A New System Architecture for Crowd Simulation

M. Lozano [a], P. Morillo [a], J. M. Orduña [*],V. Cavero [a], G. Vigueras [a],

[a]*Departamento de Informática. Universidad de Valencia. SPAIN.*
Av. Vicent Andrés Estellés, s/n. 46100 - Burjassot (Valencia). SPAIN.

Abstract

Crowd simulation requires both rendering visually plausible images and managing the behavior of autonomous agents. Therefore, these applications need an efficient design that allow them to simultaneously tackle these two requirements. Although several proposals have focused on the software architectures for these systems, no proposals have focused on the computer systems supporting them.

In this paper, we analyze the computer architectures used in the literature to support virtual environments. Also, we propose a distributed computer architecture efficient enough to support simulations of thousand of autonomous agents. This proposal consists of a cluster of interconnected computers in order to improve flexibility and robustness, as well as a hierarchical software architecture that efficiently provides consistency. Performance evaluation results show that the trade-off between flexibility and consistency allows to efficiently manage thousands of autonomous agents. Therefore, this network-based system architecture can provide the required scalability for large-scale crowd simulations.

Key words: Crowd simulation, Distributed Virtual Environments, computer architecture

PACS: 89.20.Ff Computer science and technology

1 Introduction

Crowd simulation has become an essential tool for many virtual environment applications. Extensive use of virtual crowds has been made in many commercial movies. Also, high quality crowd simulations are crucial for many virtual environment applications in education, training, and entertainment [3,15,33].

Crowd simulations can be considered as virtual environment applications with two different goals. On the one hand, crowd simulations must focus on rendering visually plausible images of the environment, requiring a high computational cost. On the other hand, complex agents must have autonomous behaviors, greatly increasing the computational cost as well. Thus, some proposals tackle crowd simulations as a particle system with different levels of details (eg:*impostors*) in order to reduce the computational cost [4,34]. Although these proposals can handle crowd dynamics and display populated interactive scenes (10000 virtual humans), they are not able to produce complex autonomous behaviors for their actors. On the contrary, several proposals have been made to provide efficient and autonomous behaviors to crowd si-

* Departamento de Informática. Universidad de Valencia.

Av. Vicent Andrés Estellés, s/n. 46100 - Burjassot (Valencia). SPAIN.

Tel. +34 96 3544489, Fax +34 96 3544768, Email: Juan.Orduna@uv.es

Email addresses: `Miguel.Lozano@uv.es` (M. Lozano), `Pedro.Morillo@uv.es` (P. Morillo), `Vicente.Cavero@uv.es` (V. Cavero), `Guillermo.Vigueras@uv.es` (G. Vigueras).

mulations [28,24,5,26,19,14]. However, they are based on a centralized system architecture, and they can only control a few hundreds of autonomous agents with different skills (pedestrians with navigation and/or social behaviors for urban/evacuation contexts). Tacking into account that pedestrians represent the slowest human actors (in front of other kind of actors like drivers in cars, for example) these results show that scalability has still to be solved in crowd simulation.

Although some scalable, complex multi-agent systems have been proposed [35], these proposals are exclusively focused on the software architecture, forgetting the underlying computer architecture (the actual implementation of the computer and the application executing on it [13,29]). As a result, important features like inter-process communications, workload balancing or network latencies are not taken into account, seriously limiting the performance of this applications.

In a previous work, we proposed a system architecture for crowd simulation that can take advantage of the underlying computer system [16]. In this paper we present, in an extended manner, the computer architectures used in the literature to support virtual environments and we propose an efficient system architecture, capable of simulating crowds of thousands of autonomous agents. In order to manage the trade off between scalability, rich behaviors, and computational cost required by crowd simulation, this system architecture is based on a hybrid computational model. It consists of a Distributed Virtual Environment (DVE) [30] based on a networked-server architecture. On top of this hardware architecture, we propose a hierarchical software architecture, in order to efficiently support consistency and autonomous behaviors: one of the servers is used to host a centralized semantic data-base, and agents are uniformly distributed among the rest of the servers in the system (denoted as *replicas*). This scheme allows to scale up the number of servers with the number of agents in the system, while the centralized database easily provides consistency. Additionally, we present an extended performance evaluation of the proposed architecture. The results obtained in the performance evaluation show that this architecture can efficiently manage up to thousands of autonomous agents.

The rest of the paper is organized as follows: Section 2 analyzes the existing computer architectures proposed in the literature for supporting Distributed Virtual Environments (DVEs). As a result of such analysis, Section 3 describes the proposed architecture for crowd simulation. Next, Section 4 shows the performance evaluation of the proposed system architecture. Finally, Section 5 shows some conclusion remarks and future work to be done.

2 Computer architectures for DVEs

Different computer architectures have been proposed in order to efficiently support DVEs: centralized-server architectures [36,25], networked-server architectures [17,23] and peer-to-peer architectures [20,18]. Figure 1 shows an example of a centralized-server architecture. In this example, the virtual world is two-dimensional and avatars are represented as dots. In DVEs based on a centralized-server architecture, there is a single server and all the client computers are connected to this server. The server is in charge of managing the entire virtual world. As a result, it becomes a potential bottleneck as the number of avatars in the system increases. In fact, the DVEs based on this architectures are the ones that support the lowest number of clients.

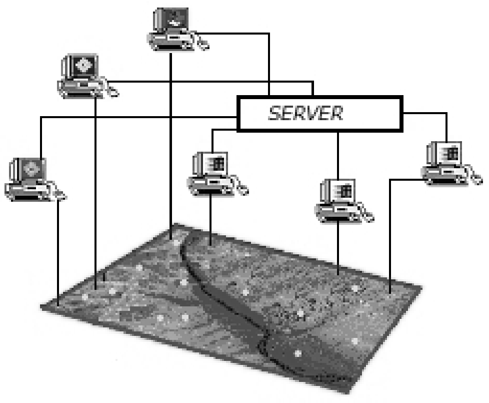

Fig. 1. Example of a client-server architecture

Figure 2 shows an example of a networked-server architecture. In this scheme there are several servers and each client computer is exclusively connected to one of these servers. This scheme is more distributed than the client-server scheme. Since there are several servers, it considerably improves the scalability, flexibility and robustness regarding to the client-server scheme. However, it requires a load balancing scheme that assigns clients to servers in an efficient way.

