I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Over the last several years, there has been tremendous growth in online gaming (i.e. playing games over the internet). The Massively Multiplayer Online Role Playing Game (MMORPG) is one type of online game. An MMORPG is played within a virtual world. Users have an in-game representation, called an avatar, that they control. Typically there are over a thousand avatars in the virtual world at one time. Users use client software to connect to an MMORPG server over the internet. If just one server is used then the number of avatars that can be supported in the virtual world at one time is severely limited. In order to overcome this, a multi-server approach is needed. Unlike traditional load balancing and partitioning schemes, which generally use task partitioning, data partitioning is required in this case. This thesis investigates schemes for partitioning and load balancing MMORPG applications on a network of processors. In particular, three different schemes were developed and examined. These are: Static Av, Static MS and Dynamic MS. Static Av assigns avatars to each server, one at a time, as they enter the simulation. Static MS assigns equal sized portions of the map of the virtual world to each server. An avatar is assigned to the server that owns the part of the map that the avatar is "standing" on. Dynamic MS divides the map into many more segments than there are servers. The map segments are dynamically distributed among the servers based on the results of a load balancing algorithm. The thesis details the algorithms and the performance associated with each of the schemes. In summary, Static Av does not perform well, whereas Static MS and Dynamic MS can be used to parallelize MMORPG applications. To the best of our knowledge, this is the first published work that looks at the issue of parallelizing and load balancing such applications.

Acknowledgments

This research was conducted using grants from the Natural Sciences and Engineering Research Council of Canada and Bell Canada University Labs.

I would like to thank my advisor Dr. Ajit Singh for his advice and support. I would also like to thank the members of the Parallel and Distributed Systems group at the University of Waterloo for their input.

Trademarks

SPARCstation and UltraSPARC are trademarks of SPARC International, Inc. and are licensed to Sun Microsystems, Inc. and Sun is a registered trademark of Sun Microsystems, Inc..

Contents

1 Introduction 1

1.1 Area of Research 1

1.2 Research Problem 5

1.3 Scope of Research 5

1.4 Research Objectives 6

1.5 Research Contribution 6

1.6 Outline of this Thesis 6

2 Background 8

2.1 Distributed Interactive Simulation 8

2.1.1 Terminology 10
2.2 DIS Characteristics 11
2.2.1 Dead Reckoning 11
2.2.2 Relevance Filtering 12
2.2.3 Terrain 14
2.2.4 Environmental Effects 15
2.2.5 Performance Monitoring 16
2.2.6 Cost Effective Testing 17
2.3 Problems with DIS 17
2.3.1 Scaling Problems 17
2.3.2 Cheating 18
2.4 Separation of Client and Server 18
2.5 Single Server Approach 19
2.6 Multi-Server Approach 19
2.7 Traditional Scheduling & Load Balancing Problem 20
2.7.1 Dynamic Load Balancing 21
2.7.2 Data Partitioning 21
2.8 Scheduling and Load Balancing for MMORPGs 22
2.9 Multi-Server Approach in Current MMORPGs 23
2.10 Relationship of Current Research to Previous Research 24
2.11 Summary 24

3 Schemes Studied 25

3.1 Static Assignment of Avatars to Servers 26

3.2 Static Assignment of Map Segments to Servers 26

3.3 Dynamic Assignment of Map Segments to Servers 27

3.4 Proposed Dynamic Algorithm 27

3.5 Overall Load Balancing Algorithm 28

3.5.1 Legend and Notes 28

3.5.2 Main Load Balancing Algorithm 29

3.5.3 Core Load Balancing Algorithm 30

3.5.4 Determination of Breaking of Connectedness 31

3.6 Explanation of Algorithm 33

3.6.1 Convergence of Algorithm 36

3.6.2 Termination of Algorithm 37

3.6.3 Run-Time Complexity 37

3.7 Summary 37

4 Experimental Setup 38

4.1 Hardware Configuration 38

4.2 Software Architecture 39

4.3 Design and Implementation 40

4.3.1 Generation of Avatars 41

4.3.1.1 Even Distribution 41

4.3.1.2 Uneven Distribution 42
4.3.2 Parameters Used During Simulation 42
4.3.3 Assumptions 43
4.4 Description of Components 43
4.4.1 Land Manager 44
4.4.1.1 Messages Sent by the Land Manager to a Server 44
4.4.2 Server 47
4.4.2.1 Messages Sent by a Server to the Land Manager 47
4.4.2.2 Messages Sent by a Server to a Super Client 49
4.4.2.3 Messages Sent by a Server to Another Server 52
4.4.3 Super Client 55
4.4.3.1 Messages Sent by a Super Client to the Land Manager 57
4.4.3.2 Messages Sent by a Super Client to a Server 58
4.5 Summary 59

5 Experimental Results and Analysis 60
5.1 Nonrandom and Random Movement 60
5.2 Even and Uneven Distributions 61
5.3 The Four Schemes 61
5.4 Nonrandom Movement Case 62
5.4.1 Even Distribution 62
5.4.1.1 Analysis 63

5.4.2 Uneven Distribution 68
5.4.2.1 Analysis 73
5.5 Random Movement Case 84
5.5.1 Even Distribution 84
5.5.1.1 Analysis 86
5.5.2 Uneven Distribution 90
5.5.2.1 Analysis 95
5.6 Summary 106

6 Conclusions and Future Directions 108
6.1 Conclusions 108
6.1.1 Contribution of the Thesis 109
6.2 Applications of This Work 110
6.3 Future Directions 110

Bibliography 113

List of Tables

5.1 System Configurations for Nonrandom Even Distribution 62

5.2 Response Times for Nonrandom Even Distribution (in seconds) 63

5.3 Total Bytes of Data Received for Nonrandom Even Distribution 63

5.4 Time to Perform LBA for Nonrandom Even Distribution (in seconds) 63

5.5 System Configurations for Nonrandom Uneven Distribution 70

5.6 Response Times for Nonrandom Uneven Distribution (in seconds) 71

5.7 Total Bytes of Data Received for Nonrandom Uneven Distribution 72

5.8 Time to Perform LBA for Nonrandom Uneven Distribution (in seconds) 73

5.9 System Configurations for Random Even Distribution 85

5.10 Response Times for Random Even Distribution (in seconds) 85

5.11 Total Bytes of Data Received for Random Even Distribution 85

5.12 Time to Perform LBA for Random Even Distribution (in seconds) 86

5.13 System Configurations for Random Uneven Distribution 92

5.14 Response Times for Random Uneven Distribution (in seconds) 93

5.15 Total Bytes of Data Received for Random Uneven Distribution 94

5.16 Time to Perform LBA for Random Uneven Distribution (in seconds) 95

List of Figures

2.1 Relevance filtering in DIS configuration with four sites 14

3.1 Four map segments assigned to four servers 27

3.2 The middle MS is owned by j, and there are zero others next to it owned by j 32

3.3 The middle MS is owned by j, and so is one other MS next to it 33

3.4 The middle MS is owned by j, and so are two other MSes next to it. 33

3.5 Map segment to server assignments 36

4.1 A configuration using four servers and three super clients 39

4.2 Process diagram for two servers and two super clients 40

4.3 An even distribution of 16 avatars arranged into 4 rows and 4 columns 42

4.4 Arrangement of groups for uneven distribution 43

4.5 An example script file 56

5.1 Maximum Number of Avatars for Nonrandom Movement, Even Distribution 65

5.2 Response Times for Nonrandom movement, Even Distribution 66

5.3 Bytes Transferred for Nonrandom Movement, Even Distribution 67

5.4 Maximum Number of Avatars for Nonrandom Movement, Uneven Configuration . . 74

5.5 Response Times for Nonrandom Movement, Uneven Distribution, 1 and 2 Servers . 75

5.6 Response Times for Nonrandom Movement, Uneven Distribution, 2 and 4 Servers . 76

5.7 Response Times for Nonrandom Movement, Uneven Distribution, 4 and 6 Servers . 77

5.8 Response Times for Nonrandom Movement, Uneven Distribution, 6 and 8 Servers . 78

5.9 Bytes Transferred for Nonrandom Movement, Uneven Distribution, 1 and 2 Servers 79

5.10 Bytes Transferred for Nonrandom Movement, Uneven Distribution, 2 and 4 Servers 80

5.11 Bytes Transferred for Nonrandom Movement, Uneven Distribution, 4 and 6 Servers 81

5.12 Bytes Transferred for Nonrandom Movement, Uneven Distribution, 6 and 8 Servers 82

5.13 Maximum Number of Avatars for Random Movement, Even Configuration 87

5.14 Response Times for Random movement, Even Distribution 88

5.15 Bytes Transferred for Random Movement, Even Distribution 89

5.16 Maximum Number of Avatars for Random Movement, Uneven Configuration 97

5.17 Response Times for Random Movement, Uneven Distribution, 1 and 2 Servers 98

5.18 Response Times for Random Movement, Uneven Distribution, 2 and 4 Servers 99

5.19 Response Times for Random Movement, Uneven Distribution, 4 and 6 Servers . . . 100

5.20 Response Times for Random Movement, Uneven Distribution, 6 and 8 Servers . . . 101

5.21 Bytes Transferred for Random Movement, Uneven Distribution, 1 and 2 Servers . . 102

5.22 Bytes Transferred for Random Movement, Uneven Distribution, 2 and 4 Servers . . 103

5.23 Bytes Transferred for Random Movement, Uneven Distribution, 4 and 6 Servers . . 104

5.24 Bytes Transferred for Random Movement, Uneven Distribution, 6 and 8 Servers . . 105

List of Algorithms

3.1 Main Load Balancing Algorithm 29

3.2 Core Load Balancing Algorithm 30

3.3 Determining of Breaking of Connectedness 32

Chapter 1

Introduction

1.1 Area of Research

The first significant computer game was Spacewar. It was written for the PDP-1 by MIT student Steve Russell in 1962. In 1972, Nolan Bushnell formed Atari and released Pong, which was the first commercially successful arcade game [9], [19], [20]. As the price of personal computers fell, the popularity of computer games grew. Originally, most computer games were single player based games. Those games that were multiplayer based usually had all players playing on the same machine. As modems increased in popularity, some games started to include the option for two people to play over a modem. Ordinary phone lines were used, and hence ordinary phone charges, particularly long distance charges, applied. As the internet became more popular, some computer games started to include the ability to play over the internet. This eliminated long distance

charges, and allowed more than just two players on separate computers.

Currently there exists a number of computer games known as massively multiplayer online role playing games (MMORPGs). There are also a number of MMORPGs under development. In a "role playing game", one takes on the role of someone (or something) within an imaginary world. Typically, the imaginary world is a fantasy world. Being "online" means that in order to play, you must be connected to a server run by the MMORPG company, usually through the internet. The term "multiplayer" means that there are two or more real humans controlling avatars in the virtual world at the same time. The "massively multiplayer" part, although a qualitative term, is generally taken to mean that it is possible for at least a few hundred other players to be in the virtual world at the same time. The term MMORPG has come to refer exclusively to graphical MMORPGs (i.e. you can see the virtual world around you on your computer screen, as opposed to just seeing text descriptions). A virtual world is a world that exists only inside a computer. Most virtual worlds are very similar to the world around us. Examples of MMORPGs that are currently in operation are Meridian 59 [17], Ultima Online [30], EverQuest [4], and Asherons's Call [1]. Examples of MMORPGs that are currently under development are Ultima Online 2 [31], Hero's Journey [11], and Middle Earth [18]. Many other games are at various stages of their development.

Within the virtual world the player is represented by an "avatar", a Sanskrit word that literally means incarnation. The avatar could take the form of a virtual human, animal, monster, tree, or some other self contained entity. A player's avatar is all that other players see of the player,

and usually they don't know anything about the player in real life. This can be done in ordinary games, but usually the others know who the person behind the avatar is. In MMORPGs, the other players usually don't know who the person behind the avatar is.

Although a player interacts in real-time within a MMORPG, there are no hard real-time constraints. The only constraint is that a player is unlikely to pay to play if the response time exceeds his own personal tolerance. This tolerance will obviously be different from person to person. Also, since the internet is involved, the response time will vary greatly from user to user, and over time.

Although MMORPGs are for entertainment, they are different from general board games and traditional computer games, in a number of ways. Generally one purchases a game and then plays it, with no further costs incurred. With MMORPGs the game is purchased and then a monthly fee is paid in order to play. MMOPRGs require one to have a constant network connection to a server run by the MMORPG company, whenever one is playing the MMORPG. Most games involve less than a dozen players at one time, but some MMORPGs can have over a thousand players at one time. Most games are generally static, and even small changes are infrequent. MMORPGs, however, frequently receive small updates. In fact, the players expect these updates to happen regularly. Most games have an overall goal, and once this goal has been reached, the game is over. MMORPGs have no overall goals, and no clear end. The virtual worlds in MMORPGs are persistent. If any one player leaves, the virtual world goes on without him. If the player returns then the world will be different than when he left (i.e. there is no "save game"

feature).

Cheating is rampant in MMORPGs. In previous non-MMORPG multiplayer games, the character data associated with a user's avatar was stored on the players own computer. Technically savvy players wrote programs to modify this data to artificially improve their avatar. For example, a cheater may give himself a large amount of virtual money. They also distributed these programs to others, resulting in rampant cheating. It was originally thought that by storing all character data on the servers run by the MMORPG company, and using simple encryption to encrypt the connection between the client used by players and the servers, that cheating would be eliminated. However, technically savvy players were able to write programs that got around the simple encryption used by the client by calling the encryption and decryption functions found in the client code. It then became possible to send arbitrary commands to the servers. The servers assumed that the command was valid, even if it would actually have been impossible for the client to send the command. For example, commands could be sent to slowdown, or crash, the MMORPG servers. As a result, all other users are negatively affected. Alternatively, a cheater could perform actions that give his avatar an advantage over other avatars. For example, he could duplicate all of his virtual money several times over, becoming much more wealthy than most other avatars, and in a short period of time. An avatar that has a significant advantage over other avatars becomes a commodity, which can be sold for real money. This showed that it is necessary to check the validity of all incoming data to the server, even if it is validly encrypted. However, small changes are constantly being made to the MMORPG, which inevitably introduces new bugs, and as a result, players are constantly finding new ways to cheat.

1.2 Research Problem

If one server is unable to host all of the avatars then multiple servers must be used. The best way to make use of multiple servers is currently unclear. This thesis attempts to determine how to make the best use of multiple servers. Multiple schemes are examined. The best scheme could depend on the distribution of avatars within the virtual world.

1.3 Scope of Research

The research presented in this thesis primarily addresses the issue of maximizing the number of avatars in a MMORPG. Consideration is also given to response times to commands received from a client, as well as the amount of data exchanged between all the processes involved in running the MMORPG. The internet is purposefully not used since response times could be affected in ways that are beyond the control of the software and hardware used. As such, the response times measured represent the low end of the range of response times that a player using the internet would experience. Relative comparisons are made between various schemes. For consistency between simulation runs, the avatars are simulated (i.e. there is not a real person behind any of the avatars, unlike a real MMORPG).

MMORPGs typically offer a wide range of in game actions and activities. The research software written for this thesis supports only movement and collision detection since this is all that is required. The issue of cheating is only considered insofar as how easily a given scheme supports anti-cheating measures.

1.4 Research Objectives

The objective of this research is to examine different schemes that are suitable for a practical implementation of an MMORPG, and to do relative comparisons of the selected schemes. During the course of the research a new scheme, involving dynamic load balancing, was developed which was compared to three other preexisting schemes. The primary criteria for comparison is the total number of avatars that can be simultaneously supported in the MMORPG. The response time of the software to users' commands, and the amount of data sent between software processes is also examined.

1.5 Research Contribution

MMORPGs have existed for a while. One server can only handle so much load. A multi-server approach is needed. There is no previous work that compares and contrasts different multi-server approaches. There are three major contributions of this thesis:

1. Three different multi-server approaches are compared and contrasted.
2. One of the multi-server approaches is presented for the first time in this thesis. It involves the use of a load balancing algorithm.
3. In order to compare and contrast the three multi-server approaches, a test bed was created.

1.6 Outline of this Thesis

Chapter 2 contains background information on previous schemes used, as well as relevant work

in load balancing. As well, the relationship of the current research to previous research is described. Chapter 3 describes the schemes studied and compared in this research. Chapter 4 describes the experimental setup used. Chapter 5 contains the experimental results and the analysis of those results. Chapter 6 contains the conclusions, and describes possible future research.

Chapter 2

Background

This chapter presents background information related to massively multiplayer online role playing games (MMORPGs). Distributed Interactive Simulation (DIS) protocol is examined. Separation of the client and server is discussed. The single and multi-server approaches are described. The traditional scheduling and load balancing problem is presented. Scheduling and load balancing for MMORPGs is discussed. The multi-server approach currently used in MMORPGs is described. The proposed load balancing issue is presented.

2.1 Distributed Interactive Simulation

Until about 1983, training simulators in the United States Department of Defense were stand-alone high fidelity devices, designed for a single individual, or single crew. The simulators were used

to train an individual or a crew in the operation of military vehicles ranging from tanks to fighter aircrafts. The engineers who designed and built these devices strived for the highest degree of realism. The cost of the simulators was between $1 million and $50 million each. These simulators were of no use for training teams to work at collective or collaborative tasks. To do this training, it was necessary to transport all participating soldiers and equipment to an exercise site to rehearse battle scenarios. During these training exercises, there were usually a small number of casualties, and sometimes some fatalities.

In 1983 the Defense Advanced Research Projects Agency (DARPA) decided to correct this situation. Part of DARPA's mission was to fund high risk/high payoff advanced simulation technologies. DARPA has been responsible for pioneering the invention of such things as stealth radar avoidance, cruise missiles, laser guided weapons, virtual reality, and the internet.

As a consequence, DARPA funded and developed SIMNET (simulator network) to allow collaborative simulation. SIMNET simulators were low cost, low fidelity, simulators that were networked together. Rather than trying to completely and accurately simulate the environment, SIMNET simulators only simulated the environment well enough to allow the required training to be accomplished. These simulators cost $250,000 each. SIMNET spawned the DIS (Distributed Interactive Simulation) protocol, a standard network protocol that enabled *heterogeneous* simulators to interact in a shared synthetic environment. The DIS protocol is defined in [12]. Over its ten year life span, SIMNET was highly successful at achieving collaborative training [6], [29].

The DIS environment consists of networked heterogeneous simulators. Each simulator acts as a client to an individual user or a group of users that control a process in a shared virtual world. Each simulator also acts as server for the overall simulation. In its simplest form, every time a process, controlled by a simulator, performs a task, all other simulators must be informed. The required network bandwidth is $O(n^n)$, where n is the number of simulators. This is the case since each simulator hosts one vehicle, and each time a vehicle's state visibly changes within the simulation, a message must be sent to every other simulator.

2.1.1 Terminology

In the United States Department of Defense, simulation is generally referred to in the context of the acronym "M&S" which stands for *modeling and simulation* or *models and simulations* depending on the context in its use. In this context the following definitions apply [21]:

- **Model**: A physical, mathematical, or otherwise logical representation of a system, entity, phenomenon, or process.
- **Simulation**: A method for implementing a model over time.
- **Modeling and Simulation**: refers to the use of models, including emulators, prototypes, simulators, either statically or over time, to develop data as a basis for making managerial or technical decisions. The terms "modeling" and "simulation" are often used interchangeably.
- **Simulato**r: (a) a device, computer program, or system that performs the simulation; (b) for training, a device which duplicates the essential features of a task situation and provides for direct human operation.

- **Game**: A simulation session in which participants seek to achieve a specified objective given pre-established resources and constraints; for example, a simulation in which participants make strategic game decisions and a computer determines the results of those decisions. This term is synonymous with constructive simulation and higher order model.
- **Monte Carlo Simulation**: A simulation in which random statistical sampling schemes are employed such that the result determines estimates for unknown values.
- **Virtual Simulation**: A simulation involving real people operating simulated systems. Virtual simulations inject human-in-the-loop in a central role by exercising motor control skills (e.g. flying an airplane, performing surgery), decision skills (e.g. committing resources to action), or communication skills.
- **Live Simulation**: A simulation involving real people operating real systems.
- **Constructive Simulation**: A simulation that involves simulated people operating in simulated systems. Real people stimulate (make inputs) to such simulations, but are not involved in determining the outcomes.

2.2 DIS Characteristics

Every DIS simulation has certain characteristics which are discussed in the following sections.

2.2.1 Dead Reckoning

Communication between simulators occurs over a network. Every time even a slight change in position of a simulated entity (e.g. an avatar, or a vehicle) occurs, this information must be sent

to all other simulators participating in the simulation. This limits the number of simulators that can be in the simulation, since network bandwidth becomes saturated. In order to reduce the number of position update messages sent across the network, a dead reckoning algorithm is used. When dead reckoning is used, position update messages are sent less frequently. Since the exact position of an entity is only known when a position update message is received, an "educated guess" as to where the entity is between update messages is made.

Dead reckoning is used as follows. Each simulator keeps track of its own vehicle's precise position and orientation over time. In addition, each simulator maintains a dead reckoning model of the positions and orientations of all other vehicles in the simulation. The dead reckoning models are updated when ever an update message is received. Each simulator also maintains a dead reckoning model of its own vehicle. When the error between the actual and dead reckoned position or orientation exceeds some threshold an update message is sent to all other simulators. The dead reckoning equation is an extrapolation formula [16].

2.2.2 Relevance Filtering

Dead reckoning substantially reduces network traffic. However, there is still a substantial amount of network traffic. The simulators participating in a training exercise must communicate with each other while carrying out the simulation. It is the responsibility of the underlying network to provide each simulator with a reliable and fast mechanism to send and receive the information to the simulated activities. The idea behind relevance filtering is to analyze the semantic content of the state update messages of a simulated entity (e.g. an avatar, or a vehicle) and transmit only the

ones found to be relevant to other entities. The primary use of relevance filtering is to overcome the bandwidth limitations of wide area networks (WANs). A WAN is used to connect together local area networks (LANs). A gateway (computer) links the LANs to the WAN. Each LAN connects together a number of simulators. Relevance filtering is done at the gateways, either at the point of transmission (onto the WAN) or the point of reception (from the WAN). Filtering at transmission is preferred, since irrelevant messages are not sent across the WAN. Each gateway knows the exact location of the entities on the simulators attached to its LAN. Each gateway also knows the approximate location of all the other entities in the simulation. If no local entities are close to an entity on another LAN then all the state update information is filtered out at transmission. If a local entity gets close to an entity on another LAN, then state update messages for that entity are sent through the WAN to that LAN. If a state update message is received, but the entity for which the state update applies is not near a local entity then it is filtered out at reception. Each gateway keeps track of the approximation of position used by the other gateways, and transmits state change information if an error threshold is exceeded. Figure 2.1 shows an example of relevance filtering. A state update message is generated by S at the first site. Gateway G1 receives this message and preforms filtering at transmission. In this case the state update message is sent to gateways G2 and G4, but not G3. Each of gateways G2 and G4 perform filtering at reception and send the message only to those local nodes that need to receive it (denoted by R) [2].

