ABSTRACT

GAVASKAR, NILESH RAMESH. A Framework for Performance Analysis of Virtual Collaborative Environments. (Under the direction of Michael Devetsikiotis.)

Performance is one of the key aspects of collaborative virtual environments which makes them gain acceptance by people. The designers on the other hand are restricted by the amount of resources they can spare for each instantiation of the virtual world. In this thesis, we make an attempt to quantify the resource requirements for obtaining satiating in-world experiences by varying the allocated resources. We present a framework for performance modeling of virtual collaborative environments (VCEs). Our model could be used as a benchmarking tool for assessing the quality of experience of participants in these platforms. Our framework is designed for virtual worlds similar to Second Life but can easily be extended to other virtual worlds. We examine the case of users communicating via instant messages and voice. We capture performance for these cases and using the response surface methodology a utility function is derived that yields the performance of the examined environment given the available computing and communication resources. We further propose a pricing scheme based on which we formulate optimization problems for optimum resource allocation for VCEs.
A Framework for Performance Analysis of Virtual Collaborative Environments

by
Nilesh Ramesh Gavaskar

A thesis submitted to the Graduate Faculty of North Carolina State University in partial fulfillment of the requirements for the Degree of Master of Science

Computer Science
Raleigh, North Carolina 2010

APPROVED BY:

_________________________  _________________________
Do Young Eun             Laurie Williams

_________________________
Michael Devetsikiotis
Chair of Advisory Committee
DEDICATION

Dedicated to my parents.
BIOGRAPHY

Nilesh Ramesh Gavaskar was born on March 6, 1986 in Mumbai, India. He graduated from the University of Mumbai, Mumbai in June 2008 with a Bachelors of Engineering degree in Computer Engineering. He has been working towards his Masters in Computer Science at North Carolina State University since August 2008. He joined the Network Performance Research Group under the guidance of Dr. Mike Devetsikiotis in January 2009. He also submitted a paper on Performance Modelling of Virtual Collaborative Environments at the IEEE 4th Workshop on Enabling the Future Service-Oriented Internet: Towards Socially-Aware Networks in July 2010. He is currently working towards his thesis defense and would graduate in December 2010...
ACKNOWLEDGEMENTS

I would like to express sincere gratitude towards Dr. Michael Devetsikiotis for giving me an opportunity to work with him and for his constant guidance in my research work. I would also like to thank Dr. Do Young Eun and Dr. Laurie Williams for expressing interest in my work and for serving on my thesis committee.

I would like to express special thanks to Dr. Michael Kallitsis for his insightful suggestions and his valuable support towards my thesis. I would also like to thank all my friends and colleagues: Kauti, Mihir, Nadu, Ritesh, Sreekanth, Vikas, Kedar, Santhosh, Deepak, Ashwin, Shrikant, Pradeep, Deepti, Keerthana, Yan and Ioannis for a fun-filled and a memorable two year journey towards my Masters degree.

Finally, I am extremely grateful to my parents for their belief in me and for their limitless support and motivation.
# TABLE OF CONTENTS

List of Tables ........................................................................... vii  
List of Figures ......................................................................... viii  

Chapter 1  Introduction ............................................................... 1  
  1.1 Overview ........................................................................ 1  
  1.2 Motivation ....................................................................... 2  
  1.3 Introduction to Virtual Worlds .............................................. 3  
    1.3.1 Avatars ...................................................................... 4  
    1.3.2 Presence: The In-World Factor/Feeling......................... 4  
    1.3.3 OpenSimulator .......................................................... 6  
    1.3.4 The Wonderland environment ...................................... 6  
  1.4 Contribution .................................................................... 7  
  1.5 Related Work ................................................................... 9  
  1.6 Thesis Organization .......................................................... 10  

Chapter 2  The Wonderland-Matlab experiment .......................... 11  
  2.1 Introduction ..................................................................... 11  
  2.2 Objective ......................................................................... 12  
  2.3 Experiment Design ............................................................ 12  
    2.3.1 The Matlab test ......................................................... 14  
    2.3.2 Experiment monitoring setup ...................................... 14  
    2.3.3 Survey ...................................................................... 16  
  2.4 Summary .......................................................................... 17  

Chapter 3  The OpenSim Experiment ........................................ 18  
  3.1 Introduction ..................................................................... 18  
  3.2 OpenSim Configuration servers .......................................... 19  
    3.2.1 Core Server ............................................................. 19  
    3.2.2 Voice Server ............................................................ 20  
  3.3 Overview of the World ...................................................... 20  
  3.4 The Maze ........................................................................ 21  
    3.4.1 The Maze Automation scripts ................................... 21  
    3.4.2 Integration of voice .................................................... 23  
    3.4.3 Automating the experiment with bots ......................... 23  
    3.4.4 The SQLite Database ............................................... 25  
  3.5 Varying the control parameters .......................................... 25  
    3.5.1 Simulation of the communication between participants .... 25  
    3.5.2 Bandwidth variation .................................................. 27  
    3.5.3 CPU load variation ................................................... 29  
  3.6 Summary ......................................................................... 29
<table>
<thead>
<tr>
<th>Chapter 4</th>
<th>Metamodelling: Framework and Case Studies</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>30</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Introduction to metamodelling</td>
<td>30</td>
</tr>
<tr>
<td>4.2</td>
<td>The Metamodelling methodology</td>
<td>31</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Design of the experiment</td>
<td>33</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Choosing an appropriate model</td>
<td>33</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Fitting the model</td>
<td>34</td>
</tr>
<tr>
<td>4.3</td>
<td>Problem Formulation</td>
<td>34</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Problem Statement</td>
<td>34</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Inputs</td>
<td>35</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Outputs</td>
<td>36</td>
</tr>
<tr>
<td>4.4</td>
<td>Case Studies</td>
<td>36</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Case Study 1: The maze with textual message traffic</td>
<td>37</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Case Study 2: The maze with voice communication traffic</td>
<td>39</td>
</tr>
<tr>
<td>4.5</td>
<td>Optimum Allocation of Resources</td>
<td>43</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Cost Functions</td>
<td>46</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Optimization Problems</td>
<td>47</td>
</tr>
<tr>
<td>4.6</td>
<td>Performance Evaluation</td>
<td>47</td>
</tr>
<tr>
<td>4.7</td>
<td>Summary</td>
<td>48</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 5</th>
<th>Conclusion</th>
<th>49</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Summary</td>
<td>49</td>
</tr>
<tr>
<td>5.2</td>
<td>Future work</td>
<td>51</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Conducting the experiments with human subjects</td>
<td>51</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Resource optimization with the marketing point of view</td>
<td>51</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Analysis of the VCE with actual voice transmissions</td>
<td>51</td>
</tr>
</tbody>
</table>

| References  |                                      | 52 |

<table>
<thead>
<tr>
<th>Appendix</th>
<th></th>
<th>55</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appendix A</td>
<td></td>
<td>56</td>
</tr>
<tr>
<td>A.1</td>
<td>Configuring the OpenSim core server</td>
<td>56</td>
</tr>
<tr>
<td>A.2</td>
<td>Configuring the OpenSim voice server</td>
<td>58</td>
</tr>
<tr>
<td>A.3</td>
<td>Scripting code - Opensim</td>
<td>59</td>
</tr>
<tr>
<td>A.3.1</td>
<td>LSL example code snippets</td>
<td>59</td>
</tr>
<tr>
<td>A.3.2</td>
<td>Bandwidth control code</td>
<td>61</td>
</tr>
</tbody>
</table>

vi
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 4.1</td>
<td>Bots to % CPU transformation</td>
<td>35</td>
</tr>
<tr>
<td>Table 4.2</td>
<td>Case 1: Chat Message Commands</td>
<td>39</td>
</tr>
<tr>
<td>Table 4.3</td>
<td>Case 2: Using voice command simulation</td>
<td>42</td>
</tr>
<tr>
<td>Table 4.4</td>
<td>Results for time minimization</td>
<td>48</td>
</tr>
<tr>
<td>Table 4.5</td>
<td>Results for cost minimization</td>
<td>48</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

| Figure 1.1 | The image shows us what an avatar looks like inside the virtual world SecondLife | 5 |
| Figure 1.2 | The image above gives us a glimpse of the wonderland world. We can see two avatars standing in front of the screen with the matlab exercise. | 7 |
| Figure 2.1 | Here we see three VNC screens embedded inside the virtual world. | 13 |
| Figure 2.2 | The CPU utilization measurements | 15 |
| Figure 2.3 | The network utilization graph | 15 |
| Figure 2.4 | The survey questionnaire | 16 |
| Figure 3.1 | Here we see a picture of experimental maze from the front | 22 |
| Figure 3.2 | The image shows us the path a packet follows through the Netfilter hooks | 28 |
| Figure 4.1 | Sending messages and receiving acknowledgements | 38 |
| Figure 4.2 | Linear response surface for chat communication | 40 |
| Figure 4.3 | Non-linear response surface for chat communication | 41 |
| Figure 4.4 | Linear response surface for chat and voice communication | 44 |
| Figure 4.5 | Non-linear response surface for chat and voice communication | 45 |
| Figure A.1 | Distance calculator code snippet | 60 |
| Figure A.2 | Bandwidth limiter code snippet | 61 |
1.1 Overview

