A Scalable Architecture for Crowd Simulation: Implementing a Parallel Action Server*

G. Vigueras, M. Lozano, C. Pérez and J. M. Orduña
Departamento de Informática
Universidad de Valencia
Spain
{Guillermo.Vigueras, Juan.Orduna}@uv.es

Abstract

Crowd simulation can be considered as a special case of Virtual Environments where avatars are intelligent agents instead of user-driven entities. These applications require both rendering visually plausible images of the virtual world and managing the behavior of autonomous agents. Although several proposals have focused on the software architectures for these systems, the scalability of crowd simulation is still an open issue.

In this paper, we propose a scalable architecture that can manage large crowds of autonomous agents at interactive rates. This proposal consists of enhancing a previously proposed architecture through the efficient parallelization of the Action Server and the distribution of the semantic database. In this way, the system bottleneck is removed, and new Action Servers (hosted each one on a new computer) can be added as necessary. The evaluation results show that the proposed architecture is able to fully exploit the underlying hardware platform, regardless of both the number and the kind of computers that form the system. Therefore, this system architecture provides the scalability required for large-scale crowd simulation.

1 Introduction

Crowd simulation can be considered as a special case of Virtual Environments where the avatars are intelligent agents instead of user-driven entities. Each of these agent-based entities can have its own goals, knowledge and behavior. In recent years, crowd simulation has become an essential tool for many virtual environment applications in education, training, and entertainment [1, 7, 17]. These applications require both rendering visually plausible images of the virtual world and managing the behavior of autonomous agents. The sum of these requirements results in a computational cost that highly increases with the numbers of agents in the system, requiring a scalable design that can handle simulations of large crowds in a feasible way.

In crowd simulation the motion of crowds and other flock-like groups has been modeled as interacting particles that display different behaviors in 2D/3D scenes [14, 5]. Beyond physically based simulations, agent-based crowd models aim to capture the nature of a crowd as a collection of individuals, each of which can have their own goals, knowledge and behaviors [12]. However, when the number of agents or particles grows so does the workload generated by the crowd, making necessary the distribution of the crowd among different computers in order to keep an acceptable degree of interactivity. Typically, there are two different approaches for distributing a crowd simulation. One of them is based on the criterion of workload [16], so that different groups of agents are executed in different computers. The other approach is region-based [10], in such a way that the virtual world is split into regions (usually a 2D cell from a grid) and all the agents located at a given region are assigned to a given computer. Both approaches should guarantee the consistency of the simulation (for example, two different agents cannot be located at the same point in the virtual world). However, to the best of our knowledge none of the existing proposals can manage crowd simulations of more than tens of thousands of autonomous agents with different graphic qualities. Therefore, the scalability of crowd simulation is still an open issue.

In a previous work, we proposed a system architec-

*This work has been jointly supported by the Spanish MEC, the European Commission FEDER funds, and the University of Valencia under grants Consolider-Ingenio 2010 CSD2006-00046, TIN2006-15516-C04-04, and UV-BVSPIE-07-1788

0190-3918/08 $25.00 © 2008 IEEE
DOI 10.1109/ICPP.2008.20

ture for crowd simulation that can take advantage of the underlying distributed computer system [8]. That architecture consists of a distributed system where one of the computing nodes contains a centralized semantic database and the Action Server controlling the whole simulation. The rest of the computers host a set of agents implemented as threads of a single process. That architecture was shown efficiently enough to support simulations up to tens of thousands of complex agents with plausible graphic quality. However, although the centralized action server allows to easily provide consistency to the crowd simulation, it represents a system bottleneck and limits the scalability of the system. Effectively, when the number of agents increases the number of client processes can increase by adding new computers to the system, but the new client processes will share the computational power of the Action Server with the existing ones.

In this paper, we propose to enhance the proposed architecture through the efficient parallelization of the Action Server and the distribution of the semantic database. In this way, the system bottleneck is removed, and new Action Servers (hosted each one on a new computer) can be added as necessary. Thus, the system architecture can scale as necessary with the number of agents by simply adding new hardware. The evaluation results show that the proposed scheme can linearly increase the number of supported agents with the number of computers in the system, regardless of both the architecture and the computational power of the processors. These results validate the proposed scheme as a scalable architecture for crowd simulation.

The rest of the paper is organized as follows: Section 2 describes the existing approaches for improving the scalability of crowd simulation. Section 3 describes in detail the proposed architecture. Next, Section 4 shows the performance evaluation of the proposed architecture. Finally, Section 5 shows some concluding remarks and future work to be done.