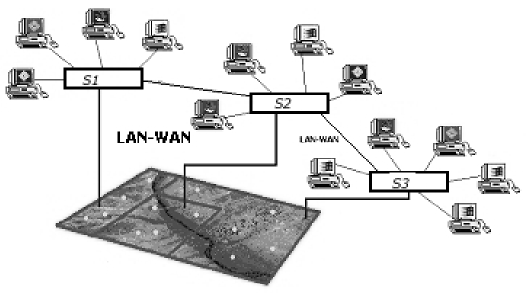

Fig. 2. Example of a networked-server architecture

Figure 3 shows an example of a peer-to-peer architecture. In this scheme, each client computer is also a server. This scheme provides the highest level of load distribution. Although the first DVEs were based on centralized architectures, during the last few years architectures based on networked servers have been the major de-facto standard for DVE systems [17,11]. However, each new avatar in a DVE system represents an increase not only in the computational requirements of the application but also in the amount of network traffic [23,22]. Due to this increase, networked-server architectures seem not to properly scale with the number of clients, particularly for the case of MMOGs [1], due to the high degree of interactivity shown by these applications. As a result, Peer-to-Peer architectures have been proposed for massively multi-player online games[20,18,10].

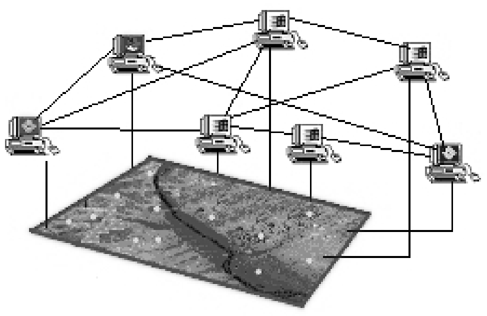

Fig. 3. Example of a peer-to-peer architecture

Nevertheless, P2P architectures must still efficiently solve the *awareness* problem. This problem consists of ensuring that each avatar is aware of all the avatars in its neighborhood [32]. Providing awareness to all the avatars is a necessary condition to provide time-space consistency (as defined in [37,9,27,31]). Awareness is crucial for DVEs, since otherwise abnormal situations could arise. For example, a game user provided with a non-coherent view of the virtual world could be shooting something that he can see although it is not actually there. Also, it could happen that an avatar not provided with a coherent view is killed by another avatar that it cannot see. In networked-server architectures, the awareness problem is easily solved by the existing servers, since they periodically synchronize their state and therefore they know the location of all avatars during all the time. Each avatar reports about its changes (by sending a message) to the server where it is assigned to, and the server can easily decide which avatars should be the destinations of that message (by using a criterion of distance). There is no need for a method to determine the neighborhood of avatars, since servers know that neighborhood every instant.

3 A System Architecture for Crowd Simulation

From the discussion above it seems that the more physical servers the DVE relies on, the more scalable and flexible it is. On the contrary, features like the awareness and/or consistency are more difficult to be provided as the underlying architecture is more distributed (peer-to-peer architecture). Therefore, we propose a networked-server scheme as the computer architecture for crowd simulation. On the one hand, this distributed scheme allows to improve scalability, flexibility and robustness when compared to centralized (client-server) architectures. On the other hand, the small number of servers in networked-server architectures makes it easy to provide awareness (and therefore time-space consistency) to the avatars moving in the virtual world. However, crowd simulations are not user-driven applications, but computer-driven applications. This means that there are not user-driven client computers. As a result, the computer is exclusively composed of the interconnected servers, and they are used to manage the crowd simulation. Figure 4 shows an example of the proposed computer architecture with three servers.

On top of this networked-server architecture, a software architecture must be designed to manage a crowd of autonomous agents. In order to easily maintain the coherence of the virtual world, a centralized semantic information system is needed. In this sense, it seems very difficult to maintain the coherence of

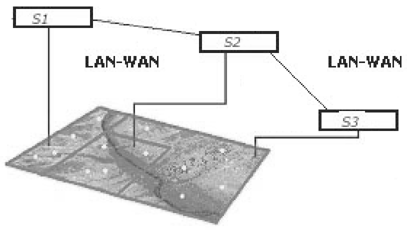

Fig. 4. An example of the proposed computer architecture for crowd simulation

the semantic information system if it follows a peer-to-peer scheme, where hundred or even thousand of computers support each one a small number of actors and a copy of the semantic database. Therefore, on top of the computer architecture shown in Figure 4 we propose the software architecture shown in Figure 5. This architecture has been designed to distribute the agents of the crowd among the different servers of a networked -server architecture.

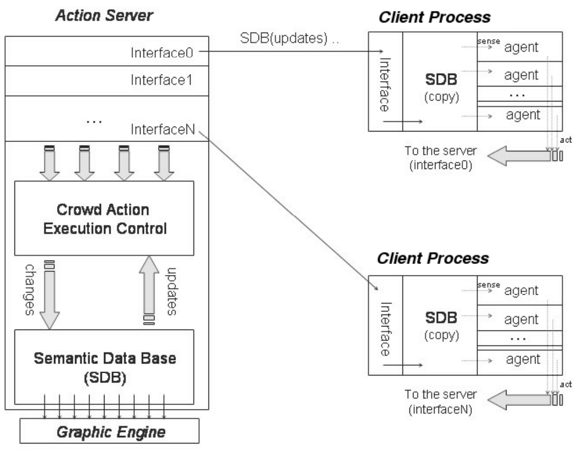

Fig. 5. The proposed software architecture

This hierarchical software architecture is composed of two kind of elements: the *Action Server (AS)* and the *Client Processes (CP)*. The AS is unique, but the system can have as many client processes as necessary, in order to properly scale with the number of agents to be simulated. In his turn, each CP manages a group of autonomous agents. In order to take advantage from the underlying computer architecture, the most suitable distribution for this software architecture consists of allocating the AS in a single server, and uniformly distributing the CPs among the rest of the networked servers. In this way, the scalability and flexibility of the networked-server scheme can be used to add more CPs (and servers) as the number of agents increases. Since each client process can manage a variable number of autonomous agents, the servers usually host one CP, although they can host several ones. On the other hand, the action server is composed of two different modules, the Semantic Data Base (SDB) and the *Action Execution Module (AEM).* The action server is hosted on another networked server.