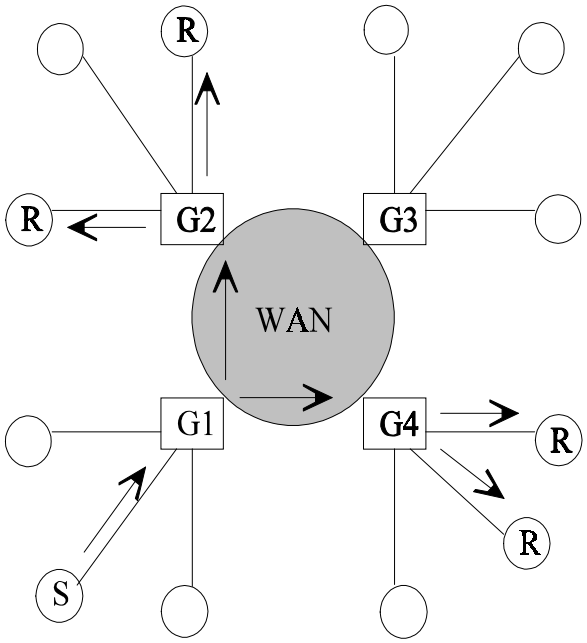


Figure 2.1: Relevance filtering in DIS configuration with four sites

2.2.3 Terrain

In DIS, the terrain is usually modeled after actual terrain found on earth. This is done to increase realism. Getting the terrain information into the simulator is generally a two step process. First the terrain data is acquired and placed in a database. This is a general purpose database of the terrain and is not specific to DIS. Second, the terrain data in the database is converted into a format suitable for a particular DIS simulator. In the process of doing this, the underlying data is altered (i.e. if the data conversion process were reversed, the result would be different than what was started with). Comparisons must be made between the two terrain data sets in order to insure that there are no significant errors. For example, if the original terrain elevation data were accurate to 0.25 meters, and the particular simulator's database had a maximum accuracy of 0.5 meters, then accuracy will be lost when the data is converted into a format suitable for the simulator.

Different DIS simulators are programmed by different teams, and hence have different native formats for terrain data. As a result, it is inevitable that the terrain data in one simulator will be different than the terrain data in another simulator. Comparisons must be made between all the different simulators to insure that differences in terrain data are not significant [22], [25].

2.2.4 Environmental Effects

The course of a simulation can be profoundly affected by the supporting environment data (e.g. the weather model). In 1992, the incorporation of realistic environmental effects into DIS simulators was essentially unknown. In response to this void, the Defense Modeling and Simulation Initiative identified the creation of synthetic environments as a major goal. Synthetic environments provide time- and space-varying information about the terrain, oceans, atmosphere, and near-space to simulations. Six tasks were identified and undertaken: 1) surveys of environmental capabilities and environmental requirements for simulators, 2) an architecture to accommodate environmental effects for distributed simulations, 3) standards for data transfer and related database repository functions, 4) environmental representations for a “level environmental playing field”, 5) environmental effect and processes, and 6) demonstrations incorporating the products of the other tasks. When discussing environmental models and representations the following definitions are used [10], [27]:

- **Embedded Process**: A dynamic process that takes place within the physical environment, but is not a consequence of normal environmental evolution (e.g. smoke, chaff, etc.).
- **Environmental Effect**: The impact that the environment or embedded process has on some component or process in the simulation exercise (e.g. the effect on the performance

of a weapon system, platform or sensor, or other non-visualized combat process).

- **Environmental Feature**: An individual element of the natural physical environment (e.g., rain system, fog, cloud, etc.).

2.2.5 Performance Monitoring

With DIS, there is an opportunity for large scale computer based training. As more simulators interact through DIS, the complexity of the training systems increases. This, in turn increases the work load on the instructor. Students are performing more complex tasks that cannot be viewed at a glance to determine their state. If an instructor has several students to monitor, it is difficult to track what each student is doing at any give time. One solution is to offload work from the instructor by interpreting and evaluating the students' actions via computer programs. This can be accomplished through the use of advanced computer interpretation schemes. This interpretation will present to the instructor information concerning what and how the student is doing, without the need for the instructor to directly monitor the student.

The process of monitoring a student consists of four steps [3]:

1. Determining the goals of the lesson or scenario that are being monitored - i.e. what the student should be doing.
2. Based on the students actions, interpreting what the student is actually doing.
3. Comparing what the student is doing against the goals that were set out in step 1.
4. Providing feedback to the student. In other words, if the student has deviated from the goals, how can the training goals still be met.

2.2.6 Cost Effective Testing

DIS is under consideration as a test tool to support developmental and operational test and evaluation (T&E). DIS is being called upon to enable cheaper, better and faster T&E. DIS has the potential to overcome numerous developmental and operational test and evaluation (DT&E and OT&E) limitations and therefore do DT&E and OT&E better. Test planning using DIS is more expensive than traditional schemes. This is because after the traditional test planning is done, the DIS simulator(s) must be programed with the scenario. Test rehearsal using DIS is expected to cut costs, since expensive resources are simulated by a DIS representation. For test conduct, DIS is expected to be cheaper and it should allow more tests to be run in the same amount of time as traditional methods. It is expected that test evaluation would be more expensive when using DIS. Overall, using DIS is cheaper, but not significantly so, and also allows more testing to be done [13].

2.3 Problems with DIS

There are problems with DIS that prevent it from being used to implement an MMORPG.

2.3.1 Scaling Problems

There are two major scaling problems: bandwidth and cost of equipment. As more simulators participate in a simulation, more bandwidth is required. Because of the DIS architecture, the increase in bandwidth is exponential. The use of bandwidth reduction schemes reduces the total bandwidth used, but it is still $O(n^n)$. Having more users in the simulation requires that there be

more DIS simulators, and hence more cost for equipment. The cost of equipment scales linearly with the number of users.

2.3.2 Cheating

DIS is unsuitable in an entertainment product that is brought into the homes of users and networked with others that have brought a simulator into their own homes. The primary reason for this is cheating. There are three basic ways to cheat using DIS. Selectively ignore incoming messages, illegally modify the environment, and generating impossible out going messages. For example, selectively ignoring incoming messages would allow a user process to become disabled in respect to an opponent's processes and reactions [14].

2.4 Separation of Client and Server

In DIS, there is no distinction between a client and a server. Each simulator fills both rolls. The required bandwidth is $O(n^n)$, where n is the number of simulators. This bandwidth is required since every simulator potentially communicates with every other simulator. As well, cheating is possible, as explained in the previous section. It is not necessary to use this approach to simulate a virtual world. A separate client/server architecture can be used instead. This requires O(n) bandwidth, since each client communicates only with the server, and is oblivious to how many other clients there are. This reduction in bandwidth means that connecting to the server through the internet is now feasible. As well, by having all interactions go through the server, cheating is not possible (assuming that the server software is bug free). A number of corporations have sold

client software to users to run on their own computers, which then connects to servers, maintained by the corporation, through the internet. The corporations charge usage fees to users for the use of the servers.

2.5 Single Server Approach

A non-DIS approach is to have all the clients connect to a single server. This approach reduces the network traffic to O(n), as described in the previous section. The server is considered to be the final authority in all disputes (i.e. if the server says that your avatar process ended, then your avatar *did* end). In effect, this approach trades processor cycles for network bandwidth (i.e. you now need a dedicated server). One server can only handle so much load. Once that limit is reached one either needs a new approach for handling further clients, or one should get a new server and have subsequent clients use the new server instead. Clients on one server would be unable to interact with clients on another server. This approach is used in Meridian 59 [17].

2.6 Multi-Server Approach

Instead of having multiple servers running separate simulations, it should be possible to partition the simulation across multiple servers. If this is done, then some kind of load balancing across the servers is desirable in order to make the most efficient use of the servers' processor cycles and memory. The ideal load distribution results in equal, and minimal, usage of processor cycles and memory for all the servers (assuming the servers are homogeneous). The ideal distribution results in no communication between the individual servers. This would be possible if no server needed

to know about an avatar that it was not hosting. The servers would of course have to communicate with the client software. It is not expected that this ideal can be met at all times. This thesis examines the multi-server approach.

2.7 Traditional Scheduling & Load Balancing Problem

A parallel program is a collection of tasks that may run serially or in parallel. These tasks must be optimally placed on the processors of a particular parallel machine if the shortest execution time is to be realized. This problem is know as the scheduling problem of parallel computing. Consider a computer workload consisting of m interacting task modules, $\{M_k\} = \{M_1,..., M_m\}$, to be run on a parallel processing system comprising p processing elements, $\{P_i\} = \{P_1,..., P_p\}$, by allocating to each P_i a module assignment A_i subset of $\{M_k\}$. Let $\mathsf{A} = (A_1,..., A_p)$ denote the vector of assignments over the p processors. A is an optimal assignment if the time to complete all the tasks is minimized. In the most general case where the workload modules are nonuniform and the system is heterogeneous, the problem is analytically intractable except in the most restrictive cases [8]. For this reason, in order to come up with A in a reasonable time, average values are often used. This results in A being at best optimal in the mean.

Scheduling can be broadly divided into static and dynamic scheduling. With static scheduling, information regarding the precedence-constrained task graph must be known beforehand. Hence, each task in the graph has a static assignment to a particular processor, and each time that task is submitted for execution, it is assigned to that processor. The main disadvantage of static scheduling is its inadequacy in handling non-determinism in program

execution. If the task graph topology is not known before program execution, the parallel processor system must attempt to schedule tasks on the fly. This is known as dynamic scheduling. The disadvantage of dynamic scheduling is its inadequacy in finding global optimums and the corresponding overhead which occurs because the schedule must be determined while the program is running [15], [24], [26].

2.7.1 Dynamic Load Balancing

As processes enter and leave the system it is desirable to rebalance the load of the processes that are there. It is highly desirable to minimize the cost of processor reallocation. In order to do this, the job and the operating system work together in processor reallocations. Two methods of processor control are equipartition and demand-driven scheduling. In equipartitioning, each job is allocated an equal fraction of the processors, up to their maximum parallelism. This assumes that all jobs have a degree of parallelism that is constant throughout their lifetime. Demand-driven scheduling uses process control both to allow the system to take into account changes in a job's parallelism and to allow the job to adapt to changes in processor allocation [23].

2.7.2 Data Partitioning

Rather than dividing a program into tasks that are then assigned to processors (task partitioning), data can be assigned to processors that run identical tasks (data partitioning). If a processor requires data that has not been assigned to it, then there is a *data dependency*, and the data must be sent across the network. There are two basic ways of doing this: single instruction, multiple

data (SIMD), and single program, multiple data (SPMD). In the SIMD case, every processor works in lock step, executing exactly the same instruction at exactly the same time (i.e. they are synchronous). Each processor operates on different data. In SPMD, each processor runs exactly the same code, but each processor can be anywhere in the code relative to another processor (i.e. it is asynchronous). As with SIMD, each processor operates on different data [15].

2.8 Scheduling and Load Balancing for MMORPGs

A logical division of an MMORPG into tasks is to assign one task to each avatar, since each avatar is independent of all other avatars. Since each avatar is more or less equivalent, equal numbers of avatars could be assigned to each processor (assuming homogeneous processors) in order to achieve load balancing. However, whenever an avatar that is hosted by one processor is close to an avatar hosted by another processor, each of those processors must exchange state information about their respective avatars, each time a state update occurs (e.g. one of the avatars moves). This is a data dependency, and network bandwidth is consumed because of it. It is noted that this consumption of network bandwidth only occurs if two avatars on different servers are near to each other. If the avatars were on the same server, then this consumption of network bandwidth would not occur. For this reason, data partitioning is used. To increase the likelihood that two avatars that are near to each other are on the same server, the underlying map of the virtual world is divided between the servers. A server hosts those avatars that are on the part of the map that the server owns (its map segment). Only when avatars are close to the edge of a map segment is there the possibility that two avatars will be on different servers.

2.9 Multi-Server Approach in Current MMORPGs

In the multi-server approach currently known to be used in MMORPGs, the system attempts to indirectly balance the load on the servers and minimize network traffic. The map of the virtual world is statically partitioned into segments, and each segment is assigned to a server (the number of map segments being equal to the number of servers). Whichever server's map segment (MS) an avatar is currently on is the server that is currently hosting that avatar, and hence also the client associated with that avatar. One server will only inform another server of the actions of an avatar it is hosting if the avatar is close to the boundary between two MSes. Thus in the best case the network traffic *between* servers is O(0) (i.e. there is none) and in the worst case is $O(m^n)$, where n is the number of avatars and m is the number of servers. This is the case since every time an avatar performs an action all the other servers may need to be informed of it. Proper arrangement of the MSes would allow the worst case network overhead to be even lower by making it "physically" impossible for an avatar to be close to all MSes, and hence all servers. The least possible load on a server in terms of number of avatars is O(0) (i.e. there are no avatars on the MSes owned by that server, and hence the server is not hosting any avatars. The worst cast load is O(n) (i.e. the server has all the avatars in the simulation). The preferred case is for all servers to have a load of O(n/m) (i.e. equal numbers of avatars on all servers. This approach is used in Ultima Online [30].

2.10 Relationship of Current Research to Previous Research

The previous research did not consider the specific case of a virtual world. Previous research has not compared and contrasted different multi-server schemes. This research considers dynamic load balancing specifically applied to a virtual world. Just as in most other load balancing algorithms, the overall goal is to keep the load on all processors equal. Unlike many load balancing algorithms, some attempt is made to minimize network traffic. Since DIS does not scale well, and has problems of cheating, the approach currently used by Ultima Online (described in the previous section) is used as a starting point. Dynamic load balancing is done in Asheron's Call [1]. However, they have not elaborated on their dynamic load balancing scheme.

2.11 Summary

This chapter has presented background information on MMORPGs. Distributed interactive simulation (DIS) was examined. Separation of the client and the server was discussed. The single and multi-server approaches were described. The traditional scheduling and load balancing problem was examined. Scheduling and load balancing for MMORPGs was discussed. The multi-server approach currently used for MMORPGs was described. The current research attempts to take advantage of specific knowledge of the problem domain in order to do load balancing. In the next section the schemes studied in this thesis are presented.

Chapter 3

Schemes Studied

There are a number of different schemes that can be used to support a massively multiplayer online roleplaying game (MMORPG). There are also schemes that appear to be capable of supporting an MMORPG, but that have one or more flaws. Some of these schemes are described in this chapter. A new dynamic load balancing algorithm for MMORPGs is also presented. In this chapter it is assumed that each avatar occupies a position in a virtual world. Server machines are responsible for tracking avatar processes, and will transfer an avatar's operating environment and parameters to another server as needed.

3.1 Static Assignment of Avatars to Servers

One possible approach to supporting an MMORPG is to evenly divide the avatars between the servers. Thus each server has equal numbers of avatars (assuming that the number of servers evenly divides the number of avatars) and the load appears to be balanced. This works by statically assigning an avatar to the server with the least number of avatars as it enters the simulation. This will result, in the worst case, in $O(m^n)$ network traffic between the servers, where n is the number of avatars and m is the number of servers. This is because every time an avatar performs an action all the other servers must be informed of this in case one of their avatars sees this action (e.g. if an avatar moves then all near by avatars will see this). No MMORPG uses this scheme, but it is studied here due to it being an approach that tends to be intuitively thought of by a person new to the field.

3.2 Static Assignment of Map Segments to Servers

Static assignment of map segments (MSes) to servers works by dividing the map of the (virtual) world into segments. The number of segments being equal to the number of servers. Each segment is statically assigned to a server, such that each server has exactly one MS assigned to it. This is illustrated in Figure 3.1. An avatar is assigned to which ever server owns the MS it is currently on. This is the multi-server scheme described in section 2.7.

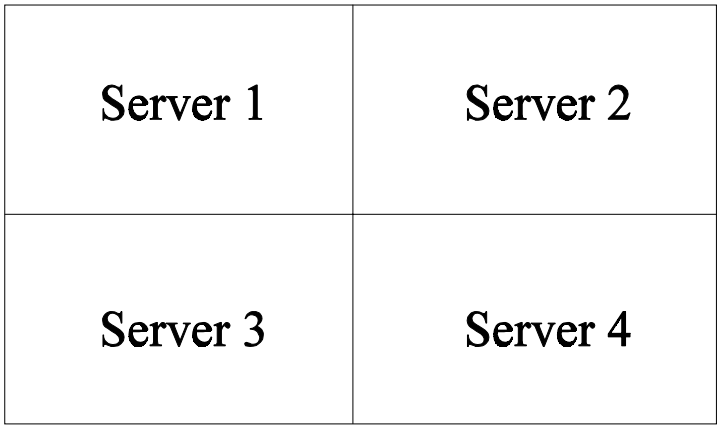


Figure 3.1: Four map segments assigned to four servers

3.3 Dynamic Assignment of Map Segments to Servers

Static assignment of MSes to servers has the potential to minimize both network traffic between servers, and server load (and hence maximize the total number of avatars in the simulation), but only if the avatars are properly distributed about the map. However, the distribution of avatars is controlled by the users, and hence is unpredictable at any given time. In order to overcome this, dynamic assignment of map segments to servers is needed.

3.4 Proposed Dynamic Algorithm

The proposed approach attempts to, partially, overcome the problem mentioned above by dividing up the map into (many) more MSes than there are servers and then assigning multiple MSes to each server, such that the resulting load on each server is as close to equal as possible. The

assignment of MSes to servers is done by a “land manager”. When needed, the land manager runs a load balancing algorithm (LBA) to redistribute the MSes, and hence the load, between the servers. If the distribution of avatars is unchanged from the last time the LBA was run then no change is made to the MS to server assignments. The algorithm attempts to keep the MSes owned by a particular server together. This makes it less likely that a given avatar will be near to an MS owned by another server, and hence no network traffic would be required to communicate the activities of that avatar.

3.5 Overall Load Balancing Algorithm

3.5.1 Legend and Notes

Notes: 1. "Next to" means the MS located next in the north, south, east, or west direction

2. "Connected component" refers to a group of MSes that are next to each other

3. The Load Balancing Algorithm (LBA) consists of the main LBA and the core LBA. The main LBA does setup between calls to the core LBA. The core LBA does the reassignment of MSes to balance the load.

S = total number of servers

N = total number of avatars on all servers

n = N/S = number of avatars ideally on each server

n(i) = number of avatars on server i

m(i) = |n - n(i)| = distance of server i from the average number of avatars

LM = Land Manager

MS = Map Segment

3.5.2 Main Load Balancing Algorithm

The load balancing algorithm to be studied in this thesis consists of the following steps:

Algorithm 3.1

1 The LM asks all servers how many avatars they have on each MS.

2 If an MS owned by a server has no avatars on it then the MS is taken away from that server and goes to the pool of MSes that are not owned by any server.

3 If one or more servers does not own any MSes then find the MS with the most avatars for which taking that MS away from its current server does not break a connected component, and that is not the only MS owned by that server. Give that MS to one of the servers that has no MSes. Repeat until all servers have at least one MS or all servers own either one or zero MSes.

4 If all servers own either one or zero MSes then goto step 12.

5 Mark all servers as not done.

6 Perform core LBA.

7 Give the least loaded server empty MSes next to its MSes (and empty MSes next to newly acquired MSes). Repeat for second least loaded server, etc.

8 Mark all servers as not done.

9 Perform core LBA.

10 Repeat step 2.

11 If a change to the map was made during step 9 then goto step 7.

12 Assign empty MSes based on a distance priority scheme (which ever server owns an MS, with avatars on it, that is closest to an empty MS gets the MS). Transmit the new distribution of MSes to the servers.

3.5.3 Core Load Balancing Algorithm

The core load balancing algorithm (called by the main load balancing algorithm) consists of the following steps:

Algorithm 3.2

1. If all servers marked as done then done.
2. Let i be the server with the largest m(x) and that is not marked as done.
3. If server i needs less avatars to reduce m(i) goto 7.
4. For the four MSes next to each of the MSes owned by server i, find the one MS (with at least one avatar on it) that if transferred to server i will decrease m(i) the most and not make the new value of m(j) larger than the new value of m(i), where server j is the server that the MS is coming from. Also, removing the MS from server j must not increase the number of connected components. If there is a tie in terms of reduction achieved in m(i), then out of those that tied, the one that is closest to a different connected component belonging to server i is chosen. If there is still a tie, it is broken based on which ever MS minimizes m(j) the most from those that still

tied. If there is still a tie then any one that is left will do.

5. If no MS was found in step 4 then mark server i as done, else transfer the MS to server i and mark server j as not done.
6. Goto 1.
7. For each MS owned by i, find the one MS (with at least one avatar on it) that if transferred to a server, j, that owns an MS, next to the MS, that will decrease m(i) the most and not make the new value of m(j) larger than the new value of m(i). Also, removing the MS from server i must not increase the number of connected components. If there is a tie then out of those that tied the one that is farthest from a different connected component belonging to server i is chosen. If there is still a tie then out of those that tied the one that minimizes m(j) the most is chosen from those that still tied. If there is still a tie then any one that is left will do.
8. If no MS was found in step 7 then mark server i as done, else transfer the MS to server j and mark server j as not done.
9. Goto 1.

3.5.4 Determination of Breaking of Connectedness

Calculating new values for m(i) (where m(i) = |n - n(i)| as defined in section 3.3.2.1) is straight forward. Determining if changing ownership of an MS breaks connectedness is not as straight forward. Here is an algorithm to do this.

Algorithm 3.3

1. The MS in question belongs to server j.
2. If server j owns exactly one or zero MSes that are next to the MS in question then done (can transfer ownership).
3. Pretend that the MS in question no longer belongs to server j.
4. Mark one of the MSes that is next to the MS in question and belongs to server j as visited.
5. Go to all the MSes marked as visited and mark the MSes that belong to j that are next to these MSes as visited.
6. If all the MSes that belong to j and are next to the MS in question are marked as visited then done (can transfer ownership of MS) else if no new MS was marked as visited then done (can not transfer ownership of MS) else goto 5.

Possible configurations encountered at step 2 of the algorithm are illustrated in Figures 3.2, 3.3, and 3.4.

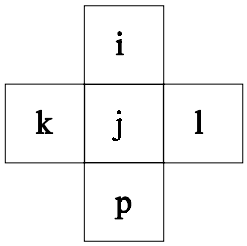


Figure 3.2: The middle MS is owned by j, and there are zero others next to it owned by j

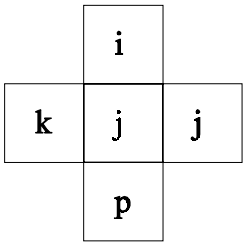


Figure 3.3: The middle MS is owned by j, and so is one other MS next to it

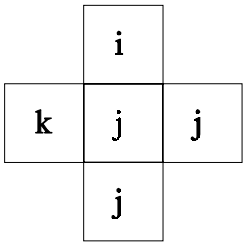


Figure 3.4: The middle MS is owned by j, and so are two other MSes next to it.

3.6 Explanation of Algorithm

When it is time to do load balancing, Algorithm 3.1 (the algorithm) is called. Algorithm 3.1 calls Algorithm 3.2 as needed. Algorithm 3.2. calls Algorithm 3.3 as needed. The algorithm does not have to operate in real-time. The servers send the number of avatars on each MS they own and then continue on. Once the land manager is finished running the LBA it sends the results to the servers. The MS to server assignments are the best assignment based on the data originally sent to the land manager. The new assignment may be communicated to the servers at a time when the data that was originally sent is no longer accurate. The sooner the new assignments are sent to the servers the less inaccurate the data used in the LBA will be with respect to the current situation. As such the time

to run the LBA must not be excessive.

The basic idea behind the algorithm is that if a server has less avatars than the average across the servers, then it takes an MS, that is beside an MS it owns, that moves it as close as possible to the average load in terms of the number of avatars. Alternatively, if a server has more avatars than the average then it will give away an MS. The MS will be given to a server that owns an MS next to the MS being given away, and that will move the number of avatars on the server closer to the average. Of course, taking and giving means that another server loses or gains an MS. This is only allowed to happen if the other server is not moved further away from the average than the new distance from the average for the server in question. Also, the LBA does not allow a group of MSes that are next to each other and belong to the same server to be broken into two groups. This makes it less likely that a given avatar will be near to an MS owned by another server, and hence no network traffic would be required to communicate the activities of that avatar.

