DYNAMIC SERVER ALLOCATION IN VIRTUAL ENVIRONMENTS, USING QUADTREES FOR DYNAMIC SPACE PARTITION.

Alex Restrepo, Alejandro Montoya and Helmuth Trefftz
Eafit University
Medellín, Colombia
{alexrestrepo, amontoya}@mac.com, htrefftz@eafit.edu.co

ABSTRACT

In actual applications of virtual environments, one of the problems that has received attention is the way to assign computational resources to those who participate in the system. In most cases, static resource allocation is used. Static allocation is simple but has several limitations, among them, the missutilization of resources.
In this paper we present a proof-of-concept of a new system and report the results of experiments with a system that accomplishes dynamic resource allocation by using Quadtrees for space partitioning, in a Shared Virtual Environment.

KEYWORDS

Dynamic Server Allocation, Quadtrees, Virtual Environments.

INTRODUCTION

In a virtual environment it is necessary to keep and share records of all participants. Generally, a server is assigned to a static virtual space for its control, the virtual space can either be empty or not. In this research we try to reduce the number of empty virtual spaces: when a partition is empty, it's not necessary to allocate a server to control it. Released servers can be assigned to other non-empty spaces. The advantage of using a dynamic space partition algorithm is better resource usage. Additionally, we claim that a dynamic allocation scheme can make a virtual environment more scalable.

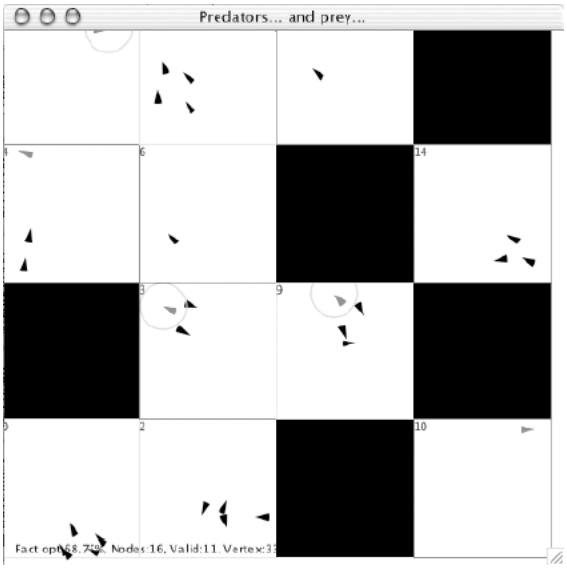
RELATED WORK

Funkhouser[4] presents several topologies intended for virtual environments, however, the presented topologies are based on a static mapping between virtual spaces and servers, just as the architectures suggested by Macedonia[5]. He reports the different factors that affect the performance of a virtual environment, latency, bandwidth, reliability, among others, and how a good performance is vital for the correct functioning of a virtual environment.
Another interesting factor in the analysis of a virtual environment is Load Balancing, where the total amount of work is split between several computers.
Srisawat and Alexandridis [1] propose a system for dynamic allocation of processing power in a mesh-connected set of parallel machines using Quadtrees. In their model, processes are allocated and deallocated using Quadtree-based searches, receiving a significative improvement of allocation over static methods. However, an application where machines are dynamically assigned to a virtual space is still open for exploration.
In most virtual environments, the space assigned to each server is allocated in a static fashion, most commonly rectangular and hexagonal meshes.
In current virtual environments, servers are assigned to different partitions of the virtual space in a static manner. To the best of our knowledge, dynamic space allocation has not been explored. In this paper we present a series of opportunities and limitations that we found in our experiments in dynamic space allocation.
In general, references to specific previous work using quadtrees for virtual space partition was not found in available literature.