Virtual reality has become one of the hot topics of the decade. The concept which Hollywood movies have been projecting since decades has finally started entering into reality. Movies like James Cameron's Avatar make us yearn to be in the hero's place and to explore the new world through the mind while being physically in the same place. We can surely say that virtual reality has made its impact on people's imagination and the interest for such get-away places is increasing. Technologically, we are still to truly arrive at such a dream world but virtual worlds like SecondLife [9], Kaneva [3], Pseudospace [6] etc. have surely taken the first step towards it. Through your computer screen, you can enter into another world where you have a different identity and where you can meet people across continents and play games and even fly! So what are these worlds about? Worlds like these constitute physical web spaces where people are represented as 3-Dimensional entities called avatars and are part of a 3D environment. Though such environments are rendered on two dimensional computer screens today, this field offers tremendous scope for development.
Virtual spaces like these are also regarded as collaborative working environments. Unlike voice and video conferences, these enable perception of the surrounding individuals as tangible entities inside the virtual workspace. They thus provide capabilities to share information such as desktop along with the provision of traditional video/voice communication. This thus enables a shift of paradigm from the traditional geographical collocation or the proximity paradigm to the virtual collocation ideology where professionals collaborate with one another regardless of their geographical location.

Social presence inside such worlds is a big factor in determination of a world being enjoyable and feeling real. One may define presence inside an environment as the ability of the domain to make its users feel comfortable and truly inside it. The inhabitants need to be able to immerse themselves into the world and feel like they are a part of it. Virtual presence which is a parallel term for virtual environments, is thus one of the key factors which can help us determine the level of user experience and acceptability of these environments.

1.2 Motivation

Combined with the the high potential encompassed by these virtual spaces for social interaction, the computing and networking needs of these arenas are also growing at an accelerated pace. It is thus necessary for us to be proactive in optimizing the processing power and networking assets at hand while making sure that we do not compromise on the social networking aspect of these work spaces. A small change in the computing parameters can spoil or enhance the user experience. The kind of objects the user interacts with inside the world and the quality of their depiction can change the way a student understands a remotely taught lesson. Analysis of the social aspect of this problem adds a new facet to the analysis of the performance of this framework.

It is the social aspect which dictates the path of the development of the virtual world and we need to model the performance characteristics according to the preferences of the subject i.e. the user. Analysis of such a world needs us to create a framework for the user. Analysis of the the perceptions of the users is a step necessary for modeling the performance parameters and also the framework. This
framework then provides us with a base to model the performance characteristics. Modulation of these characteristics can enable us to study these environments at a greater depth.

High bandwidth networks and processors play an important role in defining and controlling the quality of this term *virtual presence*. It is evident that possessing the best networking qualities and good processors can solve all our problems. But in the advent of this growing demand for resources, we need be judicious in using them. We thus need to obtain a rigorous understanding of the resource requirements and performance demands of these platforms in order to resourcefully tackle the challenge.

### 1.3 Introduction to Virtual Worlds

Virtual worlds are considered by many as the next step in helping collaboration among team members and in businesses. A virtual world can be thought of as an electronic environment that visually mimics complex physical spaces where people are represented as animated characters known as avatars. These avatars can interact with each other and with virtual objects inside these physical web spaces. This collaboration can be carried out though the means of video and voice/text chat. Virtual worlds such as Wonderland by SUN Microsystems also allow the integration of VCL (*Virtual Computing Lab*) screens through VNC (*Virtual Network Computing*: a graphical desktop sharing system).

Virtual worlds are currently being researched by many for their collaborative aspects as they are a prospective step towards remote interactive meetings. Even though they are represented as 3D worlds, virtual reality worlds are still rendered on two dimensional screens today and hopefully will find a better rendering medium in the near future. Virtual worlds provide a platform for group meetings which in the absence of a physical meeting place are currently done through remote connection software which provides for audio and video. These worlds on the other hand provide for a virtual space to hold the remote meetings generating a feeling of a virtual but 'physical' presence of ones fellow teammates through the medium of avatars. The level of satisfaction which can be derived by a user through such an interaction as compared to the one derived through live face to face conversations is still under study by various groups through human subject experiments in these
1.3.1 Avatars

Each participant is rendered as an entity with a human form also called as an avatar. Avatars add
to the virtual but physical existence of a participant in the virtual world thus making him a tangible
entity inside the world as opposed to other forms of collaborative mediums of interaction. Avatar
rendering can play an important role in such an environment. The closer an avatar resembles a
participant, the higher is the satisfaction and group efficacy, more is the sense of presence in the
group meeting and also higher is the group cohesion. In the environments where we conduct our
experiments, avatars are not custom made and we have the default 'Ruth' avatar in OpenSim and a
randomly generated avatar inside Wonderland. Both the experimental virtual worlds endow us with
the ability to modify the appearance of the avatars in terms of body structure, skin tone, hair styles,
type and color of clothes worn etc. We also have some virtual worlds which try to morph the avatar's
face to match that of the user, input with the help of a digital image.

A number of virtual worlds have emerged in the past few years. Examples include 'Active worlds',
'Kaneva', 'Second Life', 'Small Worlds', 'Onverse' etc. All these environments allow a user to have
avatars of their own, their own virtual space and a means to interact with fellow inhabitants. Worlds
like SecondLife allow the user to build structures of their own, trade their creations and buy stuff with
virtual money i.e., Linden Dollars inside SecondLife (created by Linden Labs).

1.3.2 Presence: The In-World Factor/Feeling

The goal of improving the performance of virtual worlds revolves around finding the right proportion
of the amount of resources allocated and the user experience. The user experience which is just as
vital as the optimum allocation of resources provides the X-factor in determining the quality of a
world. The user experience, also called the in-world factor is what makes users feel as if they are a
part of this new world.
Figure 1.1: The image shows us what an avatar looks like inside the virtual world SecondLife
1.3.3 **OpenSimulator**

OpenSimulator is a 3D application server which can be used to create a virtual environment similar to SecondLife and which can be accessed through a variety of clients. It allows users to log into the world with their own avatars and then explore the world. It is a world very similar to SecondLife allowing people familiar with SecondLife feel like home inside it. OpenSim allows developers to delve into the internals and design new stuff which they can then give out for free or sell for in-world money - Linden dollars. Being an opensource software is one of the primary reasons for us to choose to work on it - allowing us to play with the code for customizing our experiments.

OpenSimulator being very similar to Second Life, incorporates in it a scripting engine as well in addition to the other common to SUN Wonderland. The scripting engine can compile and run scripts embedded inside objects and can perform trivial tasks programmed by the users. A voice server can also be installed in addition to the main server. It is available through a module by the name FreeSwitch which works in parallel on the same port. The conversation can be as natural as in the real world i.e. you can listen to high quality audio and the magnitude of voice gets attenuated as you move further from the source of the sound giving you a real world feeling.

As mentioned earlier, a major difference between the worlds of Wonderland and OpenSim is that OpenSim provides a GUI and a scripting API to its users for object creation, a feature that the SUN environment has not explored in depth. Due to its cross compatibility with Second Life, one may use the Second Life clients to access and program inside OpenSimulator worlds.

1.3.4 **The Wonderland environment**

SUN Wonderland is a 3-D environment which is a toolkit provided by Sun for developing applications for virtual collaborative environment. It is a world where each person has his own avatar through which he can interact with other people. With this avatar, he can move about in the world and explore the default rooms. These default rooms can be developed as per needs to provide other sources of interaction like our experiment does. The experiment has a computer screen (VCL) embedded inside the world and is used for working on a mathematics exercise by the subjects which are undergraduate
students enrolled in a course ECE 220 at the NC State University.

A user when logged on to the world is given his own avatar using which he can move about in the world and interact with other users. Created mainly for organizing virtual meetings which can help eliminate the physical presence limitation, the concept is aimed squarely at replicating both the real-world experience of sitting in a meeting room and the experience of sharing on-screen presentations and having private back-channel conversations while watching a public presentation.

1.4 Contribution

In this thesis, we build a platform for measurement and analysis of the performance characteristics of virtual worlds equipped with high speed networking capabilities. The performance of the server is noted by varying technical factors which are the available resources such as the CPU load and the
network bandwidth. We also have a social factor for the experiment. In one of the case studies, we work with human subjects: students from the ECE 220 class taught in the NC State University. The second case study involves simulation of social traffic with the use of programmable bots. The second case study, in which we work with the OpenSimulator environment, forms the core of the thesis. Here, we vary the traffic inside the world via programmed bots and simulate and evaluate 2 modes of communication between the bots. We collect data for the second case study and after analysis we try to fit an appropriate model using the metamodeling methodology. This socio-technical metamodel is used further to derive a cost-performance relationship for the framework. This cost model is used for formulation of optimization problems for optimum resource allocation for the Virtual Collaborative Environments (VCEs). This framework can act like a metric or benchmark tool for assessing the quality of experience of participants inside a virtual world. Though the framework is designed for SecondLife [9] oriented environments, it can be extended to other virtual spaces as well.