When a tie occurs as to which MS is the best choice to be moved from one server to another, then the MS that is most effective for moving two connected components, belonging to the same server, together (or at least not apart) is chosen. That is to say, if server i is the server in question and servers j and k are two servers for which a tie has occurred, then which of j or k would allow i to move two of its connected components closer together. For example, suppose that if j is chosen then two different connected components of i go from being separated by three MSes to being separated by two MSes. Also, suppose that if k is chosen then two different connected components of i go from being separated by five MSes to being separated by four MSes. In this case j should be

chosen, since the final separation of connected components would be least. Figure 3.5 is used to illustrate another example. An MS is to be transferred to server m. Either the MS owned by server l or the MS owned by server n is to be chosen (i.e. there is a tie). The MS owned by server n should be chosen, since doing so moves the two disconnected components of m closer together (and in this example merges them into one connected component). If a tie still occurs then out of those servers that tied, the MS that best minimizes the distance from the average of the server not in question is chosen. That is to say, if i is the server in question and j and k are two servers for which a tie has occurred, then which of j or k would be moved closest to the average number of avatars across all servers. For example, if the mean number of avatars is ten, and j would be moved to eleven avatars, and k would be moved to seven avatars, then the MS belonging to j should be chosen, since it would end up closest to the mean. Any remaining ties are arbitrarily broken.

Surprisingly, dealing with empty MSes is not trivial. The first time the main LBA is run no empty MSes are assigned to a server. As a result, the empty MSes are ignored. During the middle part of the algorithm, an empty MS is assigned to whichever server owns a neighboring MS, and has the least number of avatars out of all the neighbors. After this assignment the core LBA is run. The least loaded server may now be different, so the empty MSes are reassigned accordingly and the core LBA is run again, etc. At the end of the main LBA the empty MSes are again re-assigned, this time based on a distance priority scheme (whichever server owns an MS, with avatars on it, that is closest to an empty MS, gets the MS). The reason for this is to make it less likely that an avatar will be near an MS that is owned by another server. The algorithm always completes since MSes are only transferred if doing so moves at least one server closer to having the average number of avatars

across the servers, without causing any other server to be moved further away from the average than the server in question after it has been moved closer to the average.

a	b	c	d	e
f	g	h	i	j
k	l	m	n	m
p	q	r	s	t
u	v	w	x	y

Figure 3.5: Map segment to server assignments

3.6.1 Convergence of Algorithm

The algorithm monotonically converges towards a solution. This is because a map segment with avatars on it is only transferred between servers if it results in one of the servers moving closer to having the average number of avatars across the servers, without moving the other server further away from the average than the new distance from the average for the first server.

3.6.2 Termination of Algorithm

The algorithm always terminates. This is because it will eventually reach a point where it is no longer possible to continue monotonically converging, and hence no exchanges of map segments between servers will occur within the core LBA. The main LBA will exit if there is no exchange of map segments between servers within the core LBA.

3.6.3 Run-Time Complexity

The best case run-time of the LBA is O(1). This occurs if all the servers are assigned exactly one or zero map segments. The core LBA is not run in this case. The best case run-time when the core LBA is run is O(mn), where m is the number of servers and n is the number of map segments. In this case, no map segments are exchanged between servers. The worst case run-time is $O(n^2)$. In this case, every map segment is exchanged between two servers. The average case is unknown since the probability distribution of avatars among the servers is unknown.

3.7 Summary

Three schemes for supporting an MMORPG are studied in this thesis: static assignment of avatars to servers, static assignment of map segments to servers, and dynamic assignment of map segments to servers. The dynamic assignment of map segments to servers uses a new dynamic load balancing algorithm that was presented in this chapter. In the next chapter, the experimental setup is described. In Chapter 5, the experimental results and analyses for the three schemes described here are presented.

Chapter 4

Experimental Setup

The previous chapter described the schemes that are considered in this thesis. In this chapter, the experimental setup is described. Many simulation runs are done using the experimental setup. The hardware configuration, software architecture, and design and implementation are described. At the end, a description of the software components is given.

4.1 Hardware Configuration

Figure 4.1 shows a typical hardware configuration for the experiments. The land manager and the servers were each run on their own Sun SPARCstation 2 with 64 MB of RAM. For eight servers, six of the servers were SPARCstation 2's and two of the servers were Sun SPARCstation 5's with 64 MB of RAM. Varying numbers of clients were simulated on multiple Sun UltraSPARC 170's

with 128 MB of RAM. Each machine has its own port on an Baystack 350-24T auto-sensing 10/100 Mbps ethernet switch. For 10 Mbps operation the port forwarding rate using 64 byte packets is 14,880 packets per second. Each machine uses 10 Mbps ethernet.

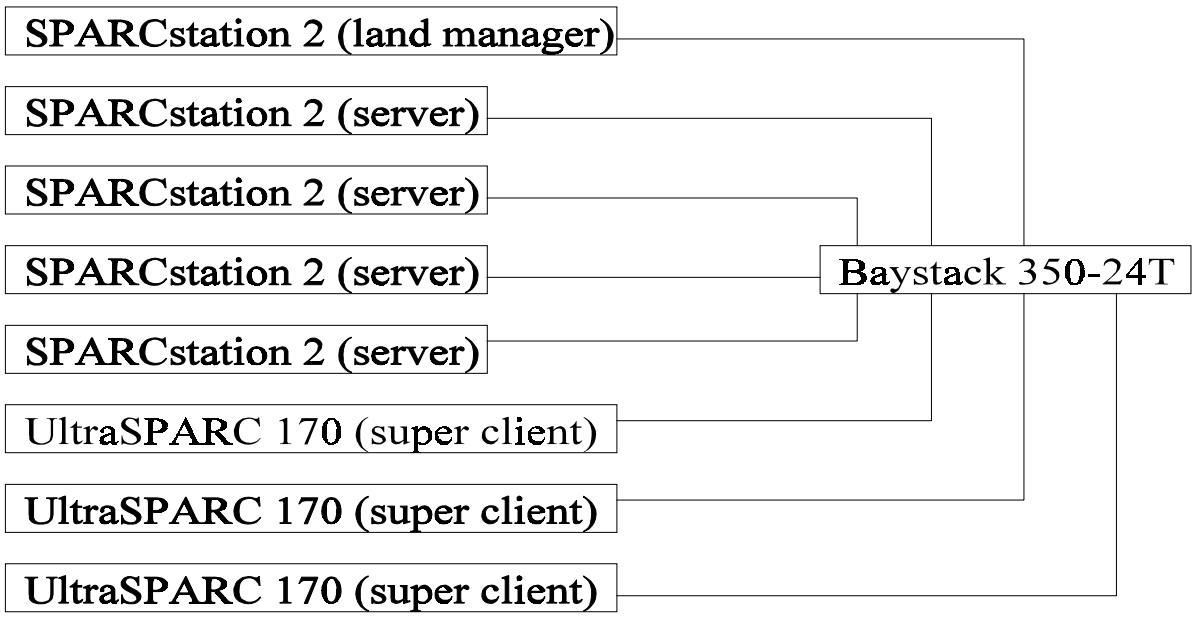


Figure 4.1: A configuration using four servers and three super clients

4.2 Software Architecture

The software architecture revolves around the “life” of an avatar. Initially, there are no avatars in the simulation. A super client generates an avatar and informs the land manager of the new avatar, and the location of the new avatar in the virtual world. The land manager determines which map segment the avatar is on, and then informs the server that owns that map segment of the new avatar. The server now owns the avatar. However, if there is more than one server, then the avatar may at some subsequent time be transferred to a new server. The avatar is always controlled by the super

client that generated it. A super client can control multiple avatars. Each avatar can be on the same or a different server than other avatars controlled by a super client. There can be one or more servers, and one or more super clients, but there is always exactly one land manager. If dynamic load balancing is used then the land manager performs the load balancing algorithm. The process diagram in Figure 4.2 illustrates the possible communication paths.

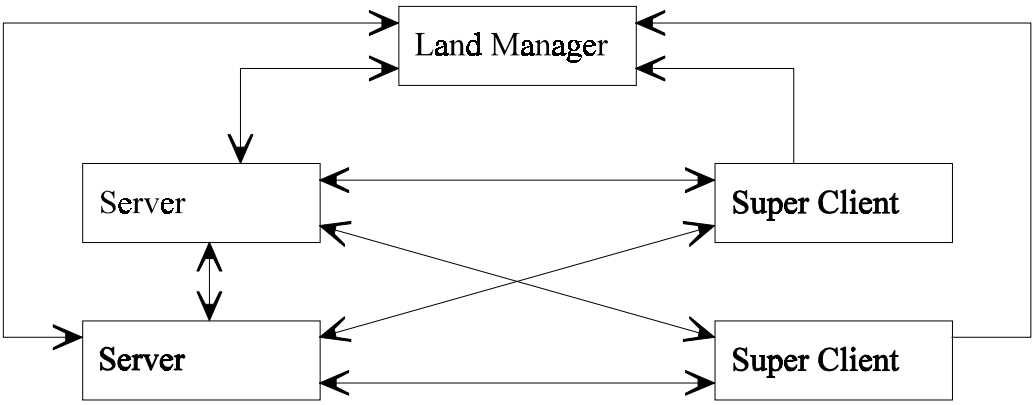


Figure 4.2: Process diagram for two servers and two super clients

4.3 Design and Implementation

The software was written in C++ [28]. Communication between processes was done using the PVM library [7]. The software is not fault tolerant. If a failure occurs then the system must be reset, losing all data. Internally, each avatar consists of a data structure, on one of the servers, that contains the avatars position and other attributes. Map segments are implemented on the servers as linked lists of avatars. The avatars in the linked list for a given map segment are the avatars that are on that map segment. An avatar can only have one of six things happen to it once it has entered the simulation:

- Move north
- Move south
- Move east
- Move west
- Be transferred to another server
- Leave the simulation

4.3.1 Generation of Avatars

Two basic distributions of avatars were used - an even distribution and an uneven distribution. Varying numbers of avatars were placed in each distribution. Movement of avatars was either random or non-random. If avatars were in a group then the whole group was moved together. In the case of random movement, each avatar moved one unit in a random direction every five seconds, wrapping around at the edges of the map. In the case of non-random movement, each avatar moved one space to the east every five seconds, wrapping around at the edge of the map.

4.3.1.1 Even Distribution

For the even distribution the avatars were evenly distributed across the map. For example, if there were 400 avatars, then the avatars were distributed into 20 rows and 20 columns. See Figure 4.3.

A A A A

A A A A

A A A A

A A A A

Figure 4.3: An even distribution of 16 avatars arranged into 4 rows and 4 columns

4.3.1.2 Uneven Distribution

For the uneven distribution 30 groups of avatars were unevenly distributed across the map. This distribution is shown in Figure 4.4. All of the groups in a quadrant contained the same number of avatars. Ten of the groups, comprising three fifths of the avatars, were placed in the upper left quadrant of the map. Ten of the groups, comprising one fifth of the avatars, were placed in the lower left quadrant of the map. The remaining ten groups of avatars, comprising one fifth of the avatars, were placed in the upper right quadrant of the map. For example, if there were 250 avatars, then there would be 10 groups of 15 avatars each evenly placed in the upper left quadrant, 10 groups of 5 avatars each evenly placed in the lower left quadrant, and 10 groups of 5 avatars each evenly placed in the upper right quadrant.

4.3.2 Parameters Used During Simulation

In the case of dynamic partitioning, ten by ten map segments were used (a total of 100 map segments) and the load balancing algorithm was run every minute. All simulations were run for one hour. During all simulations, one additional avatar was added that moved one space to the east every 0.5 seconds. This is the avatar for which specific data is collected. The response times given in the

results section refer to the delay from the time movement of this avatar is requested by the client until the time the server indicates that the movement has occurred.

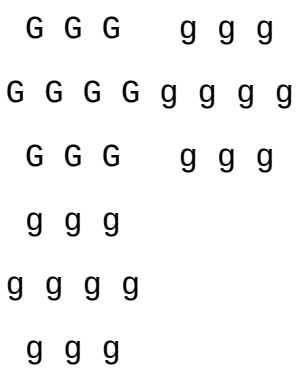


Figure 4.4: Arrangement of groups for uneven distribution (each group “G” contains three times as many avatars as each group “g”)

4.3.3 Assumptions

The following assumptions were made in order to limit the number of variables in the simulation:

- The servers are centralized (i.e. they are all in the same room)
- Each user action causes equal load (i.e. all actions are equivalent)
- The clients are all equidistant from the servers (i.e. there is equal network delay for all clients)

4.4 Description of Components

The three software components are: the land manager, the server, and the super client. There is always exactly one land manager running, and it must be started first. There can be one or more

servers. The servers are started after the land manager. There can be one or more super clients. The super clients are started last. In the following sections, "`tid`" refers to the id assigned to a process by PVM. In the implementation, there are objects and avatars. An avatar is a type of object, but an object is not necessarily an avatar.

4.4.1 Land Manager

The land manager assigns avatars to servers based on the location of the avatar and the current map segment to server assignments. Once an avatar is initially assigned to a server, it is up to the servers to determine every thing that happens to it, including determining which server should host the avatar for the cases where more than one server is used. For simplicity and uniformity, the land manager is used even when there is only one server. The land manger performs the load balancing algorithm when map segments are being dynamically assigned to servers. Each time data is received the land manager outputs, to a log file, the time at which the data was received (relative to the time at which the land manager was started) and the number of data bytes in the message. If dynamic assignment of map segments to servers is performed then the time to run the load balancing algorithm, and the time at which it is run, is recorded in a separate log file. The messages sent by the land manager are shown in the following section.

4.4.1.1 Messages Sent by the Land Manager to a Server

The following messages are sent by the land manager to a server:

LS_TID

Empty message that server uses to learn land managers's `tid`.

Contains:

-nothing

LS_AV

An avatar is being added to the simulation.

Contains:

-map number (number of times load balancing algorithm has been run)

-the avatar's object number

-the avatar’s current position

-the avatar's appearance (avatar appears as a single color ASCII character)

-the `tid` of the super client

LS_NUM_OBJS_MS_REQ

The land manager asks the server to send back the number of objects that the server has on each MS that it owns.

Contains:

-nothing

LS_NEW_MAP

Contains a map indicating the ownership of each MS.

Contains:

-The map number (initially zero, and increments each time LBA is run)

-The `tid` of the owner of each MS, in row major order

LS_USE_NEW_MAP:

All the servers now have the new map. Begin transferring objects between servers (as needed).

Contains:

-nothing

LS_SHUTDOWN

Any server that receives this message will shut down, after it no longer has any avatars.

NOTE: All map segments must be taken away from this server prior to sending this message in order for the simulation to go on without this server.

Contains:

-nothing

LS_SHUTDOWN_NOW

Any server that receives this message will shut down immediately. Continuing to run the simulation is not recommended, since this is not a “clean” shutdown. Extraneous error messages may be produced by PVM, again, this is because this is not a “clean” shutdown.

Contains:

-nothing

4.4.2 Server

The server is the most complicated piece of software. If there is one server, it keeps track of all the avatars. If there are multiple servers then each one must keep track of its own avatars and coordinate with the other servers in order to insure that all avatars that are hosted by another server and should be visible to at least one avatar that it is hosting, are made visible. A server is able to transfer avatars to another server, and receive them from another server. This would happen if an avatar moved from a map segment owned by one server to a map segment owned by another server, or after the load balancing algorithm was run the map segment that the avatar is currently on is transferred to another server. If a server sends a message for which it requires a response then it does not stop and wait for the response. It accepts new, unrelated, messages until it gets the response (e.g. a request from a super client to move its avatar would not be ignored while the server is in the process of handing off a different avatar to another server). There can be many such outstanding requests at any given time. Each time data is received, the server outputs, to a log file, the time at which the data was received (relative to the time at which the server was started) and the number of data bytes in the message. The messages sent by the server are shown in the following sections.

4.4.2.1 Messages Sent by a Server to the Land Manager

The following messages are sent by a server to the land manager:

SL_REQ_TID

Ask the land manager to send back an empty message so the server will know its `tid`.

Contains:

-nothing

SL_SAVE_AV

An avatar is leaving the simulation.

Contains:

-the avatar's object number

-the avatar's current position

SL_NUM_OBJS_MS

Sent in response to an LS_NUM_OBJS_MS_REQ message from the land manager. The server sends back the number of objects that it has on each map segment that it owns.

Contains:

-the number of objects on each map segment owned by the server, in row major order

SL_MAP_ACK

Sent back to the land manager in response to a SL_MAP message.

Contains:

-nothing

4.4.2.2 Messages Sent by a Server to a Super Client

The following messages are sent by a server to a super client:

SC_AV

When an avatar first enters the simulation the server sends this msg to tell the super client the characteristics of the avatar and its initial location.

Contains:

-avatar's object id

-The (global) position of the avatar.

-avatar's appearance

-a list of all objects visible to the super client

Contains (for each objects):

-object's id

-The (global) position of the object.

-object's appearance

SC_AV_MV

The avatar has moved one unit.

Contains:

-avatar's object id

-direction moved in (one of: NORTH, SOUTH, EAST, or WEST)

-a list of objects that have, as a result of this movement, become visible to the super client

Contains (for each objects):

-object's id

-The (global) position of the object.

-object's appearance

SC_OBJ

Whenever the server wants the super client to know about the existence of an object (which could be another user's avatar) it sends this message.

Contains:

-object's id

-The (global) position of the object.

-object's appearance

SC_OBJ_MV

An object has moved one unit.

Contains:

-the object's id

-direction moved in (one of: NORTH, SOUTH, EAST, or WEST)

SC_OBJ_RM

One or more objects is no longer visible.

Contains (for each object):

-the object's id

SC_SS

When the avatar switches servers, the old server sends this messages to the super client.

Contains:

-avatar's object id

SC_SS_NOW

First the server sends a SC_SS and waits for a CS_SS_ACK from the super client. Then the server sends a SC_SS_NOW when it transfers the avatar to the new server.

Contains:

-nothing

SC_SS_DONE

When the avatar switches servers, once the server switch is complete the new server sends this message to the super client.

Contains:

-avatar's object id

-a list of all objects visible to the super client

Contains (for each object):

-object's id

-The (global) position of the object.

-object's appearance

SC_AV_RM_ACK

After receiving a CS_AV_RM message from the super client this message is sent back.

Contains:

-avatar's object id

4.4.2.3 Messages Sent by a Server to Another Server

The following messages are sent by a server to another server:

SS_OBJS

Tell another server about the existence of one or more objects visible to it from a neighboring map segment.

Contains:

-map number

-for each object:

-the object's id

-the object's global position

-object's appearance

SS_OBJS_ACK

ACK an SS_OBJS, or SS_OBJ_MV message.

Contains:

-map number

-for each object:

-the object's id

SS_OBJ_MV

Tell a server that an object has moved.

Contains:

-map number

-the object's id

-direction moved in (one of: NORTH, SOUTH, EAST, or WEST)

SS_OBJ_RM

Tell a server that an object is no longer visible to it.

Contains:

-map number

-the object's id

SS_OBJ_RM_ACK

Acknowledge a SS_OBJ_RM message

Contains:

-map number

-the object's id

SS_XFER_OBJS

Transfer some non-avatar objects to another server.

Contains (for each object):

-map number

-object's id

-objects position

-object's appearance

SS_XFER_AVS

Transfer some avatars to another server.

Contains (for each avatar):

-map number

-object's id

-avatar’s position

-avatar's super client

-avatar’s appearance

-TRUE if all ACKs received before transfer, FALSE otherwise

-number of queued up commands

-the queued up commands

SS_SS

One or more servers is informed that this server (that is sending the message) is about to move objects from its control to the control of another server (this msg is sent to all the severs that can see this object, including the server that will receive the control).

Contains:

-map number

-object id number for each object

SS_SS_ACK

Sent in response to SS_SS message.

Contains:

-map number

-object id number for each object

4.4.3 Super Client

Rather than having hundreds of volunteers use client software that supports one avatar for each instance of the software running, a “super client” was written. As far as the land manager and server are concerned, there is no difference between a super client and a normal client. Each super client simulates multiple clients connected to the simulation. The number of simulated clients and the behavior of the avatars controlled by those clients is controlled using a script file. The super client can be configured to output, to a log file, the amount of time between the sending of a messages and the receiving of a response, and the time (relative to the time at which the super client was started)

at which the responses were received. In addition, the number of data bytes received and the time at which they were received, is output to a separate log file. Multiple super clients can be run at the same time. An example script is shown in Figure 4.2.

```
A:1,0,2
  e0.5
end A

B:2,20,22
  n1.5
  s2.3
  w2.2
  e7.3
end B

C:7,23,42
end C

LB:0,0,0
  l60
end LB

SD:0,0,0
  d3600
end SD
```

Figure 4.5: An example script file

The script file shown in Figure 4.2 works as follows. The first part (A) places one avatar at location (0, 2) which moves to the east every 0.5 seconds. The second part (B) places two avatars, centered on (20, 22), and moves them north after 1.5 seconds, south 2.3 seconds after the previous movement, west after 2.2 seconds, east after 7.3 seconds, and then the pattern of movement in this part is repeated. The avatars are not both placed in the same location, but rather spread out in a group centered on (20, 22). The third part (C) centers seven avatars at location (23, 42) and leaves them

there. The fourth part (LB) specifies that the load balancing algorithm is to be run every 60 seconds. The fifth part (SD) specifies that the script is to be terminated after three thousand six hundred seconds (one hour). The messages sent by the super client are shown in the next two sections.

4.4.3.1 Messages Sent by a Super Client to the Land Manager

The following messages are sent by a super client to the land manager:

CL_AV

Inform the land manager of a new avatar.

Contains:

-column and row number where avatar is to appear in the virtual world

-avatar's appearance

CL_SHUTDOWN

After receiving this message, the LM sends LS_SHUTDOWN messages to all servers, and then the LM shutsdown. For this to work properly there must be no avatars in the simulation.

Contains:

-nothing

CL_SHUTDOWN_NOW

After receiving this message, the LM sends LS_SHUTDOWN_NOW messages to all servers and then the LM shutsdown. Some extraneous error messages may be produced by PVM, since this is not a “clean” shutdown.

Contains:

-nothing

XL_LB

Tell the LM to run the LBA. This allows the determination of when to run the LBA to be under the control of the scripting used in the super client.

Contains:

-nothing

4.4.3.2 Messages Sent by a Super Client to a Server

The following messages are sent by a super client to a server:

CS_REQ_MV

Super client requests that its avatar move.

Contains:

-avatar's object id

-direction to move in (one of: NORTH, SOUTH, EAST, or WEST)

CS_SS_ACK

Sent in response to a SC_SS message. Super client acknowledges that its avatar is being moved to another server. Client will queue up any server requests from user until it receives a SC_SS_DONE message from the new server, then all queued up requests will be sent to the new server.

Contains:

-avatar's object id

CS_AV_RM

The avatar is leaving the simulation.

Contains:

-avatar's object id

4.5 Summary

This chapter has described the experimental setup. Many simulation runs were done using the experimental setup. The hardware configuration, software architecture, and design and implementation were described. The simulation was run on a network of Sun computers. The three software components used were the land manager, server, and super client. Avatars were arranged in either an even or an uneven distribution. A description of the software components was given. In the next chapter, the experimental results are presented and analyzed.

Chapter 5

Experimental Results and Analysis

The previous chapter examined the experimental setup. This chapter presents the experimental results and the analysis of those results. The data is collected for nonrandom and random movement of avatars. For both the nonrandom and random cases, an even distribution is examined, as well as an uneven distribution.

5.1 Nonrandom and Random Movement

In a real MMORPG, each avatar is controlled by an individual. Each individual directs his avatar towards an objective of his choosing. As such, in general, an individual avatar does *not* move randomly. It is currently unknown how to model the collective behavior of the avatars. In light of this, two extremes were chosen, a simple (arbitrarily chosen) nonrandom pattern of movement, and

a completely random pattern of movement.

5.2 Even and Uneven Distributions

In a real MMORPG, each avatar is controlled by an individual. Each individual directs his avatar towards a goal of his choosing. As such, it is not clear how to model the over all distribution of avatars within the virtual world. In light of this, two extremes were chosen, an even distribution of avatars, and an (arbitrarily chosen) uneven distribution of avatars.