3.1 The Action Server

The Action Server corresponds to the *action engine* [6,7], and it can be viewed as the world manager, since it controls and properly modifies all the information the crowd can perceive. The Action Server is fully dedicated to verify and execute the actions required by the agents, since they are the main source of changes in the virtual environment. For scalability purposes, the AS must be placed on the computer with the highest computational power. This computer should exclusively be used for this purpose. Since the AS is unique, consistency is easily provided. In this context, consistency involves the information that the agents should know to animate consistent behaviors. Additionally, another important parameter for interactive crowd simulation is the *server main frequency.* This parameter represents how fast the world can change. Ideally, in a fully reactive system all the agents send their action requests to the server, which processes them in a single cycle. In order to provide realistic effects, the

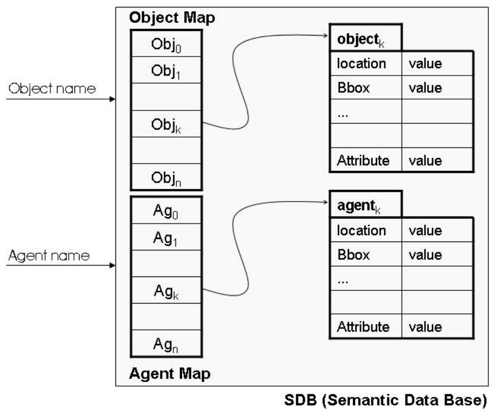

Fig. 6. The Semantic Data Base

server cycle must not be greater than the threshold used to provide quality of service to users in DVEs [21,12]. Therefore, we have set the maximum server cycle to 250 ms..

The AS consists of two modules: the Semantic Data Base and the Action Execution Module. The SDB represents the global knowledge about the interactive world that the agents should be able to manage, and it contains the necessary functionalities to handle interactions between agents and objects. In our case, we manage a simple map for objects and agents which let us to efficiently control a set of (attribute, value) pairs associated to each object/agent during the simulation. Since these attributes are centered into objects or agents, we use their names to index the correspondent map. The semantic information managed can be symbolic (eg: $object_i$ free true, $object_i$ on $object_k$, ...) and numeric (eg: $object_i$ position, $object_i$ bounding volume, ..), since it has been designed to be useful for different types of agents. Figure 6 illustrates this data base scheme.

We have decided to avoid complex spatial maps (such as quad/oct-trees) to control the SDB, since these structures can be too expensive to handle when

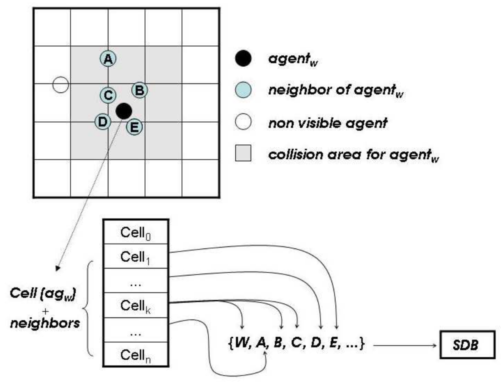

Fig. 7. A collision example.

the number of insertions and deletions grows (agents can be continually changing their location). Instead, we use a 2D Grid (Cellular Automata), which allows us to access to any object/agent efficiently and it is also useful for pathfinding behaviors, as we review below.

The AEM is devoted to guarantee the coherence of the virtual world, as it is responsible for the action checking and execution. For example, if an agent wants to change its location, it will try to perform a motor action, so a collision can occur. The agent should request the server to validate that movement by sending a message. Since the server knows the location of the agent, it accesses to its cell through a simple function (similar to the one used by hash tables), and perform an object-object collision test with the neighbors of $agent_w$ (see Figure 7). If no collision caused by that movement is detected, then the server should *update* the SDB and send an acknowledgment message to the agent.

As commented in the example, we reduce the visible area for each agent to face collision detection. Concretely, we use a Cellular Automata (CA) as an adequate formalism to handle motion behaviors for crowd simulation [2]. This formalism let us to manage collision detection and *pathfinding* behaviors in several situations (see Figure 8). As *pathfinding* behaviors clearly depends on the agent decision taking, it is reviewed in the next Section.

In order to manage the high number of changes produced, the AEM puts all the action effects in a vector *(vUpdates)* which reflects the local changes produced by each actuation (eg: an agent changes its position). Finally, when the server cycle has finished, this vector is sent to both the CPs and SDB, which will update their corresponding environmental states (see Figure 5).

Once the server cycle has been set (in our case to 250 milliseconds), the available time to process each agent action results from dividing the server cycle by the number of agents in the system. If the number of agents increases in such a way that the server is not able to process all the actions in a single cycle, then the pending actions are simply left to be processed in the next cycle. In the experiments shown in Section 4 we measure the action latency, so we can estimate the degree of reactivity achieved by this scheme.

The AEM manages the action's flow of the simulation. In order to allow the maximum flexibility, it can currently process 4 types of actions:

Motion actions : Location changes where collisions can occur, although agents were (potentially) able to navigate without colliding. If an agent wants to move to the location currently occupied by another object/agent, the environment should simply not allow it.

