Analysis of Jason's performance for crowd simulations

Víctor Fernández, Francisco Grimaldo, Miguel Lozano, Juan M. Orduña
Departament d'Informàtica, Universitat de València, Avda. Vicent Andrés Estellés s/n, 46100, Burjassot, Spain
ferbau@alumni.uv.es, francisco.grimaldo@uv.es, miguel.lozano@uv.es, juan.orduna@uv.es

Abstract

Large-scale crowd simulations require distributed computer architectures and efficient parallel techniques to achieve the rendering of visually plausible images while simulating the behaviour of crowds of autonomous agents. The Java-based multiagent platforms, devoted to provide the agents with the required lifecycle, represent a key middleware in crowd systems. However, since they are oriented to maximize portability and to reduce the development cost, they may reduce performance and scalability, two important requirements in large-scale crowd simulation systems. This paper studies the performance and scalability provided by Jason, a well known Java-based BDI-MAS platform, as a plausible framework to be used for large-scale crowd simulations. The performance evaluation results show that some improvements should be performed in order to make Jason a suitable middleware for large-scale crowd simulations.

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, mobility, safety and security in public spaces or entertainment [8]. However, simulating the realistic behavior of large crowds of autonomous agents is still a challenge for several computer research communities [9, 5].

Scalability is a key feature in crowd simulation and the simulation of high number of autonomous pedestrian-agents represents an active research area at the intersection of computer graphics, artificial intelligence and distribute computing. Any scalable model can be divided into three levels. Firstly, graphic engines must render complex crowded scenes as fast as posible. Here, the challenge is to display realistic big size crowded scenes at interactive frame rates. However, most of the current (graphics oriented) crowd systems use specific MAS with centralised architectures, so they can hardly simulate a few thousands of agents and it is also very diffucult to scale these systems up. Secondly, the MAS platform represents a middleware between the distributed computer architecture and the graphical engine. The MAS platform mainly addresses two important issues, the agents behavior modeling and their parallel lifecycle execution. Some researchers have been testing the performance of existing agent platforms [6], showing a lack of performance and scalability in many of them. The main challenge that crowd simulation offers to the MAS platforms is the ability of handling a massive and concurrent action processing at interactive rates (i.e. 250ms/action). Thirdly, large scale crowd simulation requires distributed computer architectures (eg. P2P, networked-server,...) to execute the MAS designed and to be able to increase the number of agents when required. Hence, scalability is a key issue that mainly depends on the distributed computer architecture and the degree of parallelism achieved by the software architecture.

Bearing these levels in mind, this work evaluates Jason, a well-known MAS platform, as a plausible MAS for simulating large-scale interactive crowded scenes. Although there are several works dealing with the evaluation of Java-based Multiagent Platforms [2], to our knowledge there is no such study for the Jason framework. Therefore, the results presented in this paper will be of great value to those researches considering Jason as a suitable platform to develop not only crowd simulations but also other large-scale multiagent applications. The performance evaluation carried out has been focused on the agent action response time and the percentage of CPU utilization consumed by the MAS when simulating a locomotion benchmark. The evaluation results show that some improvements should be performed in order to make Jason a suitable middleware for large-scale crowd simulations.

2. Related work

Crowd simulations must focus on rendering visually plausible images of the environment, requiring a high computational cost. Furthermore, from a MAS level point of view, complex agents must provide autonomous complex behaviors, greatly increasing the computational cost as well. Hence, this situation requires the use of distributed architectures capable of managing the corresponding workload.

Graphics oriented proposals tackle crowd simulations as a particle system with different levels of detail (e.g. impostors) in order to reduce the computational cost [7]. Although these proposals can handle crowd dynamics and display populated interactive scenes (10K pedestrians), they are not able to produce complex autonomous behaviors for their actors. On the contrary, a few graphics oriented proposals include socially complex and autonomous behaviors [13]. However, the main problem presented by these works is scalability, since they are generally based on centralised system architectures. Hence, they can only control hundreds of autonomous agents at interactive frame rates.

From the distributed computing point of view, different architectures have been applied to crowd simulation. For instance, a new approach has been presented for PLAYSTATION3 which supports simulation of simple crowds up to 15000 individuals at 60 frames per second [9]. 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 classical Reynolds's boids [10], has been also integrated in a PC-Cluster with MPI communication among the cluster processors [14]. This proposal uses different communication and partitioning strategies to finally produce small crowds simulations (512 boids), which are far from interactive.

In the middle of these two levels, a multiagent system must be located in order to: i) efficiently explode the computational resources involved and ii) to provide the required data flow to the interactive graphic application. According to this, the MAS framework constitutes a key middleware that higly influenciates the global performance and scalability of the crowd system. Some researchers have tested the performance and scalability of a few existing MAS platforms [6], showing a lack of both important issues in many of them. Since the main problem that crowd simulation provides to the MAS platform is to be able of handling a massive and concurrent action processing (all the crowd actions), a higly efficient action model will be required. Otherwise, the MAS platform will not be useful for crowd simulation purposes.