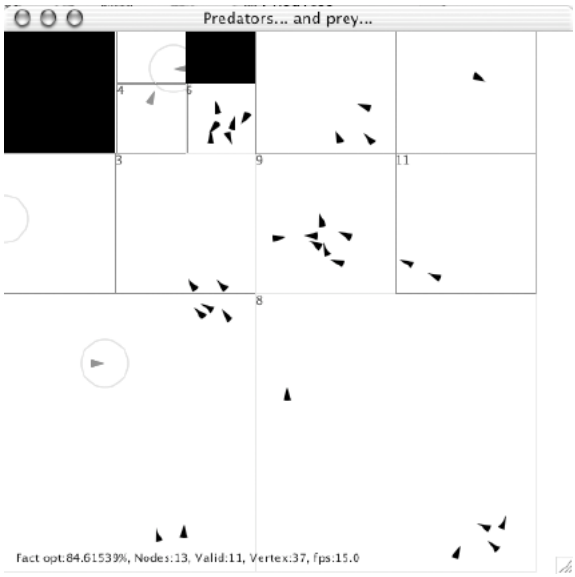
QUADTREE BASED DYNAMIC PARTITIONING

The dynamic partition using quadtrees is a well-known recursive algorithm, which is used in several areas. It works by enclosing all the working space within a square, and from the information contained in it, the algorithm determines whether or not to re-split this square into 4 equally sized sub-squares. The process is repeated recursively. At the end, a treelike structure allows access the contents of the enclosed space in a fast and efficient way, it also helps to determine what portions of space are unused. Because of this, the quadtree solution seems natural to solve the problem of dynamic server allocation.
In a shared virtual environment, communication is more likely to occur among participants that are close in the virtual space, making space partitioning practical. In the case of Quadtrees, we can assign a server to users close to each other in virtual space.
Initially, a Quadtree encapsulating all the possible virtual space in the problem is created, once participants enter this root node, it will be partitioned in 4 sub nodes, after the number of entities has reached a specified limit. Once this partition is made, some nodes may end up empty, these nodes are freed for further use. Users located close to each

other, are assigned to the same node. Picture 1 shows an environment with static allocation, all of the servers (each node represents a server) are allocated but some of them don't have entities inside, this means that they're just "sitting" wasting resources (in black). Picture 2 shows dynamic allocation: not every node is allocated, and they can be used for some other task. Those users in the same node will be allocated to the same server, minimizing the need to communicate with entities outside the node they were allocated to.



Picture 1: Static allocation of a virtual space. All the servers are allocated, however not all are performing any task. The black squares show nodes where there are no entities, inside that virtual space.



Picture 2: Dynamic allocation of a virtual space. Some of the servers are not being used, and each node has an allocated space accordingly with the number of entities under its control.

EXPERIMENTAL SETUP

In order to test the model, a virtual open environment was simulated, with two types of entities with different behaviors. The environment was implemented using both dynamic and static allocation and both results were compared. The simulation was implemented in Java, and all implementations and benchmarks were run on a MacOS X based workstation.
The static model consists in dividing the space into equally-sized nodes whose are static in size and location, each of these nodes will have an allocated server. In the dynamic model, that we propose, the quadtree algorithm is used to divide the space. The resulting nodes change both in size and location, according to the amount of work allocated to them, which is tied to the number of entities within each node. In this simulation, the dynamic partition was recalculated every 1/30th of a second.

For this experiment, the agents had the following characteristics:

Prey: slow moving agents, wandering freely through the system, but trying to move in group, using Craig Reynolds' flocking algorithm [9], when two or more of these agents meet, they continue their way in the same direction. Once they find a different type of agent (predator) in the same node, they will flee in the opposite direction.
Predators: Under normal circumstances they move at the same speed as prey, but when a prey agent enters its sight range, the predator will increase their speed by 50%, in order to catch the prey. Predators are also wandering freely through the system. If they find an agent of different type in their node, they will go after it until they catch it and "eat" it, or until it gets out of sight.

The behavior of the entities was chosen in order to ensure that the entities will be switching nodes quite often. Besides, since some of the agents will be walking as a group, we can assure that some entities will be close to each other, and the space partition will be performed.

