# Load Balancing for Massively Multiplayer Online Games

Fengyun Lu
Newcastle University
School of Computing Science
UK
+44 191 2227105
Fengyun.Lu@ncl.ac.uk

Simon Parkin
Newcastle University
School of Computing Science
UK
+44 191 2226053
S.E.Parkin@ncl.ac.uk

Graham Morgan
Newcastle University
School of Computing Science
UK
+44 191 2227983
Graham.Morgan@ncl.ac.uk

## ABSTRACT

Supporting thousands, possibly hundreds of thousands, of players is a requirement that must be satisfied when delivering server based online gaming as a commercial concern. Such a requirement may be satisfied by utilising the cumulative processing resources afforded by a cluster of servers. Clustering of servers allow great flexibility, as the game provider may add servers to satisfy an increase in processing demands, more players, or remove servers for routine maintenance or upgrading. If care is not taken, the way processing demands are distributed across a cluster of servers may hinder such flexibility and also hinder player interaction within a game. In this paper we present an approach to load balancing that is simple and effective, yet maintains the flexibility of a cluster while promoting player interaction.

## Categories and Subject Descriptors

H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems - Artificial, Augmented, and Virtual Realities C.2.4 [Distributed Systems]: Distributed Applications

## General Terms

Measurement, Performance, Design, Experimentation.

## Keywords

Keywords are your own designated keywords.

## 1. INTRODUCTION

Massively Multiplayer Online Games (MMOGs) provide gaming arenas within which hundreds of thousands of players participate. There are a number of MMOGs that have gained commercial success based on the premise of charging players to participate in large scale persistent virtual worlds. In such virtual worlds players may assume alternate identities and "live out" scenarios of their own choosing while participating in game play regulated by MMOG vendors. These types of MMOGs are commonly termed *massively multiplayer online role-playing games* (MMORPGs).

MMORPG implementations are server based, allowing vendors to regulate the provision of ever evolving alternate realities to maintain player interest and, most importantly, restrict participation to subscribed players. Player consoles connect to a server which provides players access to a virtual world. As revenue is generated on a per-player basis, the more players that can be supported by a MMORPG the more revenue may be generated. Therefore, scalability of a server, in terms of number of players supported, is of great importance to ensure commercial success.

To satisfy the demand for processing resources to provide scalable MMORPGs, clusters of servers are employed to cumulatively maintain game play by managing player interactions. The additional processing resources required to support an increase in player numbers is satisfied via the addition of servers to a cluster.

A major challenge in constructing scalable server side solutions for MMORPGs is the need to provide players with mutually consistent views of the gaming arena in a timely manner to allow fair game play. However, when a virtual world contains hundreds of thousands of players the required consistency cannot be achieved in a timely manner without localised game play. By identifying localised instances of game play the consistency of the gaming arena becomes a more manageable problem of ensuring consistency between subsets of interacting players.

The problem of satisfying the processing requirements of localised game play over a number of servers in a cluster needs to be tackled efficiently: load balancing techniques are required to ensure processing resources are allocated within a cluster to make best use of available servers. An ideal solution would be to ensure such load balancing techniques allow: (i) *equal distribution of resources* - prevent exhausting available processing resources on one server while there is spare capacity on other servers; (ii) *flexible configuration* - may afford the addition of servers during runtime to accommodate additional players with minimum disruption to game play; (iii) *promotion of game play* - does not hinder game play by overly restricting player interaction within a virtual world.

Our earlier work [1] demonstrated an approach to modelling localised game play within a virtual world that does not hinder the interaction requirements of players. This work was subsequently implemented using a network of servers and was demonstrated to be scalable [2]. In this paper we tailor our system for deployment over a cluster of servers and present a series of experimental results. We demonstrate that load may be efficiently balanced over a server cluster. In addition, our approach uses standard load balancing mechanisms common in many Internet based applications, allowing improved consistency via the addition of servers to handle increasing numbers of players with minimum disruption to the gaming experience of players.

The paper continues with a description of server side solutions to load balancing techniques that may be deployed in MMORPGs that use clustering of servers to gain scalability. Section 3 provides an abridged description of our approach to regionalisation, its implementation using server clustering and how we economically make use of existing load balancing techniques. A series of experiments and associated results demonstrating the usefulness of our approach is presented in section 4. Section 5 draws conclusions from our work and indicates future directions we expect to take in this line of research.

## 2. BACKGROUND

The technologies that combine to provide scalable online games supported by server clustering are determined by design choices made in the areas of virtual world regionalisation (with respect to identifying instances of localised game play), server clustering, and load balancing. Design choices made in each of these areas cannot be considered in isolation. For example, the choice of how to regionalise a virtual world will influence how server clustering and load balancing is achieved. Alternatively, the design of a server cluster will feedback into the manner with which regionalisation of a virtual world may be achieved. In existing literature one or more of these design choices are assumed, resulting in a narrowing of the available solutions. Therefore, in this section we afford a degree of detail we believe is a necessity for gaining a clear understanding of the possible solutions available to developers.

### 2.1 Regionalisation

There are two extremes when determining how to sub-divide a virtual world for the purposes of modelling player interaction (localised game play) and providing manageable consistency:

- **Geographic** – world divided into regions at initialisation time to reflect the structure of a virtual world.
- **Behavioural** – virtual world sub-divided to reflect the interaction patterns of players.

Geographic approaches are suited to virtual worlds that contain barriers to interaction that do not look out of place. For example, rooms in a building may be regions and only players that share a room may influence each other. Behavioural approaches are determined not by static virtual world constraints such as walls and ceilings, but by the ability of a player to express influence and other players to express interest. For example, a fighter aircraft may exert a greater degree (area) of influence than a foot soldier. When it is not convenient to use virtual world structures to define regions of a virtual world for use in a geographic approach, a behavioural approach is more appropriate.

Work on the regionalisation of a virtual world for attaining scalability and manageable consistency finds its origins in academic research commonly termed *interest management*. Regionalisation of the virtual world for interest management was first demonstrated in NPSNET, original version presented at SIGGRAPH 1991 [3] with regionalisation added in 1993/4 [4]. NPSNET divided the virtual world into static geographic regions of regular sizes (not necessarily reflecting structures in a virtual world), restricting interaction between players that exist within the same or neighbouring regions.