2 Related Work

In a crowd simulation every element (typically an agent) can be managed as a single computational task, since the decisions are taken in a distributed flavor and no central control is needed. This is the reason for crowd simulation to be studied as molecular dynamics and physically based fluid simulations [14]. For example, the Lagrangian numerical methods used to predict gas-particle distributions [5] are closely related to the particle systems that underlie crowd simulation.

Beyond physically based simulations, agent-based crowd models aim to capture the nature of a crowd as a collection of individuals, each of which can have their own goals, knowledge and behaviors [12]. In this proposal, the neighboring search was very simple, so it was very time-consuming when the flock had a large number of boids. Recently, a new approach has been presented for PLAYSTATION3 which supports simulation and display of simple crowds up to 15000 individuals at 60 frames per second [11]. This work incorporates spatial hashing techniques to improve the neighboring search and it also distributes the load among PS3-Cell elements.

Parallel simulation based on Reynolds's boids has been also integrated in a PC-Cluster with MPI communication among the cluster processors [18]. This proposal uses different communication and partitioning strategies to finally produce small crowds simulations (512 boids), which are far from interactive.

Other proposals use GPUs to compute Lagrangian methods and other algebra operators that handles flow simulations with complex boundary conditions [3]. Even a GPU-based cluster has been proposed as the hardware platform for this kind of flow simulations [4].

Other graphic approaches are focused on providing efficient and autonomous behaviors (eg. pedestrians with navigation and/or social behaviors for urban/evacuation contexts), but they are not scalable at all [9, 13].

A different work focuses on the partitioning problem [16], but the simulations shown in this paper do not contain more than one thousand agents.

Finally, another proposal uses a multicomputer to simulate large evacuation scenarios involving up to ten thousand pedestrians[10]. Using eleven processors, this system is able to update the positions of the 10,000 simulated pedestrians nearly 50 times per second by distributing the virtual world in different regions. However, the mechanism used for managing the agents located near the borders of different regions adds a huge communication overhead. This feature seriously limits the scalability when the number of adjacent regions grows. Indeed, the evaluation shown in that work is exclusively performed on a one-dimensional grid.

Given these approaches, to the best of our knowledge there is no proposal that can manage crowd simulations of more than tens of thousands of autonomous agents with different graphic qualities. Therefore, the scalability of crowd simulation is still an open issue.

3 A Scalable Software Architecture

In a previous work, we proposed a system architecture for crowd simulation that can take advantage of the underlying distributed computer system [8]. As

illustrated in Figure 1, that software architecture is mainly composed by two elements: the action server (AS) and the client processes (CP). The AS is devoted to execute the crowd actions, while a CP handles a subset of the existing agents. Agents are implemented as threads of a single process for reducing the communication cost. Each thread manages the perception of the environment and the reasoning about the next action. Since reasoning formalisms can involve a high computational cost, each client process is hosted on a different computer, in such a way that the system can have a different number of client processes, depending on the number of agents in the system. In this way, this organization allows to take advantage of the underlying distributed hardware. This scheme allows to properly simulate up to tens of thousands of autonomous agents at interactive rates.

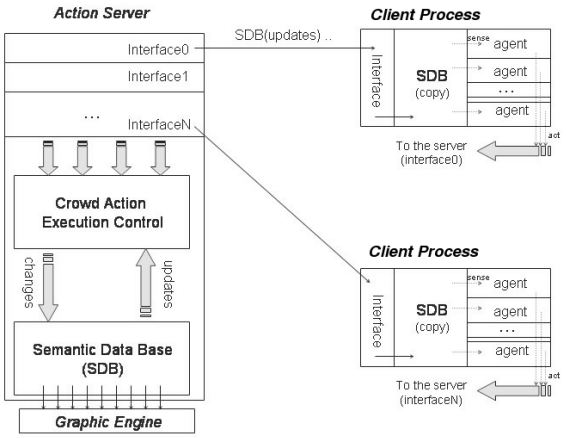

Figure 1. The previous software architecture

Nevertheless, the scalability of that architecture is limited by the centralized Action Server. Although it efficiently provides consistency to the simulation, it represents the system bottleneck when the number of agents and client processes significantly increases. Therefore, in order to improve the scalability of the proposed architecture is necessary to parallelize the Action Server so that it can be distributed among different computers. The rest of this section describes the parallelization of the Action Server and the distribution of the semantic database.