5.3 The Four Schemes

Four schemes were evaluated: single server, static assignment of avatars to servers (Static Av), static assignment of map segments to servers (Static MS), and dynamic assignment of avatars to servers (Dynamic MS). The Static Av scheme was only examined for the nonrandom case. All other schemes were examined for the random and nonrandom cases. The reason the Static Av scheme was only examined for the nonrandom case was that it faired very poorly in all experiments, and therefore it was eliminated from deeper evaluation.

In the data collected bellow, if a “yes” occurs in the “IBO” column then one of the servers crashed due to an input buffer overflow. Specifically, the server was processing data slowly, which resulted in the input buffer growing in size, until it became so large that the server crashed. This means that the configuration was unable to handle the number of avatars that it was given.

5.4 Nonrandom Movement Case

This section contains the experimental results and analysis for the case of nonrandom movement of avatars.

5.4.1 Even Distribution

This section contains the experimental results and analysis for an even distribution of avatars moving nonrandomly. The system configurations are shown in table 5.1. Response times are shown in table 5.2. The bytes of data received by each software component are shown in table 5.3. The time to perform the load balancing algorithm is shown in table 5.4.

Case	# of servers	Total # of av	Distribution	IBO
(A)	1	784	N/A	no
(B)	1	900	N/A	yes
(C)	2	324	Static Av	no
(D)	2	361	Static Av	yes
(E)	2	784	Static MS	no
(F)	2	900	Static MS	no
(G)	2	784	Dynamic MS	no
(H)	2	900	Dynamic MS	no

Table 5.1: System Configurations for Nonrandom Even Distribution

Case	Average	Min	Max
(A)	0.11	< 0.01	1.24
(C)	0.03	< 0.01	0.64
(E)	0.06	< 0.01	0.54
(F)	0.06	0.02	11.47
(G)	0.17	< 0.01	20.33
(H)	0.54	< 0.01	25.23

Table 5.2: Response Times for Nonrandom Even Distribution (in seconds)

Case	Land manager	Super Client	Server(s)
(A)	15,712	85,696	6,123,352
(C)	6,512	85,424	6,101,056
(E)	15,740	85,732	6,590,164
(F)	18,040	85,732	7,605,784
(G)	40,492	85,880	7,909,752
(H)	42,812	85,648	9,283,796

Table 5.3: Total Bytes of Data Received for Nonrandom Even Distribution

Case	Average	Min	Max	Second from max
(G)	0.02	< 0.01	0.47	0.12
(H)	0.01	< 0.01	0.49	0.04

Table 5.4: Time to Perform LBA for Nonrandom Even Distribution (in seconds)

5.4.1.1 Analysis

Figure 5.1 shows the maximum number of avatars supported. When the Static Av scheme is used

with two servers, the maximum number of avatars is less than half the maximum number of avatars for one server. This is because every time an avatar on one server moves the other server must be informed. More avatars can be supported by two servers than by one server for the Static MS and Dynamic MS schemes. Figure 5.2 shows the response times. The Static MS scheme gives a better response time than one server for the same number of avatars. The Dynamic MS scheme gives a worse response time than one server for the same number of avatars. This is likely because it is unnecessary to periodically redistribute map segments between the servers, since the avatars are evenly distributed across the map for the duration of the simulation. Figure 5.3 shows the number of bytes transferred. More bytes are transferred when two servers are used, since data must be transferred between the servers. More bytes are transferred when the Dynamic MS scheme is used, due to the overhead of the load balancing algorithm.

The LBA times are not placed in a figure since there is no discernable trend. It may seem counter intuitive that the average time to perform the LBA and second from maximum times are lower for (H) than for (G). However, the LBA is processing the same amount of data in each case, in spite of the fact that there are more avatars for (G). Also, the avatars are evenly distributed across the map, but this does not mean that there are equal numbers of avatars on each map segment and it is the distribution of avatars across the map segments that determines how the LBA runs. The reason the maximum and second from maximum times are so different is that before the LBA is run for the first time all the MSes are assigned to one server and hence a large change in the assignments of MSes to servers is required. Subsequently much smaller changes are required, and hence much less time is required for the LBA.

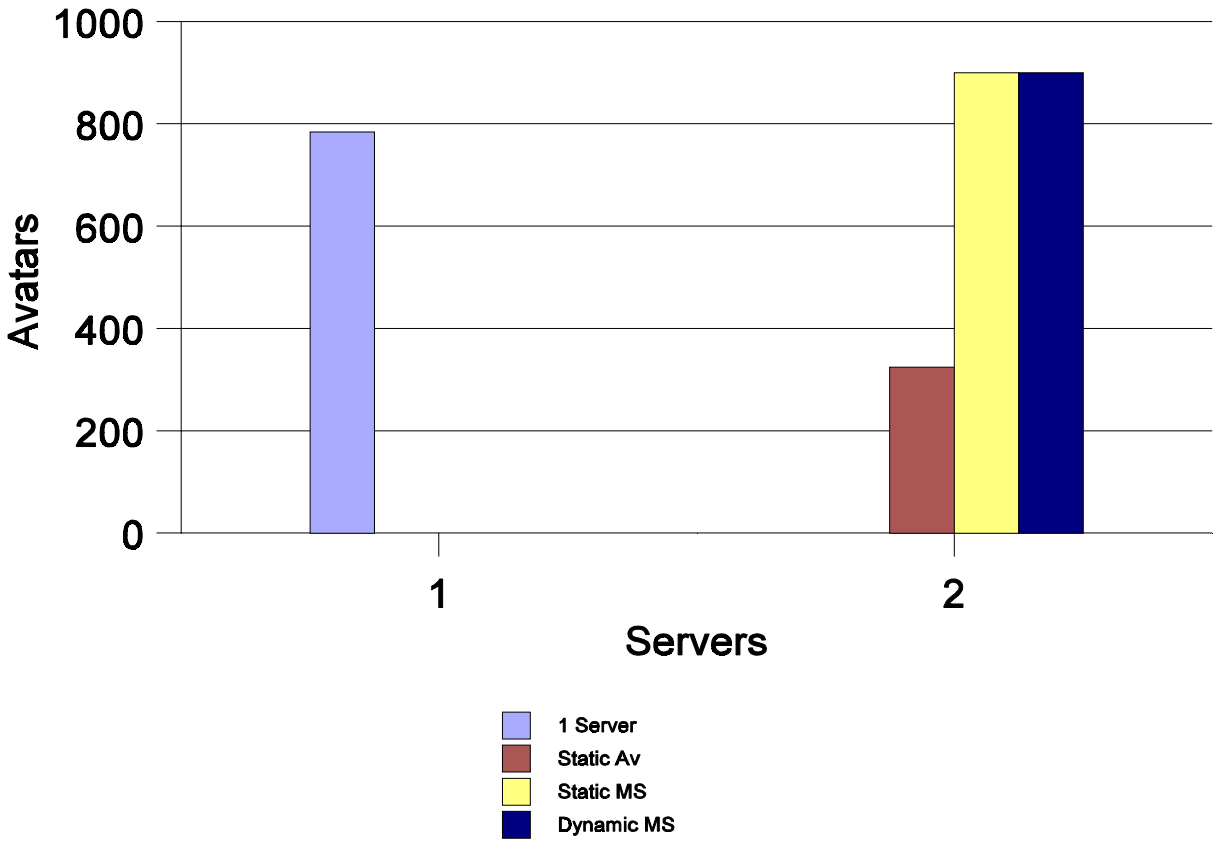


Figure 5.1: Maximum Number of Avatars for Nonrandom Movement, Even Distribution

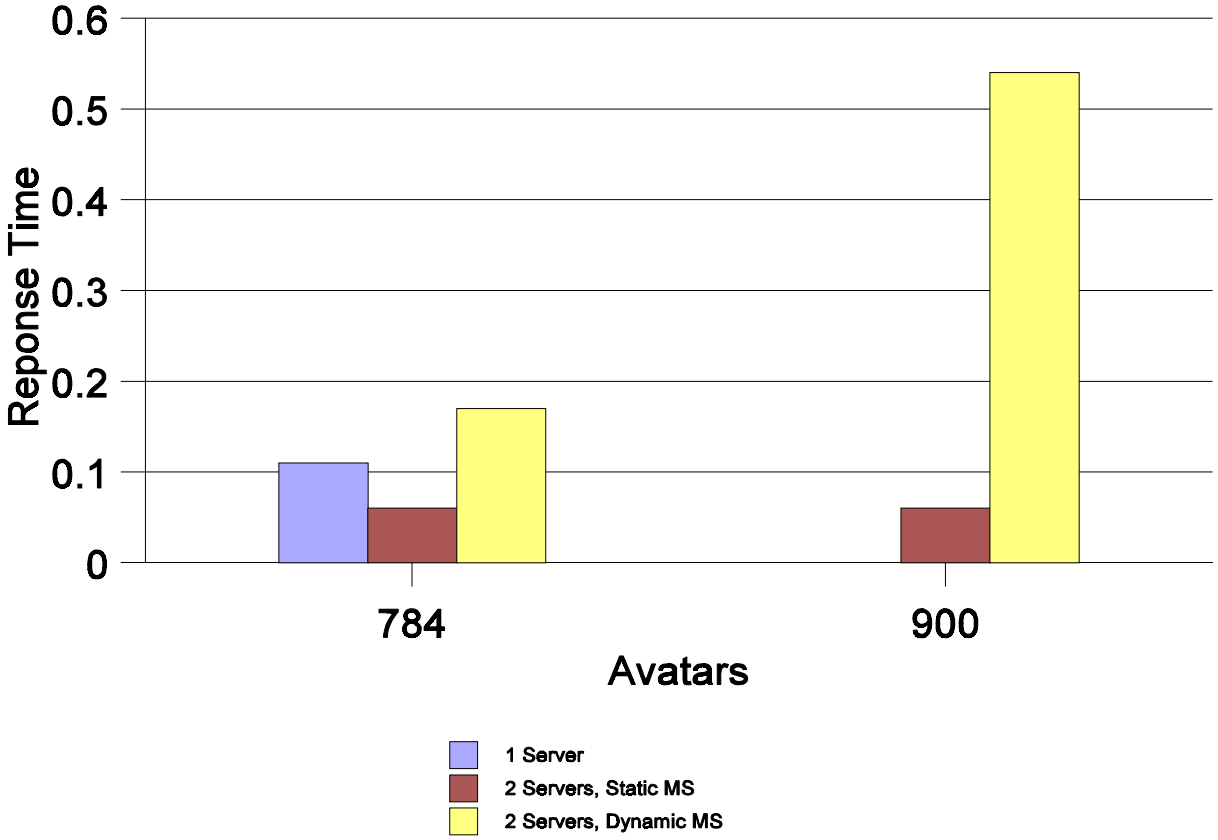


Figure 5.2: Response Times for Nonrandom movement, Even Distribution

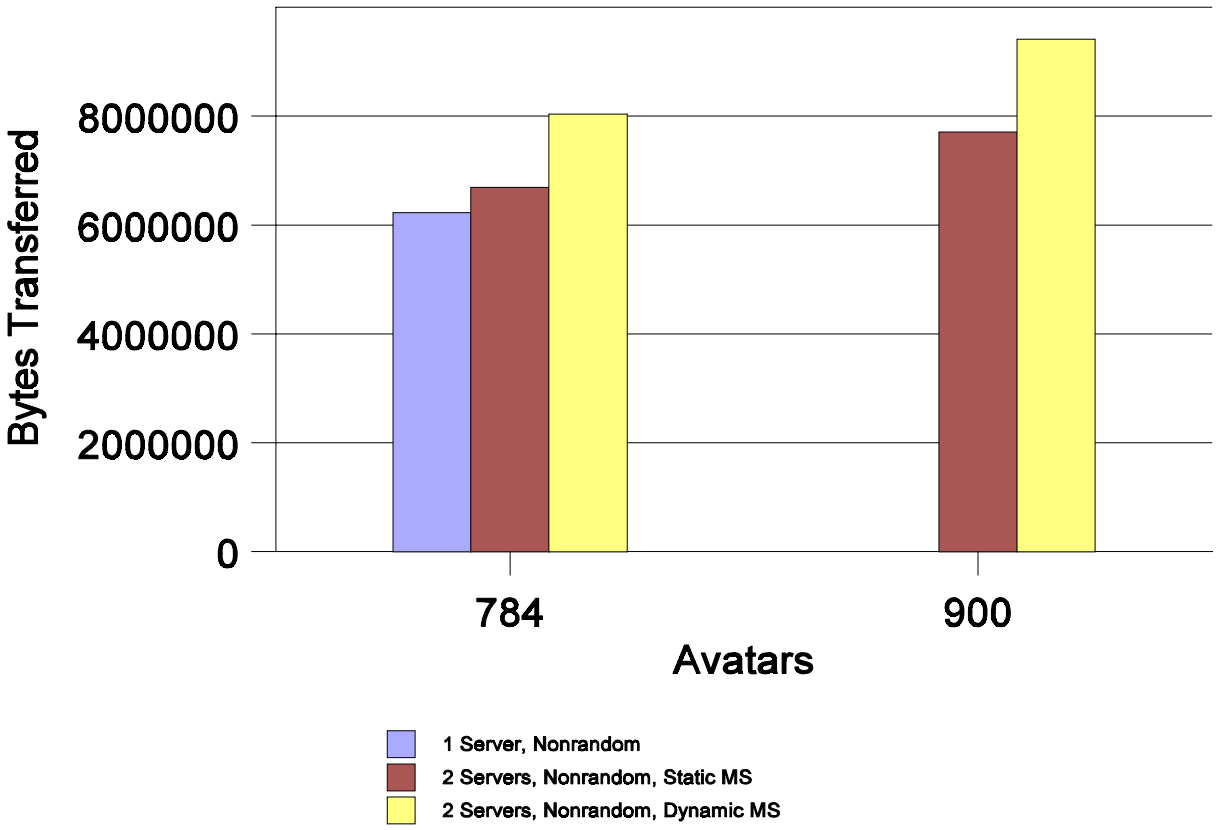


Figure 5.3: Bytes Transferred for Nonrandom Movement, Even Distribution

The network became the bottleneck before either the Static MS or Dynamic MS schemes stopped working (i.e. data generated by the super clients to be placed on the network was being generated faster than the data was actually going onto the network).

5.4.2 Uneven Distribution

This section contains the experimental results and analysis for an uneven distribution of avatars moving nonrandomly. The system configurations are shown in table 5.5. Response times are shown in table 5.6. The bytes of data received by each software component are shown in table 5.7. The time to perform the load balancing algorithm is shown in table 5.8.

Case	# of servers	Total # of av	Distribution	IBO
(I)	1	250	N/A	no
(J)	1	300	N/A	yes
(K)	2	150	Static Av	no
(L)	2	200	Static Av	no
(M)	2	250	Static Av	yes
(N)	2	250	Static MS	no
(O)	2	300	Static MS	no
(P)	2	350	Static MS	no
(Q)	2	400	Static MS	yes
(R)	2	250	Dynamic MS	no
(S)	2	300	Dynamic MS	no
(T)	2	350	Dynamic MS	no
(U)	2	400	Dynamic MS	no

(V)	2	450	Dynamic MS	yes
(W)	4	350	Static MS	no
(X)	4	400	Static MS	no
(Y)	4	450	Static MS	yes
(Z)	4	350	Dynamic MS	no
(AA)	4	450	Dynamic MS	no
(AB)	4	500	Dynamic MS	no
(AC)	4	550	Dynamic MS	yes
(AD)	6	400	Static MS	no
(AE)	6	450	Static MS	no
(AF)	6	500	Static MS	no
(AG)	6	550	Static MS	no
(AH)	6	600	Static MS	yes
(AI)	6	450	Dynamic MS	no
(AJ)	6	500	Dynamic MS	no
(AK)	6	600	Dynamic MS	no
(AL)	6	650	Dynamic MS	yes
(AM)	8	500	Static MS	no
(AN)	8	550	Static MS	no
(AO)	8	600	Static MS	no
(AP)	8	650	Static MS	yes
(AQ)	8	500	Dynamic MS	no
(AR)	8	600	Dynamic MS	no
(AS)	8	700	Dynamic MS	no
(AT)	8	800	Dynamic MS	no
(AU)	8	850	Dynamic MS	no

(AV)	8	900	Dynamic MS	yes

Table 5.5: System Configurations for Nonrandom Uneven Distribution

Case	Average	Min	Max
(I)	0.11	< 0.01	5.94
(K)	0.08	0.01	1.40
(L)	0.08	0.01	1.04
(N)	0.07	< 0.01	2.09
(O)	0.09	< 0.01	1.93
(P)	0.11	< 0.01	4.47
(R)	0.02	< 0.01	0.58
(S)	0.03	< 0.01	0.94
(T)	0.07	< 0.01	1.32
(U)	0.08	< 0.01	2.39
(W)	0.09	< 0.01	1.61
(X)	0.17	< 0.01	2.27
(Z)	0.05	< 0.01	2.27
(AA)	0.12	< 0.01	12.62
(AB)	0.23	< 0.01	19.68
(AD)	0.03	< 0.01	0.80
(AE)	0.10	< 0.01	3.91
(AF)	0.12	< 0.01	1.52
(AG)	0.17	< 0.01	2.52
(AI)	0.02	< 0.01	4.02
(AJ)	0.03	< 0.01	4.44
(AK)	0.07	< 0.01	7.35

(AM)	0.09	< 0.01	1.65
(AN)	0.16	< 0.01	3.54
(AO)	0.21	< 0.01	2.73
(AQ)	0.01	< 0.01	4.02
(AR)	0.03	< 0.01	10.92
(AS)	0.03	< 0.01	4.47
(AT)	0.05	< 0.01	2.73
(AU)	0.11	< 0.01	7.20

Table 5.6: Response Times for Nonrandom Uneven Distribution (in seconds)

Case	Land manager	Super Client	Server(s)
(I)	5,032	84,944	2,260,084
(K)	3,032	84,905	3,790,300
(L)	4,032	84,920	5,456,476
(N)	5,040	84,880	2,276,980
(O)	6,040	84,856	2,713,480
(P)	7,040	84,904	3,151,692
(R)	29,812	84,832	2,760,348
(S)	30,812	84,832	2,760,348
(T)	31,812	84,892	3,204,184
(U)	32,812	84,892	3,637,784
(W)	7,056	84,976	3,352,772
(X)	8,056	85,012	3,801,604
(Z)	33,556	84,848	3,630,876
(AA)	34,756	84,924	4,166,088
(AB)	35,756	84,908	4,633,776

(AD)	8,072	84,888	3,789,232
(AE)	9,072	84,972	4,227,684
(AF)	10,072	84,996	4,665,558
(AG)	11,072	85,044	5,105,164
(AI)	35,700	84,868	4,222,988
(AJ)	36,700	84,848	4,662,540
(AK)	38,700	84,896	5,521,620
(AM)	10,088	84,948	4,726,412
(AN)	11,088	84,996	5,167,276
(AO)	12,888	85,056	5,964,678
(AQ)	37,644	84,836	4,690,172
(AR)	39,644	84,880	5,591,864
(AS)	41,644	84,844	6,378,682
(AT)	43,644	84,940	7,313,288
(AU)	44,644	84,848	7,628,740

Table 5.7: Total Bytes of Data Received for Nonrandom Uneven Distribution

Case	Average	Min	Max	Second from max
(R)	0.02	0.01	0.30	0.02
(S)	0.02	< 0.01	0.29	0.02
(T)	0.02	< 0.01	0.31	0.02
(U)	0.02	0.01	0.29	0.03
(Z)	0.02	0.01	0.63	0.03
(AA)	0.03	0.01	0.56	0.25
(AB)	0.08	0.02	0.75	0.26
(AI)	0.02	0.01	0.33	0.04

(AJ)	0.02	0.01	0.33	0.09
(AK)	0.02	0.01	0.32	0.10
(AQ)	0.02	0.01	0.26	0.14
(AR)	0.02	0.01	0.26	0.06
(AS)	0.02	0.01	0.28	0.03
(AT)	0.03	0.01	0.45	0.09
(AU)	0.02	0.01	0.61	0.08

Table 5.8: Time to Perform LBA for Nonrandom Uneven Distribution (in seconds)

5.4.2.1 Analysis

Because the avatars are now in groups, every time one avatar moves all other members of its group must be informed of this movement, since they can all see that avatar. This results in a large drop in the number of avatars that a given configuration can hold compared to the case of an even distribution of avatars.

Figure 5.4 shows the maximum number of avatars supported. The Static Av scheme supports less avatars than one servers. This is because every time an avatar on one server moves the other server must be informed. More avatars can be supported by more servers for the Static MS and Dynamic MS schemes. The Dynamic MS scheme supports more avatars than the Static MS scheme for the same number of servers. This is because the load is becoming unbalanced when the Static MS scheme is used.

