Scalable Collision Detection for Massively Multiplayer Online Games

Graham Morgan, Kier Storey
School of Computing Science, University of Newcastle, Newcastle upon Tyne
E-mail: {Graham.Morgan, Kier.Storey}@newcastle.ac.uk

ABSTRACT

We describe approaches for satisfying the real-time collision detection requirements of distributed virtual environments. We assume a distributed virtual environment is deployed using client/server architecture typical of commercial massive multiplayer online games. We exploit the scalability provided by the clustering of servers in the development of a real-time collision detection service that may scale to satisfy the requirements of virtual environments that are constructed of many thousands of objects. We present performance figures that show our approaches to be scalable in that an addition of servers to a cluster results in an increased number of objects that can be considered for collision detection in real-time.

1. Introduction

A *distributed virtual environment* (DVE) provides a graphical representation of a virtual world that may be navigated by a number of users. User access is via a node that assumes responsibility for providing an interface to a virtual world that satisfies user/world interaction requirements. A DVE application typically presents a virtual world within which a user may be represented via an *avatar* (digital representation of a user in a virtual reality site) with additional objects populating the virtual world to promote realism (e.g., automatons, furniture, buildings). Such applications have been used for training purposes [2], computer supported collaborative work (CSCW) [1] and social play.

The ability to provide server technologies that may scale to satisfy the requirements of many thousands of players is essential to ensure the commercial success of a Massively Multiplayer Online Games (MMOG). An approach to server side scalability for Internet based applications is via the clustering of servers (usually PCs) connected via a LAN. To satisfy increased client demand additional servers are added to the cluster. Many enabling middleware technologies incorporate clustering technologies as standard (e.g., [6]).

A challenge to MMOG developers is to develop middleware services that may benefit from the scalability provided by clustered servers. A service that may benefit from such deployment is real-time collision detection. The presentation of a virtual world populated with moving objects requires real-time collision detection algorithms to identify when objects collide. This promotes realism as players do not expect to see solid objects passing through each other and allows interaction between objects/avatars to occur (e.g., picking up an object, pressing a button).

Initial research into applying clustered server technologies to MMOG development appears promising [3]. However, providing collision detection services that may benefit from the scalability offered by the clustering of servers has not been explored sufficiently in the literature to provide developers with possible solutions to scalable real-time collision detection for client/server type MMOG deployment.

Existing collision detection algorithms and implementations are developed with a single node deployment in mind. There has been limited development of parallel execution of collision detection using local hardware (e.g., [7]) with no work on tailoring collision detection services for deployment over a cluster of servers.

In this paper we present a collision detection service that may scale to satisfy the requirements of complex virtual worlds supporting many thousands of objects. Our service makes use of a cluster of servers to achieve scalability. We categorize object types within the virtual world based on their control of movement within the virtual world (e.g., player influenced, automaton, inanimate). Using our categorization, we provide a number of scenarios of object deployment in client/server architectures associated with MMOG implementation. We present performance figures that indicate the suitability of our deployment scenarios and provide evidence that our approach is scalable.

The paper is structured as follows. Section 2 describes background and related work. In section 3 we present our approach, and section 4 presents performance figures. Conclusions and future work is presented in section 5.

2. Background and Related Work

We assume a DVE represents a 3D geographic space containing objects that may navigate such a space. The DVE is deployed across geographically separated nodes connected by an underlying network. Each node may host a number of objects, their local objects, with nodes responsible for informing each other of the actions (e.g., movement) of local objects via the exchange of messages across the network (state update messages).