The previous Action Server has been divided into a set of processes so that each one can be executed in parallel in a different computer. For the sake of simplicity, each of these processes will be denoted as an Action Server while the whole set of processes will

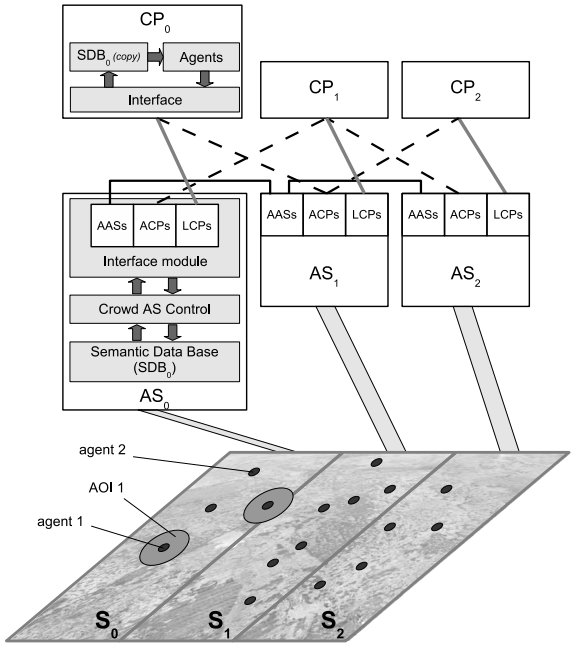

Figure 2. General scheme of the proposed architecture

be denoted as the Parallel Action Server.

In order to take advantage from the underlying system architecture, the distribution is performed at two levels. First, the virtual world is partitioned into a 2D grid, and each region of the grid is assigned to an AS process before the simulation starts. Figure 2 shows an example of the proposed architecture, and how this partitioning is performed. In this figure, the whole space is partitioned into three subregions, and each one is assigned to one AS. Once a region is assigned to a given AS, that server is responsible for checking the actions (eg. collision detection) of the agents located at that region. Once the partition has been initialized, the crowd must be also partitioned and distributed among the CPs associated to the corresponding servers. Each AS process hosts a copy of the Semantic Database. However, each AS exclusively manages the portion of the database representing the agents in its region.

In order to guarantee the action consistency near the border of the different regions, the ASs can collect information about the surrounding regions by querying the servers managing the adjacent regions. Additionally, the associated CPs are notified about the changes produced by the agents located near the adjacent regions by the ASs managing these regions.

Effectively, for each action requested by an agent a collision test is performed in the corresponding AS.

This test is computed based on the Area of Interest (AOI) [15] of the agent, using spatial hashing techniques for efficiency purposes (see [8] for details). If the AOI of the considered agent does not intersects with the region border, the corresponding AS updates the semantic database (SDB) with the new location and notifies all the local CPs about that change. The reason for broadcasting each answer to all the local CPs is that the neighborhood in the virtual world is completely independent from the assignment of the agents to each CPs. Therefore, it can happens that two agents are located very closely in the virtual world but they are managed by different CPs. That is, all the CPs should be notified about the movements of all the local agents. If, on the contrary, the AOI of the considered agent intersects with the region border, then the adjacent servers are queried. Only if all the servers answer positively the requested action is allowed, and the the semantic database (SDB) is updated. In this case the queried adjacent servers are also notified about the change, in order to guarantee the consistency among all the SDB copies.

Figure 2 shows an example of these neighborhood relationships and communications. Since in this case the region managed by the server AS_1 has two adjacent regions, the client process CP_1 should communicate with its action server AS_1 as well as AS_0 and AS_2. However, the other two regions only have a single adjacent region. Therefore, the client process CP_0 communicates with its Action Server AS_0 and with its adjacent server AS_1. In the same way, the client process CP_2 communicates with its Action Server AS_2 and with its adjacent server AS_1. Also, the Adjacent AS modules AS_0 and AS_2 are exclusively connected to AS_1. The reason is that in that example the region managed by the Action Server AS_1 has two adjacent regions, while the other two regions only have one adjacent region. Therefore, the Adjacent AS modules AS_0 and AS_2 are exclusively connected to AS_1.

We have implemented the proposed architecture using wandering agents. This type of agents is the most adequate one for testing the scalability of the system, since they do not need dynamic partitioning. Thus, the partitioning technique does not have any effects on the system scalability.

