baseline test data.

6. **16th January to 29th January:** If time allows, then implement extensions which are feasible in remaining time. Additionally, write progress report demonstrating completion of project core and evaluation data acquired in previous section.
   Deliverable: Progress report and implementation of extension(s) if time allows.

7. **30th January to 5th February:** Work on chapters introduction and preparation, adjust code from evaluation data and further work on extensions.
   Deliverable: Introduction and preparation chapter drafts and further work on extensions.

8. **6th February to 12th February:** Work on implementation chapter and further work on extensions.
   Deliverable: Implementation chapter draft.

9. **12th February to 19th February:** Work on evaluations and conclusions chapters.
   Deliverable: Entire dissertation first draft to be submitted to supervisor and DoS by March 20th.

10. **Remaining Time:** Refining dissertation and, time permitting, further extension work.

Deliverable: Dissertation final draft to be submitted to supervisor and DoS by April 20th.
Deliverable: Dissertation due on 8th May.

7 Resource Declaration

I will be using my own computer for development, I am about to get a new computer, so I cannot comment on the exact specification yet, but I will be using the new one for the majority of the project. In the event of this failing, I have several fallback machines. Firstly my current laptop running Ubuntu 18.04 is still operational. Additionally, I have a desktop PC which is also with me at university which I can use in the event of both other computers failing. Furthermore, in the unlikely event that all of these fail I can use the MCS machines in the Computer Lab, my college computer room and my college library.

I will be using Git for version control of my project and dissertation, these will be regularly pushed to GitHub. Additionally, I will be pulling the GitHub repository automatically to an external server I possess on a regular basis and also to MCS machines on a regular basis.

I will also be making use of my own dedicated server for some of the testing of the project. This is an OVH dedicated server hosted in Roubaix, France. In the event of this failing, I propose to use several Raspberry Pis for my testing. If my server proves to be insufficient then I will be using AWS services supplied by Professor Crowcroft for testing.

References

[1] "World of Warcraft subscriptions fall to 7.7 million". `https://www.gameinformer.com/b/news/archive/2013/07/26/world-of-warcraft-subscriptions-fall-to-7-7-million.aspx`. Accessed: 2019-10-16.

[2] "Sharding" on Wikipedia. `https://en.wikipedia.org/wiki/Shard_(database_architecture)`. Accessed: 2019-10-15.

[3] "Club Penguin is shutting down". `https://techcrunch.com/2017/01/31/club-penguin-is-shutting-down/`. Accessed: 2019-10-15.

[4] Miller, J. Distributed virtual environment scalability and security, chapter 5. `https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-809.pdf`. Accessed: 2019-10-15.

[5] Maymounkov, P. and Mazires, D. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. `https://pdos.csail.mit.edu/`

~petar/papers/maymounkov-kademlia-lncs.pdf. Accessed: 2019-10-16.

[6] OpenDHT. https://github.com/savoirfairelinux/opendht. Accessed: 2019-10-16.

[7] "Chunk" on the Minecraft Wiki. https://minecraft.gamepedia.com/Chunk. Accessed: 2019-10-17.

[8] OpenGL. https://www.opengl.org/. Accessed: 2019-10-17.

[9] LWJGL: Lightweight Java Game Library. https://www.lwjgl.org/. Accessed: 2019-10-17.

[10] "Perlin Noise" on Wikipedia. https://en.wikipedia.org/wiki/Perlin_noise. Accessed: 2019-10-17.

[11] "Mob" on Wikipedia. https://en.wikipedia.org/wiki/Mob_(gaming). Accessed: 2019-10-17.

[12] World of Warcraft. https://worldofwarcraft.com/. Accessed: 2019-10-17.

[13] EVE Online. https://www.eveonline.com/. Accessed: 2019-10-17.

[14] Runescape. https://www.runescape.com/. Accessed: 2019-10-17.