Collision detection in a DVE may be based on state update message exchange between nodes. A state update message indicating that local object, say obj_1, of node N_1 has moved may result in a receiving node, say N_2, identifying a collision between N_2's local object, say obj_2, and obj_1. However, before N_2 enacts collision response both nodes (N_1 and N_2) must agree that a collision has or has not taken place. If agreement is reached that a collision has occurred then N_1 and N_2 must enact a suitable collision response (e.g., change direction of movement) for their local objects. However, due to message latency and processing delays it is possible that nodes N_1 and N_2 may disagree on collision. A possible solution would be to only enact collision response if both N_1 and N_2 agree. This approach may still result in inconsistency of collision response. Inconsistency may arise due to deviations relating to collision identification carried out by N_1 and N_2. Collision response may be associated to the actual point of contact between obj_1 and obj_2 (at the polygon level) and other factors (e.g., orientation, velocity). If N_1 and N_2 both agree on collision but on different points of contact then the collision responses enacted by each node may appear inconsistent. Including in the agreement phase a mutually consistent view of point of collision would solve this problem. However, the probability of N_1 and N_2 independently determine the same point of contact would decrease the more complicated an object becomes (the number of polygons used to construct the object). Therefore, N_1 or N_2 must take the lead and determine point of contact.

There have been significant advances in the development of real-time collision detection algorithms for single node deployment. A simple algorithm for determining collisions would be to compare all objects with all other objects: a brute force approach to collision detection. Such an approach is an $O(n^2)$ problem and satisfying collision detection requirements for large numbers of objects may not be possible in real-time. This problem has been well studied in the literature and a number of algorithms have been proposed that perform better than $O(n^2)$ (e.g., [4] [5]). A common approach to collision detection is to use a two step scenario: (i) a broad phase that eliminates many pairwise comparisons due to the large distances that separate objects; (ii) a narrow phase for identifying exact collisions between object pairs highlighted by the broad phase as potential candidates for collision.

DVEs have utilized existing collision detection techniques designed for single node execution with consensus on collision between participating nodes derived from message passing. This is primarily a peer-to-peer approach, where individual nodes share collision detection responsibilities equally (based on local objects as mentioned earlier). However, for MMOGs, a client/server approach is common for DVE deployment. To enable virtual worlds to support many thousands of participants an MMOG vendor may deploy a cluster of servers over a single LAN. This provides an opportunity to re-engineer existing collision detection algorithms in a manner more suited to clustered server deployment, and so attempt to provide scalable real-time collision detection:

- **Scalability** – The addition of servers reduces the time taken to satisfy the collision detection requirements of a DVE by ensuring that collision detection is carried out only once for an object pair during an iteration of a collision detection algorithm.
- **Consistency** – As collision tests are carried out only once, nodes supporting the virtual world will receive mutually consistent views of the nature of collision detection that result from the narrow phase.
- **Timeliness** – Only interested nodes will receive collision detection notification without such nodes having to enter into any form of agreement protocol to determine the exact nature of the collision.

We consider this work to be of benefit to the community because, as of yet, there has been a lack of literature associated to the practical deployment of collision detection techniques for use with clustered server technology. By presenting a number of approaches to distributed collision detection and providing performance analysis we hope to identify appropriate avenues for further research related to satisfying the real-time collision detection requirements for client/server based DVE architectures supported by clustered servers.

3. Architecture

In this section we describe our general architecture of deployment for the provisioning of scalable real-time collision detection. The basic approach we advocate is the utilization of a cluster of machines co-located on the same LAN designed to support collision detection, with the general application level interface requirements of the virtual world satisfied by nodes that support user access. Such nodes may be large in number and geographically separated, modeling an Internet style deployment of a MMOG.

3.1 Overview

Figure 1 presents a diagram outlining the description of our architecture. We assume a number of nodes co-located in a single LAN (collision detection cluster) will satisfy the narrow phase of the collision detection requirements of the DVE. The virtual world is spatially sub-divided into regions with each node in the collision detection cluster responsible for determining narrow phase collision detection tests for a set of objects contained in a single region of the virtual world. A region is unique to a node and only a single node will attempt narrow phase collision detection tests for the same two objects during an iteration of narrow phase collision detection (between one frame of animation and the next). An object may appear in multiple regions (the case if an object overlaps region boundaries). This may result in an object pair appearing in more than one region. If this is the case then only one node will enact the narrow phase collision test. This eliminates the possibility of duplicate narrow phase collision tests carried out on distinct nodes.