Motor actions : We use simple key-framing tables to animate the actors in walking, running, and other motions. Since no constraints are allowed, the value received is simply accepted in the SDB as an internal change in an agent attribute. We consider the agent motor system as the responsible to continually read its *animation state* from the SDB. The graphic engine, which contains all the *actor's skeletons* of the crowd, performs this task according to its frame rate .

Agents interactions : Corresponds to a normal agent-agent communication scheme, which can be obtained from the system through the server. Messages can be managed as other agent attributes, so the SDB will simply route them into the correspondent slots. This scheme has made us possible to investigate on *social* crowds in the future.

STRIPS actions : STRIP is the action language used by our planning agents. A STRIPS action scheme [8] can be represented through the *Preconditions*, *Add* and *Delete* lists associated to each agent action. Before executing an action (eg: pick up object), the AEM verifies its preconditions using the SDB maps (eg: is object-k free?). When a STRIPS action is accepted, the Add and Delete lists contain the new information the SDB needs to update its state.

3.2 The Client Processes

Each client process in a replica manages an independent group of autonomous agents (a subset of the crowd). This process has an *interface* for receiving and updating the information from the server, and a finite number of threads (each thread for an agent). Using this interface, a replica (a server hosting a CP) initially connects to the server hosting the Action Server and downloads a complete copy of the SDB. From that instant, agents can think locally and in parallel with the server, so they can asynchronously send their actions to the server, which will process them as efficiently as possible (since each agent is a process thread, it can separately access to the socket connected to the server). When a server cycle finishes (every 250 ms.), the accepted changes are submitted to all the replicas interfaces, that will update their SDB copies.

This multi-threading approach is independent of the agent architecture (the AI formalism driving the agent behavior), that is out of the scope of this paper. However, the proposed action scheme guarantees the awareness for all agents [32], since all the environmental changes are checked in a central server and then broadcasted to the agents. Although time-space inconsistencies can appear due to agent asynchronies and network latencies, all these inconsistencies are kept below the limit of the server period.

Unlike in the architecture shown in [16], in the system proposed in this paper we have added a pathfinding behavioral model for agents. Thus, we have evaluated the proposed architecture with both models, wandering agents and pathfinding agents. Each kind of agents generates a different motion pattern, that is, a different system workload.

Our pathfinding approach is based on cellular automata theory. The basic procedure in cellular automata based motion is to compute a set of rules associated to each cell to determine the next cell until the goal is reached. In our system, a CA is included as a part of the SDB, and each cell has precomputed the k-best paths of length l to achieve any exit cell.

To calculate all the paths (k paths per cell; cell = 1m side square), we are using a variation of the $A*$ algorithm which can be precomputed initially in an empty environment. The algorithm starts from each goal cell, and by inundation we select and store the k best paths that arrive to each cell. Parameters such as k and the cell size allow to reduce the memory required for managing large environments, avoiding memory problems. Furthermore, the calculation of complete paths is not interesting generally, as agents can only evaluate the

k-first steps before deciding its next cell. The algorithm we have used to pre-compute all the possible paths to reach the exit cells for a specific environment could be the following one:

```
VAR

Lab  /* Auxiliary list of cells */
Mop  /* Map of paths (MoP) for each cell */
goal /* exit cell */



begin
   /*Initialization*/
   Lab <- goal;
   // Stores the next cell with its distance, for a specific goal
   recNext(goal, goal, NULL, 0.0);

   while(Lab != NULL)                 // Lab is not empty
      cell_i = Lab.pop();
      // Computes distance from all the neighbor cell to the goal
      for (next_cell_i in neighbours(cell_i))
        if (next_cell_i is not an obstacle)
          d_next = distance(next_cell_i, goal);
          dthrogh_next = distance(cell_i, portal) +
                         distance(next_cell_i, cell_i);
          if(d_next > dthrough_next)
             recNext(goal, cell_i, next_cell_i, dthrough_next);
             Lab <- next_cell_i;
          // Creates the best path going backward from cell to a goal
          path_i = buildPathfromCell(cell_i, longPath);
          MoP <- addPath(path_i,dthrogh_next, MAX_PATHS = k);
end
```

As a result of this algorithm, each cell contains a map with the best k paths (MoP) to reach any goal (exit) cell. These paths will be used by the agents. However, along the simulation time these paths can be either empty (not used by any agent) or busy. Therefore, each agent should evaluate all the paths to obtain the best next cell for any situation (state of the simulation). Since the purpose of the application is the evacuation of the virtual world, it is necessary to balance the distance for each path to any goal cell with the waiting time associated to each one. This waiting time is composed of the total sum of cycles that agents are waiting in a path and also the path congestion percentage (the percentage of busy cells in a path). In order to perform this balancing, we have defined an evaluation function H that measures the quality of each of the paths p_i.

$$H(p_i) = \alpha * Dist(p_i, goal) + (1-\alpha) * (Wait(p_i) + Cong(p_i)) \quad (1)$$

where the function $Dist$ measures the distance for each path i to any goal cell, the function $Wait$ measures the total sum of cycles that agents are waiting in path i, the function $Cong$ measures the path congestion percentage, and α is a weighting parameter to balance both terms of the sum. Each agent should minimize the function $H(path_i)$ in order to find the best way to exit the virtual world.

As an example, Figure 8 shows two snapshots of the system running an evacuation simulation with 8000 agents in a square two-dimensional world corresponding to a urban environment. In this picture, the virtual world seen from above, the buildings are represented as black rectangles, and the avatars are represented as grey dots. The avatars must leave the virtual world through 8 exit cells that are located on the perimeter of the virtual world. These exit cells can be easily recognized by following the different crowd flows. Concretely, Figure 8 -a) shows the initial situation of the crowd, which is randomly distributed in the virtual world. Figure 8 -b) captures the crowd situation at the cycle 100. At this point it can be clearly seen different crowd flows and some congested areas (all agents try to follow the shortest non-busy paths).