3.1 Internal Structure of Each Action Server

Each process the Parallel Action Server can be viewed as a partial world manager, since it controls and properly modifies the information in a region of the whole simulation space. Thus, it can be considered as the system core. Each AS process contains three basic elements: the Interface module, the Crowd AS Control module and the Semantic Data Base (SDB). Figure 3 illustrates a detailed scheme of an AS.

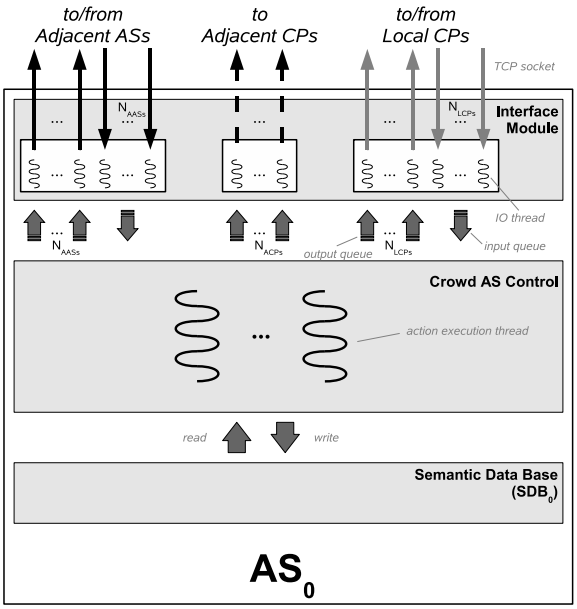

Figure 3. Internal structure of an Action Server

The main module is the Crowd AS Control module, which is responsible for executing the crowd actions. This module contains a configurable number of threads for executing actions (action execution threads in Figure 3). For an action execution thread (AE thread), all messages sent to or received from other ASs and CPs are exchanged asynchronously (the details are hidden by the Interface module, see below). This means that the AE threads only may have to wait when accessing shared data structures such as the semantic database. Thus, a single AE thread may be appropriate when an AS runs on a computer with a single core without hyperthreading support. However, experimental tests have shown that having more AE threads than cores (even many more) does not significantly affect execution times, because these extra AE threads simply stay blocked waiting for actions requests to arrive. This multithreaded scheme allows each AS to take advantage of several cores.

Most actions are executed from start to end by the AE thread that extracts the request from the corresponding input queue (see below). The exception are those actions that occur in the border of the region (border actions) and thus require confirmation from

all of the adjacent servers. For these actions, a two-phase algorithm is used. In the first phase, an AE thread checks that the action is feasible locally, blocks the position that the agent would take if all adjacent servers agree and marks the action as pending. Then, a request is sent (by the same AE thread) to all the adjacent ASs, and the thread looks for another action to process. In the second phase, each time a response from an adjacent AS arrives, an AE thread gets the response (which is dealt with as any other action request) and the answer is annotated in the action object. As soon as a negative answer is received, the action is considered unfeasible and this result is immediately sent to all local CPs. If no negative answer is received, when the last positive one is received, the response is sent to all local and adjacent CPs and ASs in order to maintain consistency among all the SDB copies.

The Interface module hides all the details of the message exchanges. This module provides the Crowd AS Control module with the abstraction of asynchronous messages. For receiving messages, AE threads just get one from the input queues (arrows pointing to the Crowd AS Control Module). Of course, the AE threads may have to wait if the corresponding queue is empty. Two separate input queues exist, one for messages coming from local CPs (action requests) and the other one for messages coming from adjacent ASs (responses to requests issued because of local border actions or requests for remote border actions from adjacent ASs). Having two separate input queues is an efficient way of giving a higher priority to messages from adjacent ASs. The reason for improving the priority of these messages is that the border actions are the ones whose processing takes longer, and we should reduce as much as possible their response time to provide realistic interactive effects.

In order to process messages as soon as they arrive, the Interface module contains one IO thread dedicated to getting incoming messages from each TCP socket. There are no input threads associated with sockets connecting one AS to their adjacent CPs, because CPs only send messages to their local AS. In the same way, there is one IO thread and one output queue per TCP socket, so that messages are sent as soon as the corresponding TCP socket is ready for writing.