We can summarize the contributions as follows:

- Creating a framework for the human subject experiments inside SUN Wonderland.
- Developing monitoring tools for the Wonderland environment.
- Building a platform which can act as a metric for analyzing the performance of virtual worlds.
  1. Building the maze platform using toolkits inside OpenSim including the Linden scripting language and C#.
  2. Generating response surfaces using the metamodeling methodology which give us an understanding of the resource dependencies.
  3. Using the response surfaces to formulate and solve cost and time optimization problems for our proposed metric.
1.5 Related Work

The development of Virtual worlds which looks like a recent occurrence probably has its roots in the first Multi-user Dungeon (MUD) concept of the late 1970s. These simple text based or simply animated games like Adventure and Zork acted like a spark-plug to the technological developments which followed. Research on virtual worlds can be classified into a number of fields. An example of an organizational/social research aspect can be seen in [10] which presents an avatar business value analysis and a tool for managing value creation in these worlds. Virtual gaming has also been one of the traditional motivational areas for development of virtual worlds with massively multi-player online role playing games like World of Warcraft and Counter Strike having a large audience. [14] focusses on resource allocation in such massively multiplayer games and aims towards a fair allocation metric. [31] and [15] also give emphasis to resource allocation in such environments with regards to the population of the world and cost estimates for the resources. A more recent development is the application of the virtual world concept to the field of education, which can be seen in [11] and [19]. Virtual presence has been studied in [20] where the authors stress upon the importance of presence in educational collaborative environments through experimental evaluations. An example of the research being conducted in the individual/social area is the article about User Acceptance of Virtual Worlds [22] which assesses the reasons to use social-research oriented virtual worlds, in particular Second Life. Through their empirical analysis, we can see that factors such as collaboration and co-operation play a crucial role in influencing user intention and acceptance of such worlds.

We can see the development of tools for virtual world event analysis in [24] in which the creators at Metaverse Business focus on analyzing popularity and presence of the virtual work-spaces. We, in our framework focus on the networking and processing power aspect of the server while managing virtual presence. Research is also been followed in private firms who own 3D worlds like Second Life [9], Kaneva [3] etc., education focussed virtual worlds like ScienceSim [8], Quest Atlantis [7] etc. and OpenSimulator based virtual worlds like Cyberlandia [2] and Pseudospace [6]. There also are institutions like Open Source Metaverse Project [5], the Croquet Project [1] etc., who have invested in virtual world building and development tools. Finally, it is research in areas like collaboration and
education which raises challenges for computer and information sciences and which suggests new potential for development.

1.6 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 gives a brief overview of the case study of the Wonderland environment and the Matlab experiment conducted inside it with the ECE 220 course participants. Chapter 3 describes the OpenSimulator environment and the design of the framework for the experiment. Chapter 4 continues the discussion in Chapter 3 with an overview of the metamodelling methodology followed by the metamodel formulation, case studies and finally the problem formulations for optimum resource allocation and some examples using those problems. Finally, Chapter 5 concludes the thesis with future work.
The Wonderland-Matlab experiment

2.1 Introduction

SUN Wonderland is a virtual environment designed by SUN Microsystems for developing creating applications for collaboration inside virtual work spaces. It is believed that a 3D environment is a more natural way to organize multiple simultaneous conversations. The special layout of the 3D world coupled with immersive audio provides strong cognitive cues that enhance collaboration. MPK20 is a sample virtual world which is built using the open source toolkit provided by the Wonderland application. The toolkit provides the capabilities for creating the world, avatars and embed other media inside it. Every user has his own avatar which he uses for interacting with other entities which may be other avatars or objects with scripts or embedded media. A few default rooms have been provided in MPK20 for use and can be used for organizing meetings or seminars. These default rooms can be developed as per needs to provide other sources of interaction like our experiment does. The Wonderland experiment comprises of a VCL(Virtual Computing Lab) computer screen of a remote server embedded inside the world with the help a VNC (Virtual Network Computing) client. The
experiment would be described in the upcoming sections.

A user when logged on to the world is given his own avatar using which he can move about in the world and interact with other users. Created mainly for organizing virtual meetings which can help eliminate the physical presence limitation, the concept is aimed squarely at replicating both the real-world experience of sitting in a meeting room and the experience of sharing on-screen presentations and having private back-channel conversations while watching a public presentation.

2.2 Objective

The experimental setup consists of a framework for supporting collaborative work inside Wonderland, a toolkit provided by SUN for creation of virtual workspaces. The objective was to setup Virtual Computing Laboratory (VCL) screens inside Wonderland with the help of VNC clients for the purpose of conducting human subject experiments with the help of students enrolled in the course ECE 220. The students had to complete a matlab exercise collaboratively on the screen inside the world and monitoring the entire activity was also a part of the objective.

2.3 Experiment Design

The experimental design involved setting up the environment with VNC screens which could incorporate VCL images located on a remote server. The VCL images were to have the Matlab software installed on them beforehand. Figure 2.1 shows a VNC screen with the Matlab instance open inside it. The setup enabled the students to make changes simultaneously to the same content on the screen. The participants were allotted time slots in which they could login as pairs and work together on the exercise. The means of interaction for the participants was text and voice communication. A special emphasis was placed on the collaborative aspect and students were not permitted to submit work which was done by a sole participant.
Figure 2.1: Here we see three VNC screens embedded inside the virtual world.
2.3.1 The Matlab test

The Matlab test comprised of a simple question set consisting of six items which the students had to complete within a stipulated amount of time. Collaborating with each other was an important aspect of the exercise as one of the main objectives of the whole experiment was to measure the virtual presence of the environment and the experience of the participants while solving the exercise.

2.3.2 Experiment monitoring setup

Scripts were written for measuring the technical aspect of the experiment. The scripts included are:

1. User login recorder script: A script for recording the identity of the users logged in along with their time of entry and exit. The script scanned the logs recorded by the server process for certain patterns which listed the names of the users. These names were then used for searching for the login and logout times of the users who were present in the virtual world.

2. Utilization Measurement script: A script for measuring the CPU and memory utilization of the server process on the server machine. The above script scanned the logs recorded by the server process for its process id (pid) and also pids of the child processes spawned by it. The process identifiers were then supplied to the top command which listed the utilization over specific time intervals. We can see an example of the result of the script in Figure 2.2.

3. Network utilization measurement. The *tcpdump* command on Linux was chosen for measuring the network utilization. The amount of network traffic related to the Wonderland experiment was measured by collecting all the packets flowing in and out of the server machine. This data was collected and stored on an external storage disk due to its large size. The analysis of these packets was to be done with the means of *Wireshark*. An example of the results obtained in this case can be seen in Figure 2.3.
Figure 2.2: The CPU utilization measurements

Figure 2.3: The network utilization graph
2.3.3 Survey

The main subjects in this experiment were students from the ECE 220 class. The collaborative experience of these students while completing the Matlab exercise inside the world was to serve as a metric for measurement of the social aspect of this experiment. This was to be completed through a survey via Survey Monkey by Dr. Mitzi Montoya (Dept of Business Management, NC State University) and was to measure the students’ view of the collaboration achievable and its effects on their exercise. The survey enquires about the experience of the participants inside the world and also while collaborating with their partners. A snapshot of the survey document can be seen in Figure 2.4.

Figure 2.4: The survey questionnaire
2.4 Summary

The Wonderland experiment was aimed at obtaining an understanding of the concept of virtual presence and of how the variation of technical parameters of the experiment can play a significant factor in the overall experience of an individual inside such collaborative environments. Unfortunately, the analysis of the data has not been completed and we proceed to the next chapter which explains the experiment inside OpenSimulator and in which we have more control over the input and the outputs.
CHAPTER 3

The OpenSim Experiment

3.1 Introduction

In Chapter 1 we had a look at what constitutes a virtual world and its dynamics. Chapter 2 gave us a brief overview of the Wonderland environment and also an understanding of how server performance and the factor of virtual presence are related. Here, we take a look at another similar environment called OpenSimulator which is much more versatile as compared to our previously encountered virtual world. OpenSimulator as previously described is an open-source software tool created with stark similarity to Second Life which is much more widely known. With the back-end code being entirely in C# as compared to Wonderland which uses Java, it also incorporates a scripting engine, giving its developers much more control in fiddling with the development of the user-side of the world. In this chapter, we take a look at how the environment was setup for the experiment to take place. We also take a look at the configuration servers, the designing of the maze which includes the automation scripts, automated bots etc., and finally the variation of the control parameters with the help of which we carry out the experiment.
3.2 OpenSim Configuration servers

An OpenSim configuration consists of regions run by region simulators and 5 core back-end services (which manage users, the grid, assets, inventories, and grid-wide messaging). The server can be configured to run in two different modes: Standalone mode or Grid mode. An OpenSim server running in Standalone mode is one which runs everything (all the above mentioned services) in a single executable. It is suited primarily for situations where very few users are using the system. In the Grid mode, the five services (User, Grid, Asset, Inventory and Messaging) are not part of the region server. Instead they can be run separately on either the same machine or can be spread our across different computers. Here our primary executable (OpenSim.exe) acts solely as the region server serving one or more regions which communicate with the core servers for inventory, asset and other management functions. We have used the server in the Standalone mode as we are not using it for testing on a very large scale and because of the core requirement of running just one region and not multiple ones.