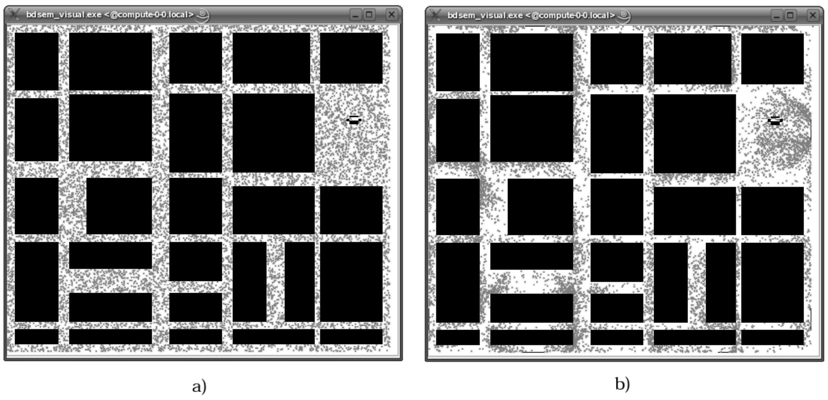
bdsem_visual.exe <@compute-0-0.local>
bdsem_visual.exe <@compute-0-0.local>
a)
b)

Fig. 8. Evacuation test with 8000 agents

On other hand, it is important to ensure that the system can render visually plausible images of the virtual world. In this sense, Figures 9 and 10 show two different three dimensional views of a urban simulation environment inhabited by 8000 agents. Figure 9 shows a general view of the city when the simulation starts, while figure 10 shows a more detailed view of an avenue. These figures show that the graphical quality provided during the simulation is acceptable enough.

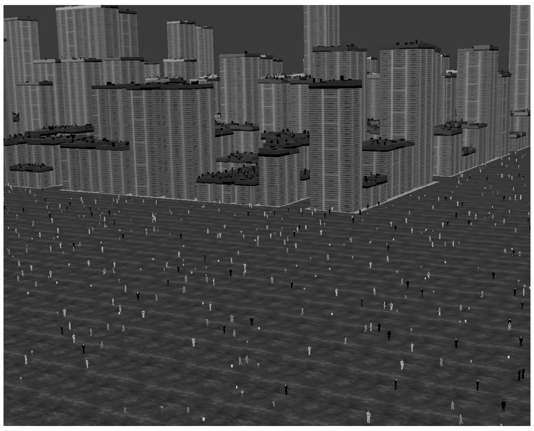

Fig. 9. Urban environment with 8000 agents

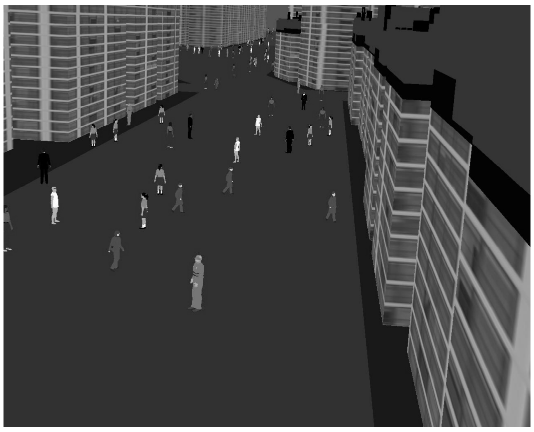

Fig. 10. Detailed view of the urban environment (8000 agents)

4 Performance Evaluation

This Section shows the performance evaluation of the architecture described in the previous Section. We have performed different measurements on a real system with this architecture, following different behavioral models. Concretely, we have performed simulations with wandering agents, as well as simulations with complex avatar behaviors (pathfinding). As cited in the previous Section, the AS cycle (the maximum period of time an interactive actor can wait for it action response) has been set to 250ms., and every 2.5 seg. statistics are computed, resulting in 30 different samples. Each point of the plot and each value of the tables in this Section has been computed as the average value of the 30 samples.

We have used wandering agents for the first phase of the performance evaluation wandering agents, since this type of agents is the one that generate the highest workload to the AS (since they simply move, they require the server to validate their movements in each server cycle). The computer platform 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. Depending on the test, we have used from one to five nodes of the cluster.

Like in other distributed systems, the most important performance measurements in DVE systems are latency and throughput. First, we have focused on system throughput (the maximum number of agents that the system can efficiently support), that is limited in our architecture by the AS throughput. Concretely, we have measured the AS throughput when it is fully dedicated to collision detection tasks. The rationale of this test is to evaluate the number of actions that the server is able to carry out in a single cycle, since this could be a plausible bottleneck.

When an action is requested by an agent, the server must access to its cell and check out the resulting distance to all the agents which are sharing the same cell, as Figure 7 shows . If no collisions are produced, then the same process is performed in the 8 neighbor cells. A positive acknowledgment can be sent back to the requesting agent only if no collisions are produced in the nine cells checked.

Figure 11 shows the results obtained in a collision detection test performed in a single server where 10,000 wandering agents demand a random position as soon as they can, in order to saturate the server. The purpose of this experiment is to know the maximum server throughput, in terms of the average number of actions that it is able to process in a single cycle. As Figure 11 shows, this value mainly depends on the density of the crowd. Nevertheless, we have measured this parameter as the percentage of finally executed actions (ACK's), since it is more informative for our purposes. Thus, an ACK percentage of 0% occurs when no motion is allowed because the crowd is completely full and no one can move. On the other hand, when all the agents completely pass a collision test all the actions are allowed (100% of ACK's) and all the agents finally move. This is the worst case for the server, because in order to pass each collision test it should access to 9 cells. However, if the collision test fails it may require to access to less cells, depending on which is the cell where the test fails. Thus, the worst case for the server in the plot shown in Figure 11 is when 94% of ACK's are provided.