3.2 Client Processes

Client processes have been modified with respect to the CPs described in [8]. In this new scheme, when an agent enters a different region, the corresponding CP thread is migrated to another CP. This procedure is performed as follows: the AS controlling that agent detects that it is entering into a new region and it requests to the corresponding CP to terminate that thread. Next, it notifies the AS controlling the destination region the entering of that agent, and in turn that AS selects a CP to host that agent. Once a new CP is selected, it is requested to create a new agent thread. Since no new connections are established, the migration process does not add a significant overhead.

4 Performance Evaluation

This Section shows the performance evaluation of the architecture described in the previous section. We have performed different measurements on different real systems using this architecture. Like other distributed systems, the most important performance measurements in DVE systems are latency and throughput [2]. Since we are focusing on the system scalability, we have performed simulations with different number of agents and we have measured the response time provided to the agents. In this way, we can study the maximum number of agents that the system can support while providing a response time below a given threshold value. In order to define an acceptable behavior for the system, we have considered 250 ms. as the threshold value, since it is considered as the limit for providing realistic effects to users in DVEs [6].

We have performed crowd simulations with wandering agents because all their actions should be verified by an AS. Each simulation consists of the crowd moving within the virtual world following k-length random paths. Since to our knowledge this is the first proposal of distributed action servers and semantic databases for crowd simulations, we cannot use our previously proposed architecture neither other proposed architecture for comparison purposes. Nevertheless, in order to obtain reproducible and comparable results, we saved the paths followed by the agents in the first execution of each configuration and used the same paths for all the executions tested with the same number of avatars. For all the populations tested, the average response time has become stable within the first minute of execution time. Therefore, we have used one minute simulations for all the configurations tested.

In order to ensure that the scalability of the proposed architecture does not depend on the underlying hardware, we have performed experiments using two different computer platforms. One of the platforms has been a cluster of computers based on AMD Opteron (2 x 1.56 Ghz processors) with 3.84GB of RAM, executing Linux 2.6.9-1 operating system. The interconnection network in the cluster was a Gigabit Ethernet network. The other platform was a set (labora-

tory) of interconnected PCs, each one with a 2.5 GHz Celeron processor and 1GB of RAM. The interconnection network in this case was a Fast Ethernet switched network. We simulated an Action Server distributed among different numbers of computers. When using the cluster, we implemented a Parallel Action Server distributed among one, two and four computers. We have denoted these computers as servers. For that case we used up to twelve cluster nodes (four of them for hosting the AS and eight of them for hosting eight clients). Using this platform, we simulated up to twenty three thousands agents. When using the PCs in the lab, we implemented a Parallel Action Server distributed among one, two, four, and eight computers (servers). For that case, we used up to sixteen PCs (eight of them for the eight servers, and eight of them for hosting eight clients). Using this platform, we simulated up to eleven thousand agents.

Figure 4 shows the response times provided by the proposed architecture when distributing the Action Server among four different nodes of the cluster platform. On the X-axis, this figure shows the number of thousands of agents in the system for a configuration of four servers and eight client computers. The Y-axis shows both the average and the maximum response times provided to the agents. Each point in this Figure has been computed as the average value of thirty different simulations. The label "AVG. RT" corresponds to the plot showing the average response time, while the label "MAX. RT." corresponds to the maximum response time provided to an agent. Figure 4 shows that the response time provided to agents linearly increases with the number of agents in the system. That is, the design of the proposed architecture allows to host new agents with linear cost. As an average, the system can support more than 20,000 agents while providing an average response time not greater than 250 milliseconds (the threshold value for providing realistic effects). Regarding the maximum response time, the number of avatars supported is around 19,000 agents.

In order to test if the behavior of the proposed architecture depends on the underlying hardware platform or the number of servers, we have also implemented the same crowds in the set of interconnected PCs. In this case, we have implemented the system with one, two, four and also eight servers. Figure 5 shows the results for that platform when using eight servers. Figure 5 shows that the behavior of the proposed architecture is similar to the one shown in Figure 4. That is, the response time provided to agents linearly increases with the number of agents. However, since the computational power of the computers is lower in this platform than in the cluster platform, the threshold value in the average response time (250 milliseconds) is reached when the system supports only around 10,600 agents, instead of 20,000 agents.