The aura/nimbus approach, used by MASSIVE in the mid 90s [5], modelled influence on a per player basis [6]. An aura describes the area of a virtual world a player may exert influence with a nimbus identifying an area of the virtual world a player may express interest. Although this approach is still reliant on the notion that players interact if they are geographically close to each other in a virtual world, more accurate modelling of interaction between players is possible compared to the NPSNET approach. However, the additional processing resources required to determine each player influence individually made this approach not as scalable as the region based approach [7] [8]. Attempts have been made to reconcile the scalability of regions with the accuracy of auras with some success [9, 10]. However, the scalability required for commercial MMORPGs is not achieved by such systems.

### 2.2 Server Clustering

Popular games in the MMORPG genre (e.g., EverQuest, Asher's Call, Ultima Online, City of Heroes, and Star Wars Galaxies) all employ clustered server solutions to achieve scalability while managing consistency. The techniques used to implement their interest management solutions in a server cluster is not described in detail in a published article for general viewing (which is to be expected for a commercial enterprise in a competitive market). However, there is an article describing EverQuest's approach in general terms: a mixture of regions and "duplicate worlds" with each duplicate world supporting approximately two to three thousand players with each world divided into regions based on the geography of the virtual world [11]. As regionalisation is associated to virtual world geography, this approach is closely related to geographic virtual world sub-division schemes.

In EverQuest a duplicate world is itself supported by a cluster of servers, with regions used to aid in allocating the processing requests originated from player actions amongst such servers as and when required. Due to the similarities in game play and the existence of duplicate worlds; one may assume that all other commercial MMORPGs approaches to implementation of interest management are similar, conceptually, to that of EverQuest.

Duplicate worlds and geographic influenced regionalisation present a three step approach to reducing the consistency problem to a manageable size: (i) players do not interact across different duplicate worlds; (ii) players do not interact across different regions; (iii) Players interact intricately with other players they specifically target (e.g., click on with mouse). This approach provides two distinct forms of interaction: (i) a general, viewing type style, where players can see the actions of others in their region (assuming appropriate line of sight); (ii) an intricate manner where players directly interact with each other in a user directed way. The latter form of interaction requires consistency to be greater as ordering of events are usually crucial in intricate game play (the server must resolve player interaction). The consistency can be weaker in the general style of interaction as summary information could be propagated between players. For example, in a fight between two players in a virtual world attacks must be regulated (e.g., ordered, not lost in transit) between engaged players (e.g., spells, hitting, shooting) to provide an outcome (e.g., decreased health, loss of inventory). However, for players watching a fight between other players there is only a need to view a series of fighting moves and the end result (that may or may not reflect the actual fight moves as enacted between the fight participants).

Commercial MMORPGs aside, there are a number of other works in the area of scalable server side solutions that may be appropriate for MMORPGs. A notable contribution is work carried out by IBM. IBM has produced region based services that are capable of supporting MMORPGs [12] that attempt to make use of standards such as Web/Grid services. Regions are again used in this work, providing a platform that would allow a similar approach to implementation that would be expected in the commercial MMORPGs already discussed. Other works (e.g., RING [16]) do employ multiple servers, allocating regions of virtual worlds to different servers, providing a similar approach to scalability (regions to servers) as advocated in commercial MMORPGs.

There are a large number of academic works that have advocated the client/server approach to virtual world implementation that, with tailoring, may be suited to MMORPGs. BrickNet [15] is an example of academic work that employs a server side solution. However, in such works scalability is limited without the ability to support server clustering.

## 2.3 Load Balancing

Load balancing is a term used to describe an attempt to efficiently distribute an application's processing requirements across a number of servers. Considering server clustering for MMORPGs, there are two ways of achieving load balancing:

- **Player** – Players are allocated to different servers (or mini-clusters of servers) as and when they join a game.
- **Interaction** – Servers manage allocation of processing recourses based on the interaction patterns of players.

The player oriented approach to load balancing is similar to standard load balancing techniques in many server based applications found on the Internet (e.g., search engines, shopping carts, and auctions). These approaches rely on a *network address translator* (NAT), or software equivalent, to allocate clients to servers efficiently using a number of load balancing techniques (e.g., round robin). The NAT "remembers" which server a particular client is attached to and directs all requests from a client to the same server during the lifetime of a *session*. A session is simply an application dependent classification of related client requests. The term sticky session is used to describe how a session should "stick" to the same server throughout its duration. In MMORPG a session may be identified as a prolonged period of unbroken game play of a player.

Using a NAT alone for load balancing is most viable given the ability of a single server to satisfy all a client's requests (*homogenous approach* to server clustering). Using this approach to load balancing allows servers in the cluster to be removed for maintenance or added as and when required without hindering players on other servers. In MMORPGs, allocation of players to duplicate worlds (and associated mini-clusters) is a close relation to this form of load balancing, apart from the fact that the players themselves, not a NAT, chooses which duplicate world they will visit.

Once players are allocated to a duplicate world, there is still a need to balance load across the server cluster supporting such a world. If players are allocated to servers, as in the player centric approach to load balancing, there would be a need for servers to inter-communicate as players hosted on different servers interact with each other. This increase in server side message exchange may exhaust available bandwidth and processing resources if an attempt is not made to limit such message exchange. This is where the use of interest management becomes pivotal in the role of load balancing for MMORPGs: interest management may identify interacting players and be used to limit inter-server communications while still allowing player interaction to occur.

The geographic approach of virtual world duplication and regionalisation found in MMORPGs lends itself to load balancing as design time decisions can be made as to which servers may satisfy the processing requirements of different regions of a virtual world. In this approach there is no requirement for inter-server communications to model player interactions as all players will be located in the same region, and therefore, be on the same physical server. In addition, convenient breaks in game play (e.g., set piece animation of travelling through a tunnel) can be introduced to hide the delay encountered when a player crosses geographic boundaries and associated processing resources are handed over to different servers.

Due to the ease with which the geographic approach to interest management may be mapped to processing resources there has been little interest in mapping the behavioural approach to servers.

## 2.4 Crowding

Allocating processing resources to different geographic regions of a virtual world can result in *crowding*. Crowding is a phenomenon that occurs in online gaming when the number of players that congregate in the same area of a virtual world inhibits the successful execution of interest management in a timely manner. The effects of crowding may be a slowdown in game play or, in worst case scenarios, a complete inability to enact player interaction. This may be considered the same problem of consistency management that regionalisation is attempting to alleviate: without regionalisation the virtual world itself (single region) may become populated by a sufficiently large number of players as to make the consistency problem unmanageable.