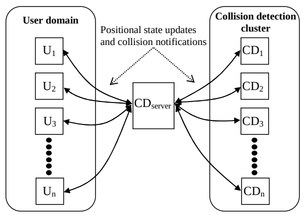

User domain
Positional state updates and collision notifications
Collision detection cluster
U_1
U_2
U_3
U_n
CD_{server}
CD_1
CD_2
CD_3
CD_n

Figure 1 – Collision detection via cluster.

A single node exists (collision detection server) that assumes responsibility for identifying which region(s) an object is associated with and informing the appropriate collision detection node(s) of the objects that they must consider for narrow phase collision detection.

We assume a set of nodes that support the DVE and allow users to interact with the DVE (user domain). Via a node, each user may control one or more avatars that may interact with objects that populate the DVE. We classify objects into the three following categories:

- **Avatars** – objects that move under the direct control of a user.
- **Active** – objects that move in some pre-determined manner as directed by some programmed logic associated with the virtual world.
- **Non-active** – objects that do not normally move but may move due to collision with another object.

Given our assumptions on the types of objects that may exist in a virtual world we may support such objects in a distributed manner in the following way:

- **User domain** - All objects in the virtual world are equally distributed across nodes in the user domain. Nodes associated to users are responsible for updating the location of their local objects and managing the collision response of their local objects.
- **User/Server domain** - Nodes in the user domain are only responsible for updating the location of avatars and managing collision response of avatars with the collision detection server responsible for updating the location of active and non-active objects and managing the collision response of such objects.
- **User/collision domain** - Nodes in the user domain are only responsible for location updates and the collision response of their avatars with active and non-active objects equally distributed across nodes in the collision detection cluster. A node in the collision detection cluster is responsible for location updates and collision response management of their local active and non-active objects.

We identify the above object distribution models based on the fact that active and non-active objects may interact infrequently with avatars and may only move in the virtual world via computer controlled calculations, be it a path finding algorithm for active objects or collision response for non-active objects. Therefore, it is possible for the positional update calculations to be achieved at arbitrary nodes for active and non-active objects. The remote hosting of avatars may introduce unnecessary delay in the time taken for an avatar to react to user interaction. As such interaction tends to be frequent this would reduce the realism of the virtual world. Although a delay may be witnessed when an avatar interacts with a remotely hosted object, this would be the case in any DVE that used distributed hosting of objects. Therefore, we deem it appropriate that avatars are hosted on nodes accessed by associated users.

There are two distinct types of messages contained within our system: (i) messages associated with collision detection updates and; (ii) messages associated with application level state update propagation. We assume a single message may contain information associated to multiple objects. For example, an application level state update message may contain positional update information relating to all objects that have moved since the last positional update message was sent and a collision update message may contain multiple identifications of pairwise collisions.

Messages associated with collision detection serve the purpose of informing nodes in the user domain of collisions whereas messages associated with application level state updates inform nodes in the user domain of changes in state of the virtual world. For example, a collision detection message, say M_1, may be issued by a node in the collision detection cluster, say by CD_1, that identifies a collision between objects hosted at U_1 and U_2, requiring the message M_1 that originates at CD_1 to be sent to U_1 and U_2. An example of application level state update may require U_1 to issue a message M_2 to U_2 if a state change event in the virtual world enacted by U_1 may be deemed influential to the objects hosted by U_2.

Messages associated to collision detection also contain information relating to the positions of objects when collision was determined. This allows user domain nodes to position their objects appropriately when enacting collision response.