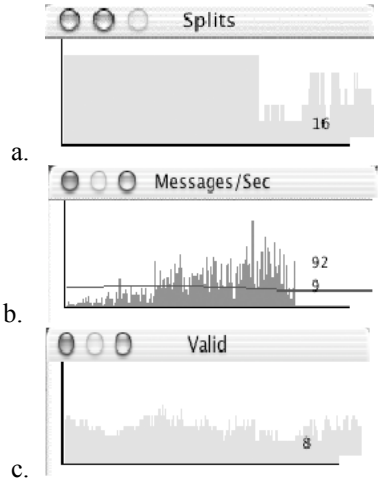
By modifying some of the parameters, such as speed, we gathered the following information:

a. Partition Number per time unit:
For this experiment, each partition had an allocated server,. The experiments were performed with a limit of 16 servers.
Regarding the number of partitions per time unit, we found that in the dynamic model not all of the servers were allocated, however, in the static partition, the total number of servers was allocated all the time (regardless of having any entities in their area or not).

b. Number of messages between servers per time unit:
Each time an entity changes nodes, the server it was assigned to sends a message to the node it is going to. This process works the same for both models, static and dynamic.
This measurement showed that the static model actually involves a smaller number of circulating messages.

c. Optimization factor:
The main problem we tried to solve is the optimization in the use of resources, by trying to maximize the number of allocated servers that are indeed busy.
The optimization factor is the number of available servers divided by the number of allocated servers, and it's given as a percentage. This factor is better as it gets closer to 100%.

The results of the experiments show that the proposed solution is better than the old model, in certain cases. It's important to note that the entities had a relatively big size, compared to their virtual space, making them switch nodes more frequently, therefore increasing the number of transmitted messages.
The results show that the dynamic model makes a better use of resources, however it increments the number of messages in the network, as explained next.

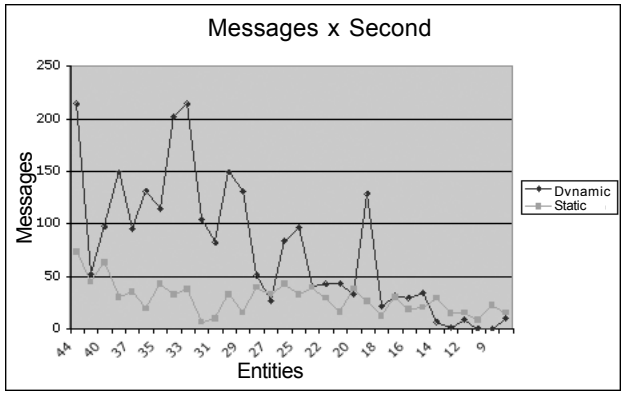


Picture 3: Evolution of the system in time.
a. Number of allocated servers in time.
b. Number of messages sent by unit-time and
c. Number of servers with actual clients.
It's important to note that as time passes by, the number of entities decreases.

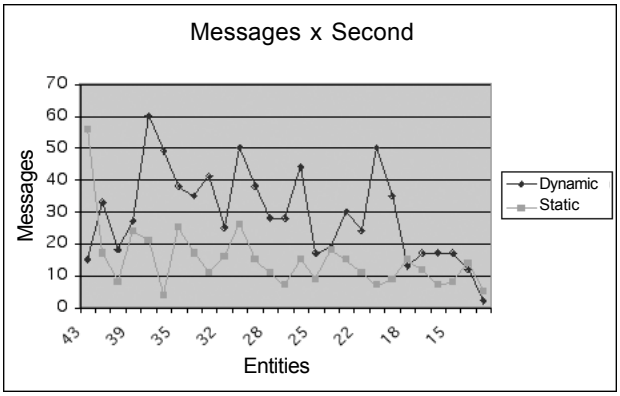
RESULTS

For the measurement of the results, the speed of the entities was set to a random number that does not obey any particular reason, only to test the experiment under different conditions, since at a greater speed, the probability that an entity changes nodes increases, and therefore, the whole space partition needs to be recalculated, for the dynamic algorithm.