In the presence of server clustering, there is an opportunity to alleviate the crowding problem by dynamically associating processing requirements generated by player actions during runtime. This takes the form of load balancing player activities across servers with respect to regions. The literature provides a number of solutions to load balancing across server clusters suitable for MMORPGs. Regions may be reduced in size by sub-dividing them further (allocating servers to these additional sub-divisions) [17]. Other methods distribute responsibility for region execution to a particular server at runtime based on the volume of players in a region [18], while other methods dynamically resize regions during runtime [19]. Such approaches may be fine tuned further to ensure that the cost of moving responsibility for execution to another server is minimised [20].

EverQuest also describes runtime allocation of resources from within small clusters of servers responsible for a duplicate virtual world. Although no great technical detail is provided on how this is achieved [11], the premise of this approach appears to be player driven: when player enacts a particular action (e.g., opening a door, entering into battle) processing resources are allocated to satisfy the increased processing requirements.

## 2.5 Discussion

We find a contradiction in the direction of research concerned with the approach to server side load balancing in MMORPGs: (i) is based on geographic regionalisation to minimise server side inter-communications to promote scalability; (ii) requires inter-server communications to alleviate process exhaustion due to crowding.

The behavioural approach to interest management has been overlooked as it did not lend itself to load balancing in the same, obvious manner, as geographic approaches to interest management. However, with the problem of crowding we encounter the same need for inter-server communication, yet without the intricate game play afforded by behavioural approaches to interest management. In addition, the allocation of server resources dependent on interactions in a virtual world requires quite elaborate techniques compared to the traditional NAT load balancing approaches that are commonplace, increasing processing resources required for the load balancing mechanism.

We compare the geographic approach to load balancing using the three points relating to an ideal solution for load balancing described in the introduction of this paper:

(i) *equal distribution of resources* – crowding can exhaust server resources on one server while other servers are lightly loaded;

(ii) *flexible configuration* – as virtual world geography is linked to server configuration, removing or adding servers is not straightforward (even if no players exist, re-allocation of server responsibilities regarding virtual world geography is required within a cluster when servers are added or removed).

(iii) *promote game play* - geographic regionalisation is not as appropriate as behavioural approaches when modelling player interaction. Requiring additional “highlight by click” intervention of a player.

## 2.6 Contribution of Paper

We wish to clarify and simplify an approach to load balancing for MMORPGs and other, similar, games that depend on clustered server solutions for scalability. We believe that online games that gain scalability from server clustering will inevitably require communications between servers, irrelevant of what techniques are used for load balancing. Therefore, deriving ever more elaborate techniques for mapping geographic regionalisation to server allocation in a bid to prevent inter-server communications, we believe, is not the appropriate avenue to take. We make this statement for two reasons: (i) geographic regionalisation does not afford the greatest potential for game play (player interaction); (ii) the eventual cost, in terms of processing overhead, of elaborate techniques of allocating processing resources in this manner comes at a high price (process intensive).

In previous work we developed behavioural type approaches to interest management that can scale [1] and be implemented in a distributed server model (where servers are geographically separated) [2]. We now advance this work into the area of clustered server solutions.

We disregard all load balancing techniques based on mapping geographic regions to servers. Instead, we restrict ourselves to only using standard, “off the shelf”, sticky session type load balancing common with a NAT based infrastructure. This allows our technique to be economically employed with existing load balancing technologies. Furthermore, as our technique is behaviourally based, it affords more opportunity for introducing rich interaction into game play than a geographic approach to interest management. This added bonus may improve game play substantially as players can more naturally interact with each other without having to point and click at other player/artefacts to invoke intricate game play. We demonstrate that our approach is scalable via a series of experiments.

An additional contribution this paper makes to the community is to provide a comprehensive overview of the state of the art in scalable load balancing techniques for MMORPGs (described in this section). We do this by clarifying, via categorisations, the topics of interest management and load balancing in MMORPGs. These two topics are intricately linked, and a clear understanding of both is a necessity for any researcher in this area.

## 3. IMPLEMENTATION

We now describe our approach to clustered server deployment of our system. We start by describing our approach to load balancing and then continue with descriptions of our interest management and server clustering implementation. We provide descriptions in this paper only in sufficient detail to understand how our approach is deployed over a cluster of servers. Extended descriptions of our interest management scheme and its implementation may be found in [1] [2], only the changes that have been made to accommodate server clustering are highlighted here.

## 3.1 Load Balancing

Our approach to load balancing is typical in the area of clustered server solutions and relies on the allocation of client machines (player consoles) to servers. We allow servers to communicate player actions to each other as and when required but do not move responsibility for processing player actions from the server they are initially allocated. The diagram in figure 1 describes our server cluster implementation.

In figure 1 a player’s console ($C_1$) connects to the server cluster via a load balancer (NAT), and is then associated to a particular server in the application tier (e.g., $S_1$) for the duration of this session of interaction (sticky session). The application tier satisfies the runtime requirements of game play. Via the database tier, an application server may gain access to persistent artefacts that constitute a gaming arena (e.g., virtual world constructs, players’ statistics). A load balancer may exist between the application tier and the database tier, presenting a single “image” of a database to the application tier, simplifying the implementation of the application tier (no need for application tier to be concerned with database load balancing).

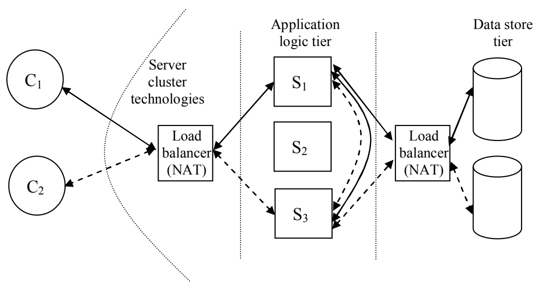

**Figure 1 – Server clustering in n-tier systems**

We assume that the load balancers that are present (client-to-application and application-to-data) are standard "off the shelf" NAT type load balancers. Any load balancing scheme may be enacted, however, we assume a simple round robbin approach that attempts to equally distributed players to servers.

## 3.2 Interest Management

Our interest management scheme, *predictive interest management*, may be considered behavioural in its approach, as player interactions are associated to player expressiveness as apposed to static geographic regionalisation of the virtual world. We use auras (as described in [5]) for determining when players should exchange messages. For clarity, we describe predictive interest management by describing inter-player interaction only. For a more detailed description of predictive interest management the reader is directed to [1] [21]. Our scheme does not rely on the presence of a server (acting as an oracle) and is suitable for peer-to-peer deployment. We use the term avatar to denote a player's representation in a virtual world.