There is the possibility that the drawn scene (as viewed by a user) may deviate from an expected collision response. For example, a state update message may relate to positional information of an object and the receiving user node may actually draw a collided scenario due to the positional update associated with its own object and that described in the received message (a possibility due to inconsistency associated with message delay/jitter). However, as no collision detection is carried out by the receiving node then notification of such a collision must be received from the collision detection domain before the receiving node may enact a suitable collision response for its objects. Alternatively, a collision detection message may be received before a node actually views a drawn collision, requiring collision response to be enacted in anticipation of a collision. Although this may appear to highlight an inconsistency in providing collision response when associated to the timely drawing of a scene, we admit such an inconsistency in an effort to ensure all user nodes interested in a collision maintain a mutually consistent view of such a collision.

We now consider the different object distribution scenarios and their associated message passing requirements in detail in the following sections.

3.2 User Domain Deployment

User domain deployment is typical of many peer-to-peer deployments associated with DVEs and is commonly referred to as frequent state regeneration in online gaming communities. A user domain node maintains the state information of a subset of objects contained within the virtual world and assumes the responsibility of informing other user domain nodes of virtual world updates associated to their local objects.

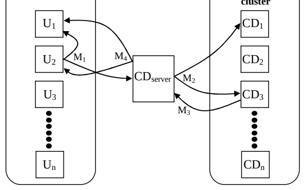


Figure 2 – User deployment scenario.

Figure 2 aids in the description of message flow associated with user domain deployment. The collision detection server participates in application level state update message passing as a receiver only. The collision detection server consumes all messages associated to positional update information of all objects associated to all user nodes. As positional state update messages are received (M_1) the collision detection server identifies which region the objects described in M_1 are located (for collision detection purposes) and issues the positional update message (M_2) to the appropriate CD nodes (CD_1, CD_3) indicating such objects have moved. If any collisions have been identified by a CD node then a message is sent to the collision detection server indicating such collisions (CD_3 sending M_3). The collision detection server then issues a message to the nodes where collided objects have been identified (M_4).

3.3 User/Server Domain Deployment

In user/server domain deployment the collision detection server is responsible for active and non-active object management. This requires the collision detection server to move active objects using a well defined algorithm (such as path finding) and implementing collision response for both active and non-active objects. This scenario is commonplace in MMOGs, where a server is responsible for controlling all non-user controlled characters.

The collision detection server must send state update messages to user domain nodes because it is responsible for updating virtual world state (active and non-active objects). This is in addition to collision detection update messages that may be issued to user domain nodes by the collision detection server.

Message passing associated to user/server domain deployment is similar to that described (via figure 2) in user domain deployment. However, messages relating to active and non-active object state will only be sent by the collision detection server within the user domain.

An advantage in the user/server domain deployment scenario over that of user domain deployment is the ability to inform collision detection nodes of state updates related to active and non-active objects directly by the collision detection server. In the user domain deployment scenario such messages had to be propagated in two steps: (i) initial message from user node followed by; (ii) additional message from collision detection server to appropriate collision detection node.

3.4 User/Collision Domain Deployment

In user/collision domain deployment all active and non-active objects are distributed across nodes in the collision detection domain and responsibility for avatars is distributed across nodes in the user domain. Each node in the collision detection domain is responsible for the location updates and collision responses of a subset of active and non-active objects located in the virtual world. This approach requires that the application state level update messages associated to active and non-active objects be propagated to the appropriate user nodes from the collision detection nodes. One approach would be to allow collision detection nodes to send messages directly to user nodes. A drawback to this approach is the increased dependency between the collision detection domain and the user domain. Ideally, we would like to implement scalable collision detection that is generic in nature and may suffer only the minimum tailoring to work with arbitrary numbers of user domain nodes. Making user domain nodes aware of collision detection nodes would inhibit the ability to vary collision detection node availability (e.g., increase to handle additional complexity of world or decrease for routine server maintenance). Therefore, we utilize the collision detection server in our approach to propagating application state messages from the collision detection domain to the user domain.

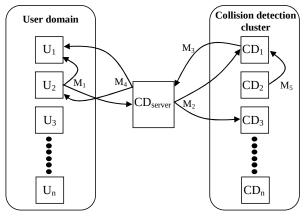


Figure 3- User/collision domain deployment scenario.