Messages per second
Pictures 4.1 and 4.2 show the number of messages per second being generated by the amount of existing entities.
In the first case (picture 4.1) the entities have a higher speed, and we found a significative difference between the static model (black) and the static model (Gray)
In the second case (picture 4.2) the entities have a slower speed, and the behavior was as expected, the number of messages decreased for both models, but the dynamic model showed a better improvement. However this measurement turned out to be better for the static model.



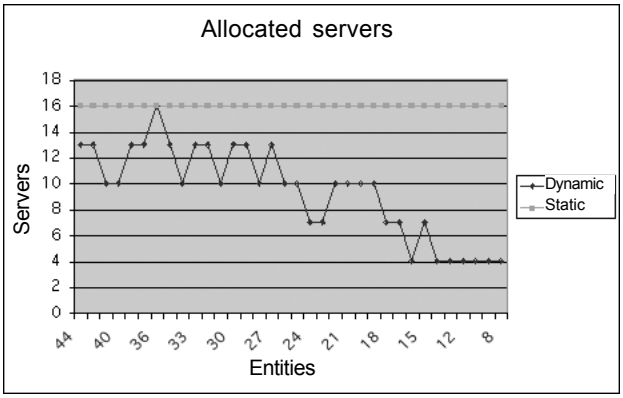
Picture 4.1: Fast moving entities



Picture 4.2: Slow moving entities.

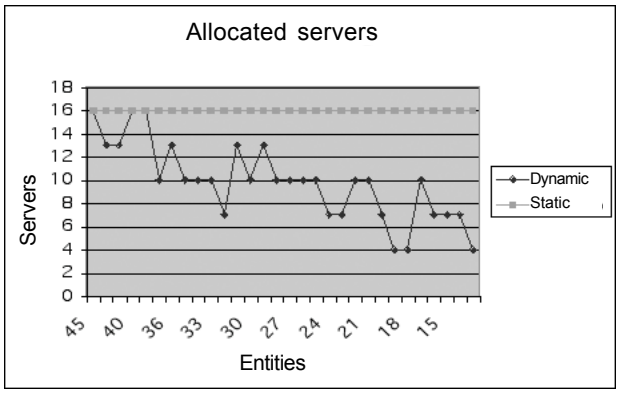
It's clear that the proposed model increases the message number per second (in the case when the number of the entities was 50, it is increased in an average of 30 messages per second, based on the static model), this result was also expected since the partitioning of a node causes its sub nodes to become too small, increasing the probability of passing from one node to another. However this difference decreases when the entities move slowly and when the number of entities is low, since the partitions are be performed less often.

Allocation



Picture 5.1: Fast moving entities

Pictures 5.1 and 5.2 show that in the static model all the servers remain allocated, whereas in the dynamic model not all of them are allocated and can be used for some other task. It is also important to note that once the number of entities decreases, the utilization of servers also decreases in the dynamic model. The results also show that the proposed model behaves in a similar way, independently of the speed of the entities.