3.2.1 Core Server

The core server was installed as a standalone version as mentioned above on our machines which had Ubuntu 9.0 installed. The server communicated with external machines via port 9000 for all kinds of traffic. We had to create a region for the server to deploy and our server could run only one region at a single time. Designing the maze involved designing the root region for deploying the structure. The region design was done with the help of Photoshop and in-world design tools. Most of the designing for the maze was done using the clients designed for SecondLife servers - SLClient and HippoOpenSimViewer.

The installation and configuration of the OpenSim core server has been explained in Section A of the Appendix.
3.2.2 Voice Server

The voice server is an independent entity which we install supplementary to the main standalone server which assists us in carrying our voice communications between the user-avatars inside the OpenSim world. The voice server module by the name 'FreeSwitch module' is primarily available for a 32bit machine and for both windows and Linux environments. Voice capability was initially supplied by Linden Labs along with OpenSim in a module called Asterix and by SecondLife in the form of SLVoice.exe. The new FreeSwitch module is much easier to install in the number of dependency installations and also to maintain in terms of previous extensions.

3.3 Overview of the World

OpenSimulator is an environment very similar to the well known SecondLife. OpenSim is an open-source project and has its own internal scripting language known as Linden Scripting Language (LSL). It also incorporates code snippets written in commonly known languages such as C#, Visual Basic and JavaScript in addition to LSL. The scripts are interpreted by the in-built OpenSim interpreter. The project has a large and constantly growing API for LSL and has enthusiasts contributing to the project in their own way for eg. constructing structures such as houses, writing scripts for clocks and creating objects with animation. The world allows developers to design objects through the OpenSim client and to place them in the world. The objects support embedded scripts and give the developer better control inside the world in dictating certain events happening in the region. The world provides special channels which can be used for communication between avatars(default channel - 0) and other 4,294,967,293 channels which can be used for private communication between the user-created objects and/or other user avatars. The objects use these channels for message passing as required by their embedded code for completion of various functions and duties. Every entity in the asset server is represented by a Universally Unique Identifier or UUID which is a 16-byte (128 bit) number - standardized by the Open Software Foundation (OSF) [4].
3.4 **The Maze**

The Maze remains the core entity of the experiment with all other factors designed with the construction of the maze in mind. The precursor for the maze comprised of sample creations such as walls, doors etc., with *LSL* scripts embedded in them. We had a number of different scripts being written and employed for creating effects such as walls which open and close at fixed time intervals, objects which react when touched, walls which emit light when the virtual sun inside the world sets etc. The scripts in the final version of the maze up for the experiment are the time measurement script, the distance recorder script which has been described later, the script which controls the open-close of the doors restricting re-entry into the maze after the experiment has started and prohibiting anyone from entering through the exit door before the experiment starts.

3.4.1 **The Maze Automation scripts**

As mentioned above, the maze includes a number of automation scripts which help us keep a track of the various measurement parameters of the experiment. These include the Time calculator script, the distance measurement scripts and the door open/close script.

3.4.1.1 **Time Calculator**

The Time Calculator scripts are embedded inside objects at the start and the end of the maze. The script starts with a simple timer object which is triggered through an avatar-object collision event when an avatar enters the maze. This signal is sent over a private communication channel to the main time tracker script embedded in another object. The time keeps ticking till the avatar exits the maze and the timer is stopped and at this point, the elapsed time is also stored inside a file with the help of one of the *Mazebot* instances which will be elaborated in 3.4.3.1.

3.4.1.2 **Distance Monitor**

The distance calculator code is similarly embedded inside objects in the maze. These objects are invisible to the eye and are spread out all over the maze only to be triggered when an avatar walks over
Figure 3.1: Here we see a picture of experimental maze from the front
them. As soon as the event is triggered, the co-ordinates of this object are passed over to the main
distance calculator object and it finds out the relative distance between itself and the last received
signal thus adding on to the distance traversed by the avatar since its inception into the maze. The
total distance traveled by the avatar is evaluated at each point and can be found by querying the
Distance Monitor object located on the top floor of the maze.

3.4.1.3 Magic Door

We have automated the entry and exit doors of the maze as well. It is primarily to restrict anyone
access inside the maze through the exit before the experiment starts and to disallow the person lost
in the maze from leaving the maze through the entry gate. As the avatar enters the maze, a collision
object signals the entry of the avatar and the exit door is opened via a signal sent through one of
the communication channels. A similar event is scheduled at the end of the ongoing instance of the
experiment in which the entry door is opened and the exit door is closed when the avatar exits the
maze.

3.4.2 Integration of voice

The installation and configuration of the OpenSim voice server has been explained in Section A of the
Appendix.

3.4.3 Automating the experiment with bots

The experiment was automated with the help of programmable avatars known as bots. These bots
follow a set of instructions as put down in their programming logic and perform iterations around the
maze with us varying the control parameters: bandwidth and CPU load. The bots are an extension of
the OpenSim executable `pCampbot.exe` which necessarily is a template for creating dummy bots for
use inside the world. The programming for the bots was essentially done in C# and forms the crux of
the automation section. This segment of code is responsible for extracting the time taken by the bots
to traverse the maze. The C# code accommodates commands from the Linden Scripting Language
which are the ones used for transacting inside the OpenSim environment. The `pCamphbot` template was modified accordingly to create two different types of bot objects essentially the bots responsible for being a part of the maze traversal, the `Mazebots` and the bots responsible for adding to the overall traffic in the environment i.e. the `jumpingBots`.

### 3.4.3.1 The Main bots: Mazebot and Guidebot

The Maze bots and Guide bot are the ones who are responsible for the maze traversal. The first one traverses the maze and the second guides or issues instructions to prior. The instructions are issued in the form of text chat most of which are commands to go to certain physical co-ordinates inside the maze. The traversing bot is responsible for completing the tasks issued and for sending back acknowledgements to the guide bot. The procedure thus continues till the entire maze traversal is completed. The time taken for traversing the maze using this strategy is expected to vary as we change the total load on the CPU and the amount of bandwidth assigned to a certain instance of the world.

### 3.4.3.2 The Dummy bots: JumpingBots

The jumping bots are the ones who serve the purpose of adding to the traffic inside the maze and to the overall system. Their assigned task is to jump around the maze and generate text traffic in the form of text chat which they broadcast inside the world. We use these bots as a way to linearly increase the load on the OpenSim server and as a form of varying the CPU assigned. It has been noted that the variance in the number of jumping bots floating around has a certain effect on the overall curve of the traversal time as shown in Chapter 4. The OpenSim client employs a very good user interface through which we can design objects and set properties. External editors such as `LSLEditor` are available which allow us to write code off-line and embed it inside objects for testing. The design of the maze was made with the thought of generation of collaboration in mind. Here we have 2 participants: one who traverses the maze and another who guides the person traversing the maze. The guide is able to see the maze from the top floor and guides the traverser to take the correct paths out of the maze.
3.4.4 The SQLite Database

The OpenSim server also includes with it a database where it stores the assets, user information, region information etc. The database is created, maintained and used by the ongoing instance of the world. Most of the data stored inside the database is about the users registered with the region, information about the assets i.e. the objects present inside the region (their access permissions, their features, links to embedded scripts etc.)

3.5 Varying the control parameters

The control parameters for our experiment are the bandwidth allocated and the amount of CPU utilized by a certain instance of the world. For simulation purposes, the effect achieved by variation of the CPU available to the server is provided by increasing the internal load on the OpenSim server. It has been explained in more detail in Section 4.

3.5.1 Simulation of the communication between participants

Bots were used to simulate the experiment in place of actual users. It was hard for us to generate a conversation between the bots in terms of voice commands. An idea which we came up with was to simulate the voice traffic in this case by transferring files between the bots while they exchanged instructions through instant messages. The idea was to replicate the occupation of channel bandwidth through file transfer instead of filling it with voice data. Depending on the length of an instruction, we propose a relative variation in the size of the file transferred so as to simulate the fluctuation in voice activity. The true randomness of voice traffic was not replicated by a voice generator algorithm as it was found that the generated samples do not generate a noticeable difference in the expected outcomes.

Here we have two flavors to the experiment. Firstly we conduct the trials with traffic generated by the instant messages sent from one bot to another for completing the maze and also the messages sent between the random jumping bots which add to the load on the CPU. In this case, the conversation
carried out between the bots is represented in the form of instant messages.

The second part of the trials, as suggested above are conducted with simulation of voice. Since it is cumbersome and time consuming to create and process voice commands between the bots, we use file transfer to represent the bandwidth occupied by the voice during the process. Here, at each command sent from one bot to another, we transmit files of definite sizes. Now by varying the size of the files in an instance, we are able to replicate (to a certain degree) the effect achieved by variation in the quality of voice channel used for communication. Both these case studies have been explained in detail in Section 4.4.