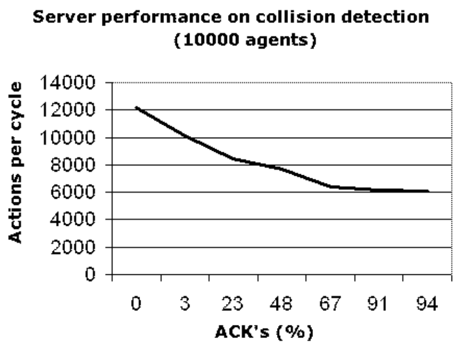

Fig. 11. Server performance on collision tests

Figure 11 shows that in the worst case (94% of ACK's) the server is able to process around 6000 actions in a single cycle (250 ms). However, when the density of the crowd increases then the percentage of ACK's decreases because the probability of collision grows in very dense worlds. This will produce that the server can finish the cell checking without visiting all the neighbors cells. As a consequence, the server can process a higher number of action requests per cycle (12000 actions for a percentage of 0% ACKS). It is also worth mention that for a medium case (48% ACK's), the system can manage around 8000 agents.

Additionally, we have evaluated the throughput and the latency of the entire system in a configuration composed of 4 replicas and 1 server. For both cases, agents continuously demand a random position to the server. The purpose of these evaluations is to check if the proposed architecture can improve the latency and throughput provided by other general purpose DVEs [23]. In these experiments, the existing agents are distributed among the existing CPs in such a way that all the replicas manage a crowd subset (group of agents) of similar size. We have increased the number of agents until the whole system has reached saturation. The saturation point is reached when the CPU utilization reaches 100% in any computer, and it results in a huge increase of the system latency [23].

Table 1 shows the results for different simulations with the same AS cycle (250 ms.) but with different numbers of agents. Each column in this table shows two different values, except for the server. The column labeled with S0 shows the percentage of CPU utilization reached in the computer hosting the AS. The columns labeled with Cx show the percentage of CPU utilization reached in that computer (replica) and the average response time (measured in AS cycles) for the agents supported by that computer. The left column shows the number of agents used in each simulation. Each row in this table shows the results for a simulation with a different number of agents.

Table 1 shows that if a distributed (cluster-based) computer is used then the

	S0	C1	C2	C3	C4
Agents	%	%(cy.)	%(cy.)	%(cy.)	%(cy.)
1000	11	1(1)	1(1)	1(1)	1(1)
2000	22	4(1)	4(1)	4(1)	4(1)
3000	32	10(1)	10(1)	11(1)	10(1)
4000	43	18(1)	18(1)	18(1)	17(1)
5000	54	24(1)	27(1)	25(1)	23(1)
6000	64	31(1)	32(1)	30(1)	32(1)
7000	79	39(1)	40(1)	37(1)	36(1)
8000	94	45.0(1)	48(1)	45(1)	46(1)
9000	97	50(1.5)	49(1.3)	50(1.4)	49(1.3)

Table 1

System throughput for a fixed agent cycle of 250ms.

proposed architecture takes advantage of the distributed hardware, in such a way that the use of multiple computers allows to improve the number of supported agents until the server hosting the AS becomes the system bottleneck. Effectively, this table shows that the system provides acceptable response times with up to 8000 agents, when the computer hosting the AS reaches a CPU utilization of 94%. Although the CPU utilization in the AS is close to saturation (94%), the response times in all the replicas are kept below the AS cycle. That is, the AS is able to process all the requests in a single cycle. However, when the system is supporting 9000 or 10000 agents then the AS can only serve part of such requests, increasing the average response time up to 2.0 cycles (replica C1 for ten thousand agents). It is worth mention that none of the computers reaches a CPU utilization of more than 50%, showing that there is a single bottleneck (the AS).

Table 2 also shows the results for different simulations performed with different numbers of agents. However, in these cases we have studied the minimum response times that can be achieved. Concretely, for each number of agents (each row) we have adjusted the AS cycle until either it has reached a CPU utilization close to saturation (90-97%) or until the CPU utilization did not increase (that number of agents was not enough to saturate the AS). Therefore, the column S0 in table 2 shows values equal or higher than 90% for the last eight rows. The average response times for the agents supported by each replica (the values in parenthesis) are expressed in milliseconds.

Table 2 shows that for 3000 or less agents in the system the average response times in all the replicas is below 0.1 s, and these average response times increase as more agents are in the system. When comparing tables 1 and 2 and Figure 11, it can be seen that they provide coherent results. Effectively, the results in table 1 are obtained with an AS cycle of 250 ms., and that table shows that the system can support up to 8000 agents while providing average response time of 1 cycle in all the replicas. Table 2 shows that for 8000 agents the average response times in all the replicas are 250ms. That is, the system can manage up to 8000 autonomous agents if the AS cycle is 250ms.. These results are obtained with a 50% of positive server acknowledgment, and therefore they agree with the ones in Figure 11, where the server process around 8000 actions (one per agent and cycle) for a moderately dense world (48% of ACKs). Since the agents considered for performance evaluation exclusively

	S0	C1	C2	C3	C4
Agents	%	%(ms.)	%(ms.)	%(ms.)	%(ms.)
1000	28	2(100)	3(100)	4(80)	3(110)
2000	59	12(100)	13(100)	13(90)	16(100)
3000	90	36(100)	35(100)	43(80)	37(100)
4000	93	49(110)	48(110)	48(130)	48(130)
5000	96	50(150)	50(150)	50(160)	50(150)
6000	94	51(190)	51(220)	52(180)	52(180)
7000	95	52(230)	52(210)	51(230)	53(210)
8000	97	47(250)	48(250)	46(250)	46(250)
9000	97	45(290)	47(290)	46(310)	46(300)

Table 2

Response times obtained at maximum throughput

move following a random pattern, they generate the highest number of requests as possible to the AS, that is the resource that can potentially become a bottleneck. Therefore, if the number of agents is increased then the time between successive requests to the AS will also increase, thus allowing the system to support a higher number of agents.

In order to prove the real improvement that the proposed architecture provides in regard to classical approaches, where a centralized application is executed on a single computer, table 3 shows the throughput achieved when the proposed software architecture is executed on a single node of the cluster. Table 3 shows

Agents	CPU(%)	Av-RT(ms)
1000	35	0.11
2000	73	0.16
3000	77	0.24
4000	82	0.32
5000	82	0.40

Table 3

Evaluation results for a fully centralized system configuration

that for 3000 agents the average response time practically reaches the 0.25 s. threshold, and with 4000 agents the average response time exceeds this threshold.