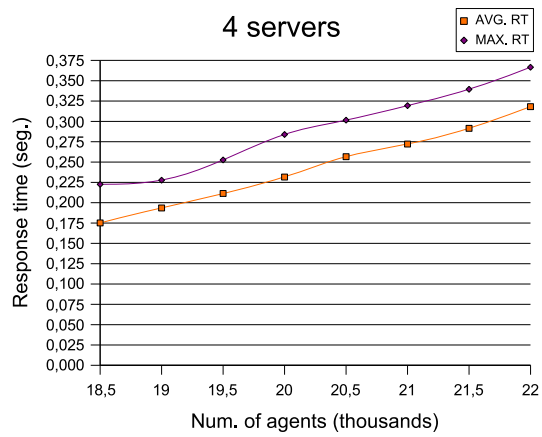

Figure 4. Response times for the cluster platform

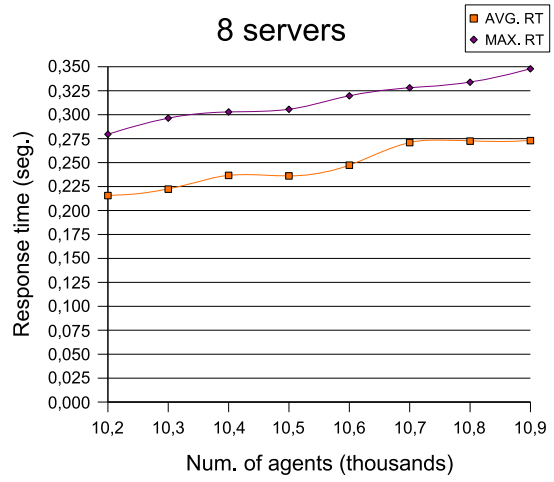

Figure 5. Response times for the interconnected PCs

Although these results show that the proposed scheme efficiently supports the workload generated by the agents regardless of the underlying hardware platform, the scalability of the proposed architecture has still to be proven. Therefore, we have measured in both platforms the number of agents supported with different number of computers while still providing an average response time below the threshold value of 250 milliseconds. Figure 6 shows the scalability of the proposed scheme when implemented on the cluster plat-

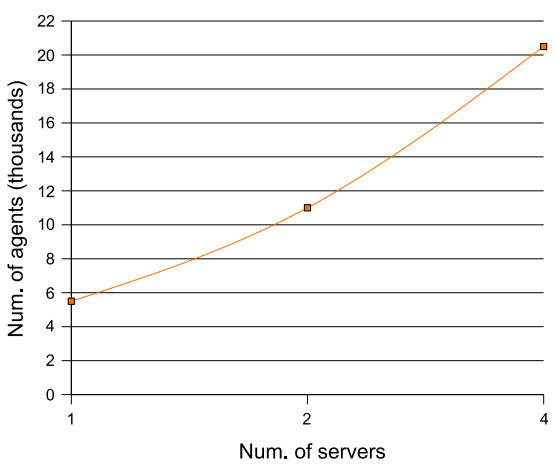

Figure 6. System throughput for different configurations of the cluster platform

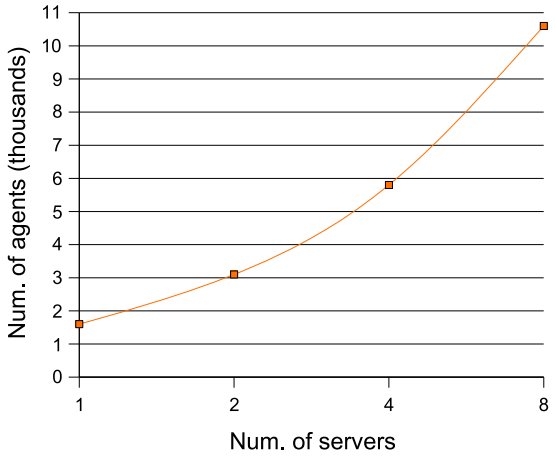

Figure 7. System throughput for different configurations of the interconnected PCs

form. This figure shows on the X-axis the number of servers for each configuration considered, and it shows on the Y-axis the number of thousands of agents that each configuration can support.

Figure 6 shows that when the AS is implemented on a single computer (that is, it is centralized) then the system properly supports more than five thousand agents. When the AS is distributed between two nodes, then the system can support more than ten thousand agents, and when distributing the AS among four servers then system correctly supports more than twenty thousand agents. That is, the number of supported agents linearly increases with the number of computers used for hosting the AS. Therefore, these results prove that the AS was the system bottleneck and that the proposed scheme properly scales with the number of servers.

In order to prove that the scalability of the proposed scheme neither depends on the hardware platform, Figure 7 shows the scalability results for the set of interconnected PCs. In this case, we have distributed the Parallel AS among one, two, four and eight servers.