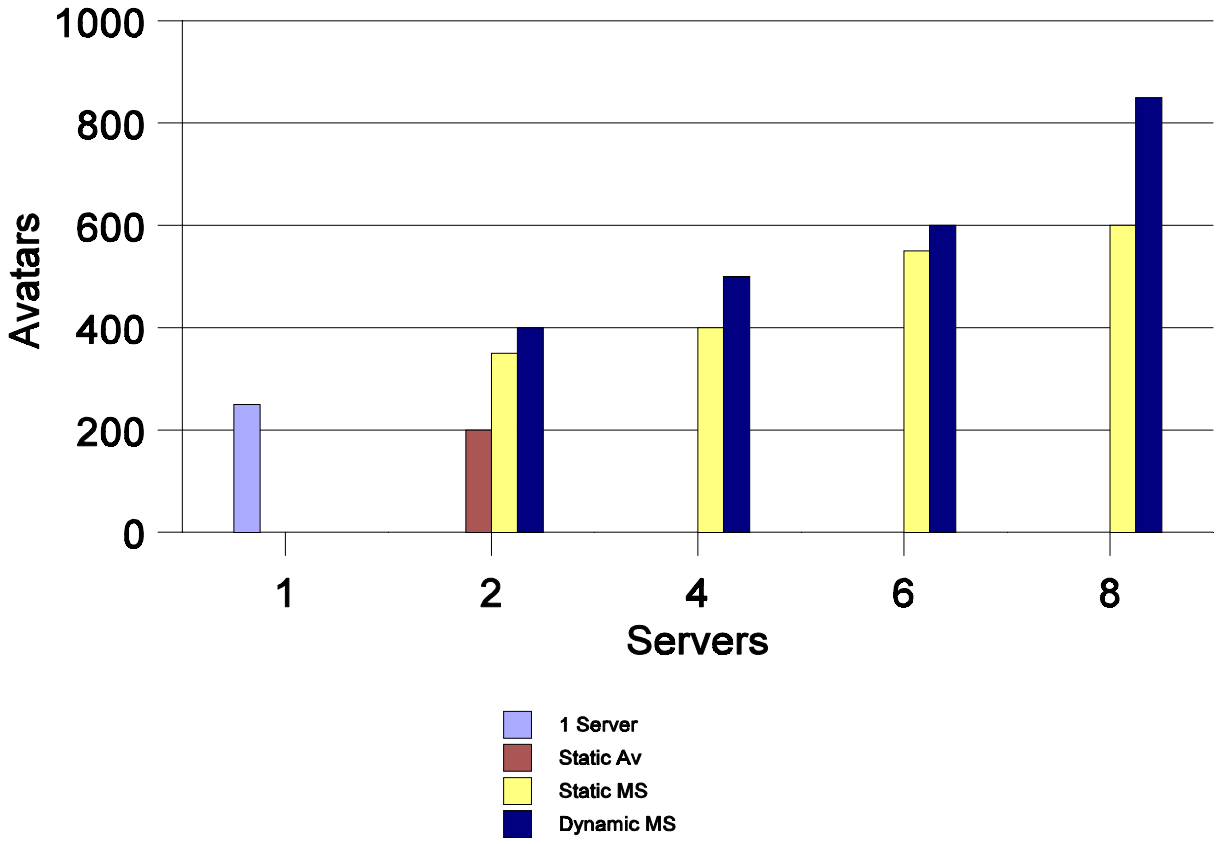


Figure 5.4: Maximum Number of Avatars for Nonrandom Movement, Uneven Configuration

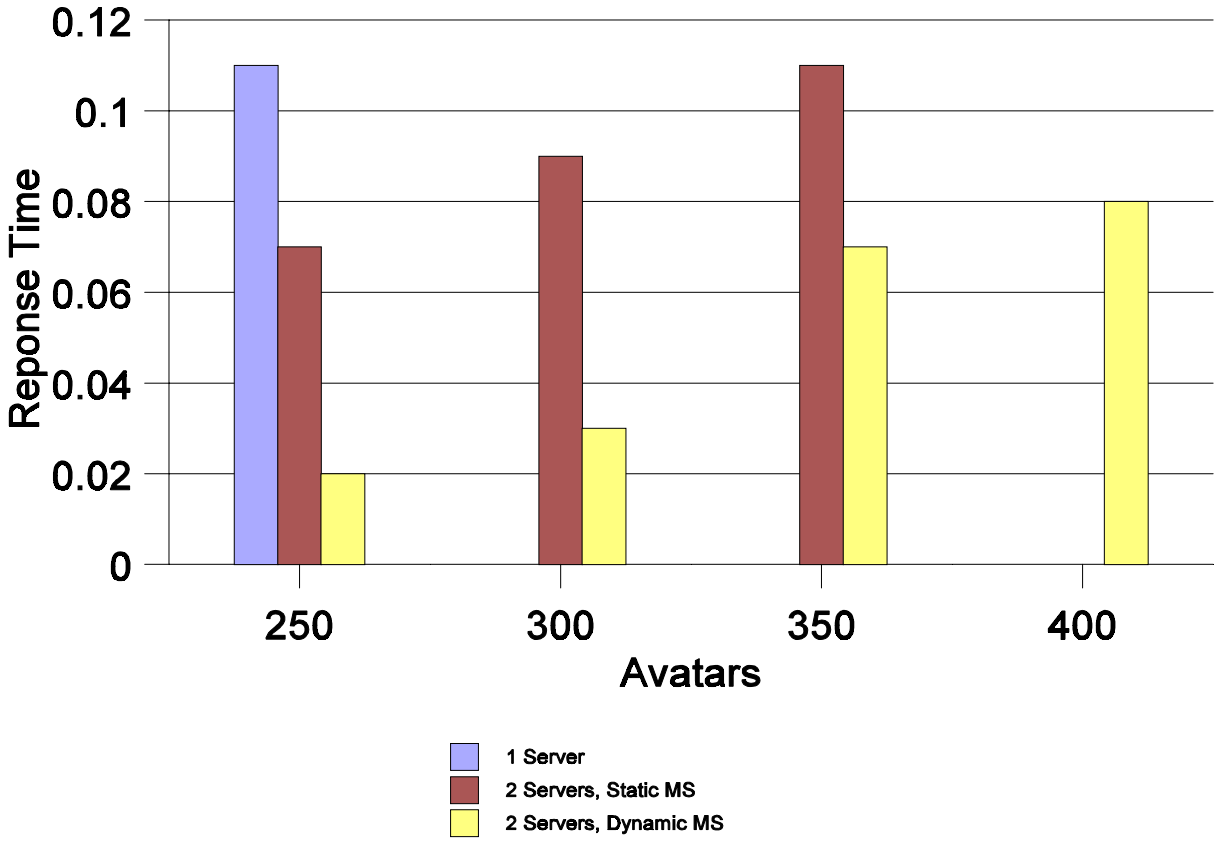


Figure 5.5: Response Times for Nonrandom Movement, Uneven Distribution, 1 and 2 Servers

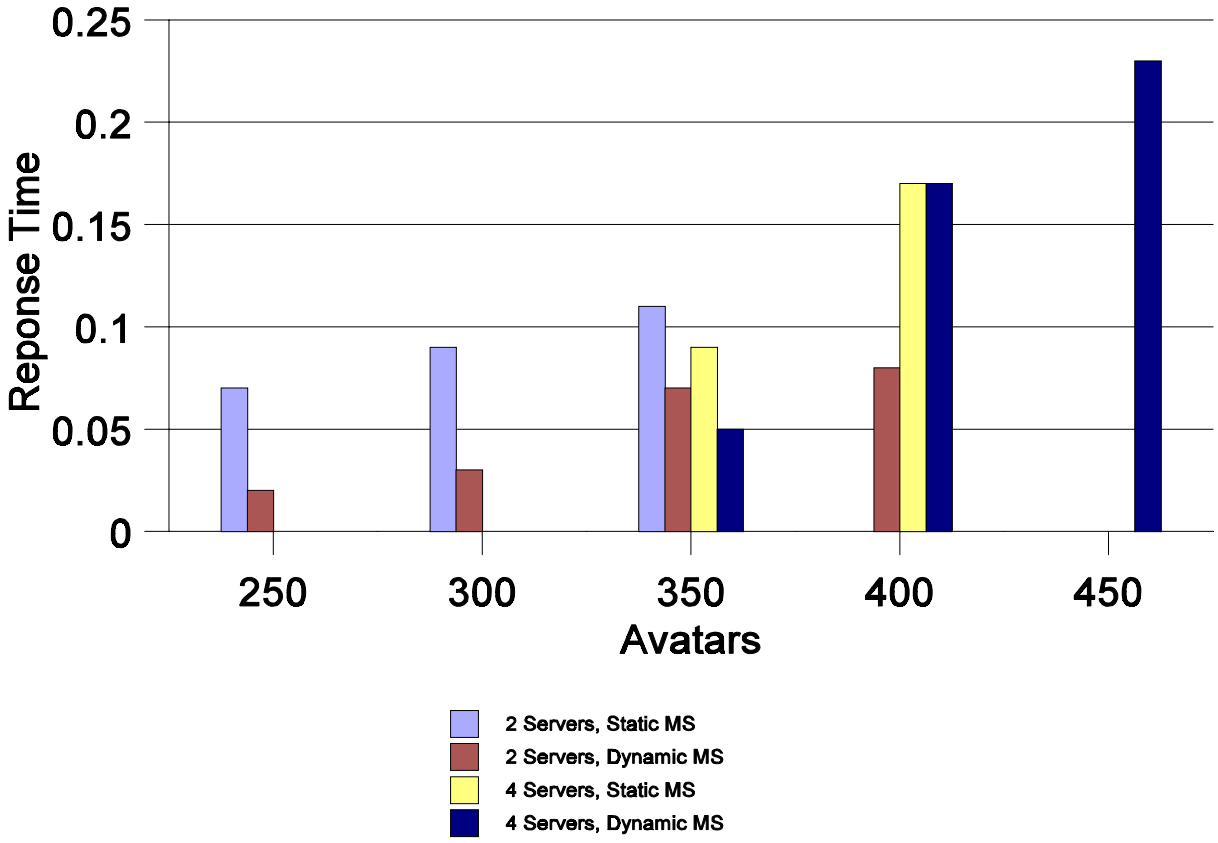


Figure 5.6: Response Times for Nonrandom Movement, Uneven Distribution, 2 and 4 Servers

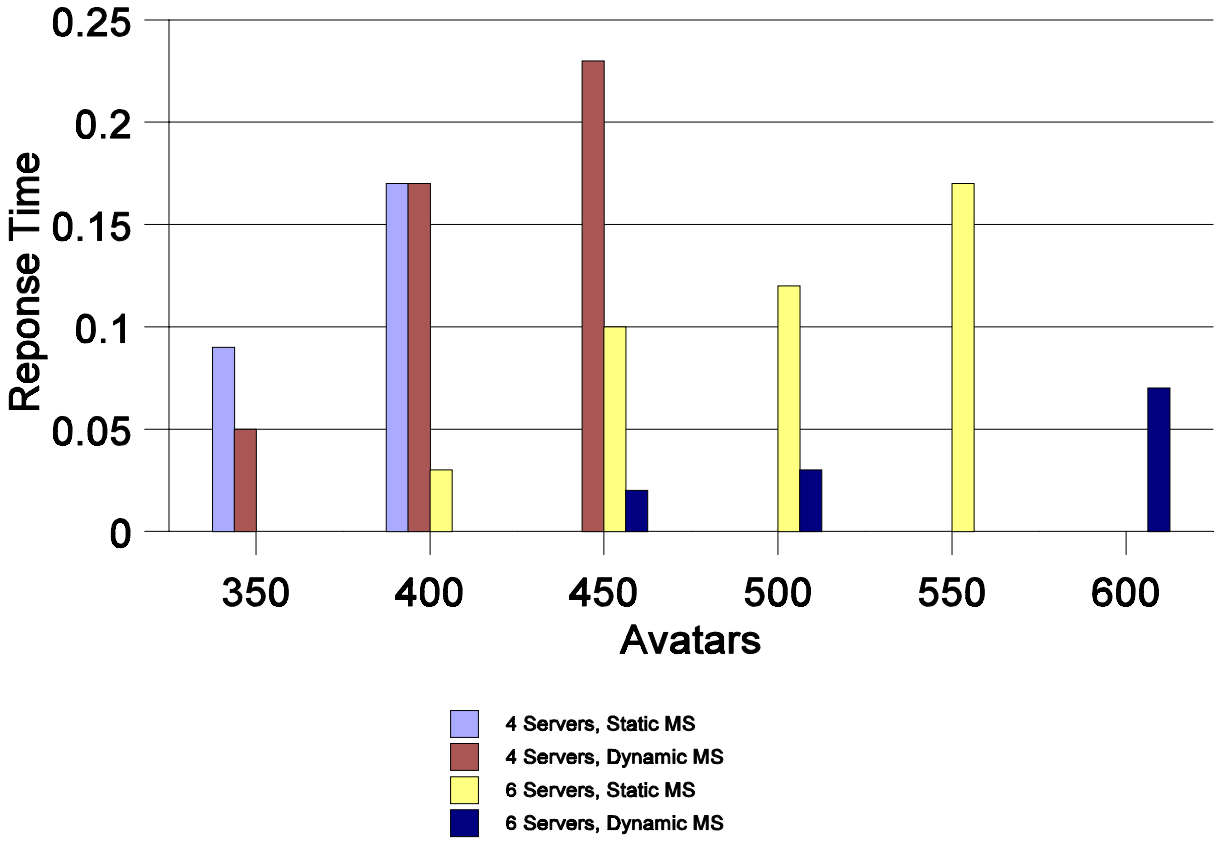


Figure 5.7: Response Times for Nonrandom Movement, Uneven Distribution, 4 and 6 Servers

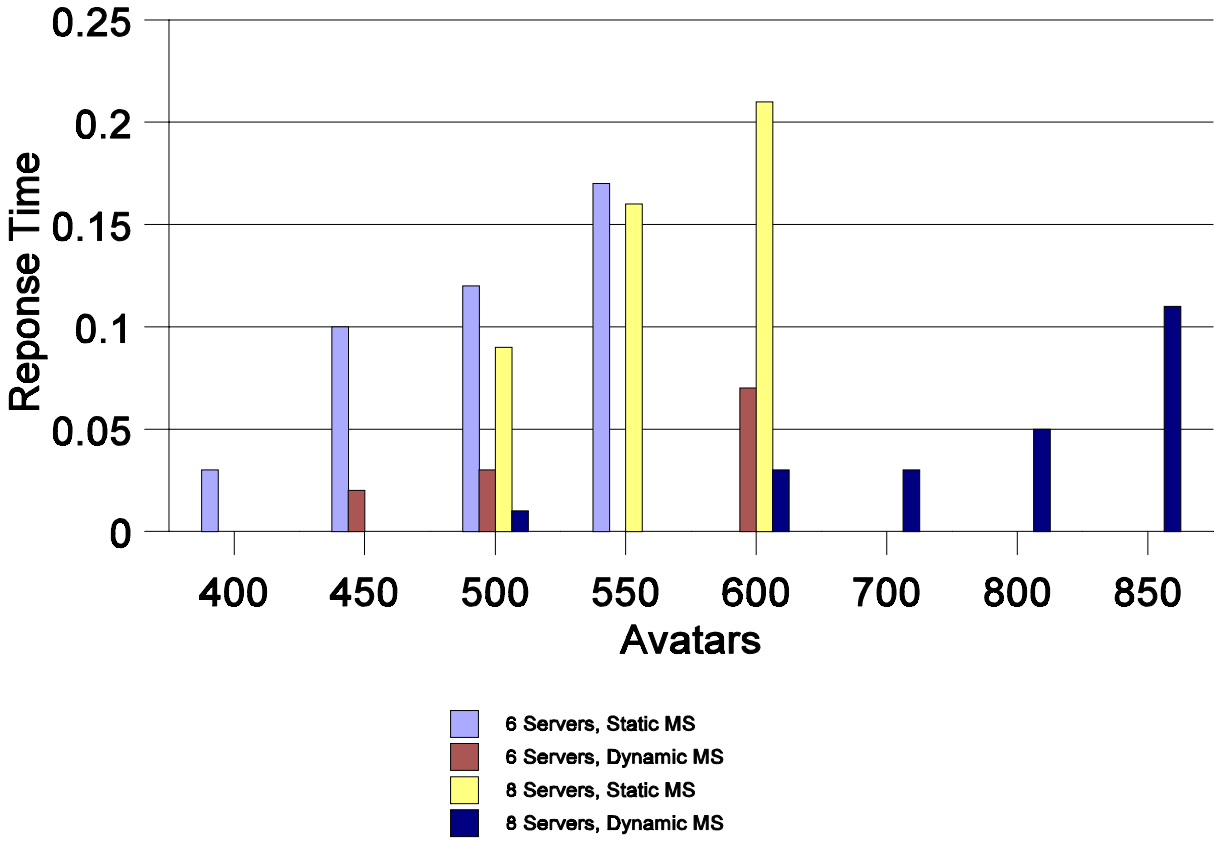


Figure 5.8: Response Times for Nonrandom Movement, Uneven Distribution, 6 and 8 Servers

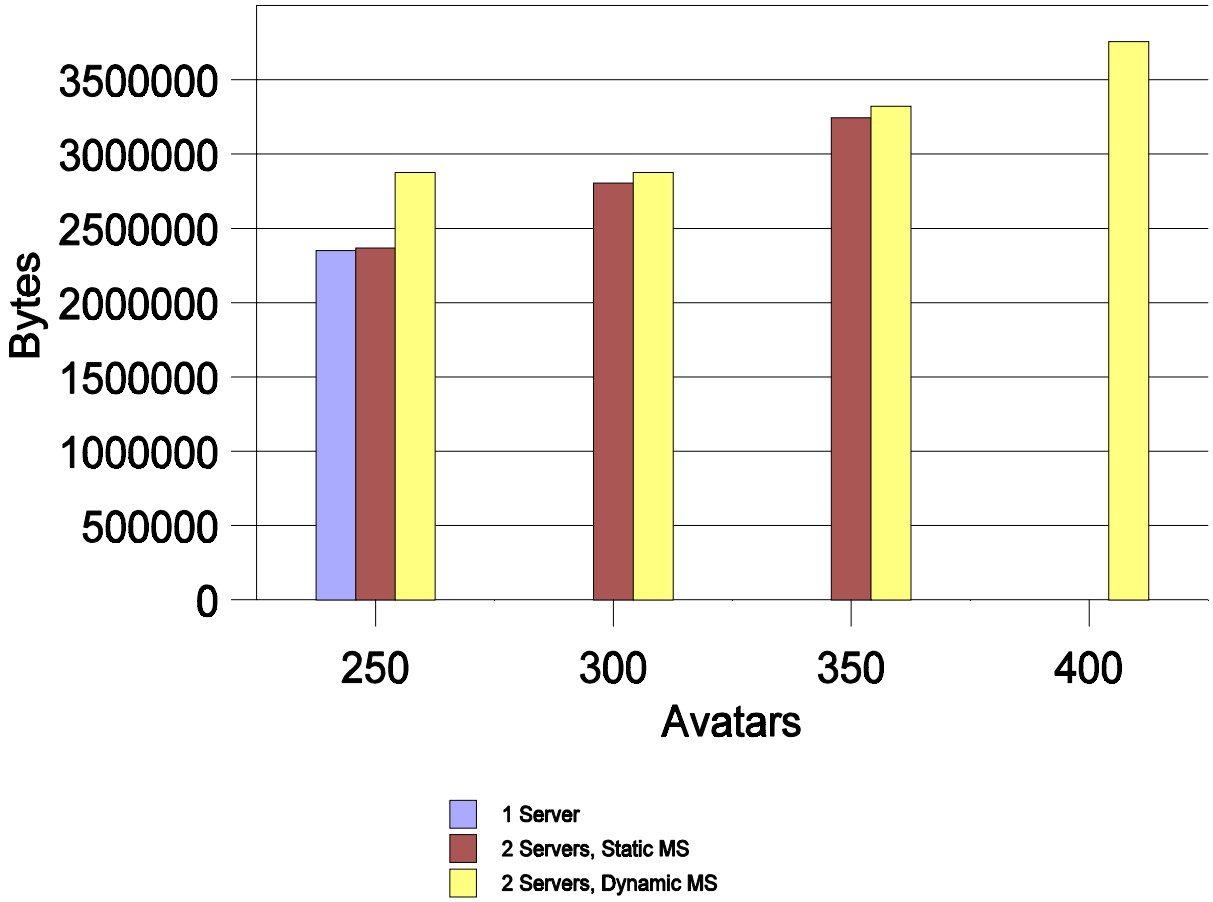


Figure 5.9: Bytes Transferred for Nonrandom Movement, Uneven Distribution, 1 and 2 Servers

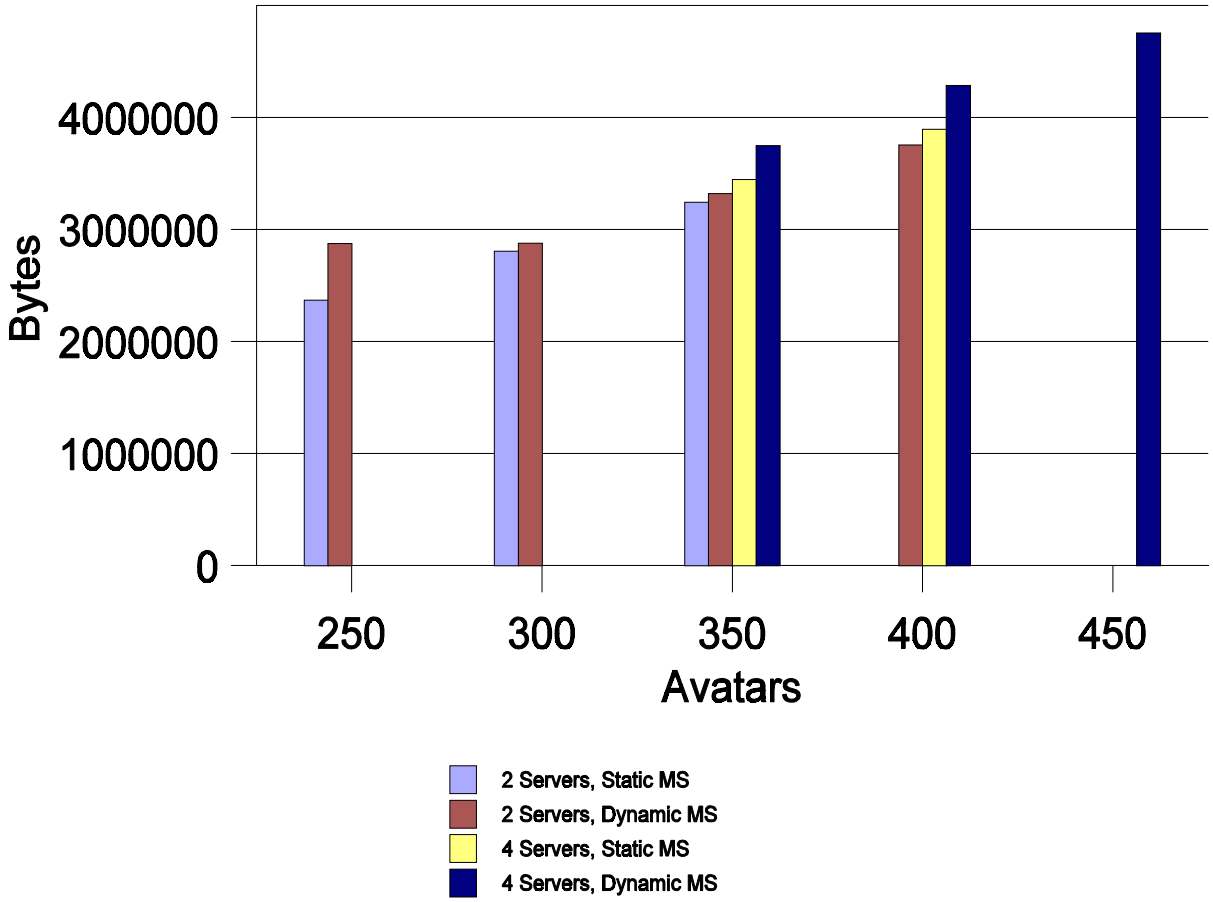


Figure 5.10: Bytes Transferred for Nonrandom Movement, Uneven Distribution, 2 and 4 Servers

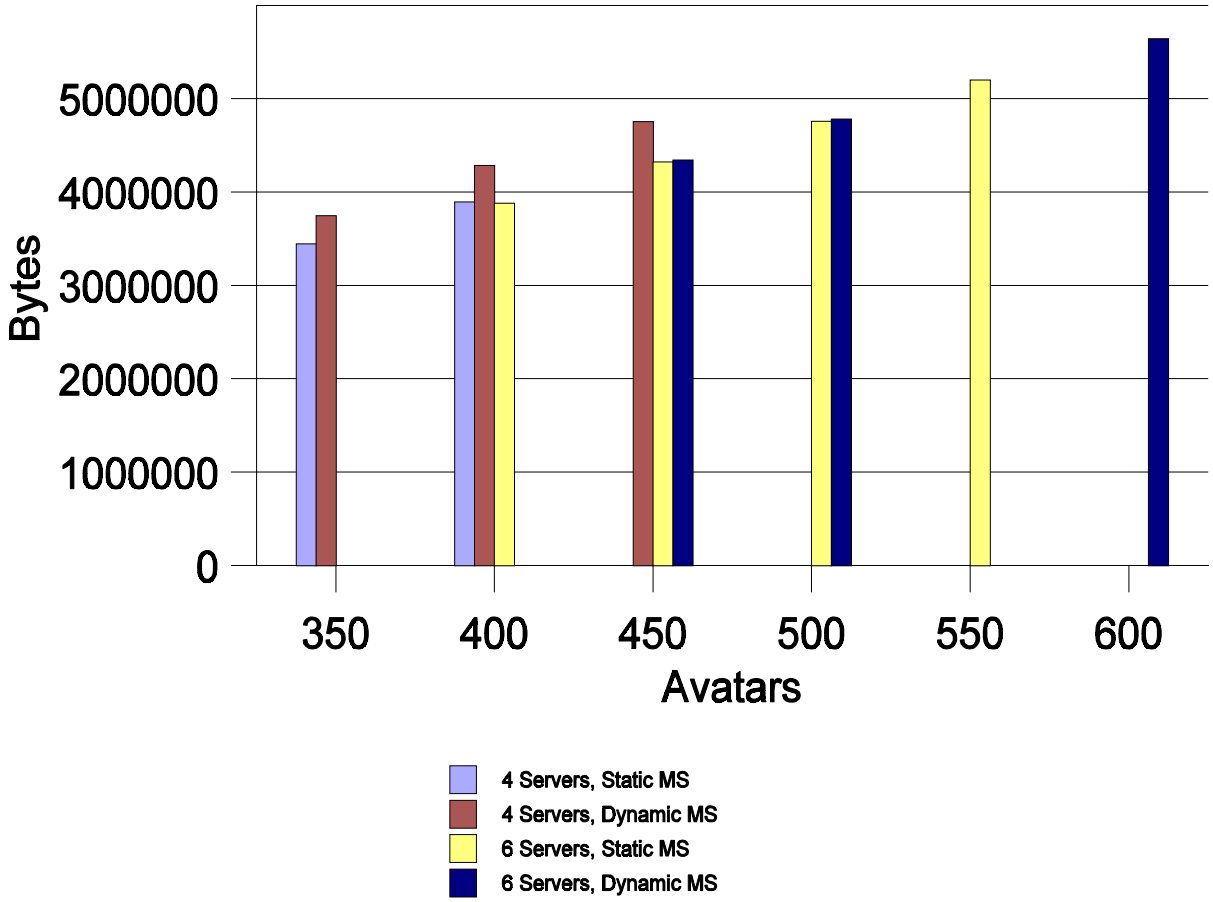


Figure 5.11: Bytes Transferred for Nonrandom Movement, Uneven Distribution, 4 and 6 Servers