The aura of an avatar describes an area of the virtual world enclosed by a sphere (Figure 2). The radius of an aura is specified on a per avatar basis and is fixed at avatar creation time. Avatars have the ability to influence each other when their auras collide via the exchange of messages.

A predicted area of influence (*PAI*) identifies the extent of an avatar's aura over a period of time given the distance an avatar may travel in a straight line in any direction (assuming an avatar's maximum speed).

Based on how PAIs and auras are overlapping in the virtual world we may regulate message exchange between avatars:

- **Aura overlap** – aura overlap indicates interacting avatars requiring high frequency *positional update messages* (PUMs) to be exchanged between them. PUMs carry positional information of the sending avatar, but may also carry other game dependent data.
- **PAI overlap** – if PAIs overlap but not auras then there is a possibility that such avatars may interact in the near future, requiring *admin PUMs* (APUMs) to be exchanged between them at a frequency that relates to the degree of PAI overlap witnessed.
- **No aura or PAI overlap** – avatars exchange APUMs at a low frequency, allowing for possible PAI/aura overlap in the future to be realised.

In summary, the more PAIs overlap (but not auras) the higher the frequency of message exchange. This provides a model where avatars increase their message exchange frequency gradually until auras overlap, when they continue by exchanging high frequency messages. Alternatively, avatars decrease their message exchange frequency gradually until they only exchange low frequency messages.

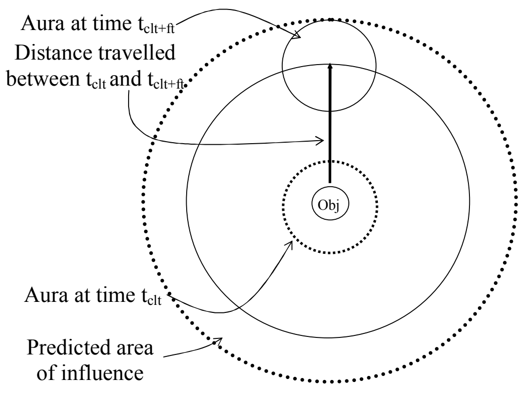
Aura at time $t_{clt+ft}$
Distance travelled between $t_{clt}$ and $t_{clt+ft}$
Obj
Aura at time $t_{clt}$
Predicted area of influence

**Figure 2 – Defining Predicted Area of Influence (PAI).**

Two avatars may come close to each other over time in a virtual world (resulting in increased APUM exchange), but never encounter aura overlap. This message exchange overhead is accepted by us as necessary to avoid missing when avatar auras are overlapping. In effect, we spread the processing requirements related to the detection of aura overlap over a longer period of time to avoid non-detection of aura overlap and promote a more realistic interaction.

## 3.3 Server Clustering Implementation

Our concern, for this paper, is on clustering technologies related to predictive interest management. Therefore, we perceive the data store as a commercial database (e.g., Oracle) that comes complete with its own load balancing technologies and concentrate our discussion on the application tier.

Player consoles (clients) periodically send *PUM*s to the load balancer. As a client may manage multiple avatars (we provide flexibility in our approach in that we do not limit a client to a single player representation in the virtual world), a single message may contain multiple *PUM*s. These messages are synchronous calls (implemented as RPC), with the return part of the message containing one or more *PUM*s relating to avatars that are hosted on other clients. A server may send *PUM*s to clients that have not sent *PUM*s for a substantial length of time (i.e., due to player inactivity – timeout determined by client). Our approach to client/server interaction eases client participation in a virtual world as clients only need send *PUM*s, not *APUM*s: the burden of interest management implementation is solely within the application server tier.

Between servers individual *APUM*s are periodically combined into single messages and distributed on a per-server basis.

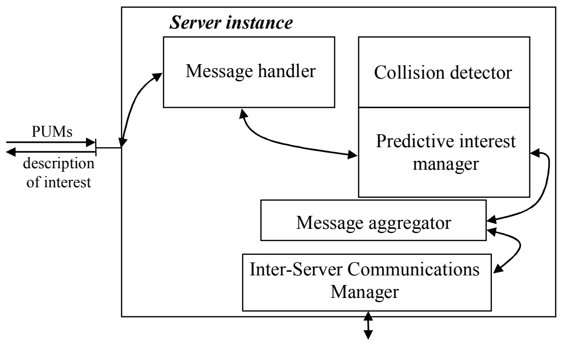

**Figure 3 – Components of a server instance.**

Figure 3 describes the main server components that contribute to satisfying the interest management requirements:

- **Message handler** - receives and returns messages to load balancer. If necessary, registering new player information using data store tier.
- **Predictive interest manager** - uses predictive interest management to construct appropriate *APUM* messages.
- **Collision detector** - identifies aura and *PAI* overlap to aid predictive interest manager in constructing appropriate *APUM* messages.
- **Message aggregator** - composes single messages from multiple *APUM* messages for distribution to other servers.
- **Inter-Server Communications Manager** - supports message exchange between servers.

The message handler receives *PUM*s from the load balancer and returns to the load balancer descriptions relating to player interests. The interest manager implements the predictive interest management scheme and calls on the collision detector to identify aura and *PAI* overlap. The collision detector implements a collision detection algorithm that we specifically designed for use with predictive interest management [22]. The interest manager constructs *APUM*s and passes them to the message aggregator, which in turn composes single messages from multiple *APUM*s on a per server basis and passes such messages to the appropriate servers via the inter-server communications manager (implemented at socket level).

*APUM*s are received at a server's message aggregator and are passed to the predictive interest manager to aid in determining the interest of avatars. Information relating to the interest of avatars is passed to the message handler by the predictive interest manager. The message handler then informs the load balancer of updated avatar interests.

Our peer-to-peer approach to interest management has been directly mapped to the application server tier in our clustering solution. Message aggregation is used to conserve bandwidth between servers, and so aid scalability.

## 4. Performance Analysis

In this section we present a series of experiments to determine the suitability of our approach to load balancing and interest management to satisfy the requirements of an MMORPG. The requirement we are specifically interested in is that of scalability: can our approach scale to a level similar to that found in commercial MMORPGs while satisfying timely and consistency requirements.