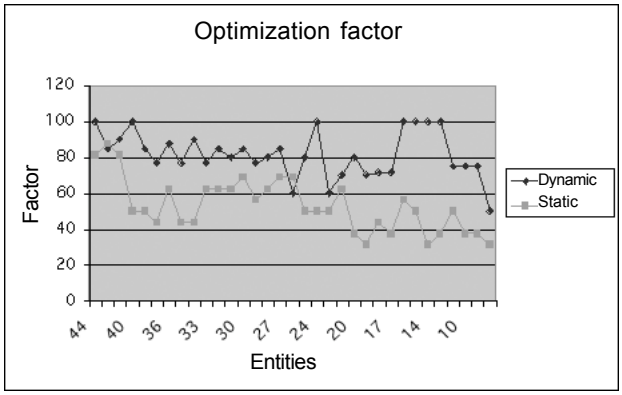


Picture 5.2: Slow moving entities

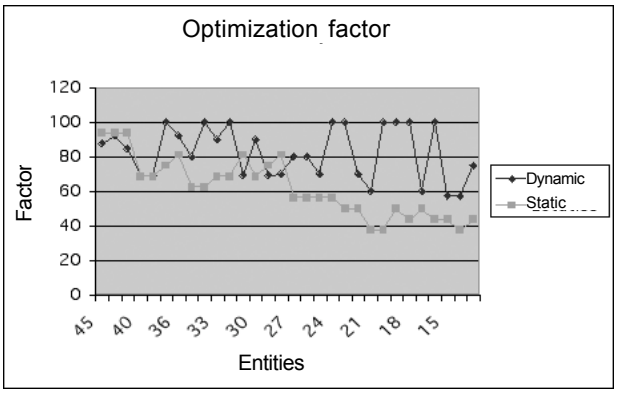
In the static partition model, the number of allocated severs shouldn't change (gray), but in the dynamic model (black), the number of allocated servers decreases at the same time that the number of entities decrease.

Optimization Factor

The results described in pictures 6.1 and 6.2 show a better utilization of resources in the proposed model, independently of speed. Besides, utilization improves as the number of entities decreases. In the static model a peak factor of close to 90% is reached, while in the dynamic model a factor of 100% is often achieved. In general the dynamic model shows an improvement of near a 24% compared to the static model, in the resource optimization factor.



Picture 6.1: Fast moving entities



Picture 6.2: Slow moving entities

The optimization factor measures the number of servers assigned that are actually being used, the higher the better.

CONCLUSIONS

In this article we introduced a new method for dynamic allocation of servers using Quadtrees. This method proves to be more powerful in the resource optimization factor than in the static model. The simulation shows that the proposed Quadtree based model results in the best utilization of the system, showing an improvement of 24% compared to the static model.

However, the cost of this level of optimization is high, since it can saturate the traffic of messages in the network more easily than a static model, because there are more messages per time-unit in the dynamic model than in the static model.

FUTURE WORK

The problem can be extended to be used in three dimensional environments, using Octrees[6]. Also, the model can be evaluated using systems where the entities are controlled by humans, i.e. games, where users are also more interested on their immediate environment than on distant entities. Dynamic allocation can also be used in other areas such as communications, where it's essential to have resources to satisfy demand. The use of the model could also be explored in other types of load balancing problems, such as the allocation of signals in the case of cellular phone networks.

REFERENCES

[1] Srisawat y Alexandridis, "A Quadtree Based Dynamic Processor Allocation Scheme for Mesh-Connected Parallel Machines" Conference in Computational Applications (CAINE 98), Las Vegas, Nevada, USA, November 1998.

[2] Srisawat y Alexandridis, “A New Quadtree Based subsystem Allocation Technique for Mesh-Connected Parallel Machines” ACM-SIGARCH conference in super computing (ICS 99), june 20-25/1999, Rhodes, Greece

[3] Broll, “Distributed Virtual Reality for Everyone - a framework for Networked VR on the internet” IEEE Computer Society Press, 1997

[4] Thomas A. Funkhouser, Network Topologies for scalable multi-user virtual environments. Bell Laboratories

[5] Michael R. Macedonia, A Taxonomy for networked virtual environments. Fraunhofer Center for Research in Computer Graphics
Michael J. Zyda. Computer Science Department
Naval Postgraduate School

[6] Jonathan Ferraris, Quadtrees
http://www.gamedev.net/reference/programming/features/quadtrees/

[7] Ben Humphrey (DigiBen), Octree tutorial
http://www.gametutorials.com/Tutorials/OpenGL/Octree.htm

[8] James D. Foley, Andries Van Dam, Steven K. Finer, John F. Hughe. Computer Graphics: Principles and Practice in C (2nd Edition)

[9] Craig Reynolds, Boids
http://www.red3d.com/dwr/boids/