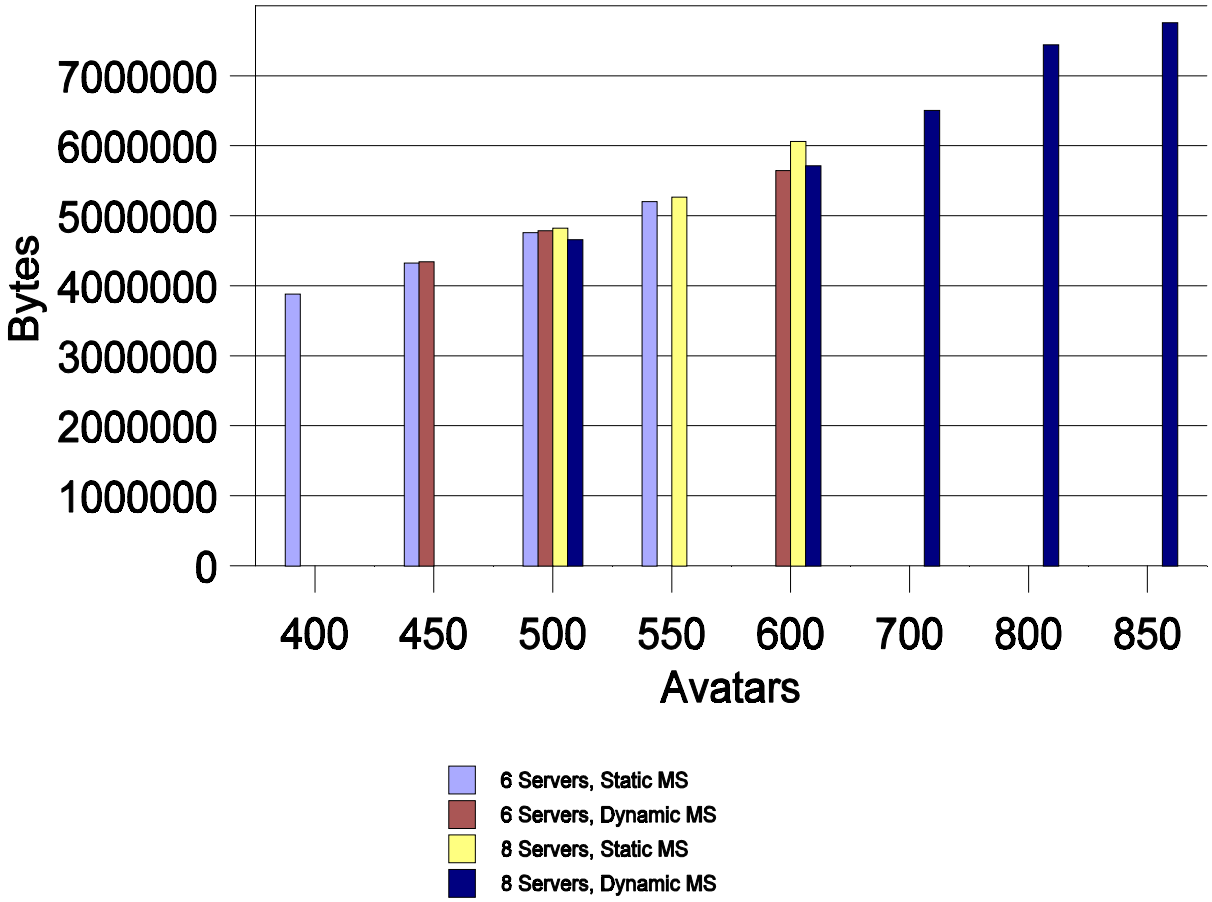


Figure 5.12: Bytes Transferred for Nonrandom Movement, Uneven Distribution, 6 and 8 Servers

Figure 5.5 shows the response times for one and two servers. Using two servers gives better response times than using one server for the same number of avatars. The Dynamic MS scheme gives lower response times than the Static MS scheme for the same number of avatars. Only the Dynamic MS scheme supports 400 avatars. Figure 5.6 show the response times for two and four servers. The Dynamic MS scheme gives lower response times than the Static MS scheme for the same number of servers and avatars. For 400 avatars using two servers the Dynamic MS scheme give the best response time. Using four servers with the Dynamic MS scheme is required to support 450 avatars. Figure 5.7 shows the response times for four and six servers. The Dynamic MS scheme gives the same or lower response times than the Static MS scheme for the same number of servers and avatars. Using six servers with the Dynamic MS scheme is required to support 600 avatars. Figure 5.8 shows the response times for six and eight servers. The Dynamic MS scheme gives lower response times than the Static MS scheme for the same number of servers and avatars. Using eight servers with the Dynamic MS scheme is required to support 850 avatars. In general, the Dynamic MS scheme gives lower response times and can support more avatars because it keeps the load balanced.

Figure 5.9 shows the number of bytes transferred for one and two servers. More bytes are transferred when two servers are used. More bytes are transferred when the Dynamic MS scheme is used. Using two servers with the Dynamic MS scheme is required to support 400 avatars. Figure 5.10 shows the number of bytes transferred for two and four servers. More bytes are transferred when four servers are used, for the same number of avatars. More bytes are transferred when the Dynamic MS scheme is used, for the same number of servers and avatars. Using four servers with

the Dynamic MS scheme is required to support 450 avatars. Figure 5.11 shows the number of bytes transferred for four and six servers. More bytes are transferred when four servers are used for the same number of avatars. More bytes are transferred when the Dynamic MS scheme is used, for the same number of servers and avatars. Using six servers with the Dynamic MS scheme is required to support 600 avatars. Figure 5.12 shows the number of bytes transferred for six and eight servers. About the same number of bytes are transferred for the same number of servers and avatars. Using eight servers with the Dynamic MS scheme is required to support 850 avatars. In general more bytes are transferred across the network when the Dynamic MS technique is used because of the overhead of the load balancing algorithm.

5.5 Random Movement Case

This section contains the experimental results and analysis for the case of random movement of avatars.

5.5.1 Even Distribution

This section contains the experimental results and analysis for an initially even distribution of avatars moving randomly. The system configurations are shown in table 5.9. Response times are shown in table 5.10. The bytes of data received by each software component are shown in table 5.11. The time to perform the load balancing algorithm is shown in table 5.12.

Case	# of servers	Total # of av	Distribution	IBO
(A)	1	784	N/A	no
(B)	1	900	N/A	yes
(C)	2	784	Static MS	no
(D)	2	900	Static MS	no
(E)	2	784	Dynamic MS	no
(F)	2	900	Dynamic MS	no

Table 5.9: System Configurations for Random Even Distribution

Case	Average	Min	Max
(A)	0.11	< 0.01	4.24
(C)	0.05	< 0.01	3.31
(D)	0.09	< 0.01	8.94
(E)	0.01	< 0.01	0.81
(F)	0.08	< 0.01	4.55

Table 5.10: Response Times for Random Even Distribution (in seconds)

Case	Land manager	Super Client	Server(s)
(A)	15,712	85,120	6,299,824
(C)	15,720	85,796	7,526,860
(D)	18,040	86,116	8,604,408
(E)	32,672	84,904	7,526,860
(F)	42,812	85,916	9,993,836

Table 5.11: Total Bytes of Data Received for Random Even Distribution

Case	Average	Min	Max	Second from max
(E)	0.02	< 0.01	0.09	0.03
(F)	0.03	< 0.01	0.81	0.11

Table 5.12: Time to Perform LBA for Random Even Distribution (in seconds)

5.5.1.1 Analysis

Figure 5.13 shows the maximum number of avatars supported. More avatars can be supported my two servers than by one. Figure 5.14 shows the response times. Using two servers gives better response times than using one servers for the same number of avatars. Figure 5.15 shows the number of bytes transferred. More bytes are transferred when two servers are used. More bytes are transferred when the Dynamic MS scheme is used, due to the overhead of the load balancing algorithm.

The network became the bottleneck before either the Static MS or Dynamic MS schemes stopped working (i.e. data generated by the super clients to be placed on the network was being generated faster than the data was actually going onto the network).

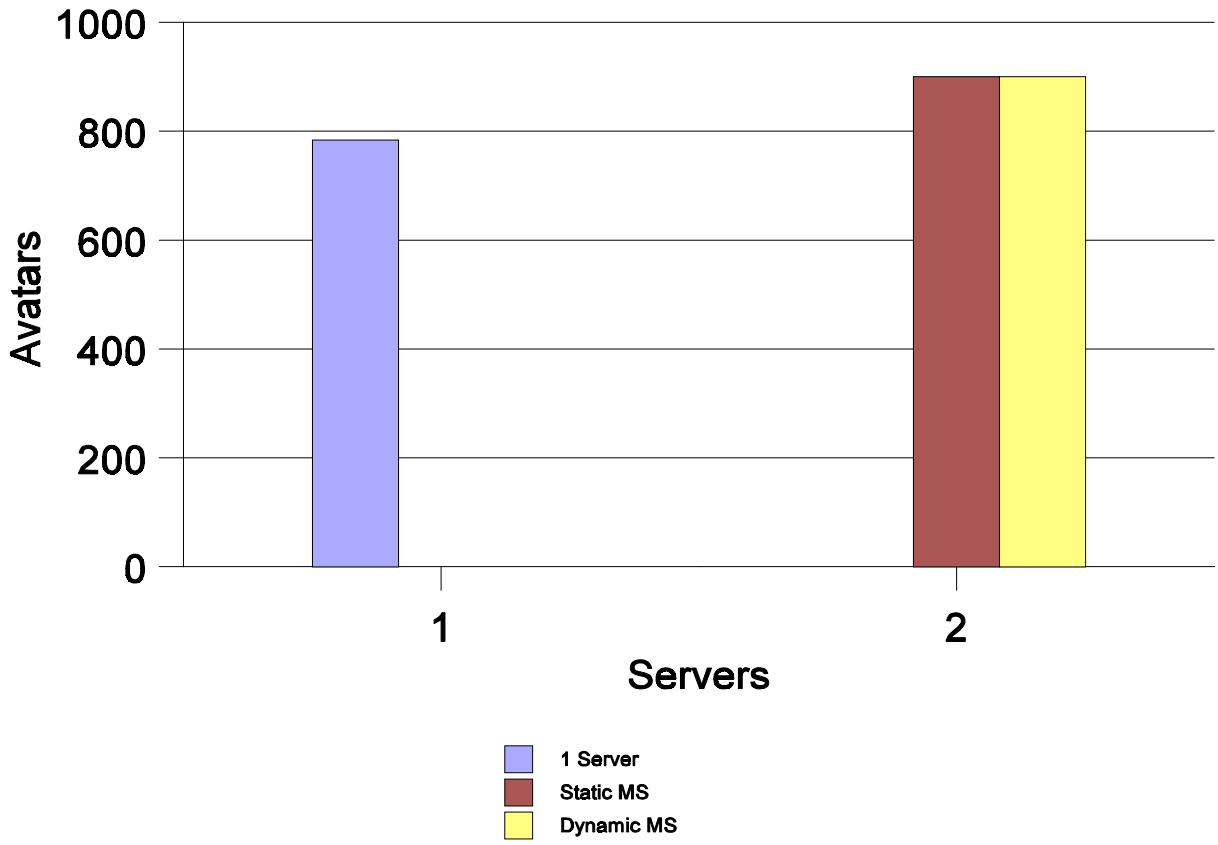


Figure 5.13: Maximum Number of Avatars for Random Movement, Even Configuration

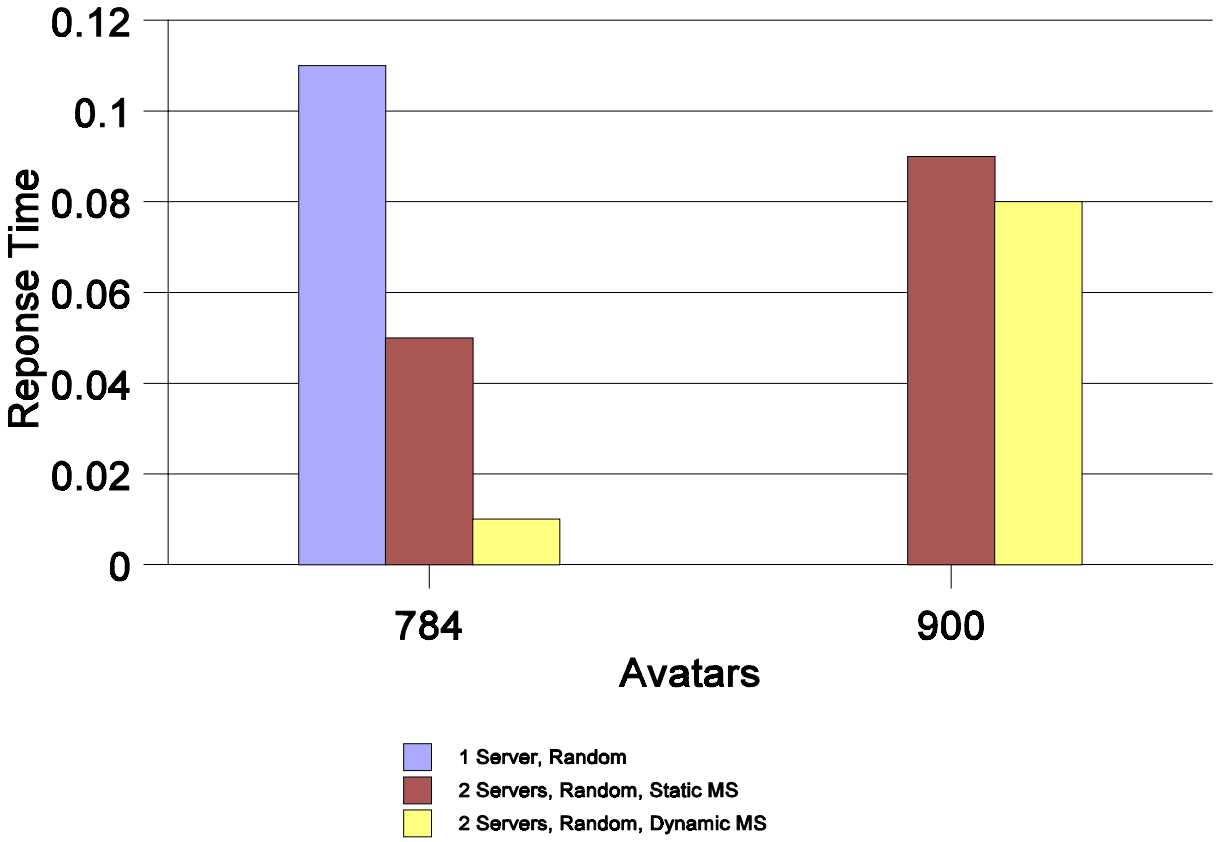


Figure 5.14: Response Times for Random movement, Even Distribution

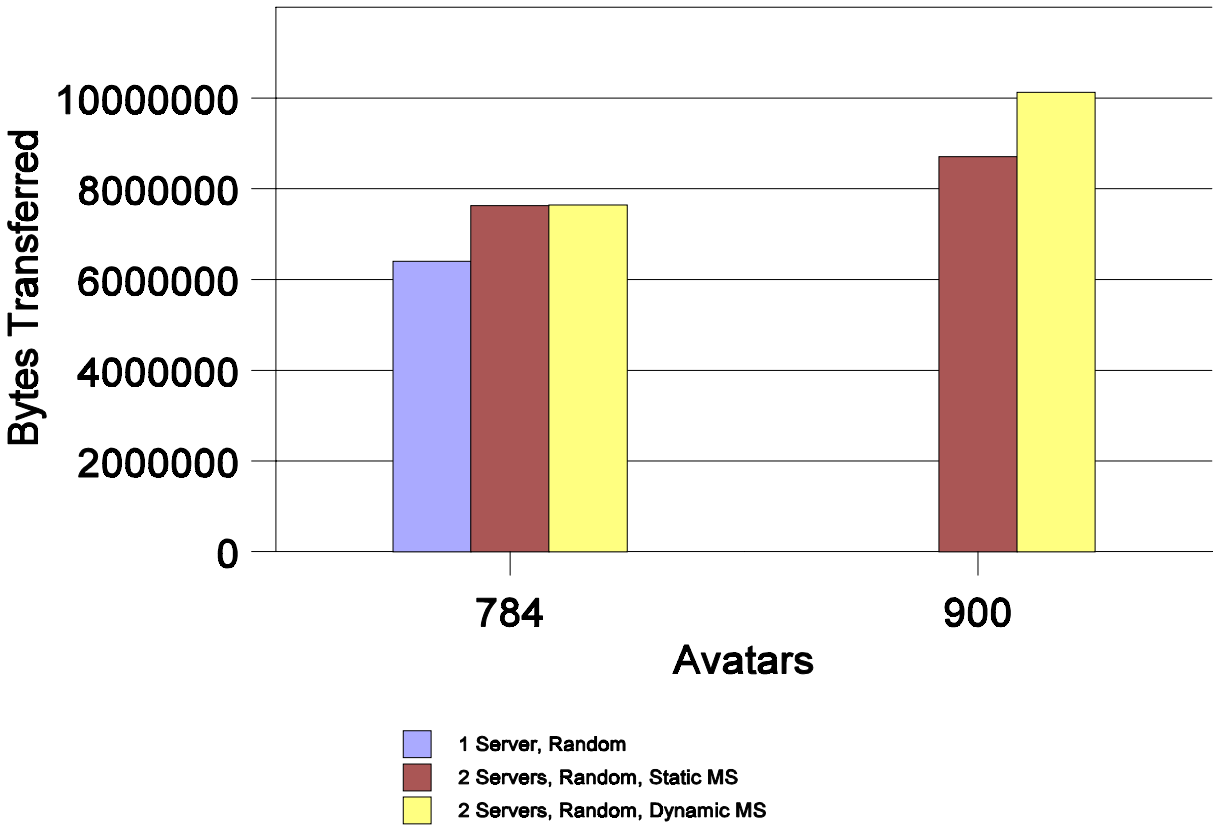


Figure 5.15: Bytes Transferred for Random Movement, Even Distribution

5.5.2 Uneven Distribution

This section contains the experimental results and analysis for an uneven distribution of avatars moving randomly. The system configurations are shown in table 5.13. Response times are shown in table 5.14. The bytes of data received by each software component are shown in table 5.15. The time to perform the load balancing algorithm is shown in table 5.16.

Case	# of servers	Total # of av	Distribution	IBO
(G)	1	250	N/A	no
(H)	1	300	N/A	yes
(I)	2	250	Static MS	no
(J)	2	300	Static MS	no
(K)	2	350	Static MS	yes
(L)	2	250	Dynamic MS	no
(M)	2	300	Dynamic MS	no
(N)	2	350	Dynamic MS	no
(O)	2	400	Dynamic MS	yes
(P)	4	300	Static MS	no
(Q)	4	350	Static MS	no
(R)	4	400	Static MS	no
(S)	4	450	Static MS	yes
(T)	4	300	Dynamic MS	no
(U)	4	350	Dynamic MS	no
(V)	4	400	Dynamic MS	no
(W)	4	450	Dynamic MS	no

(X)	4	500	Dynamic MS	no
(Y)	4	550	Dynamic MS	yes
(Z)	6	400	Static MS	no
(AA)	6	450	Static MS	no
(AB)	6	500	Static MS	no
(AC)	6	550	Static MS	no
(AD)	6	600	Static MS	yes
(AE)	6	400	Dynamic MS	no
(AF)	6	450	Dynamic MS	no
(AG)	6	500	Dynamic MS	no
(AH)	6	550	Dynamic MS	no
(AI)	6	600	Dynamic MS	no
(AJ)	6	650	Dynamic MS	yes
(AK)	8	550	Static MS	no
(AL)	8	600	Static MS	no
(AM)	8	650	Static MS	no
(AN)	8	700	Static MS	no
(AO)	8	750	Static MS	yes
(AP)	8	550	Dynamic MS	no
(AQ)	8	600	Dynamic MS	no
(AR)	8	650	Dynamic MS	no
(AS)	8	700	Dynamic MS	no
(AT)	8	750	Dynamic MS	no
(AU)	8	800	Dynamic MS	no
(AV)	8	850	Dynamic MS	no
(AW)	8	900	Dynamic MS	no

(AX)	8	950	Dynamic MS	yes

Table 5.13: System Configurations for Random Uneven Distribution

Case	Average	Min	Max
(G)	0.10	< 0.01	7.59
(I)	0.11	< 0.01	19.07
(J)	0.02	< 0.01	3.49
(L)	0.04	< 0.01	1.31
(M)	0.04	< 0.01	6.63
(N)	3.78	< 0.01	86.38
(P)	0.11	< 0.01	21.57
(Q)	0.36	< 0.01	39.16
(R)	1.32	< 0.01	70.29
(T)	0.01	< 0.01	3.05
(U)	0.04	< 0.01	9.98
(V)	0.17	< 0.01	26.15
(W)	0.24	< 0.01	32.92
(X)	0.66	< 0.01	86.13
(Z)	0.08	< 0.01	14.26
(AA)	0.10	< 0.01	13.01
(AB)	0.55	< 0.01	47.30
AC)	1.00	< 0.01	67.61
(AE)	0.06	< 0.01	8.68
(AF)	0.09	< 0.01	7.65
(AG)	0.08	< 0.01	19.98
(AH)	1.16	< 0.01	84.77

(AI)	0.04	< 0.01	8.65
(AK)	0.03	< 0.01	1.39
(AL)	0.18	< 0.01	28.03
(AM)	0.18	< 0.01	23.87
(AN)	0.86	< 0.01	62.45
(AP)	0.02	< 0.01	4.88
(AQ)	0.03	< 0.01	10.62
(AR)	0.01	< 0.01	2.01
(AS)	0.01	< 0.01	2.47
(AT)	0.08	< 0.01	11.68
(AU)	0.03	< 0.01	3.46
(AV)	1.48	< 0.01	87.92
(AW)	0.33	< 0.01	33.49

Table 5.14: Response Times for Random Uneven Distribution (in seconds)

Case	Land manager	Super Client	Server(s)
(G)	5,032	84,896	2,258,752
(I)	5,040	86,032	2,310,904
(J)	6,040	86,444	2,745,356
(L)	30,132	84,824	2,313,492
(M)	31,132	84,832	2,751,896
(N)	30,884	84,972	3,208,820
(P)	6,056	86,700	2,780,364
(Q)	7,056	86,292	3,222,088
(R)	8,056	86,188	3,627,144
(T)	32,076	84,820	2,807,492

(U)	33,076	85,348	3,256,756
(V)	34,076	84,992	3,700,488
(W)	35,076	85,372	4,146,164
(X)	36,076	84,960	4,577,524
(Z)	8,072	86,540	3,650,208
(AA)	9,072	86,272	4,071,232
(AB)	10,072	85,880	4,504,596
(AC)	11,072	86,012	4,940,692
(AE)	34,700	85,636	3,858,704
(AF)	35,860	84,808	4,185,176
(AG)	37,020	86,140	4,673,548
(AH)	38,020	86,016	5,108,084
(AI)	39,020	85,104	5,512,268
(AK)	11,088	85,964	6,309,428
(AL)	12,088	86,044	6,800,120
(AM)	13,088	86,068	7,315,564
(AN)	14,088	86,672	7,688,668
(AP)	38,964	85,116	5,104,220
(AQ)	40,012	86,128	5,603,584
(AR)	40,964	85,104	5,920,528
(AS)	41,964	85,116	6,360,028
(AT)	42,964	85,268	6,754,168
(AU)	44,124	85,916	7,223,844
(AV)	44,676	84,876	7,613,128
(AW)	46,124	86,256	8,030,900

Table 5.15: Total Bytes of Data Received for Random Uneven Distribution

Case	Average	Min	Max	Second from max
(L)	0.02	0.01	0.29	0.02
(M)	0.02	0.01	0.22	0.02
(T)	0.06	0.01	0.63	0.09
(U)	0.03	0.01	0.56	0.03
(V)	0.07	0.01	0.75	0.26
(W)	0.03	0.01	0.57	0.06
(X)	0.06	0.01	0.75	0.26
(AE)	0.02	0.01	0.30	0.02
(AF)	0.02	0.01	0.32	0.09
(AG)	0.02	0.01	0.30	0.03
(AH)	0.02	0.01	0.30	0.09
(AI)	0.03	0.01	0.31	0.09
(AP)	0.02	0.01	0.30	0.04
(AQ)	0.02	0.01	0.27	0.13
(AR)	0.03	0.01	0.31	0.13
(AS)	0.03	0.01	0.30	0.07
(AT)	0.03	0.01	0.45	0.04
(AU)	0.02	< 0.01	0.44	0.04
(AV)	0.04	0.01	0.59	0.16
(AW)	0.03	0.01	0.41	0.12

Table 5.16: Time to Perform LBA for Random Uneven Distribution (in seconds)

5.5.2.1 Analysis

Because the avatars are now in groups, every time one avatar moves all other members of its group must be informed of this movement, since they can all see that avatar. This results in a large drop

in the number of avatars that a given configuration can hold compared to the case of an even distribution of avatars. Figure 5.16 shows the maximum number of avatars supported. More avatars can be supported my more servers. The Dynamic MS scheme supports more avatars than the Static MS scheme for the same number of servers, because the Dynamic MS scheme keeps the load balanced.