Typically, when a server nears exhaustion of its processing resources due to excessive client induced load a slowdown in server performance is witnessed. If client load is increased further server failure will follow. As we have strict timely requirements we wish to avoid such a slowdown in a server: it would be misleading to indicate that a server is supporting many thousands of players when such support is ineffective due to real-time requirements not been met. Therefore, as soon as a server cannot satisfy the real-time requirements of its clients a server fails. Failure of a server is apparent in the graphs when a line stops short of the maximum number of players supported (denoted by the x axis).

We measure the percentage of messages dropped by a server and place a finite size on a server's message queues. In this approach, a server may maintain real-time requirements at the expense of dropping messages. The percentage of messages dropped by such queues forms the basis of our measurements in the experiments presented in this paper.

### 4.1 Testing Environment

This testing is based on 20 useable machines on the same LAN segment. Each machine has a 2GHz Intel Xeon processor (equivalent of 2x2GHz Pentium 4 processors with Hyper Threading) with 1GB RAM running Red Hat Linux 7.2.

Servers are located on different machines on the same LAN segment. Client (simulated player) machines are located on different machines outside this server cluster (but connected via 100 Mb Ethernet to the LAN cluster). Using the client machines, synthetic networking traffic for representing players is created. Player numbers are increased in increments of 500 from 500 to 6000 (depending on experiment), with measurements taken at each increment.

Each experiment's duration was one hour to ensure the initialisation overhead does not skew the results (e.g., player registering and stream socket setup). Additionally, the machines used for this experiment are a shared resource. As such, the performance of the machines and the available network bandwidth can vary considerably depending on the number and nature of the processes running on each machine at the time each experiment.

### 4.2 Experiments

Four experiments have been conducted to test different aspects of the system:

1. **Single Server** - The maximum number of players which can be supported by one server;

2. **Player Interaction** - The upper bound of message frequency a player console can send PUMs to the cluster;
3. **Prediction Overhead** - The overhead of APUM in the predictive interest management scheme compared with a traditional aura-based interest management scheme;
4. **Scalability** - The scalability of the system in terms of the number of players that can be supported simultaneously.
5.

The first two experiments' results can be used to assist game developers to estimate appropriate system variables (PUM frequency, number of servers, maximum number of players supported) to provide acceptable performance. For example, given a threshold maximum drop rate and a PUM transmission frequency, the results of the first two experiments can be used to estimate the number of servers required to achieve acceptable performance for a given number of players.

As mentioned in section 3.2, predictive interest management is a peer-to-peer approach and so relies on message exchange to realise when aura overlap occurs. To ensure this is achieved in a timely manner additional messages are sent when auras near overlap, producing a message overhead beyond that of a simple aura based approach. Experiment 3 determines the cost of such an overhead. To encourage a like-for-like comparison we make use of the same message aggregation techniques used in predictive interest management for our standard aura approach (we simply identify an avatar's PAI to be the same size as an avatar's aura).

The fourth and final experiment is to determine the overall scalability of the system. Additional servers are added to determine if player numbers can be maintained. In the EverQuest article [11], individual clusters of servers may support 2500 – 3000 players. Therefore, we are seeking to surpass this figure. We admit to not providing the detailed game play as EverQuest (we are a proof of concept academic work), but we at least hope to demonstrate scalability in the same league as commercial games.

## 4.3 Virtual World Simulation

To avoid the need to manually manipulate each individual player avatar in a virtual world we simulate avatar movement. We attempt to re-create the phenomena of periodic crowding throughout an experiment to identify that our approach is suitable in such scenarios. Deriving a suitable simulation of avatars to exhibit the type of behaviour expected in a virtual world is not documented in the literature. Therefore, we afford a reasonable description of our technique to allow reproduction of our experiments by others.

A program, called *RandomWayPointWorld*, is used to simulate the movement of player's avatars. A number of static points in the virtual world are generated, *markers*, at virtual world creation time. Each player's avatar chooses a marker at random and moves towards the marker for a random amount of time, termed marker selection time (*MST*). During *MST*, the avatar's position is updated at the same frequency as the PUM messages sent to the cluster of servers. Once the *MST* has been exceeded, an avatar selects another marker at random, and continues the process. Each marker remains at a position for a random amount of time, called marker relocation time (*MRT*). Once *MRT* is exceeded a marker relocates to a new position in the world. In order to determine the *MST* and the *MRT*, four values are used to calculate the minimum and maximum range of *MST* and *MRT*. As the x, y and z dimensions are identical in a cubic world; the diagonal size of this world can be calculated as:

$$Size_{dia} = \sqrt{3W_{size}^2}$$

$MRT_{lower}$ is the lower bound of the *MRT* and it is defined as the time taken for an avatar travelling with its maximum speed to cover a distance equal to half the diagonal size of the world. $MRT_{upper}$ is the upper bound of the *MRT*. Compared with the $MRT_{lower}$, $MRT_{upper}$ is the time taken for an avatar travelling a distance, which is the same as the full diagonal size of the world, with its top speed. These two variables are represented as the formulas below:

$$MRT_{lower} = (\frac{1}{2} * Size_{dia}) \Big/ Speed(top)$$

$$MRT_{upper} = Size_{dia} / Speed(top)$$

*MRT* is a random time selected within the range [$MRT_{lower}$, $MRT_{upper}$] and can be decided based on the formula below:

$$MRT = CurrentTime() + (MRT_{lower} + Random() \\ * (MRT_{upper} - MRT_{lower}))$$

CurrentTime() is a function to get the current time of the system; Random() returns a decimal number uniformly distributed between 0 and 1. After the previous selected *MRT* has passed, the *MRT* is recalculated. The process will repeatedly occur during the lifetime of an avatar. This selection ensures that the time a marker remains in a given position is a sufficient time, with respect to the size of the world, to avoid markers repositioning too frequently. If markers reposition too frequently, the avatar's movement towards the markers exhibits strange behaviour: when the avatars are initialised, they are uniformly distributed within the virtual world but, as time passes, the majority of the avatars crowd together in the centre of the world. This is because, once an avatar reaches the centre of the world, the direction they travel changes sufficiently rapidly that it is unlikely they will be able to move to the extremities of the world before they change direction

*MST* is chosen within the range of [ $MST_{lower}$, $MST_{upper}$ ]. $MST_{lower}$ and $MST_{upper}$ should be less than $MRT_{lower}$ and $MRT_{upper}$ respectively. Therefore, an avatar can trace one marker and change to a different marker before the marker relocation happen. $MST_{lower}$ and $MST_{upper}$ can be defined as below:

$$MST_{lower} = (MRT_{lower} + MRT_{upper}) \big/ 4$$

$$MST_{upper} = (MRT_{lower} + MRT_{upper}) \big/ 2$$

Based on the calculated $MST_{lower}$ and $MST_{upper}$, *MST* can be determined:

$$MST = CurrentTime() + (MST_{lower} + Random() \\ * (MST_{upper} - MST_{lower}))$$

As the same as the *MRT*, *MST* will dynamically change during the lifetime of the avatar.