3.5.1.1 Implementation details

The OpenSim installation supports an asset server which in the standalone mode is managed by the sole server and by a separate machine in the grid mode. The asset server maintains a track of all the objects belonging to a particular avatar. Many of these objects are provided along with the installation whereas there may be a few created by the user himself. The objects created by the user include LSL Scripts, image files for creation of textures for particular in-world objects, sound files to play in the case of a supporting voice server implementation etc. Every entity in the asset server is represented by a Universally Unique Identifier or UUID which is a 16-byte (128 bit) number - standardized by the Open Software Foundation (OSF). Every such entity is stored by the server either in one of two repositories - one common to all users and one private to the current user. These assets can be loaded in to the world either by copying the assets into the folders in server back-end and or by uploading the corresponding files through an OpenSim client. In the former case, it is necessary to correctly link a couple of xml files which enable the server to correctly find and load the new assets added into the world. Uploading the new objects using an OpenSim client is much easier and more convenient. A specific UUID needs to be assigned to every object when we create it using the xml method but the UUID is automatically generated in the latter method. A time consuming task was to code the file transfer in this case for which the UUID of the file being transferred turned out to be different when viewed in the client browser and when searched by the bot program through the inventory node.
traversal. The corresponding code snippet has been supplied in the appendix section of this thesis for future reference. The size of the files being transferred were varied so as to simulate the different voice traffic qualities. – put this in the results section as we design the model for the experiment.

3.5.2 Bandwidth variation

Another core parameter of our model is the allowed server bandwidth. The OpenSim server communicates with its clients through a limited number of ports. These are five in number in the case of the grid mode of the server and only one in the case of the standalone server which is port number 9000. All information in the case of the standalone server passes through this endpoint. Throttling the flow of data to the server can be achieved by making sure that the packets on this port are limited to a certain value.

3.5.2.1 Trickle - Bandwidth limiting tool

Trickle is a process level bandwidth limitation tool which was used by us for a considerable part of the testing phase after which we came up with a custom tool for limiting the bandwidth on a specific port. The tool works by taking advantage of the unix loader. It provides to the application, a new version of the functionality required to send and receive data through sockets. The traffic limitation is based on delaying the sending and receiving the data over a socket. The work on Trickle can be found here [18].

3.5.2.2 Limiting port bandwidth

We propose another approach of limiting bandwidth on a particular port as opposed to limiting the traffic for a particular program or process. The process starts with finding out the ports on which a particular application functions which was port 9000 in our case. We mark the packets coming to that port using the Netfilter framework and then pass those packets onto a Linux utility called the traffic controller (tc). The traffic controller is the responsible for throttling the packets coming to it by the means of the hierarchical token bucket algorithm which it utilizes.

Netfilter as mentioned above is a framework (a part of Iptables) that provides hook handling within
the Linux kernel for intercepting and manipulation of network packets. These hooks allow us to capture packets, work on them and insert them back onto the channel via which they are able to reach their original destination. The **Netfilter** utility provides us with three tables: NAT, Mangle and Filter. Each table consists of 5 chains essentially Pre-routing, Post-routing, input, output and forward. With the help of those chains, we are able to manipulate the contents of the packets or in our case just mark them with a specific identifier. In our tool, we have used the Mangle table for capturing the packets with the help of the Forward and Input chains. The chains as mentioned above are used for marking them with a particular identifier if they are destined for or leaving our machine through the port numbered 9000. These marked packets are inserted back on the channel and allowed to reach the application process. Here we capture them again using the traffic controller utility (**tc**). The traffic controller utility works as follows. We create a class for each genre of packets we want to operate on. The class created in our earlier step can be passed on to another utility function belonging to ‘**tc**’

**Figure 3.2:** The image shows us the path a packet follows through the **Netfilter** hooks
called as 'filter'. This filter utility function captures those marked packets and applies on it one of the many output algorithms defined for it. In this case we use the hierarchical token bucket algorithm which employs a certain queueing model and manages the throttling for us.

### 3.5.3 CPU load variation

Variation of the CPU load is one of the core parameters of our model. It gives us a measure of the response of the system when we selectively increase the background noise. This background noise represents the numerous background processes which will be assumed to run while a server instance is running. By selectively varying the total background noise traffic in combination with the other parameters, we obtain a graph of the system response over a range of values.

The CPU load variation is implemented with the help of a set of dummy bots whose function is to jump about in the world and increase the inherent traffic by sending text chat messages to each other. As expected, the more the number of dummy bots we insert into the world, the more is the load on the server and the experiment in general. These bots have been programmed in C# by modifications to the pCampbot code provided in the OpenSim package. The concept has been explained in Section 4.3.2 of Chapter 4.

### 3.6 Summary

The chapter gives us an overview of the background work required for creating the framework necessary for carrying out the experiment. The tools and methodologies used for monitoring of the experiment which are explained above give us a base for proceeding to the next stage of the experiment. In the next chapter, we use and build upon the concepts and tools explained here and advance to the simulation stage. Here, by running detailed simulations, we obtain readings which are further given as input to a process called metamodelling. This is explained in detail in Chapter 4.
4.1 Introduction

Chapter 4 rests on the analysis of the data obtained by running the simulations based on the experimental setup explained in Chapter 3. It begins by providing a brief introduction to the metamodelling approach of approximating system performance. The case studies which have been briefly mentioned in Chapter 3 are elaborated upon further and through detailed analysis of the observations and the methodology described below, we proceed towards building of suitable models. These models, which give us response surfaces are further used for the formulation of optimization problems for cost and time minimization.

4.1.1 Introduction to metamodelling

Metamodelling techniques are quite popular in the domain of engineering design for enhancing the efficiency in simulation and optimization of design systems involving simulation programs. Simulation models are usually the preferred way for system analysis as they are able to capture the
system dynamics very closely. It is when we use computationally expensive simulation models that it becomes imperative to look for a more practical approach of performance evaluation. Metamodelling, which is an approach of building approximation models is a strategy in which we build a mathematical closed form model of the simulation model. It thus provides an alternative approach through which we can find an approximate mathematical representation of the system performance and can replace the expensive simulation model.

Metamodelling has been studied with regards to computer simulation modeling for over the past three decades even though the term metamodelling was coined in 1987 by Kleijnen [25]. We can find good study resources for this topic in [26] and [17].

Response Surface Modeling (RSM) is the one of the most commonly used methodologies for metamodelling. It was introduced by Box and Wilson in 1951 in their seminal paper [12] on this topic. In their work, followed by their explanation of the concept, they also describe the applications of this concept to chemical processes. RSM has also been discussed in review papers by Hill and Hunter (1966) [21] and Mead and Pike(1975) [27]. The topic has grown over the decades and the techniques have become the topic of entire books as in Box and Draper [13] and Khuri and Cornell [16]. A comprehensive review of the methodology and its applications from 1966 to 1988 has been given in Myers et al. [28].

4.2 The Metamodelling methodology

The Metamodelling methodology is a way of analyzing and deriving a relation between the set of input (CPU and Bandwidth) and output (Time) parameters. We use this method to come up with a mathematical model called a Response Surface Model (RSM) to represent this relation and subsequently a response surface which maps together, the input and output values from our experiments.

RSM is the process of creating a polynomial model that relates the input-output behavior of the simulation model as a "black box" [29] and which can be used to obtain an approximate mathematical representation for our problem. [23] suggests this approach for being able to derive explicit mathematical relationships between the inputs and outputs of a such an experiment. A simulation model is
generally used to approximate the behavior of a system as it can simulate the dynamics of the system in greater detail when compared to an analytical model. Simulation models can be thought of as simple functions that transform input parameters into output values. We however require a closed form response function to correctly understand the behavior of a system. We thus turn to the method of metamodelling which can help us build such a closed form expression and can help us fulfill the purpose of creation of the simulation model by effectively relating the output data of the simulation model to its input parameters. If we have a computational equation representing the input-output relationship of a system as:

\[ Y = f(X_1, X_2 \ldots X_k) \]

We can realize it as a metamodel, which in its basic form would be represented as:

\[ Y_G = G(X_1, X_2 \ldots X_k) \]

\[ Y = Y_G + \epsilon \]

where \( Y_G \) stands for the model and \( \epsilon \) represents both measurement and approximation errors. The most common way to approach metamodelling can be given as follows:

1. Use the experimental design to simulate a number of efficient computer runs and generate output performance data.

2. Choose a metamodel to represent the observed values i.e., the data.

3. Analyze the output data and fit an appropriate model to it.
4.2.1 Design of the experiment

An experiment design consists of constructing the experiment through a series of detailed trials. There are a number of ways of doing this, examples of which would be the factorial analysis, orthogonal arrays, the D-optimal method; which is used for the realization of a stochastic process, the select by hand method which is used in neural networks etc. One of the most basic experimental designs and the one which we have used in the analysis is the full factorial analysis. This analysis depends on the number input of factors and the number of levels which we evaluate the input factors at. In our case, the CPU load and bandwidth allocation represent the two factors and we evaluate them at four levels each.