When comparing tables 2 and 3 it can be clearly seen that the proposed system architecture can take advantage of distributed computer architectures distributing the agents between the existing servers. In this way, the number of supported agents can be improved.

Additionally, we have evaluated the performance of the proposed scheme with the pathfinding behavioral model. For evaluation purposes, we have used a range of values of α between 0.4 and 0.6, and we have simulated a urban environment (the one shown in Figures 9 and 10) with a single exit. If these behavioral model and virtual world are used together with an initial uniform distribution of the avatars(agents), initially avatars move in such a way that a very high percentage of positive requests are sent to the server. These effects can be seen on Figure 12. This Figure show on the X-axis the simulation time, while on the Y-axis it shows the percentage of positive answers provided by the server. The percentage of positive acknowledgments to the requests sent by the agents to the AS is very high during the first cycles, reaching around 90%. As the simulation proceeds this percentage linearly decreases, since avatars tend to follow similar paths.

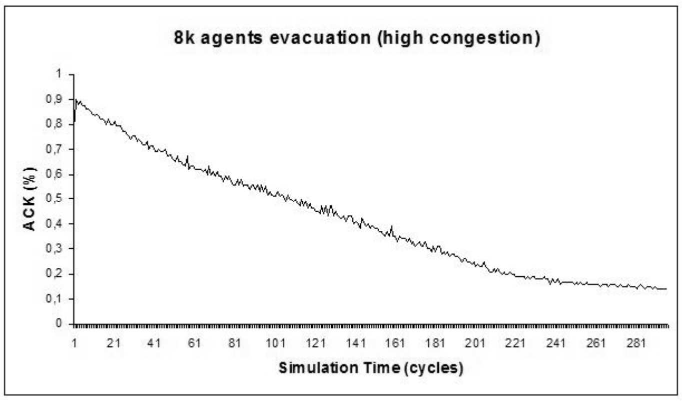

Fig. 12. Percentage of positive ACKs during the evacuation simulation

Taking into account these results, we have measured the average time response provided to the agents in each client computer during the simulation time. Table 4 shows the results for an evacuation simulation with 8000 complex agents (using pathfinding behavioral model). This table shows in each row the results for a different simulation time (measured in the number of sample periods from the beginning of the simulation, 1 sample = 2.5 seconds). Starting from the left, the first column in this table shows the simulation time. The second column shows the total number of agents in the virtual world for that simulation instant. The four remaining columns show the average performance results for the four client computers. Concretely, each of these columns shows the average time response provided to the agents hosted in that computer.

Table 4 shows that for the first simulation instants all of the clients are provided with an average response time higher than two or more server cycles. The

Sim. Time	Number of Agents	C1 ms.	C2 ms.	C3 ms.	C4 ms.
7	7535	504	518	1004	837
11	7153	473	493	1032	751
19	6144	249	249	1014	517
23	5493	250	249	754	309
30	4249	250	250	318	250
35	3431	250	250	249	250

Table 4

Average response times obtained with pathfinding agents

reason for this performance is the percentage of positive answers that the server must initially support, together with the number of current agents. Since the virtual world is well-dimensioned for the amount of agents, the percentage of positive answers is very high during the whole simulation. According to Figure 11, this is very close to the worst case, and as an average the server only can provide a response time lower than one cycle to around six thousand agents. Since this simulation contains eight thousand agents, the average response times provided to the agents is much higher. As the simulation proceeds (second and third row) the number of agents significantly decreases, reducing the server workload accordingly. As a result, in the third row two of the clients already have an average response time below the server cycle. The high numbers of clients $C3$ and $C4$ in the first three rows are due to the semantic data base implementation. In order to show that the server performance is critical for the performance of the whole system, we have implemented the AS in such a way that it sorts the received request by the client process, instead of balancing the requests from the different processes. Therefore, the requests coming for the same process are processed first, in such a way that we can identify the requests of how many client processes can actually answer the AS in a single cycle. Finally, for a simulation time corresponding to 35 sample periods (87.5 seconds) the agents in any client computer are provided (as an average) with a response time below 250 milliseconds.

These results show that even when using agents with complex behaviors the system is capable of supporting thousands of agents. These results validate the proposed scheme (a computer architecture based on networked servers and a hierarchical software architecture) as a trade-off between scalability and consistency.

5 Conclusions and future work

In this paper, we have analyzed the computer architectures used in the literature to support virtual environments. Also, we have proposed an efficient, distributed architecture for crowd simulation. On the one hand, this architecture consists of a computer system based on networked servers, in order to improve scalability but also as a trade-off for efficiently providing awareness and time-space consistency. On the other hand, it consists of a hierarchical software architecture that can easily maintain the coherence of the virtual world (there is a single copy of the semantic database). Performance evaluation results show that this architecture can take advantage of distributed architectures, efficiently managing thousands of autonomous agents. Concretely, for the case of an AS cycle of 250ms., this scheme can handle from 6000 to 8000 autonomous agents when using 5 computers, depending on the behavioral model of the agents.

As a future work to be done, we plan to efficiently distribute the action server in different machines by using distributed databases techniques, in order to improve the scalability. Also, we plan to characterize the requirements of different kinds of autonomous agents. The idea is to use each replica for supporting one (or more) kind of agents, according to the computational power of the computer and the requirements of the agents. Thus, by properly balancing the existing load among the computers we expect to improve the system throughput.

6 Acknowledgments

This work has been jointly supported by the Spanish MEC and European Commission FEDER funds under grants Consolider Ingenio-2010 CSD2006-00046 and TIN2006-15516-C04-04.

References

[1] T. Alexander, Massively Multiplayer Game Development II, Charles River Media, 2005.

[2] S. Chenney, Flow tiles, in: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation, ACM Press, 2004.