In order to simulate the movement of an avatar, the trajectory can be predicted based on a set of formulas, which are used to calculate the current position, velocity and acceleration of the avatar according to the previous relevant information. Given a fraction time (*dt*), if the distance (*dis_marker_obj*) between the avatar and the marker is less than the distance (*dis_travelled*) an avatar can travel based on the previous velocity, the position of the avatar is set as the marker position and the velocity is set as 0. The avatar will stay at the marker once its' position is set as the marker and it remains still until next marker is selected. This will give an avatar variable speed before its speed reaches its maximum speed. If *dis_marker_obj* is larger than *dis_travelled*, the avatar is still moving towards the selected marker, the new position and velocity of this avatar is required to be calculated. To simplify the calculation process, the acceleration is set as a fixed value, 10 meters per second in each dimension.

## 4.4 Single Server

This experiment is intended to determine the number of players (represented as avatars) a server can support simultaneously. Due to physical restrictions, such as CPU speed and the amount of free memory, the number of players a server can support simultaneously has a limit.

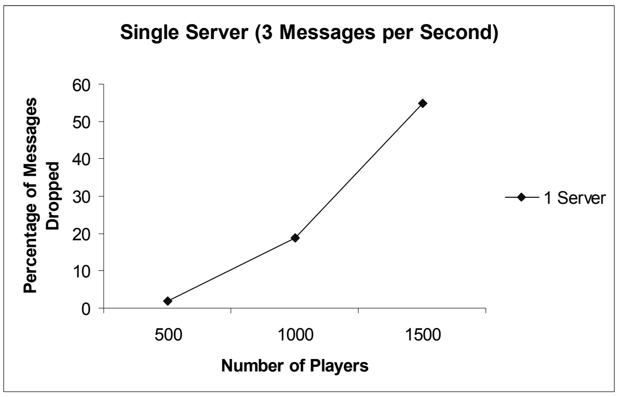

**Figure 4 – Single server**

According to Figure 4, the message drop rate increases steeply as the number of players increases. As can be seen from the graph, with 500 players, the drop rate was just 1.96%; with 1000 players, the drop rate increases to 16.9%; with 1500 players, the drop rate reaches 54.8%. The performance of the system sharply degrades. The reason for this is that the server received more messages than it can handle per second. Therefore, under the current test conditions, the maximum number of players a server can support is 1500 players. However, considering the percentage of dropped messages, player numbers of less than 1000 would be more appropriate.

## 4.5 Player Interaction

The frequency a client sends *PUM* messages to a server must be limited to some extent to avoid intolerable drop rates. Therefore, the purpose of this experiment is to determine the maximum acceptable frequency a node can send *PUM*s to a server cluster that is of a fixed size. We compare server cluster sizes consisting of 1, 2 and 3 servers.

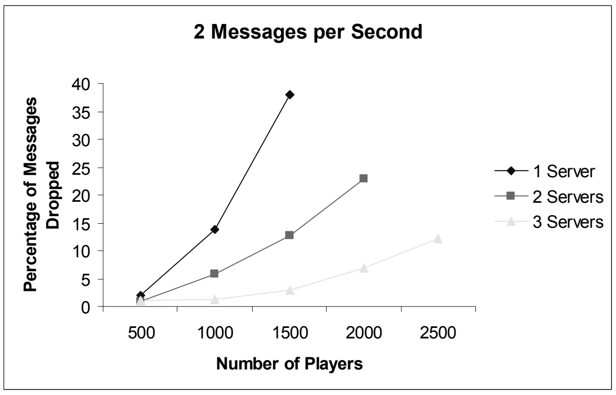

**Figure 5 – 3 Servers, 2 messages per second**

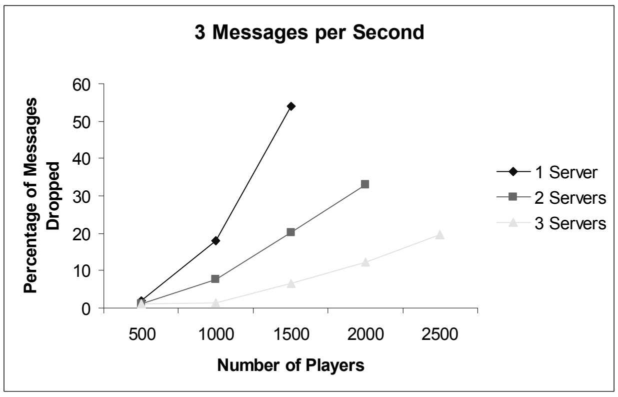

**Figure 6 – 3 Servers, 3 messages per second**

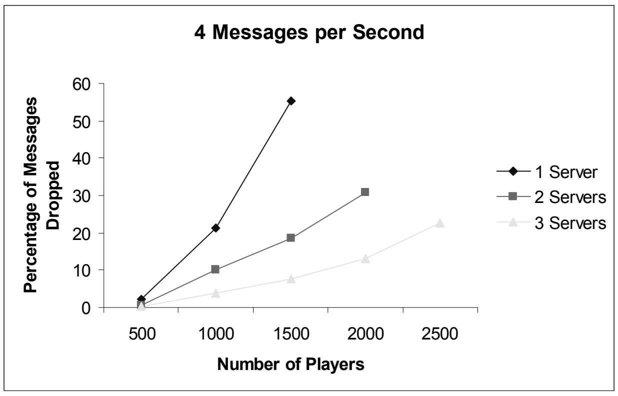

**Figure 7 – 3 Servers, 4 messages per second**

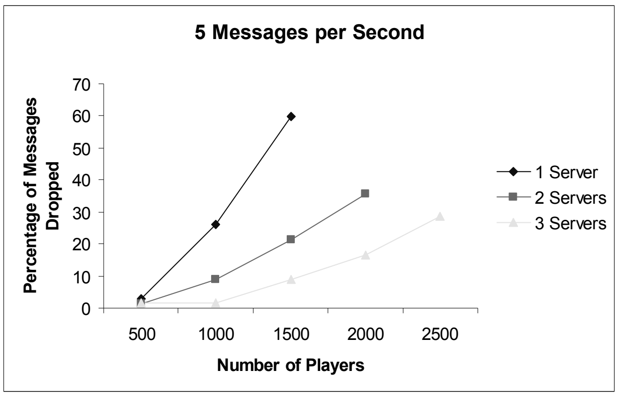

**Figure 8 – 3 servers, 5 messages per second**

As can be seen from Figures 5 through 8 inclusive, the message drop rate increases when the *PUM* transmission frequency is increased. In order to analyse the results, only the tests with the largest number of players for each number of servers will be discussed (1500, 2000 and 2500 players for 1, 2 and 3 servers respectively):