4.2.2 Choosing an appropriate model

The next step is the choice of an appropriate model for approximation of the mathematical and functional relationship between the inputs and outputs. The most commonly used models are response surfaces, neural networks, inductive learning models etc. We are going to use the response surface modeling approach which can be described using polynomial equations of the first order, first order with interactions between its parameters and so on. The standard mathematical polynomial model for a response surface can be written down as:

\[ M(X) = \beta_0 + \sum_{i=1}^{k} \beta_i X_i + \sum_{i=1}^{k} \beta_i X_i^2 + \ldots + \sum_{i=1}^{k} \sum_{j=1}^{k} \beta_{ij} X_i X_j + \ldots \]  

(4.1)

This model is based on the observed data and is an empirical model.

We can further derive first and second order models with interactions from the general form given above. The equation of the form first order with interactions would be:

\[ M_{L}(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_{12} X_1 X_2 \]  

(4.2)

Here, \( X \) stands for the vector of input factors which are defined below.
$X_1$ stands for the number of bots, $X_2$ for the bandwidth while the $\beta$s represent constants to be determined.

Further, we can also define the second order equation for forming a model as:

$$M_{NL}(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 + \beta_4 X_2^2 + \beta_5 X_1 X_2$$

These equations are used in the following sections to analyze the observations and to derive response surfaces from them. Further reading material on the topic can be found in [30].

4.2.3 Fitting the model

The conclusive step is to fit the data to the model to approximate the input-output relationship. The fitting method varies according to the choice of the model to be built. (Here we try to build a Response Surface Model.) The usual steps employed by the RSM involve first/second order polynomial equations with factor interactions and least square regression analysis. Other models like neural networks use back-propagation to fit the data.

4.3 Problem Formulation

4.3.1 Problem Statement

The problem can be described as representing the data obtained from the experiments inside the OpenSimulator world as a response surface (as described earlier) and to derive explicit relationships between the input and the output factors. Finally, we need to derive a relationship between the amount of CPU and bandwidth allocation and a proposed cost function.

Since we need to develop an understanding of the relationship between the amount of resources required and the performance of the virtual world, We follow the steps in Section 4.2 to build an appropriate model, determine its fit using the R-square value and obtain the corresponding response surface for both cases.
4.3.2 Inputs

The inputs as discussed earlier are the variation of the allocated bandwidth and the CPU load. Readings are taken for these two factors at four levels each. The bandwidth is assigned in magnitudes of 16, 32, 64 and 128 kbps and the CPU load is varied in the form of bots from 20 bots to 80 bots in increments of 20 units. The stress on the server due to the introduction of these bots represents the overall load and the server performance is seen to decrease as we increase these dummy jumping bots.

Due to unfortunate circumstances, CPU limiting tools like cpulimit have been proven incompatible with our OpenSim platform so we have had to adhere to alternative methodologies for controlling the available CPU for serving the virtual world needs. The CPU load variation is achieved with the introduction of these additional dummy bots as mentioned above, whose function is to jump around the world and broadcast random text messages. Varying the number of these bots instantiated at a particular time gives us a variance in the CPU load. Table 4.1 shows the mapping between the amount of dummy bots and the available CPU cycles dedicated to our server. The transformation function between the number of bots and the allocated CPU cycles is shown below:

\[ \lambda = \alpha + \beta \ast Bots \]  

(4.4)

Here, \( \lambda \) stands for the percentage of available free CPU cycles and \( \alpha \) and \( \beta \) are constants. Experimental values for these constants in our case turned out to be \( \alpha = 105.6 \) and \( \beta = -1.45 \).

<table>
<thead>
<tr>
<th>#Bots</th>
<th>CPU cycles (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>74</td>
</tr>
<tr>
<td>30</td>
<td>63</td>
</tr>
<tr>
<td>40</td>
<td>52</td>
</tr>
<tr>
<td>50</td>
<td>32</td>
</tr>
<tr>
<td>60</td>
<td>17</td>
</tr>
</tbody>
</table>
4.3.3 Outputs

The outputs of the experimental simulations are readings which represent the amount of time needed for completion of each run. The variation of the input factors has an effect on the time for traversal and we note this effect for gauging the pattern of the response of the server to our input sequences. The measured time readings are observed to range from around 180 sec to 380 sec for the input values provided. The observations have been noted and described in section 4.4 for two different cases, one in which the communication between the bots is assumed to proceed using textual instant messages and another where they are to communicate using something similar to voice communication.

4.4 Case Studies

The framework containing the maze is completely based inside the virtual world OpenSimulator. As described earlier, there are two bots which form the subject of the experiment. One bot who is responsible for guiding i.e. giving directions to another bot and the other one who is responsible for interpreting those instructions and reacting appropriately. These directions are sent through the means of instant messages from the guide bot to the bot inside the maze whom we may call as the lost bot. These messages are to be interpreted correctly by the lost bot and the trial should culminate into a successful completion of the maze. The time for completion for each trial is measured from the point at which the lost bot teleports to the start of the maze till the finishing point of the maze while successfully interpreting and correctly executing the transmitted instructions. As mentioned in the factorial analysis earlier, we have a number of levels which we use for obtaining the observations. We vary the CPU load and also the allocated bandwidth of the server and observe the effect of the combinational variation has on the output in each scenario.

We thus change the processor load and the allocated bandwidth and note their effects on the performance of the server and on the time taken to complete the maze. This maze traversal time is expected to give us an understanding of the relation between the allocated resources and the time for maze completion.
4.4.1 Case Study 1: The maze with textual message traffic

This case study describes the section of the experiment in which we conduct the experiment with the medium of information exchange being only textual instant messages. As mentioned above, the chat messages exchanged by the bots contain instructions for execution of specific actions. These actions include moving from one location to another on the positional grid of the virtual world by walking to that location, using teleportation to arrive at a certain location on the map i.e., without walking to that spot. (This particular instruction is used at the start of the traversal when we want to align the lost bot to the start of the maze.), instructions for correct receipt of an instruction and also for completion of a particular one. Each message of instruction, as mentioned above, is followed by an acknowledgement message which represents successful completion of that particular individual activity i.e. of going from one location to another. An example of a chat message would be Go 156 128. The two numbers following the word Go represent the location of the destination point with respect to the X and Y axis of the virtual world. The third axis which is the height of the bot above the ground is assumed to remain the same and hence this piece of information is not transmitted in the instant message.

4.4.1.1 Observations

The observations from the experiment for this scenario have been shown in Table 4.2. We take an average of 15 readings per combination of CPU and the allocated bandwidth for determination of each sample trial result. The observations for the above case as mentioned above are obtained by running the simulations of maze traversal with the medium of exchange being only textual instant messages. The observed values, which represent the total time for completion of a single maze traversal, depict an increasing trend as we increase the number of bots i.e., decrease the amount of assigned CPU cycles and show a decrease in magnitude as we increase the amount of allotted bandwidth.

4.4.1.2 Choosing and fitting the metamodel

We approach the problem by choosing an appropriate metamodel.
Figure 4.1: Sending messages and receiving acknowledgements
### Table 4.2: Case 1: Chat Message Commands

<table>
<thead>
<tr>
<th>Bandwidth</th>
<th>Bots#20</th>
<th>Bots#40</th>
<th>Bots#60</th>
<th>Bots#80</th>
</tr>
</thead>
<tbody>
<tr>
<td>16kbps</td>
<td>239.05</td>
<td>253.52</td>
<td>297.94</td>
<td>348.2</td>
</tr>
<tr>
<td>32kbps</td>
<td>226.32</td>
<td>241.05</td>
<td>284.05</td>
<td>346.19</td>
</tr>
<tr>
<td>64kbps</td>
<td>202.05</td>
<td>226.71</td>
<td>283.62</td>
<td>319.73</td>
</tr>
<tr>
<td>128kbps</td>
<td>194.62</td>
<td>216.39</td>
<td>273.13</td>
<td>314.69</td>
</tr>
</tbody>
</table>

The evaluation of the first order model described in Eq. (4.2) gives us the following resulting equation:

\[
M_{L}^{1}(X) = 189.22 - 0.37X_1 + 1.9X_2 + 0.002X_1X_2 \tag{4.5}
\]

The analysis of variance (ANOVA) evaluation of the above model, Eq. (4.5) gives us an R-square value of 0.9529. The closer the value of R-square to 1, the better is the model.

We further proceed towards the evaluation of the second order model shown in Eq. (4.3). The new model results in a much higher R-Square value of 0.9854. We thus choose the second model which shows a much better fit, for our RSM. By plugging \(\beta_0 = 238.525, \beta_1 = 0.106, \beta_2 = -0.949, \beta_3 = 0.018, \beta_4 = 0.004\) and \(\beta_5 = 0.002\), we can obtain the final equation for our experiment as follows (see Figure 4.3).

\[
M_{NL}^{1}(X) = 238.525 + 0.106X_1 - 0.949X_2 + 0.018X_1^2 + 0.004X_2^2 + 0.002X_1X_2 \tag{4.6}
\]

### 4.4.2 Case Study 2: The maze with voice communication traffic

The following case study describes the simulations of the experiment with the medium of exchange of information between the two bots being simulation of voice traffic in addition to chat messaging. Here, in order to have the effect of voice communication, we transmit files between the bots for each
Figure 4.2: Linear response surface for chat communication
Figure 4.3: Non-linear response surface for chat communication
instant message transferred between them. These files occupy extra bandwidth in addition to the occupation of the channel by the textual messages. Since it was tedious to actually transmit voice, and interpret the sent conversational clip inside the virtual world, we have preferred to use the file transfer scheme.