Figure 5.17 shows the response times for one and two servers. Using two servers, Dynamic MS gives a better response time than using one server, but using two servers, Static MS gives a worse response time. The Dynamic MS scheme gives a lower response time than the Static MS scheme for 250 avatars, but a higher response time for 300 avatars. Only the Dynamic MS scheme supports 350 avatars. Figure 5.18 shows the response times for two and four servers. The Dynamic MS scheme gives lower response times than the Static MS scheme for the same number of servers and avatars, except for 300 avatars and two servers. Using four servers with the Dynamic MS scheme is required to support 500 avatars. Figure 5.19 shows the response times for four and six servers. The Dynamic MS scheme gives lower response times than the Static MS scheme for the same number of servers and avatars, except for 550 avatars with six servers. Using six servers with the Dynamic MS scheme is required to support 600 avatars. Figure 5.20 shows the response times for six and eight servers. The Dynamic MS scheme gives lower response times than the Static MS scheme for the same number of servers and avatars, except for 550 avatars with six servers. Using eight servers with the Dynamic MS scheme is required to support 900 avatars. In general, the Dynamic MS scheme gives lower response times and can support more avatars because it keeps the load balanced.

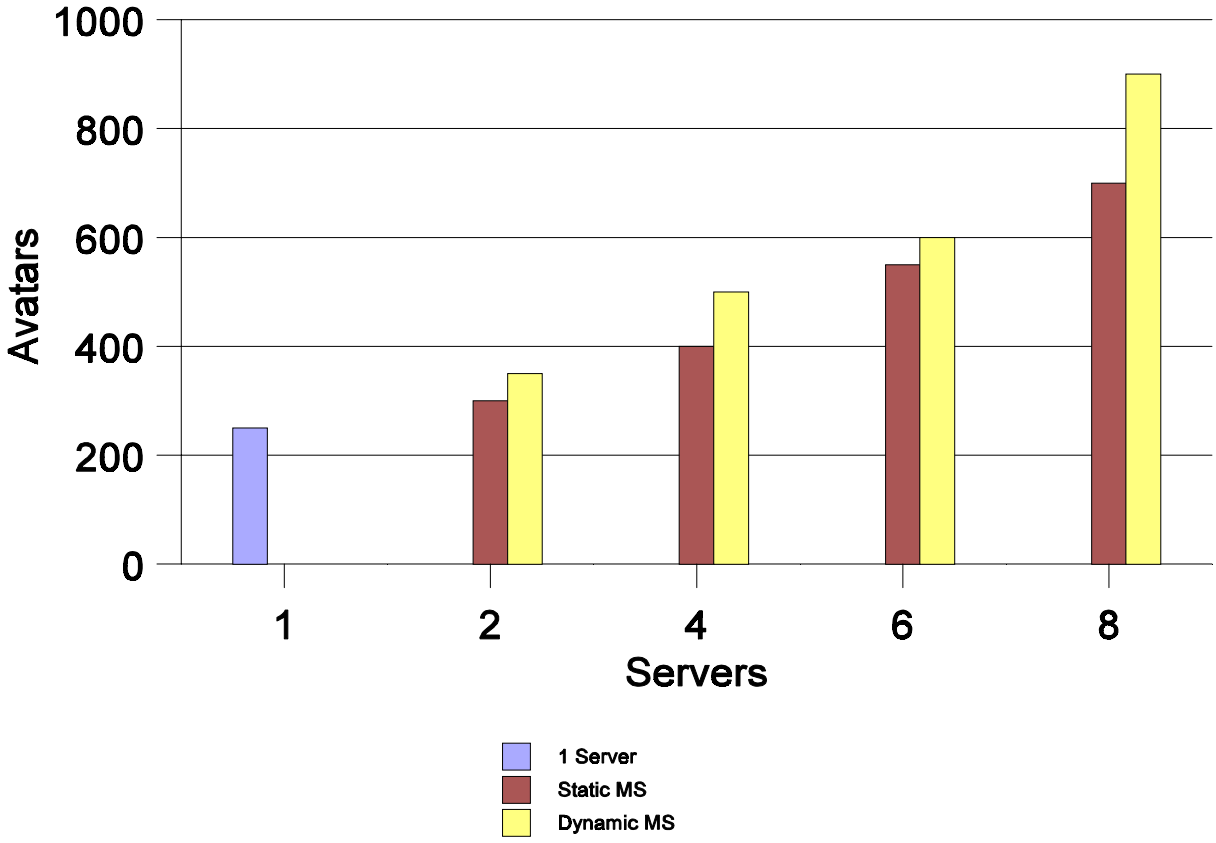


Figure 5.16: Maximum Number of Avatars for Random Movement, Uneven Configuration

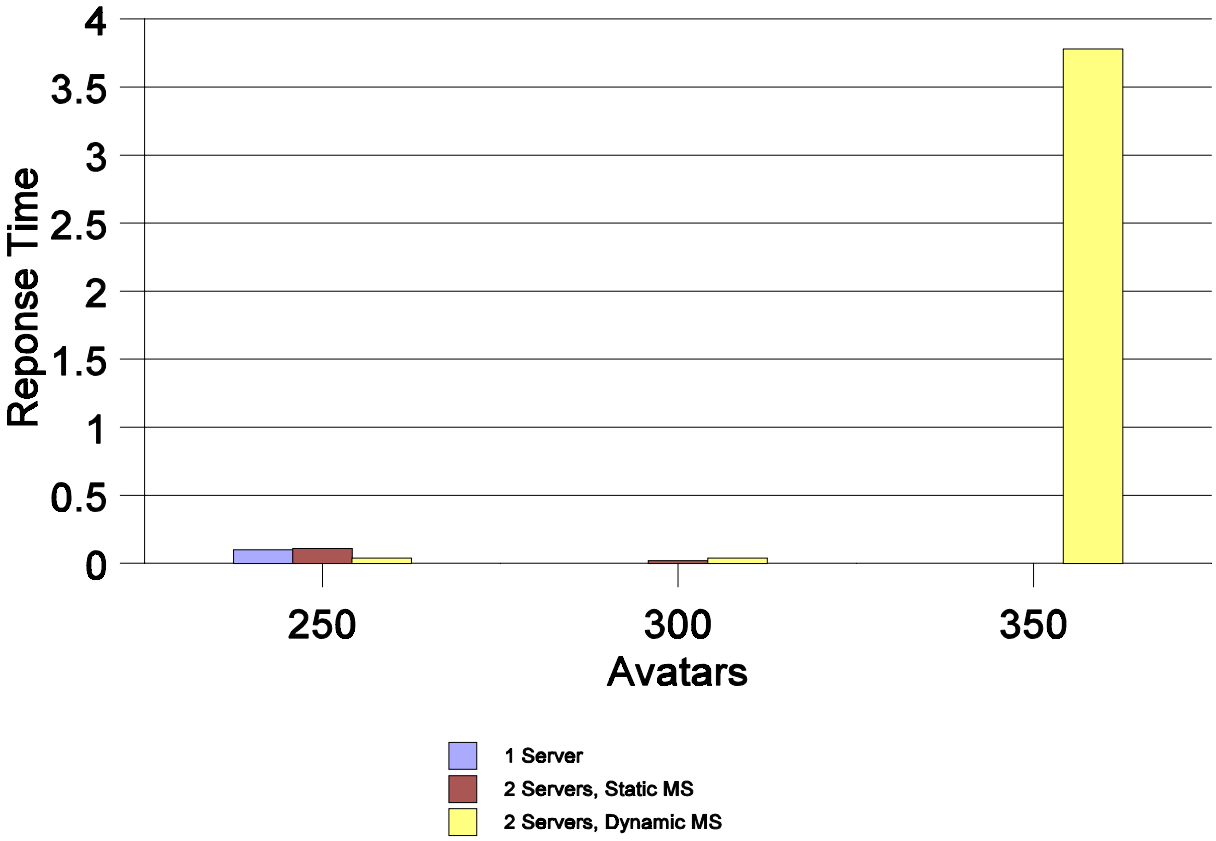


Figure 5.17: Response Times for Random Movement, Uneven Distribution, 1 and 2 Servers

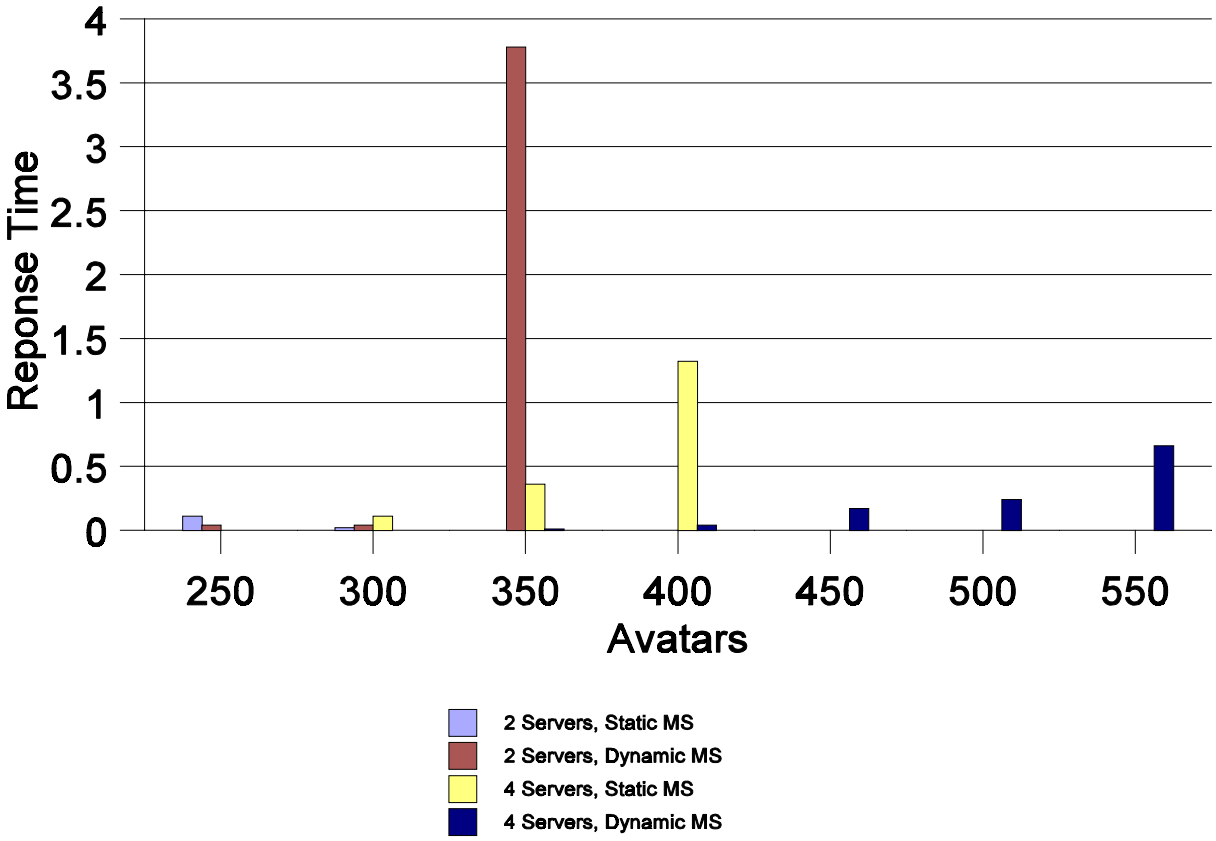


Figure 5.18: Response Times for Random Movement, Uneven Distribution, 2 and 4 Servers

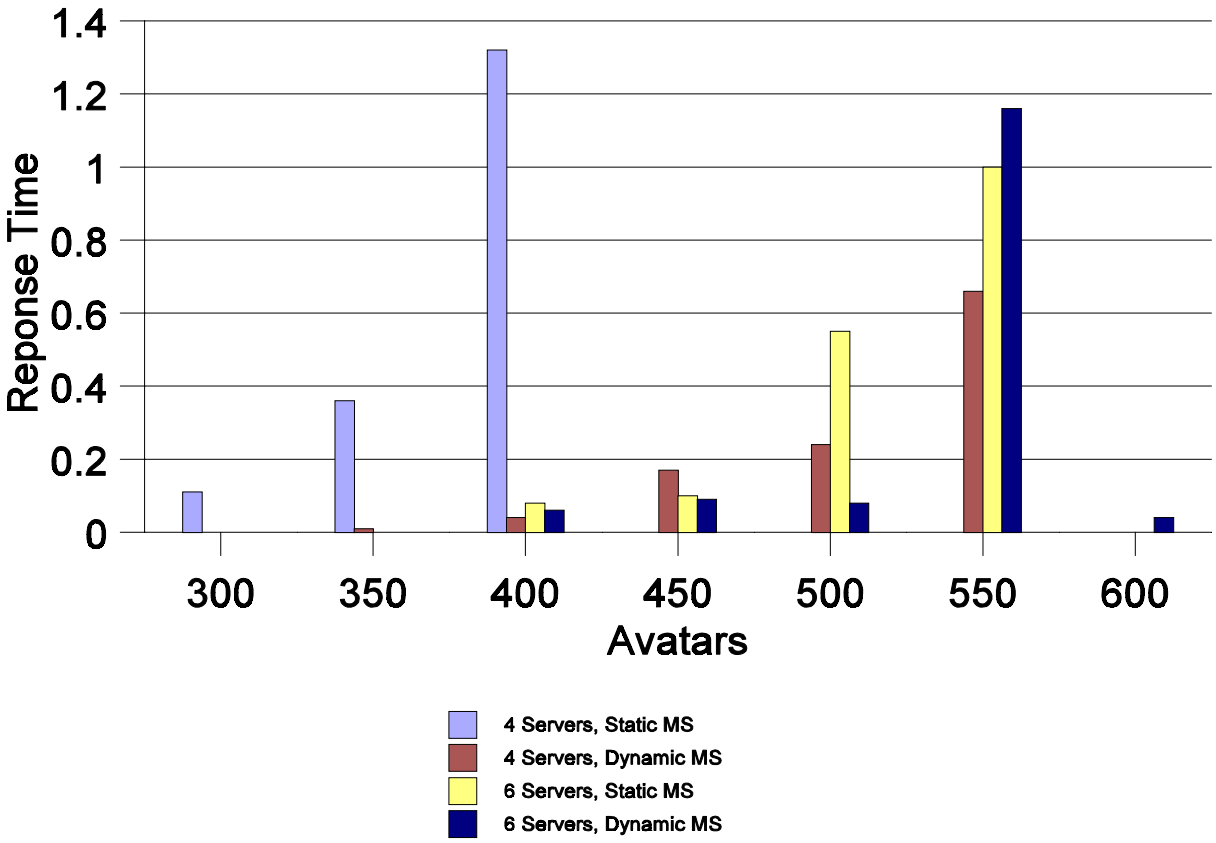


Figure 5.19: Response Times for Random Movement, Uneven Distribution, 4 and 6 Servers

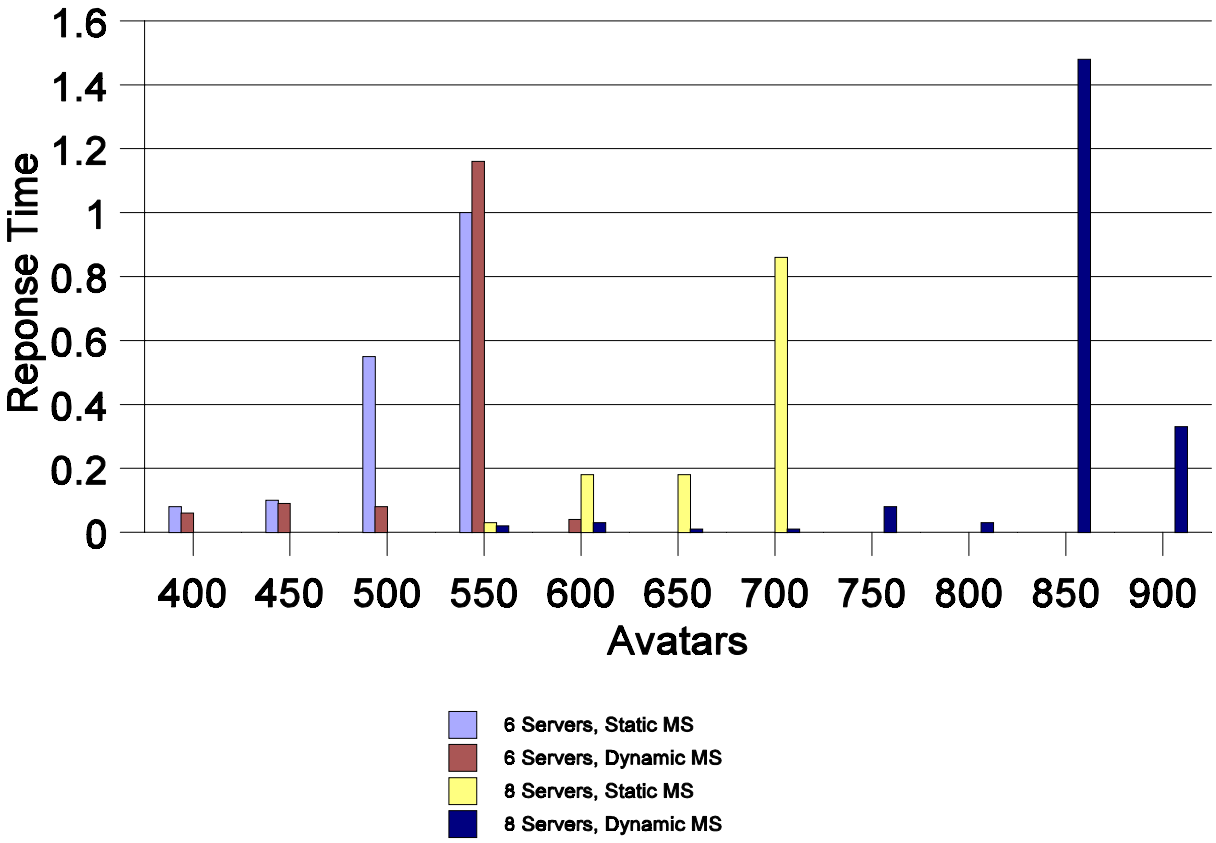


Figure 5.20: Response Times for Random Movement, Uneven Distribution, 6 and 8 Servers

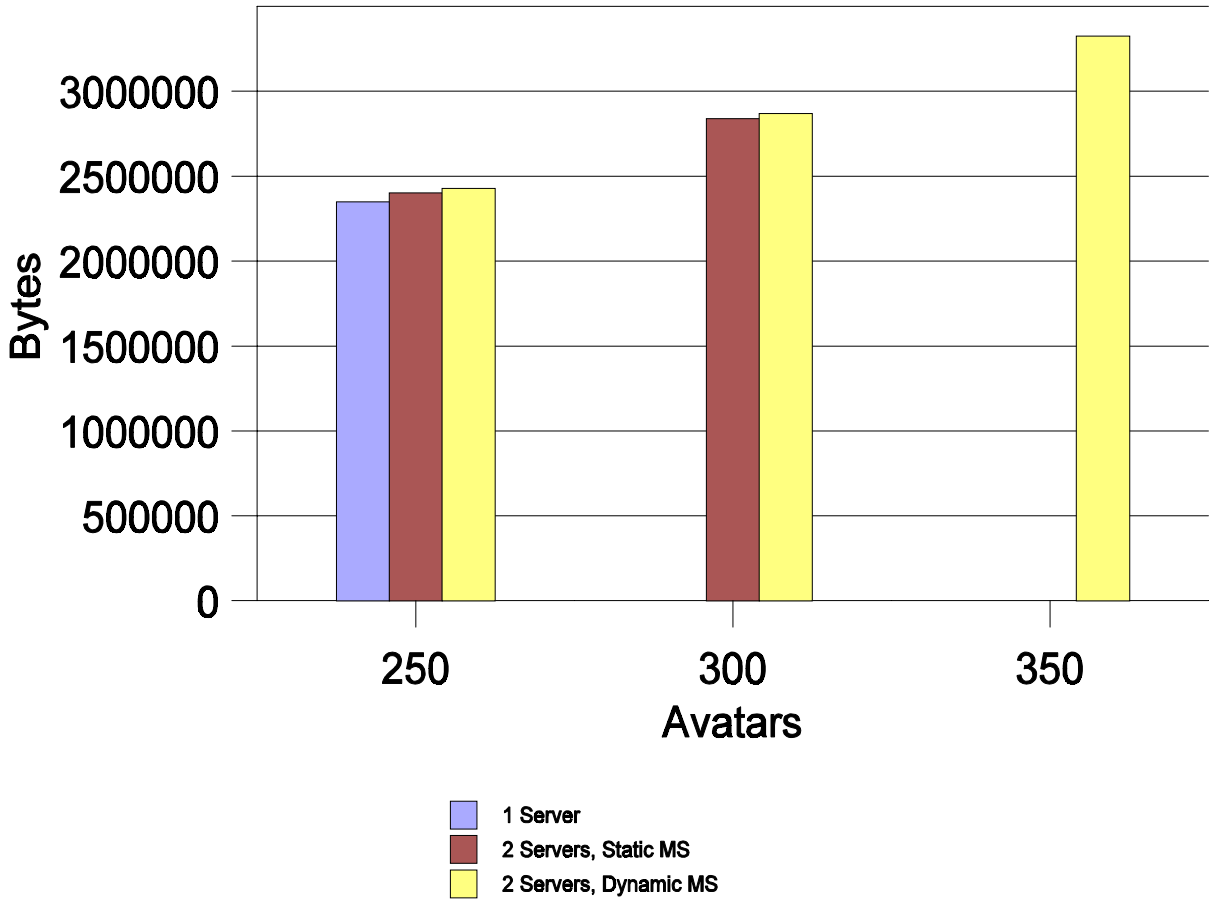


Figure 5.21: Bytes Transferred for Random Movement, Uneven Distribution, 1 and 2 Servers

