A Specific System Design for Crowd Simulation

G. Viguera∗, M. Lozano∗, C. Pérez∗, J. M. Orduña∗

∗ Departamento de Informática, Universidad de Valencia, Spain

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. The sum of these two requirements results in a computational cost that exponentially increases with the number of agents in the system. Thus, crowd simulation requires a scalable design that can handle large simulations in a feasible way. In this paper, we propose a system design that can manage the simulation of large crowds of autonomous agents at interactive rates.

KEYWORDS: Crowd simulation; Parallel simulation

1 Introduction
In recent years, crowd simulation has become an essential tool for many virtual environment applications in education, training, and entertainment. 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 exponentially increases with the numbers of agents in the system, requiring a scalable design and the distribution of the crowd among different computers in order to keep an acceptable degree of interactivity. Some approaches have been proposed to address the problem of distributed crowd simulations [Rey06, ZZ04, QMHZ03], but this approaches can manage up to tens of thousands of autonomous agents. Therefore, the scalability of crowd simulation is still an open issue.

In a previous work, we proposed a system architecture for crowd simulation that can take advantage of the underlying distributed computer system [LMOC07]. In this paper, we propose to enhance the proposed architecture through the efficient parallelization of the Action Server. In this way, the system bottleneck is removed, and new Action Servers (hosted

1E-mail: {Guillermo.Viguera, Juan.Orduna}@uv.es
2This 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-BVSPFE-07-1788.
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 rest of the paper is organized as follows: Section 2 describes the proposed architecture. Finally, Section 3 shows the performance evaluation of the proposed architecture.

2 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 [LMOC07]. As illustrated in Figure 1a, 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.

Nevertheless, the scalability of that architecture is limited by the centralized Action Server that represents the system bottleneck when the number of agents and client processes significantly increases. Therefore, 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 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 1b shows an example of the proposed architecture, and how this partitioning is performed. In this
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 (e.g., 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 exclusively manages the agents that fall within 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 in figure 1b) of the agent. If the AOI of the considered agent does not intersects with the region border (agent1 in figure 1b), the corresponding AS (AS0 in figure 1b) updates the location of the agent and notifies the change to the local CP (CP0 in figure 1b) that sent the action request. If, on the contrary, the AOI of the considered agent intersects with the region border (agent2 in figure 1b), then the adjacent servers are queried (AS1 for the example in figure 1b). Only if all the servers answer positively the requested action is allowed. In this case the queried adjacent servers are also notified about the change, in order to guarantee the consistency of the simulation.

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 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. 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 (system throughput) while providing a response time below 250 ms., since it is considered as the limit for providing realistic effects to users in DVEs.

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 (laboratory) 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 (servers) and two cluster nodes (clients) per server (up to twelve nodes). 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) and one computer (client) per server (up to sixteen compu-
Using this platform, we simulated up to eleven thousand agents.

Figure 2 shows the scalability of the proposed scheme when implemented on both the cluster and the interconnected PCs platforms. In this figure, both plots, show on the X-axis the number of servers for each configuration considered, and on the Y-axis the number of thousands of agents that each configuration can support.

Figure 2.a shows that when the AS is implemented on a single computer 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. In order to prove that the scalability of the proposed scheme neither depends on the hardware platform, Figure 2.b shows the scalability results for the set of interconnected PCs. In this case, Figure 2.b shows that the proposed architecture allows to properly scale the number of supported agents with the number of servers also for this hardware platform.

References