### 4.4.2.1 Observations

The observations for this scenario have been given in Table 4.3. These readings are also obtained by averaging about 15 readings per sample. The information transmitted in this case constitutes both textual messages and file transfer. The textual messages account for the actual instructions and the file transmission adds to the bandwidth occupancy of the transfer channel. The observed values, which represent the total time for completion of a single maze traversal, depict an increasing trend as we increase the number of bots, i.e., decrease the amount of allocated CPU cycles and show a decrease in magnitude as we increase the amount of allocated bandwidth.

### 4.4.2.2 Choosing and fitting the metamodel

We form the metamodels for Case 2 as done in Case 1, checking the efficacy of fit of both the first and second order equations as given in Eq. (4.2) and Eq. (4.3). The R-square value of the first order model for the second case based upon the ANOVA results is 0.9669. The equation obtained is as follows (see
Upon evaluating the second order model as represented in equation (4.3), we find that this model has a much better fit with an R-square value of 0.9776. Thus evaluating the model further, we come up with values for our factors, $\beta_0 = 149.314, \beta_1 = 4.154, \beta_2 = -0.225, \beta_3 = -0.013, \beta_4 = 0.002$ and $\beta_5 = -0.009$. Thus the resulting surface for Case 2 can be obtained as shown in Figure 4.5.

$$M^2_{L}(X) = 169.69 + 2.8X_1 + 0.04X_2 - 0.009X_1X_2$$

$$M^2_{NL}(X) = 149.314 + 4.154X_1 - 0.225X_2 - 0.013X_1^2 + 0.002X_2^2 + 0.009X_1X_2$$

ANOVA analysis of the observations shows us a 3% dependence on bandwidth, 32% dependence on the CPU, 1% variations due to interaction and 64% unexplained variation. Focussing on the dependence of the performance on CPU and bandwidth, we still see a major inclination of the server towards allocation of the CPU cycles as compared to the bandwidth.

### 4.5 Optimum Allocation of Resources

Here, we propose our pricing models for CPU and Bandwidth and introduce our optimization problems. The first optimization problem minimizes the performance metric of our VCE (i.e., the time traversal) subject to resource and budget constraints. The second optimization problem minimizes the cost of operation subject to performance guarantees (i.e., time traversal constraints). The decision variables in both cases are the allocated bandwidth and CPU.
Figure 4.4: Linear response surface for chat and voice communication
Figure 4.5: Non-linear response surface for chat and voice communication
4.5.1 Cost Functions

In this section, we describe our cost functions. The CPU allocation and bandwidth (the two input factors) are used for deriving the cost function. The cost for the bandwidth could be derived with the help of market values for Internet services. Here is the proposed form:

\[ P_{bw} = \gamma + \delta \times \log_{10}(\phi_{bw}) \]  

(4.9)

where \( P_{bw} \) is the cost for the allocated bandwidth, \( \phi_{bw} \) represents the input bandwidth in kbps and \( \gamma \) and \( \delta \) are constants whose values were found by fitting the above equation to the data obtained from an internet service provider in our area as, \( \gamma = -27.944 \) and \( \delta = 11.402 \). Alternatively, we could use a linear pricing model which can be shown as:

\[ P_{bw} = \phi_{bw} \times p_{bw} \]  

(4.10)

where, \( p_{bw} \) represents the pricing parameter for bandwidth allocation.

Correspondingly, we propose a cost function for the CPU allocation. The cost function can be written down as:

\[ P_{CPU} = \phi_{CPU} \times p_{CPU} \]  

(4.11)

where \( P_{CPU} \) is the total cost for the CPU, \( \phi_{CPU} \) is the allocation of CPU in cycles and \( p_{CPU} \) is the pricing parameter for CPU.

The total cost \( P_{Total} \) is thus:

\[ P_{Total} = P_{CPU} + P_{bw} \]  

(4.12)
4.5.2 Optimization Problems

The optimization problems can be stated as follows.

\[
\begin{align*}
\min_{\phi_{CPU}, \phi_{bw}} & \quad \text{Completion Time}(\phi_{CPU}, \phi_{bw}) \\
\text{Subject to} & \quad P_{Total} \leq M,
\end{align*}
\]

where \( P_{Total} \) is the total cost and \( M \) is the monetary budget. Here, we aim to minimize the time needed for completion of the maze as a function of the CPU load and the bandwidth while we are given a certain monetary budget and completion time could be any any of the response surface proposed in Section 4.4.

The second problem involves minimizing the cost for running the server given that the time for completion of the maze is less than a given time threshold.

\[
\begin{align*}
\min_{\phi_{CPU}, \phi_{bw}} & \quad \text{Cost}(\phi_{CPU}, \phi_{bw}) \\
\text{Subject to} & \quad \text{Completion Time} \leq T_{\text{threshold}}
\end{align*}
\]

where \( T_{\text{threshold}} \) is the maximum possible time for completion.

4.6 Performance Evaluation

In Table 4.4, we solve the optimization problem shown in Eq. 4.13 where the cost function is the linear one shown in Eq. 4.10 and Eq. 4.11. The pricing parameters are \( p_{CPU} = 1000 \) CPU pricing units and \( p_{bw} = 5 \) bandwidth pricing units. The results show that with a higher budget, we can have a higher allocation of CPU/BW that leads to superior performance.

Table 4.5 shows the results for cost optimization for the problem shown in Eq. 4.14. The equation for the relation between time for traversal and the input factors can be seen in Eq. 4.6 with the cost factors being the same as in the evaluation for time optimization. We see that as our performance
Table 4.4: Results for time minimization

<table>
<thead>
<tr>
<th>Budget ($)</th>
<th>Bandwidth (kbps)</th>
<th>CPU cycles (%)</th>
<th>Time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
<td>0.5</td>
<td>358</td>
</tr>
<tr>
<td>150</td>
<td>4</td>
<td>12</td>
<td>341.69</td>
</tr>
<tr>
<td>350</td>
<td>19</td>
<td>25</td>
<td>283</td>
</tr>
<tr>
<td>500</td>
<td>29.8</td>
<td>35</td>
<td>263</td>
</tr>
<tr>
<td>800</td>
<td>51</td>
<td>54</td>
<td>229</td>
</tr>
<tr>
<td>1000</td>
<td>65</td>
<td>67</td>
<td>211</td>
</tr>
<tr>
<td>1500</td>
<td>101</td>
<td>99</td>
<td>184</td>
</tr>
</tbody>
</table>

Table 4.5: Results for cost minimization

<table>
<thead>
<tr>
<th>Time Budget (sec)</th>
<th>Bandwidth (kbps)</th>
<th>CPU cycles (%)</th>
<th>Cost ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>185</td>
<td>99.55</td>
<td>97.25</td>
<td>1470</td>
</tr>
<tr>
<td>220</td>
<td>58.31</td>
<td>60.47</td>
<td>896</td>
</tr>
<tr>
<td>250</td>
<td>37.5</td>
<td>41.9</td>
<td>607</td>
</tr>
<tr>
<td>280</td>
<td>20.88</td>
<td>27.9</td>
<td>375.4</td>
</tr>
<tr>
<td>310</td>
<td>6.6</td>
<td>14.35</td>
<td>176.5</td>
</tr>
<tr>
<td>335</td>
<td>1</td>
<td>2.36</td>
<td>28.7</td>
</tr>
</tbody>
</table>

metric (i.e., time completion) is relaxed the cost decreases but at the same time the resource allocation decreases as well.

4.7 Summary

The chapter gives us an understanding of the relationship between the input and output parameters through a response surface created with the help of a metamodel. It assists us in the formulation of the proposed metric which can be used for evaluating the performance capabilities of a Virtual Collaborative Environment (VCE) setup on a high speed network. The cost optimization model derived with the help of these response surfaces can be customized according to the specifications of the Virtual Collaborative Environment (VCE) and thus can be used for the budget estimation of such platforms.
Conclusion

We conclude the thesis with a brief summary of the work conducted by us on the topic and some related future work.

5.1 Summary

Virtual worlds as we know are growing at a tremendous pace and so are the computing requirements of such environments. It is thus imperative that we be proactive in optimizing the networking assets and processing power at hand while making sure that we do not compromise on the social networking aspect of these work spaces. A small change in the technical parameters can change the user experience. Analysis of the social aspect of this problem adds a new facet to the analysis of the performance of this framework. The kind of objects the user interacts with inside the world and the method and quality of their depiction can dictate the way a DE student understands a lecture taught remotely.

As we have seen earlier, it is the social aspect that dictates the path of the development of the virtual
world and that we needed to model the performance characteristics according to the preferences of the subject i.e. the user. We needed to analyze such a world in order to create a framework keeping the user's preferences in mind. Thus, analysis of the perceptions of the users was a step necessary for modeling the performance parameters and also the framework. The proposed framework provided us with a base to model these performance characteristics and modulation of these characteristics enabled us to study these environments at a much greater depth.