- **1 Server**: at 2 messages per second transmission rate, a single server with 1500 players drops less than 40% of the messages. At 3 messages per second, a single server was observed to drop 53% of messages. At 4 messages per second, the server was observed to drop 56% of messages. At 5 messages per second, the server dropped 60% of messages.
- **2 Servers**: the average drop rate observed with two servers and 2000 players were 24%, 33%, 31% and 36% for 2, 3, 4 and 5 messages per second respectively. The deviation between 3 and 4 message per second from the expected trend can be attributed to variations in the external processing demands on the test machines.
- **3 Servers**: the trend in drop rate is obvious in the 3 server tests. The message drop rates grow proportionally to the frequency of message transmission. The drop rates were 12%, 20%, 23% and 29% for 2, 3, 4 and 5 messages per second respectively.

We show that when more servers are present fewer messages are dropped, irrelevant of the frequency of PUM message exchange. However, there is an anomaly present with 2 servers exhibiting similar drop rates for 3 and 4 messages per second. We put this anomaly down to external machine usage when the series of experiments was recorded for 2 servers.

## 4.6 Prediction Overhead

This experiment is designed to determine the overhead of *APUM* message exchange. As mentioned previously, the difference between the predictive interest management approach and a traditional aura-based interest management system is predictive interest management's utilisation of an additional message, *APUM*, to pre-empt detection of potential aura intersections. However, this additional message exchange may degrade the system's performance. Therefore, it is necessary to determine to what extent the additional message, *APUM*, affects performance.

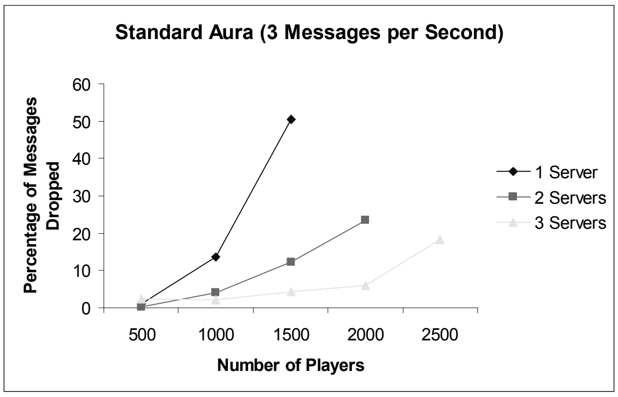

**Figure 9 – Standard aura, 3 messages per second**

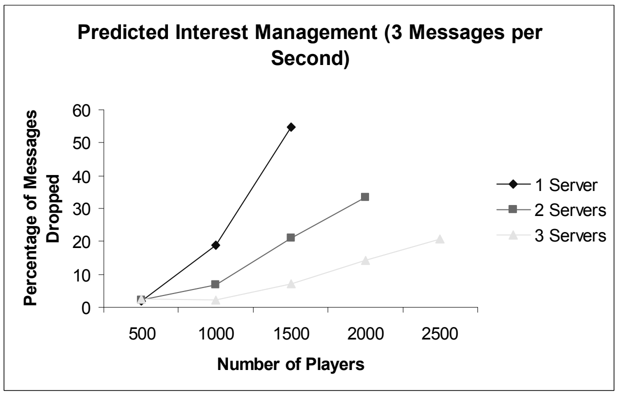

**Figure 10 – Predictive Interest Management – 3 messages per second**

Figure 9 and figure 10 present the results of standard aura and predictive interest management respectively. In both approaches, the drop rate decreases as server numbers increase. However, compared with the aura-based system, the message drop rate in predictive interest management is higher. This indicates that there is an overhead in our approach to interest management compared to when the standard aura approach is used. This overhead is directly related to the servers assuming responsibility for generating, sending, receiving and processing *APUM*s. The drop rate differences between the two ranges (for minimum and maximum player numbers recorded per server) are [0.9% to 5.18%] in the 1 server experiments, [1.77% to 10.13%] in the 2 servers experiments and [0.03% to 8.22%] with 3 servers.

In the single server scenario, as there are no inter-server communications involved, one may assume the overhead arises due to the processing required to determine PAI overlap and identify the appropriate frequency of APUM exchange.

With 2 and 3 servers the overhead increases above that of the single server, however, in the worst case scenario the additional overhead incurred by predictive interest management (2 server

2000 players) overhead is only 10% for maximum number of players supported.

The increase in the drop rate, compared with the traditional aura-based interest management system, does appear detrimental to the appropriateness of predictive interest management on first inspection. The overhead is tolerable (it is not that great in that the system is unusable) and the addition of servers can alleviate the overhead (which is the purpose of clustering).

## 4.7 Scalability

According to the results displayed in the first experiment in this section, the maximum number of players a server can support simultaneously is 1500 under our test conditions. In the following experiments, the number of players our system may support is increased as more servers are made available.

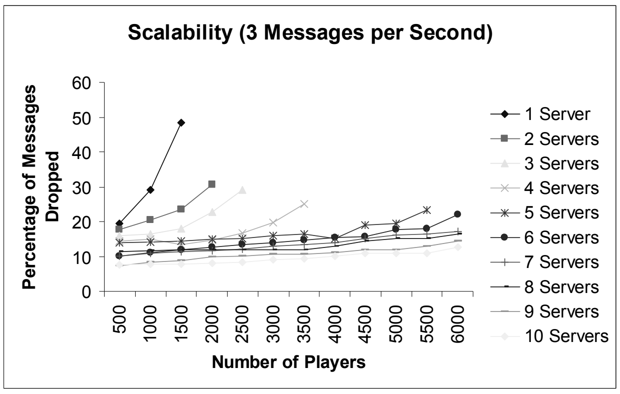

**Figure 11 - Scalability**

Figure 11 shows the performance results for up to 6000 players on 10 servers. The ideal trend should show the following when servers are added to a cluster: (i) the drop rate decreases or remains constant for a specific number of players; (ii) additional players may be supported. From observations of the graph shown in figure 11 we can see that what we consider to be an ideal trend has been achieved. With ten servers present our system can support 6000 players with drop rates of less than 10%.