Figure 7 shows that the proposed architecture allows to properly scale the number of supported agents with the number of servers also for this hardware platform. Effectively, the system can support around one thousand and five hundred agents when the AS is implemented on a single server, and the number of supported agents is directly related with the number of servers.

5 Conclusions and Future Work

In this paper, we have proposed a system architecture for crowd simulation that properly scales up with the number of agents in the system. This proposal consists of a set of interconnected computers in order to improve scalability and flexibility, as well as a distributed software architecture based on parallel servers. The evaluation results show that the proposed architecture is able to fully exploit the underlying hardware platform, regardless of both the number and the kind of computers that form the system. The distribution of the AS among parallel servers efficiently provides consistency and properly scales up with the number of servers. Therefore, this system architecture provides the scalability required for large-scale crowd simulations.

As a future work, we plan to study different partitioning methods for dynamically assigning the agents to the different servers in the system. These methods can help to achieve dynamic reconfigurations when the movement pattern of the agents is not uniform. Also, we plan to improve the proposed architecture to take full advantage of multi-core nodes.

References

[1] D. Diller, W. Ferguson, W. Leung, A. Benyo, and D. Foley. Behavior modelling in comercial games. In *BRIMS '04: Proceedings of the 2004 Behavior Re-*

presentation in Modelling and Simulation Conference, 2004.

[2] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks: An Engineering Approach*. IEEE Computer Society Press, 1997.

[3] U. Erra, R. De Chiara, V. Scarano, and M. Tatafiore. Massive simulation using gpu of a distributed behavioral model of a flock with obstacle avoidance;. In *VMV '04: 9th International Workshop on Vision, Modeling, and Visualization*, 2004.

[4] Z. Fan, F. Qiu, A. Kaufman, and S. Yoakum-Stover. Gpu cluster for high performance computing. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 47, Washington, DC, USA, 2004. IEEE Computer Society.

[5] T. Frank, K. Bernert, and K. Pachler. Dynamic load balancing for lagrangian particle tracking algorithms on mimd cluster computers. In *PARCO'2001 - International Conference on Parallel Computing 2001.*, 2001.

[6] T. Henderson and S. Bhatti. Networked games: a qos-sensitive application for qos-insensitive users? In *Proceedings of the ACM SIGCOMM 2003*, pages 141–147. ACM Press / ACM SIGCOMM, 2003.

[7] P. A. Kruszewski. A game-based cots system for simulating intelligent 3d agents. In *BRIMS '05: Proceedings of the 2005 Behavior Representation in Modelling and Simulation Conference*, 2005.

[8] M. Lozano, P. Morillo, J. M. Orduña, and V. Cavero. On the design of an efficient architercture for supporting large crowds of autonomous agents. In *Proceedings of IEEE 21th. International Conference on Advanced Information Networking and Applications (AINA'07)*, pages 716–723, May 2007.

[9] N. Pelechano, J. M. Allbeck, and N. I. Badler. Controlling individual agents in high-density crowd simulation. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 99–108, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[10] M. J. Quinn, R. A. Metoyer, and K. Hunter-Zaworski. Parallel implementation of the social forces model. In *In Proceedings of the Second International Conference in Pedestrian and Evacuation Dynamics*, pages 63–74, 2003.

[11] C. Reynolds. Big fast crowds on ps3. In *sandbox '06: Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, pages 113–121, New York, NY, USA, 2006. ACM.

[12] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34, New York, NY, USA, 1987. ACM.

[13] W. Shao and D. Terzopoulos. Autonomous pedestrians. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 19–28, New York, NY, USA, 2005. ACM.

[14] K. Sims. Particle animation and rendering using data parallel computation. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 405–413, New York, NY, USA, 1990. ACM.

[15] S. Singhal and M. Zyda. *Networked Virtual Environments*. ACM Press, 1999.

[16] A. Steed and R. Abou-Haidar. Partitioning crowded virtual environments. In *VRST '03: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 7–14, New York, NY, USA, 2003. ACM.

[17] M. Sung, M. Gleicher, and S. Chenney. Scalable behaviors for crowd simulations. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 519–528. ACM Press, 2004.

[18] B. Zhou and S. Zhou. Parallel simulation of group behaviors. In *WSC '04: Proceedings of the 36th conference on Winter simulation*, pages 364–370. Winter Simulation Conference, 2004.