We realized that high bandwidth networks and processors really play an important role in defining and controlling the quality of this term: *virtual presence*. It is obvious that providing the best networking qualities and good processors can give us the best performance. But in the advent of this growing demand for resources, it is necessary we utilize them in an optimum way and that being judicious holds the key. We thus have gone ahead with our design of the framework to obtain an understanding of the performance demands of these environments.

Summarizing our work, we know that the construction of a framework for the performance analysis of VCEs with high speed networks was the core topic of the thesis. We took a brief introductory glimpse of the Wonderland environment through the experiment in Matlab. Performance monitoring tools were developed for the Wonderland platform as a step towards developing an awareness of the resource utilizations of the server. It also gave us an understanding of the concept of virtual presence. We proceeded to another environment by the name OpenSimulator where we had more control on the inputs and results as compared to the initial example. Here, our first step was to create a maze inside the environment with the help of built-in toolkits and custom scripts. This was developed further to be used for a start to end traversal (of the maze) and thus for performance evaluation. The metamodelling methodology helped us transform the output results of the traversals into response surfaces which gave us an idea of the overall response of the system with regards to the resources used. Finally, we used the results to formulate equations for resource optimization with respect to time for completion of the task and also with respect to the cost of the allotted resources. The metric which we have proposed thus aims at expressing the relationship between the input resources and the performance of these platforms.
5.2 Future work

In this section, we propose a number of areas for future work in the topic of performance analysis of VCEs using networking capabilities.

5.2.1 Conducting the experiments with human subjects

The framework which was originally designed with the engineering of these experiments with human subjects in SecondLife (in the place of bots). Conducting the trials with human subjects could give us a new insight in this domain.

5.2.2 Resource optimization with the marketing point of view

Detailed analysis of the cost and time optimization with regards to the performance metric of VCEs could be carried out with respect to the marketing aspect of the area.

5.2.3 Analysis of the VCE with actual voice transmissions

Since current analysis in terms of voice transmissions involved only a simple simulation of the voice traffic, as future work, one can repeat the experimental case studies with actual voice transmissions between the bots instead of the transfer of files.
REFERENCES


[14] Luis Diego Brice no, Howard Jay Siegel, Anthony A. Maciejewski, Ye Hong, Brad Lock, Mohammad Nayeem Teli, Fadi Wedyan, Charles Panaccione, Chris Klumph, Kody Willman, and Chen Zhang. Robust resource allocation in a massive multiplayer online gaming environment. In


APPENDIX
A.1 Configuring the OpenSim core server

We start with the basic installation guidelines for configuring the server. The steps are given below. 1. The OpenSim installation file can be downloaded from [put the reference here]. We extract the file on our machine and run the runPrebuild.bat file in the top directory. 2. We next open the resulting .sln file with visual studio and build it in place or run the compile.bat file. 3. Running nant in the same subdirectory will build the file. 4. We start the server with the command - Replace SYSTEMIP with the ip address of your server.

Post-Installation.

1. Inside bin directory, we make a copy of the OpenSim.ini.example file as OpenSim.ini
   Making walls opaque
   Open the file OpenSim.ini
   Search for the keyword physics and change its value to physics = OpenDynamicsEngine.
   This solidifies in-world objects.

2. If no one other than the localhost is able to login to the server, we need to make changes to
Regions.ini or default.xml located inside the directory /bin/Regions
Changes: change the internal IP address to 0.0.0.0
confirm that SystemIp is the IP address of the system.

3. Flattening the terrain

   \texttt{change region <Region\_Name>}
   \\
   \texttt{terrain fill 21}

4. Loading and saving the terrain

   \texttt{load oar <Terrain\_Name>}
   \\
   \texttt{save oar <Terrain\_Name>}

5. Disable authentication checks

   Inside OpenSim.ini, modify value of authentication to:
   \\
   \texttt{authentication = false}

6. Changing the gravity of the region to correct the posture of bot and avoid bent legs. Inside
   OpenSim.ini, change \texttt{av\_capsule\_standup\_tensor\_linux = 1700000} from its old value - 500000.

7. Changing the landing point. To make sure that the inception point of all bots does not coincide
   with the maze.

   (a) In each region, go to the place where you would like yourself and others to arrive at.

   (b) From your viewer go to \textit{About Land} (by clicking on the Region name at the top of your
       viewer screen in the middle of the menu bar).

   (c) click on the \textit{Options} tab.

   (d) set the \textit{Teleport Routing} to \textit{Landing Point}.

   (e) press the \textit{Set} button. Do the same for each region.
A.2 Configuring the OpenSim voice server

Voice capability can be installed on the OpenSim server through the installation of a module called FreeSwitch.

The FreeSwitch module is available externally for download and is available for 32bit systems. The installation steps are as follows.

1. Install FreeSwitch by executing the FreeSwitch binary downloaded.

2. Open 
   `/FreeSwitch/conf/autoload_configs/modules.conf.xml`

3. Uncomment `<load module = "mod_siren"/>`.

4. Uncomment `<load module = "mod_xml_curl"/>`.

5. Save the file `modules.conf.xml`.

6. Open 
   `/FreeSwitch/conf/autoload_configs/xml_curl.conf.xml`

7. Change the configuration to match `http://opensimulator.org/wiki /FreeSwitch_Module` and save it.

8. Open `FreeSwitch/conf/vars.xml`.

9. Add a new codec to `global_codec_prefs:G722132000h`, save the file.

10. Open and setup `OpenSim.ini`.

11. Search for the `[FreeSwitch]` section and make `enabled=true`.

12. Make sure all instances of `ip.address.of.freeswitch.server` are modified to equal your server's ip_address, save the file.

13. Start FreeSwitch.

14. Start the OpenSim server.
15. Start the client and go to user preferences.

16. Uncheck the enable voice chat option.

17. Go to estate options and make sure that the allow voice chat option is checked.

18. Go to About Land, click on the media tab and check the Use a private spatial channel option.

19. Go again to user preferences, voice chat tab and check the Enable Voice chat option.

20. Voice chat should be enabled inside OpenSim now and you can use it by pressing the talk button on the bottom right of your client.

A.3 Scripting code - Opensim

The OpenSimulator environment uses its own internal scripting language for programming objects inside the world. For eg. If your avatar comes up against a door inside the world and you want to open the door to entire or exit the structure, clicking on the door can activate a script which would open the door for you. Such scripts are embedded in virtual objects inside these environments.

A.3.1 LSL example code snippets

A.3.1.1 Distance Monitor

The distance calculator code is similarly embedded inside objects in the maze. These objects are invisible to the eye and are spread out all over the maze only to be triggered when an avatar walks over them. As soon as the event is triggered, the co-ordinates of this object are passed over to the main distance calculator object and it finds out the relative distance between itself and the last received signal thus adding on to the distance traversed by the avatar since its inception into the maze. The total distance travelled by the avatar is evaluated at each point and can be found by querying the Distance Monitor object located on the top floor of the maze. The code snippet has been illustrated in fig A.1
vector lPos = <135,114,22.6>;
//Modify acc to start position <133.29,114.081,20.906>
float dist_traveled = 0;
integer listen_handle;
default
{
    state_entry()
    {
        listen_handle = llListen(12,"","","");
    }
}
//Listen for distance broadcasts
listen(integer channel, string name, key id, string message)
{
    if(message != "reset distance")
    {
        vector pos = (vector)message;
        float xDist = pos.x - lPos.x;
        float yDist = pos.y - lPos.y;
        dist_traveled += llSqrt(xDist*xDist + yDist*yDist);
        llSay(0,"Distance traveled: "+(string)dist_traveled);
        lPos = pos;
        if(pos.z == 1 && xDist==0 && yDist==0)
        {
            dist_traveled = 0;
        }
        else if(pos.z == 1 && xDist!=0)
        {
            //llShout(0,"Distance traveled: "+(string)dist_traveled);
        }
    }
    if(message == "reset distance")
    {
        dist_traveled = 0;
        lPos = <135,114,22.6>;
    }
}
// If someone clicks on the object, display the last noted distance.
touch_start(integer total_number)
{
    // Display the distance.
    llSay(0,"Last measured Distance:"+(string)dist_traveled);
}

Figure A.1: Distance calculator code snippet
A.3.2 Bandwidth control code

Bandwidth limitation was done by throttling the packet arrival and departure rate on a certain port on the system. The packets were marked using the Netfilter module and then their rate of flow was restricted using the code snippet displayed in Figure A.2.

```
#Create the root handle for qdisc
sudo /sbin/tc qdisc add dev eth0 root handle 1:0 htb

#Add a class with the rate
sudo /sbin/tc class add dev eth0 parent 1:0 classid 1:1 htb rate 128kbit
#Add a filter with the class id of the above class
#sudo /sbin/tc filter add dev eth0 protocol ip handle 6 fw classid 1:1

sudo /sbin/tc filter add dev eth0 protocol ip handle 7 fw classid 1:1

#Display the created filter
sudo tc -s filter show dev eth0

#Display the created class
sudo tc -s class show dev eth0
```

Figure A.2: Bandwidth limiter code snippet