The collision detection server receives all application state update messages sent from the collision detection domain and relays these messages to the appropriate recipient nodes in the user domain. This has the result of the collision detection server appearing to host all active and non-active objects in the user domain (similar to user/server domain deployment). However, the processing burden is lessened on the collision detection server compared to the user/server domain deployment due to the responsibility for active/non-active object management undertaken by nodes in the collision detection domain.

Figure 3 describes an example of message passing that may occur in the user/collision deployment scenario. We assume an optimization in that nodes in the collision detection domain may inform each other of active/non-active object traversals of spatial division boundaries associated to collision detection. For example, in figure 3 CD_2 notices that an object, say obj_1, it hosts has traversed a spatial division boundary that indicates that obj_1 and associated collision detection should now be hosted by CD_1. Therefore, CD_2 sends a message (M_5) to CD_1 informing CD_1 of the change in object hosting and collision detection responsibility. This approach to object hosting ensures that collision detection nodes only need to inform each other when objects traverse spatial boundaries and not when objects move as would be the case if a collision detection node assumed hosting responsibilities for an object throughout the lifetime of the virtual world.

Apart from the addition of inter-collision domain node interaction (e.g., M_5) the description of message passing shown in figure 3 remains the same as that for figure 2.

4. Experiments

This section describes the experiments carried out to assess the performance of our different approaches to determining collisions (as described in section 3). The primary reason for our experiments is to deduce the scalability of our approaches. We consider scalability to relate to the complexity (number of objects present) of the virtual world. Ideally, we would like the addition of collision detection nodes to provide a speedup in the time taken to detect and inform user domain nodes of collisions. For comparative analysis we ran the narrow phase collision detection on a single node (this is shown in the graphs by the line '1 Node').

We are interested in the time it takes to determine all collisions within the virtual world and inform nodes of such collisions. Therefore, we manipulated the movement of objects so that they would only update their position in the virtual world given a signal from the collision detection server that all collisions have been identified and appropriate user nodes have been informed of such collisions. The time taken between state updates for nodes located in the user domain is taken as our measurement of performance (we aim to progress object movement whenever we are told by the collision detection server that all collisions have been identified). This measurement includes message latency overheads. We timed how long it took to advance a frame of animation for all user nodes and took the mean value.

User domain nodes were co-located on a different LAN to that used for the collision detection domain. However, both LANs are geographically co-located. Message latency times between user domain and collision detection server are less than 1ms. In a real MMOG deployment, user domain nodes may be geographically distributed over the Internet with message latencies in excess of 100ms in some instances. However, the purpose of our experiments is to identify the scalability offered by clustered servers for collision detection.

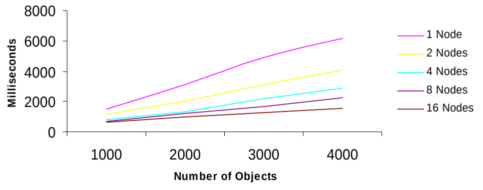
Both LANs contain the same type of machines and configuration (Pentium III 700MHz PCs with 512MB RAM running Red Hat Linux 7.2) with nodes connected via 100 Mbits fast Ethernet. We implemented our system in Java. However, as we are predominantly interested in collision detection and not graphical representation we draw no graphics.

We simulate avatar movement as well as active object movement within our experiments. An attempt is made to provide realistic movement of objects within the virtual world. A number of targets (T) are positioned within the virtual world that objects (O) travel towards. Each target has the ability to relocate during the execution of an experiment. Relocation of targets is determined after the elapse of some random time (between T_t^{min} and T_t^{max}) from the time the previous relocation event occurred. Furthermore, objects may change their targets in the same manner (random time between O_t^{min} and O_t^{max}). Given that the number of targets is less than the number of objects and T_t^{min}, T_t^{max}, O_t^{min} and O_t^{max} are set appropriately, objects will cluster and disperse throughout the experiment.