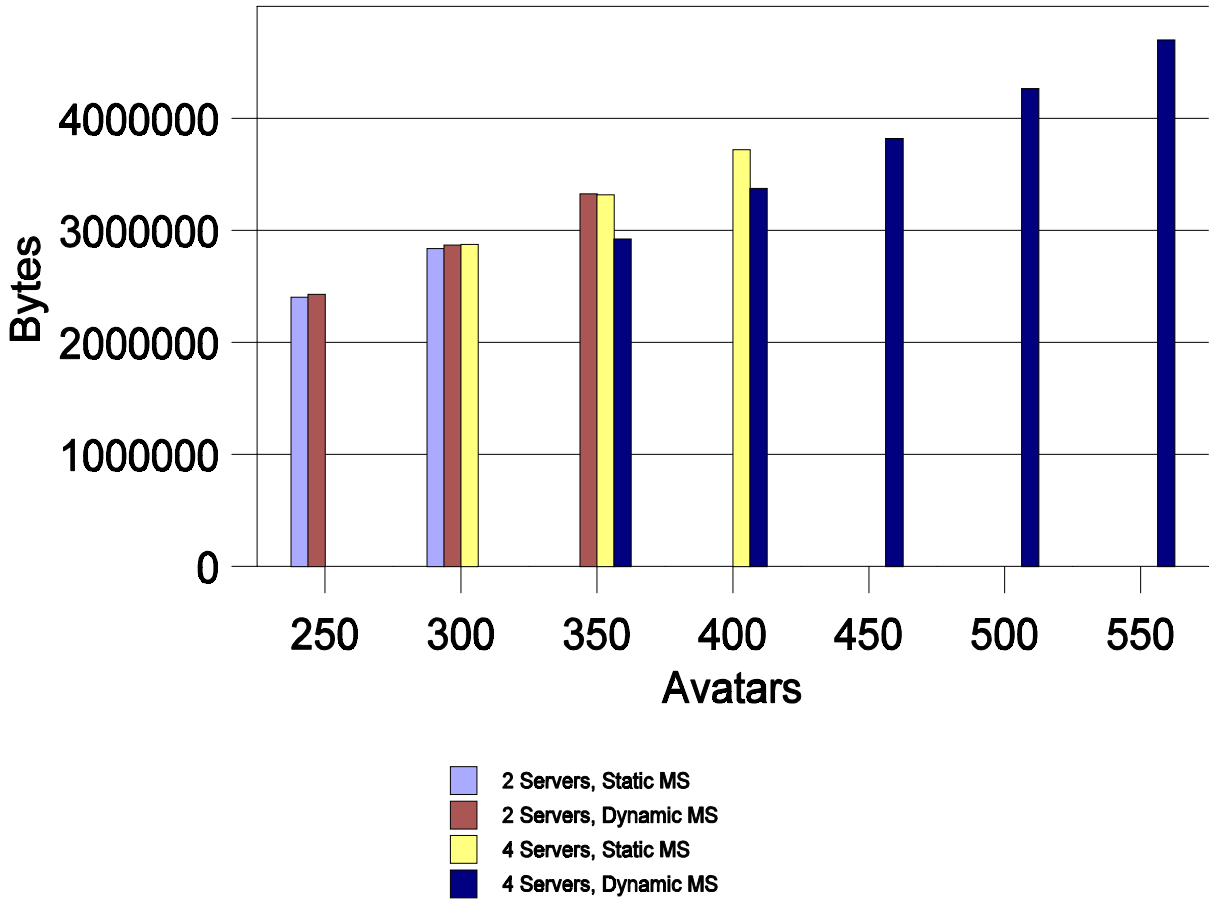


Figure 5.22: Bytes Transferred for Random Movement, Uneven Distribution, 2 and 4 Servers

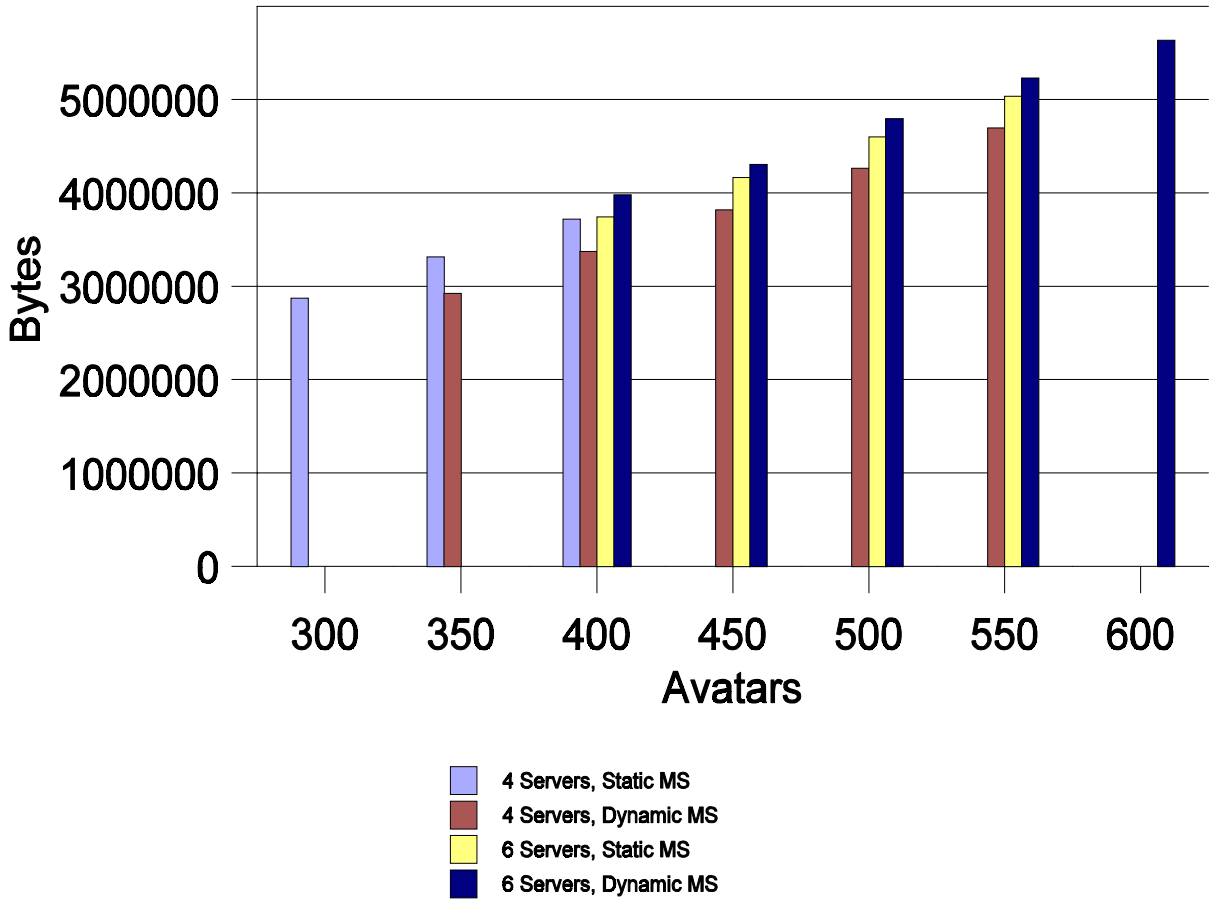


Figure 5.23: Bytes Transferred for Random Movement, Uneven Distribution, 4 and 6 Servers

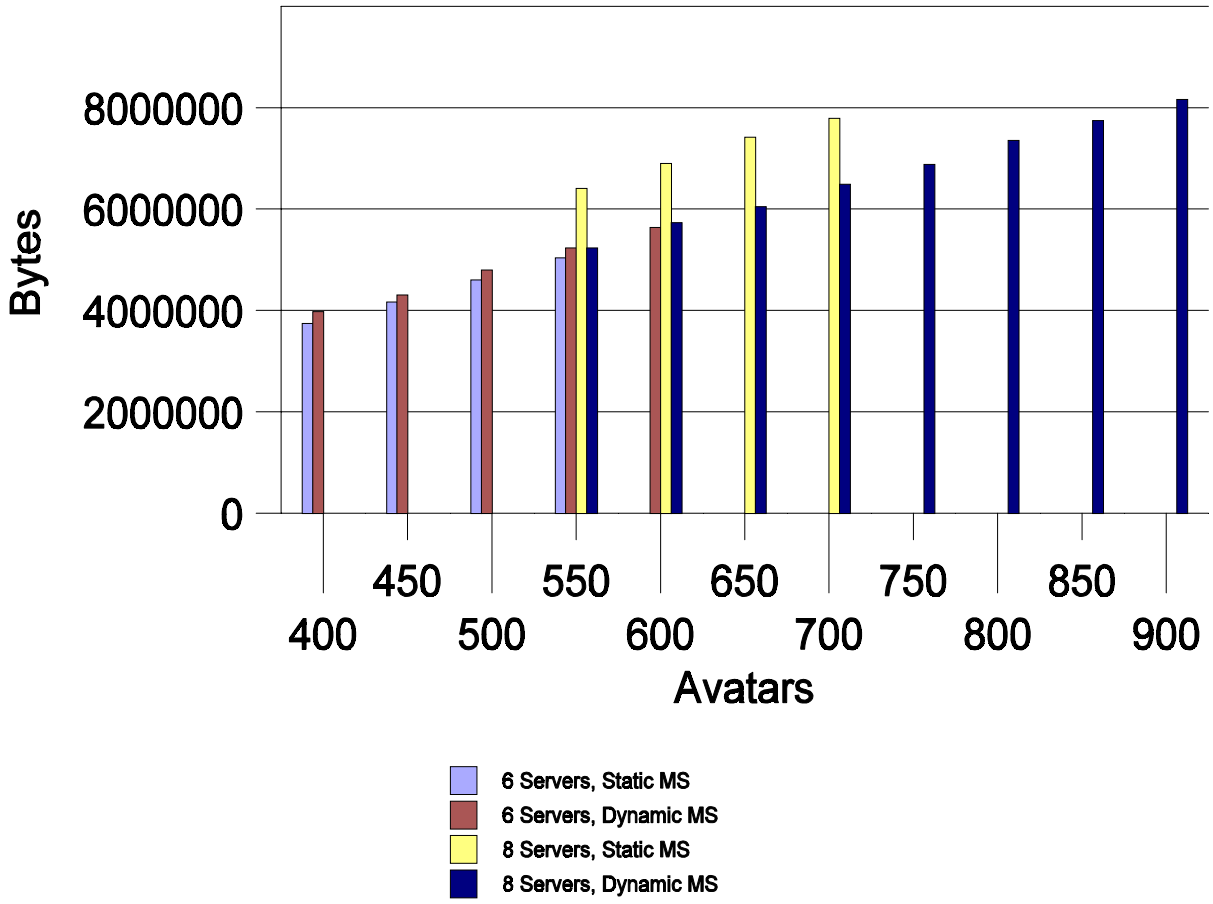


Figure 5.24: Bytes Transferred for Random Movement, Uneven Distribution, 6 and 8 Servers

Figure 5.21 shows the number of bytes transferred for one and two servers. More bytes are transferred when two servers are used. More bytes are transferred when the Dynamic MS scheme is used. Using two servers with the Dynamic MS scheme is required to support 350 avatars. Figure 5.22 shows the number of bytes transferred for two and four servers. About the same number of bytes are transferred for two servers with the same number of avatars. More bytes are transferred when the Static MS scheme is used, for four servers and the same number of avatars. Using four servers with the Dynamic MS scheme is required to support 550 avatars. Figure 5.23 shows the number of bytes transferred for four and six servers. More bytes are transferred when six servers are used for the same scheme and number of avatars. For two servers more bytes are transferred for the Static MS scheme for the same number of avatars. For four servers more bytes are transferred for the Dynamic MS scheme for the same number of avatars. Using six servers with the Dynamic MS scheme is required to support 600 avatars. Figure 5.24 shows the number of bytes transferred for six and eight servers. More bytes are transferred when eight servers are used for the same scheme and number of avatars. For six servers more bytes are transferred for the Dynamic MS scheme for the same number of avatars. For eight servers more bytes are transferred for the Static MS scheme for the same number of avatars. Using eight servers with the Dynamic MS scheme is required to support 900 avatars. In general, more bytes are transferred across the network when the Dynamic MS technique is used because of the overhead of the load balancing algorithm.

5.6 Summary

The Static Av scheme supports less avatars on two servers than one server supports, for both the nonrandom even and nonrandom uneven distributions. This is because whenever an avatar performs

an action, the server that is not hosting it most be informed. One server supports less avatars than either the Static MS or Dynamic MS schemes, but it requires the least amount of data to be sent across the network, when it can support all the avatars. As expected, more servers can support more avatars than one server, but there is more network traffic, since the servers have to communicate with each other. For the even distribution, the Static MS scheme produces better response times and requires less data to be sent across the network than the Dynamic MS scheme. The lower response times are because the avatars are evenly distributed across the map, and hence statically assigning equal sized portions of the map to the servers gives the best load distribution. The lower network traffic than the Dynamic MS scheme is because the Dynamic MS scheme has the overhead of the load balancing algorithm. For the uneven distribution, the Dynamic MS scheme gives better response times, but requires more data, relative to the Static MS scheme, to be sent across the network. The lower response times occur because the load is periodically rebalanced by the load balancing algorithm. As before, the Dynamic MS scheme requires more data to be sent across the network due to the overhead of the load balancing algorithm. For an uneven distribution, the Dynamic MS scheme always supported the most avatars for a given number of servers. This is because the load is kept balanced by the load balancing algorithm.

Chapter 6

Conclusions and Future Directions

6.1 Conclusions

A variety of schemes can be used to support a massively multiplayer online role playing game (MMORPG). Three schemes were examined: static assignment of avatars to servers, static assignment of map segments to servers, and dynamic assignment of map segments to servers. In the case of dynamic assignment of map segments to servers a new load balancing algorithm (LBA) was presented. Four different scenarios were covered - an even distribution of avatars with nonrandom movements, an even distribution with random movements, an uneven distribution with nonrandom movements, and an uneven distribution with random movements of avatars. The conclusions based on the analysis of the data collected from the simulations are given below.

Never use static assignment of avatars to servers, since it supports less avatars on two servers than are supported by one server. If possible, only one server should be used, since it requires the least amount of data transfer, and has the least hardware cost.

In the case of an even distribution of avatars, static assignment of map segments to servers should be used, with the least number of servers that can handle the expected number of avatars. This gives a better average response time than dynamic assignment of avatars to servers and results in less data being transferred.

In the case of an uneven distribution, dynamic assignment of map segments to servers should be used, with the least number of servers that can handle the expected number of avatars, since this gives the lowest average response time, and more avatars can be handled than in the case of static assignment of map segments to servers for the same number of servers. There will be more data transferred over the network than for the case of static assignment of map segments to servers due to the overhead of the LBA.

6.1.1 Contribution of the Thesis

In this thesis three different multi-server approaches were compared and contrasted. Moreover, a new multi-server approach was presented in this thesis. It involves the use of a load balancing algorithm. In order to compare and contrast the three multi-server approaches a novel test bed was created that can be readily applied in other similar experiments. Results were found for various distributions of avatars and partitioning strategies.

6.2 Applications of This Work

This work is applicable to any application that requires a virtual world with a large number of avatars in it. Entertainment products that are currently in development could benefit. A simulation of emergency response to a natural disaster could be simulated. This would allow for much more diverse conditions than could be simulated in a live exercise. A virtual conference could be held. Conference participants would not have to travel to the conference. A virtual shopping mall could be created. A person could go to a shopping mall without leaving home. A historical simulation could be created. A user's avatar would take on the role of person living in a particular time and place in the past. For example, a Roman citizen in ancient Rome. It is anticipated that other applications will be found.

6.3 Future Directions

There are several areas of possible future research:

- A model based approach could be taken. The model would have to take into account the operating system and networking hardware and software in addition to the actual control of the avatars. It is anticipated that in the most general case the problem is NP-complete.

- Different actions of an avatar could be allowed to have different processing requirements (i.e. not all actions would be equivalent). One of the tasks in this case would be to determine what the variation in processing requirements are for different actions by an avatar.

- The simulator could simulate different network delays for clients. One of the tasks in this case would be to determine what the variation in network delay is for clients.

- More variety of distributions of avatars could be examined. Due to limited time only two distributions were examined in this thesis. If many more distributions were examined then trends relating distributions may be found.

- A study of actual MMORPGs to attempt to determine what distributions of avatars are common, could be done. Since no study has been done to determine what distributions occur in MMORPGs this thesis simply examined two extremes. It is unknown if either of these extremes is representative of what happens in an actual MMORPG.

- The effect of various map segment sizes could be examined. Larger map segments allow the LBA to run faster, but the load will generally be less evenly distributed across the servers. Smaller map segments will result in a longer running LBA, but generally more even distribution of load across the servers. The longer the running time of the LBA the more out of date the resulting new distribution of MSes is, since it is based on old data. By varying the sizes of the map segments the above effects could be studied.

- The LBA could be modified to deal with heterogeneous servers. The LBA in its current form assumes that all servers are homogeneous. By extending the LBA to support heterogeneous

servers the LBA becomes more flexible and useful under more circumstances.

- A different networking architecture could be used, such as that found in a cluster computer. The network became the bottle neck for the even distribution. Using a different network architecture may have eliminated this bottle neck. As well, response times will potentially be lowered.

- The effect of using shared memory multiprocessing could be examined. This research assumed the use of message passing due to the use of a network of computers. If a shared memory multiprocessor is used than message passing may not be the best way for processes to communicate.

- The impact of small changes to the LBA could be examined. The LBA sometimes has to break ties. The impact of arbitrarily breaking all ties, or using some criteria other than what is currently used could be examined. Other minor modifications are possible.

- An entirely different LBA, still using map segments, could be proposed. Once the decision to use map segments is made, it is possible to come up with many different ways to do dynamic load balancing. Most of these will be ineffective, but one or more may actually be more effective than the LBA used in this thesis.

Bibliography

[1] Asheron's Call (http://www.asheronscall.com). Note: Asheron's Call is an example of a massively multiplayer online roleplaying game.

[2] M. A. Bassiouni, M. H. Chiu, M. Loper, M. Garnsey, J. Williams, "Performance and Reliability Analysis of Relevance Filtering for Scalable Distributed Interactive Simulation", ACM Transactions on Modeling and Computer Simulation, Vol. 7, No. 3, July 1997, pp. 293 - 331.

[3] P. Drewes, A. Gonzalez, "Automatic Performance Monitoring and Evaluation", Proceedings of the Fifth Annual Conference on AI, Simulation and Planning in High Autonomy Systems: Distributed Interactive Simulation Environments, Gainesville, Florida, December 7 - 9, 1994, pp. 274 - 280.

[4] EverQuest (http://www.everquest.com). Note: EverQuest is an example of a massively multiplayer online roleplaying game.

[5] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, P. Wong, “Theory and Practice in Parallel Job Scheduling”, Proceedings of the IPPS ‘97 Workshop: Job Scheduling Strategies for Parallel Processing, Geneva, Switzerland, April 1997, pp. 1 - 34.

[6] D. Fullford, “Distributed Interactive Simulation: It’s Past, Present and Future”, Proceeding of the 1996 Winter Simulation Conference, pp. 179 - 185.

[7] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, "PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing", MIT Press, 1994, ISBN 0-262-57108-0.

[8] E. Haddad, “Load Distribution Optimization in Heterogeneous Multiple Processor Systems”, Proceedings of the Workshop on Heterogeneous Processing, April 13, 1993, pp. 42 - 47.

[9] K. Hafner, M. Lyon, “Where Wizards Stay Up Late: The Origins of the Internet”, Simon & Shuster, 1996.

[10] L. A. Hembree Jr., R. Siquig, “Distributed Interactive Simulation Atmosphere Subgroup

Rationale for Synthetic Atmosphere and Near-Space Environments", Proceedings of the 1994 Summer Computer Simulation conference, pp. 789 - 794.

[11] Hero's Journey (http://www.herosjourney.net). Note: Hero's Journey is an example of a massively multiplayer online roleplaying game that is currently under development.

[12] IEEE Standard for Information Technology - Protocols for Distributed Interactive Simulation Application IEEE Std. 1278-1993.

[13] E. Keck, "Cost Effective Testing Using DIS", Proceedings of the 1996 Summer Computer Simulation Conference, Portland, Oregon, July 21 - 25, 1996, pp. 360 - 365.

[14] The Laws of Online World Design (http://www.legendmud.org/raph/gaming/laws.html). Note: These "laws" were compiled by Raph Koster, who was lead designer of Ultima Online from its initial conception until two years into its public release.

[15] T. G. Lewis, H. El-Rewini, "Introduction to Parallel Computing", Prentice Hall, 1992.

[16] K. C. Lin, D. E. Schab, "The Performance Assessment of the Dead Reckoning Algorithm in DIS", Proceedings of the 1994 Summer Computer Simulation Conference, pp. 848 - 853.

[17] Meridian 59 (http://www.meridian59.com). Note: Meridian 59 is an example of a massively

multiplayer online roleplaying game.

[18] Middle Earth (http://www.middle-earth.com). Note: Middle Earth is an example of a massively multiplayer online roleplaying game that is currently under development.

[19] M. Morrision, “The Magic of Interactive Entertainment”, Sams Publishing, 1994.

[20] The Museum of Classic Games and Classic Gaming (http://www.classicgaming.com/museum). Note: This site contains historical information on computer gaming in general.

[21] E. H. Page, R. Smith, “Introduction to Military Training Simulation: A Guide for Discrete Event Simulationists”, Proceedings of the 1998 Winter Simulation Conference, pp. 53 - 60.

[22] Y. E. Papelis, “Terrain Modeling on High-Fidelity Ground Vehicle Simulators”, Proceedings of the Fifth Annual Conference on AI, Simulation and Planning in High Autonomy Systems: Distributed Interactive Simulation Environments, Gainesville, Florida, December 7-9, 1994, pp. 48 - 54.

[23] E. W. Parsons, K. C. Sevcik, “Multiprocessor Scheduling for High Variability Service Time Distributions”, Proceedings of the IPPS ‘95 Workshop: Job Scheduling Strategies for Parallel Processing, Santa Barbara, CA, USA, April 1995, pp. 125 - 145.

[24] M. J. Quinn, “Parallel Computing: Theory and Practice”, McGraw-Hill, 1994, ISBN 0-07-051294-9.

[25] G. A. Schiavone, S. Sureshchandran, K. C. Hardis, “Terrain Database Interoperability Issues in Training with Distributed Interactive Simulation”, ACM Transactions on Modeling and Computer Simulation, Vol. 7, No. 3, July 1997, pp. 332 - 367.

[26] S. M Shatz, J. P. Wang, “Introduction to Distributed-Software Engineering”, IEEE Computer, October 1987, pp. 23 - 31.

[27] R. A. Siquig, L. A. Hembree Jr., “Environmental Effects for Distributed Interactive Simulation: Standards and Environmental Representations Tasks”, Proceedings of the 1994 Summer Computer Simulation Conference, pp. 795 - 800.

[28] B. Stroustrup, "The C++ Programming Language: Second Edition", Addison-Wesley, 1991, ISBN 0-201-53992-6.

[29] R. W. Tarr and J. W. Jacobs, "Distributed Interactive Simulation”, 1994 Summer Computer Simulation Conference, Orlando FL, July 18 - 20, 1994.

[30] Ultima Online (http://www.uo.com). Note: Ultima Online is an example of a massively multiplayer online roleplaying game.

[31] Ultima Online 2 (http://www.uo2.com). Note: Ultima Online 2 is an example of a massively multiplayer online roleplaying game that is currently under development.