[3] D. Diller, W. Ferguson, W. Leung, A. Benyo, D. Foley, Behavior modelling in comercial games, in: BRIMS '04: Proceedings of the 2004 Behavior Representation in Modelling and Simulation Conference, 2004.

[4] S. Dobbyn, J. Hamill, K. O'Conor, C. O'Sullivan, Geopostors: a real-time geometry/impostor crowd rendering system, ACM Trans. Graph. 24 (3) (2005) 933–933.

[5] S. Donikian, Informed virtual environments, in: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation, ACM Press, 2004.

[6] A. I. F. Luengo, Framework for simulating the human behavior for intelligent virtual agents, Lectures Notes in Computer Science 3039 (2004) 229–244.

[7] A. I. F. Luengo, Framework for simulating the human behavior for intelligent virtual agents. part ii: Behavioral system., Lectures Notes in Computer Science 3039 (2004) 237–244.

[8] R. Fikes, N. Nilsson, Strips: a new approach to the application of theorem proving to problem solving, Artificial Intelligence 5 (2) (1971) 189–208.

[9] R. M. Fujimoto, R. Weatherly, Time management in the dod high level architecture, in: Proceedings tenth Workshop on Parallel and Distributed Simulation, 1996.

[10] L. Gautier, C. Diot, Design and evaluation of mimaze, a multi-player game on the internet, in: Proceedings of IEEE Multimedia Systems Conference, 1998.

[11] C. Greenhalgh, A. Bullock, E. Fr¿on, D. Llyod, A. Steed, Making networked virtual environments work, Presence: Teleoperators and Virtual Environments 10 (2) (2001) 142–159.

[12] T. Henderson, S. Bhatti, Networked games: a qos-sensitive application for qos-insensitive users?, in: Proceedings of the ACM SIGCOMM 2003, ACM Press /

ACM SIGCOMM, 2003.

[13] J. L. Hennessy, D. A. Patterson, Computer architecture: a quantitative approach (3rd. Edition), Morgan Kaufmann Series in Computer Architecture, Elsevier Science, 2002.

[14] A. Iglesias, F. Luengo, New goal selection scheme for behavioral animation of intelligent virtual agents, IEICE Transactions on Information and Systems, Special Issue on 'CyberWorlds' E88-D (5) (2005) 865–871.

[15] 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.

[16] M. Lozano, P. Morillo, J. M. Orduña, 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), 2007.

[17] J. C. Lui, M. Chan, An efficient partitioning algorithm for distributed virtual environment systems, IEEE Trans. Parallel and Distributed Systems 13.

[18] D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, Z. Xu, Peer-to-peer computing, Tech. rep., Technical Report HPL-2002-57, HP Laboratories, Palo Alto (2002).

[19] J. S. Monzani, A. Caicedo, D. Thalmann, Integrating behavioral animation techniques, in: Proceedings of EUROGRAPHICS 2001, Computer Graphics Forum, Vol. 20, Issue 3, 2001.

[20] S. Mooney, B. Games, Battlezone: Official Strategy Guide, BradyGame Publisher, 1998.

[21] P. Morillo, J. M. Orduña, M. Fernández, J. Duato, A method for providing qos

in distributed virtual environments, in: 13th Conference on Parallel, Distributed and Network-based Processing (PDP'05), IEEE Computer Society, 2005.

[22] P. Morillo, J. M. Orduña, M. Fernández, J. Duato, On the characterization of distributed virtual environment systems, in: Euro-Par' 2003 - Lecture Notes in Computer Science 2790, Springer-Verlag, 2003.

[23] P. Morillo, J. M. Orduña, M. Fernández, J. Duato, Improving the performance of distributed virtual environment systems, IEEE Transactions on Parallel and Distributed Systems 16 (7) (2005) 637–649.

[24] H. Nakanishi, T. Ishida, Freewalk/q: social interaction platform in virtual space, in: VRST '04: Proceedings of the ACM symposium on Virtual reality software and technology, ACM Press, New York, NY, USA, 2004.

[25] Quake: http://www.idsoftware.com/games/quake/quake/.

[26] S. Raupp, D. Thalmann, Hierarchical model for real time simulation of virtual human crowds, IEEE Transactions oon Visualization and Computer Graphics 7 (2) (2001) 152–164.

[27] D. Roberts, R. Wolff, Controlling consistency within collaborative virtual environments, in: Proceedings of IEEE Symposium on Distributed Simulation and Real-Time Applications (DSRT'04), 2004.

[28] W. Shao, D. Terzopoulos, Autonomous pedestrians, in: SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation, ACM Press, New York, NY, USA, 2005.

[29] D. Sima, T. Fountain, P. Karsuk, Advanced Computer Architectures : A Design Space Approach, Addison Wesley, 1997.

[30] S. Singhal, M. Zyda, Networked Virtual Environments, ACM Press, 1999.

[31] J. Smed, T. Kaukoranta, H. Hakonen, A review on networking and multiplayer computer games, Tech. rep., Turku Centre for Computer Science. Tech Report 454. (2002).

[32] R. B. Smith, R. Hixon, B. Horan, Collaborative Virtual Environments, chap. Supporting Flexible Roles in a Shared Space, Springer-Verlag, 2001.

[33] M. Sung, M. Gleicher, S. Chenney, Scalable behaviors for crowd simulations, in: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation, ACM Press, 2004.

[34] F. Tecchia, C. Loscos, Y. Chrysathou, Visualizing crowds in real time, Computer Graphics Forum 21.

[35] H. Tianfield, J. Tian, X. Yao, On the architectures of complex multi-agent systems, in: Proc. of the IEEE/WIC International Conference on Web Intelligence / Intelligent Agent Technology,, IEEE Press, 2003.

[36] Unreal Tournament: http://www.unrealtournament.com/.

[37] S. Zhou, W. Cai, B. Lee, S. J. Turner, Time-space consistency in large-scale distributed virtual environments, ACM Transactions on Modeling and Computer Simulation 14 (1) (2004) 31–47.