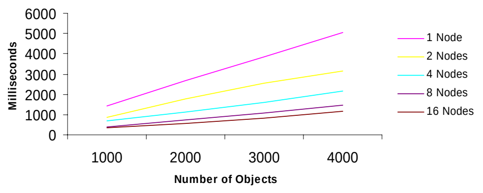
We assume that each user node maintains a single avatar with the remaining objects present in the virtual world equally split between active and non-active objects. This scenario presents the expected approach to participation of a single avatar per user in a DVE.

Object numbers were increased gradually from 1000 to 4000 with the time taken to advance a frame recorded at each increment. The coverage of the virtual world was set at approximately 6%. That is, all object volumes cumulatively cover approximately 6% of total virtual world space. In all experiments we had 100 nodes in the user domain.

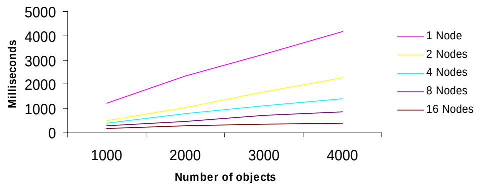
Graph 1 - User Domain Deployment



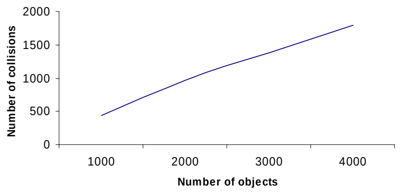
Graph 2 - User/Server Domain Deployment



Graph 3 - User/Collision Domain Deployment



Graph 4 - Collisions



Graphs 1 through 3 present performance figures in the three deployment scenarios described in section 3 for 1, 2, 4, 8 and 16 nodes in the collision detection domain. Graph 4 provides an indication (average over the run of all experiments and scenarios) of the number of exact collisions detected for 1000 through 4000 objects. When 4000 objects are present there are in excess of 1600 exact collisions between object pairs. This identifies that the computational requirements for the narrow collision detection phase rises given increasing numbers of objects.

The first observation to be made is that increasing the number of collision detection nodes does improve performance of the overall system (i.e., time taken to calculate all collisions and inform nodes in the user domain). Doubling the number of collision detection nodes leads to almost a doubling in performance. These results are promising as the addition of computational resources in the collision detection domain relates to an expected speedup in the overall system. That is, the cost of additional resources in the collision detection domain is justified due to the rate of performance increase.

A significant observation is that a single node performs at a rate that is not acceptable for a DVE. For 1000 objects a single node provides rates of approximately 1500 milliseconds rising to over 6000 milliseconds for 4000 objects in user domain deployment. However, this does reduce slightly in user/collision deployment, giving approximately 1200 and 4100 milliseconds for 1000 and 4000 objects respectively. When 16 nodes are present such figures do not exceed 1500 milliseconds for 4000 objects and provide no more than 600 milliseconds for 1000 objects in the worst case (user domain deployment).

For all node configurations the user/collision deployment outperforms all other scenarios. With 4000 objects and 16 nodes the user/collision deployment scenario manages to achieve performance of approximately 300 milliseconds. Although this only equates to a collision detection being achieved at a frame rate of 3 (3 frames of animation may be drawn in a second) this still provides users with what appears to be a reactive system while still maintaining consistency of collision response. If we allow user domain nodes to process frame rate independently of collision detection then the inconsistency of collision will be acceptable for objects that do not exhibit high velocities. For example, assume user domains may achieve 40 frames per second, then only 10 frames of animation may be shown that are free from collision detection. If an object travels a limited distance in these ten frames then the collision response will appear appropriate.

We assume the overhead of message passing to be the main reason why user deployment performs less well than user/server and user/collision deployments. In user deployment all user nodes must propagate their state updates for all objects (avatar, active and non-active objects) to the collision detection server that in turn propagates this information to the appropriate collision domain nodes. In user/server and user/collision deployments only avatar state updates need to be propagated to the collision detection nodes (active and non-active by collision detection server in user/server deployment).