## 5. CONCLUSIONS AND FUTURE WORK

We have presented an approach to integrating behavioural style interest management and off the shelf load balancing techniques to provide an efficient approach to scalable online gaming using clustered server solutions. By taking a behavioural approach to load balancing we are affording a greater degree of interactivity while minimising the problem of crowding commonly found when geographic regionalisation is used for governing player interaction. In addition, our approach is less complicated (application dependent) than related works associated to mapping load balancing to virtual world regionalisation. Our series of experiments demonstrate that our approach is scalable while maintaining real-time requirements.

Our discussion of interest management, clustered server solutions, and load balancing provides a comprehensive description of how different techniques and technologies combine to provide scalable server side solutions for MMORPGs. We provide clarification of the techniques available, how they relate to each other, and their justification for use. As such, we believe such a discussion provides clarity of understanding for researchers new to this area by providing a focus of detail that is difficult to attain in any one co-located piece of text. This in itself, we believe, is a contribution to the research community.

There is a need to conduct much more research to derive ideal load balancing techniques for use in clustered server solutions for MMORPGs. Techniques that are based on geographic regionalisation may appear an appropriate approach to allocating processing resources/servers and have been explored at length. However, we have demonstrated that there are other opportunities available to achieve similar results without using geographic regionalisation.

Our future work will concentrate on extending the use of behavioural interest management techniques to derive greater scalability. We acknowledge that geographic regionalisation is not without its merits. Therefore, we intend to explore combining geographic and behavioural techniques to provide a unified approach to efficient load balancing, suitable for a wide range of different online gaming scenarios.

## 6. ACKNOWLEDGMENTS

This work is funded by UK EPSRC EP/D037743/1 “Networked Computing in Inter-Organisation Settings” and UK EPSRC Grant GR/S04529/01: “Middle-ware Services for Scalable Networked Virtual Environments”.

## 7. REFERENCES

[1] G. Morgan, F. Lu, “Predictive Interest Management: An Approach to Managing Message Dissemination for Distributed Virtual Environments”, Richmedia2003, Switzerland, 2003.

[2] F. Lu, K. Storey, G. Morgan, “Message Oriented Middleware Services for Networked Games”, In Proc. of the I3D 2005. ACM Symposium on Interactive 3D Graphics and Games, Washington DC, 2005

[3] M. J. Zyda and D. R. Pratt, “NPSNET: A 3D visual simulator for virtual world exploration and experience”, In Tomorrow’s Realities Gallery, Visual Proceedings of SIGGRAPH 91, p. 30, USA, July 1991

[4] M. R. Macedonia, D. R Pratt, and M. J. Zyda, "A Network Architecture for Large Scale Virtual Environments," Proceedings of the 19th Army Science Conference, Orlando, Florida, June 1994.

[5] C. M. Greenhalgh, "Awareness Management in the MASSIVE Systems", Distributed Systems Engineering, Vol 5, No 3, September 1998, pp. 129-137, IOP Publishing.

[6] C. Greenhalgh, S. Benford, “MASSIVE: a distributed virtual reality system incorporating spatial trading”, In Proc. International Conference on distributed computing systems (DCS 95), Vancouver, 1995.

[7] S. Singhal and M. Zyda, “Networked Virtual Environments, Design and Implementation”, Addison Wesley, 1999.

[8] S. E Parkin, P. Andras, G. Morgan, “Managing Missed Interactions in Distributed Virtual Environments”, to appear

in Eurographics Symposium on Virtual Environments, Portugal, May 2006

[9] J. W. Barrus, R. C. Waters, D, B. Anderson, "Locales: Supporting Large Multiuser Virtual Environments", IEEE Computer Graphics and Applications, 16,6, p 50-57, Nov 1997.

[10] S. Han and M. Lim, "ATLAS-II: A Scalable and Self-tuneable Network Framework for Networked Virtual Environments," In Proc. The Second Young Investigator's Forum on Virtual Reality (YVR'03), Kangwon Province, Korea, 2003.

[11] D. Kushner, "Engineering Everquest", IEEE Spectrum Magazine, July 2005.

[12] A. Shaikh et al, "On Demand Platform for Online Games", IBM Systems Journal, Vol 45, No1, 2006

[13] A. Bharambe, J. Pang, S. Seshan, "A Distributed Architecture for Interactive Multiplayer Games", In CMU CS Technical Report Number CMU-CS-05-112, 2005

[14] D. Bauer, S. Rooney, P. Scotton, "Network Infrastructure for Massively Distributed Games", In Proc. of the 1st workshop on Network and system support for games, Germany p. 36 – 43, 2002

[15] G. Singh, et al., "BrickNet: Sharing object behaviors on the net", In Proc. of the IEEE Virtual Reality Annual International Symposium, pp 19-25. Los Alamitos CA, IEEE Computer Society Press

[16] T. A. Funkhouser, "RING: A Client-Server System for Multi-User Virtual Environments", Computer Graphics (1995 SIGGRAPH Symposium on Interactive 3D Graphics), pp 85-92, Monterey, CA, 1995

[17] B. De Vleeschauwer, et al., "Network and System Support for Games", Proc. of 4th ACM SIGCOMM workshop on Network and system support for games, pp 1 – 7, Hawthorne, NY, 2005

[18] T. Das, et al., "NEtEffect: A Network, Architecture for Large Scale Multi-User Virtual Worlds", Proc. ACM VRST, pp. 157-163, 1997.

[19] M. Hori, et al., "Scalability Issues of Dynamic Space Management for Multiple-Server Networked Virtual Environments", Proc. IEEE Pacific Rim Conf. on Communications, Computers and Signal Processing, pp. 200-203, 2001

[20] J. Chim, R. Lau, H.V. Leong, and Antonio Si, "CyberWalk: A Web-based Distributed Virtual Walkthrough Environment," IEEE Transactions on Multimedia, 5(4):503-515, Dec. 2003.

[21] F. Lu, "Monitoring Middleware for Distributed Virtual Environments", PhD Thesis, May 2006, Newcastle University

[22] K. Storey, F. Lu, and G. Morgan, "Determining Collisions between Moving Spheres for Distributed Virtual Environments", In Proc. of the Computer Graphics International (CGI '04), pp. 140-147, June 16-19, 2004