The scenario providing the optimum performance is the user/collision scenario. Spreading the management of active and non-active objects across collision domain nodes and making such nodes responsible for informing each other when such objects move appears to utilize the overall processing capabilities of the system more appropriately than the other two scenarios. As collision detection nodes are co-located on a single LAN the overhead of message passing is relatively low compared to the requirement to propagate messages related to such objects from the nodes in the user domain. We may assume that in user/server deployment the responsibility of maintaining active and non-active objects on the server will consume the processing resources of the collision detection server at the expense of carrying out broad phase collision detection.

The performance results presented here indicate that there is benefit to be gained from distribution of collision detection responsibilities across nodes that are specifically dedicated to satisfying collision detection requirements for DVEs. This is most evident in the case where objects that are not under the direct control of the user, and can therefore be distributed arbitrarily in the system, are distributed on nodes that assume responsibility for collision detection.

5. Conclusions and Future Work

We have presented a number of approaches for providing real-time collision detection for DVEs. We demonstrate, via experimentation, that our approaches are scalable in terms of virtual world complexity (i.e., number of objects and detail of objects) and do provide performances that may be acceptable to DVEs that use client/server based architectures for their deployment. We concentrate on ensuring that participants of a virtual world share a mutually consistent view of exact collision detection and so aim to alleviate virtual world inconsistencies related to collision detection and response. The aim of our approach is to provide an architecture within which existing collision detection methods may be applied in a traditional two phase approach.

Future work will concentrate on extending our architecture for scenarios where a single LAN deployment of collision detection nodes may not be suitable. For example, when user nodes are distributed around the world (as would be the case in a commercial MMOG deployment). In addition, we are currently investigating mechanisms to enhance our collision detection algorithm [8] and services [9] with Quality of Service (QoS) adaptability to enable QoS guarantees to be applied to the approaches outlined in this paper.

Acknowledgements

This work is funded by the UK EPSRC under grant GR/S04529/01: “Middleware Services for Scalable Networked Virtual Environments”.

6. References

[1] C. Greenhalgh, S. Benford, “MASSIVE: a distributed virtual reality system incorporating spatial trading”, Proceedings IEEE 15th International Conference on distributed computing systems (DCS 95), Vancouver, Canader, June 1995.

[2] D. Miller, J. A. Thorpe. “SIMNET: The advent of simulator networking”, In Proceedings of the IEEE 83(8), p 1114-1123, August 1995.

[3] IBM, “Butterfly.net: Powering Next Generation Gaming with Computing On-Demand”, www.ibm.com , as viewed March 2004

[4] S. Gottschalk, M, C. Lin, D. Monocha, “OBB-Tree: A Hierarchical Structure for Rapid Interference Detection”, SIGGRAPH 93, p247-254, USA, 1993

[5] J. D. Cohen, M. C. Lin, D. Manocha, M. K. Ponamgi, “I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments”, In Proceedings of the 1995 symposium on Interactive 3D graphics, pages 189–196, 218. ACM, Press, 1995

[6] S. Labourey, B. Burke, “JBossClustering UNIX Edition March 2004”, JBoss Documentation, http://www.jboss.org/, as viewed September 2004.

[7] G. Zachmann, “Optimizing the Collision Detection Pipeline”, Proc. of the First International Game Technology Conference (GTEC), Hong Kong, 18-21 January 2001.

[8] K. Storey, F.Lu, G. Morgan, “Determining Collisions between Moving Spheres for Distributed Virtual Environments”, Computer Graphics International (CGI'04), Crete, June 16 - 19, 2004, pp. 140-147, EEE Computer Society Press 2004

[9] G. Morgan, K Storey, F Lu, “Expanding Spheres: A Collision Detection Algorithm for Interest Management in Networked Games”, In Proceedings ICEC 2004, The Netherlands, Lecture Notes in Computer Science Volume 3166 pp. 435 – 440, Springer-Verlag 2004