3. Jason infrastructures

Jason is a Java-based interpreter for an extended version of AgentSpeak, a BDI agent-oriented logic programming language [1]. It offers an elegant notation based on logic programming to design agents capable of managing long-term goals (goal-based behavior), reacting to changes in a dynamic environment and handling multiple intentions concurrently. Beyond the implicit overload that a virtual machine adds to any system, here we are focused on scalable crowd architectures, which

are mainly affected by the possibility of distributing the MAS on different computers to increase the crowd size without loosing agent performance.

Jason provides three infrastructures to execute a MAS: *Centralised*, SACI and JADE. On the one hand, the *Centralised* infrastructure places all the components of the MAS in the same host. On the other hand, it is also possible to distribute these components in several hosts using either SACI or JADE technologies. Next, we review these three approaches.

3.1. Centralised

The *Centralised* infrastructure executes both the environment and the agents of a MAS within a single computer. Figure 1 depicts the general scheme of this infrastructure. On the left hand side, the environment has its own execution thread and it is provided with a configurable pool of threads (PThE) to attend the actions requested by the agents. This way, the enviroment is able to deal with several agent requests concurrently. On the right hand side, each agent owns by default a thread in charge of executing the agent reasoning cycle. In this manner, all the agents can run concurrently within the MAS. However, this approach limits the number of agents that can be executed, since the total number of threads will be limited by the JVM and finally by the operating system. In order to overcome this limitation, Jason offers the possibility to optionally add another configurable pool of threads (PThA), so that the set of agents can share a smaller number of execution threads but reducing the level of concurrency. By default, the PThE holds 4 threads whereas the PThA is disabled.

Agent-agent as well as agent-environment communication is made through event passing in the *Centralised* infrastructure. That implies that the message is not serialized but cloned, and the new object reference is passed to the receiver. The structure of the messages in Jason is based on the speech act paradigm and follows the protocols proposed by the KQML [3] and FIPA [4] agent communication languages.

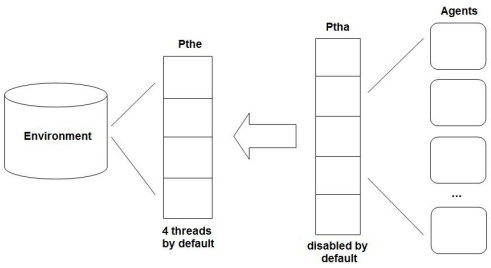

Figure 1: Overview of the Centralised infraestructure

3.2. SACI

The Simple Agent Communication Infrastructure (SACI) is an API of Java with a set of tools for the development of societies of distributed agents [11]. SACI offers a programming communication API oriented to two important features: i) composing, sending, and receiving messages; and ii) getting the agents designers rid of the underlaying computer architecture. As shown in figure 2, the agents in SACI group in societies and they are provided with a mailbox to interact among them. Every society has a *Facilitator* agent who makes the location of the agents transparent in the network. This way, agents can exchange messages between their queues of incomming and outgoing messages. Then, the programmer can specify in which host to run each agent through the configuration file of the multiagent system (i.e. the *mas2j* file).

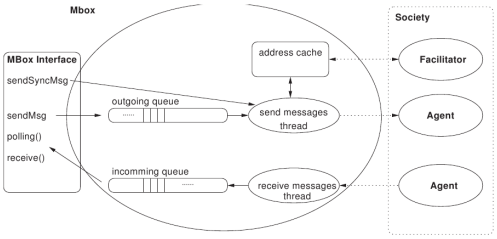

Figure 2: Overview of the SACI infraestructure [11]

With regard to the agent-agent and agent-environment communication, the SACI infras-

tructure uses Java Remote Method Invocation (Java RMI). RMI uses object serialization to marshal and unmarshal the messages being exchanged. Besides, as messages in SACI are formatted in KQML, every Jason message will have to be recoded by the sender in order to be delivered.

3.3. JADE

The latest infrastructure available in Jason is the well-known Java Agent DEvelopment Framework (JADE) [12]. JADE simplifies the implementation of multiagent systems through a middleware that complies with the FIPA specifications [4]. Thus, the agent platform can be distributed across machines as shown in figure 3. The agent execution environments in JADE are called *Containers* and the active set of *Containers* that form the MAS is known as the *Platform*. Each *Platform* has a *Main Container* with two special agents: i) the *Agent Management System* (AMS), that provides the naming service and represents the authority in the platform; and ii) the *Directory Facilitator* (DF), that provides a Yellow Pages service. Currently, the specification of the JADE settings to run Jason projects over JADE is not fully supported by the *mas2j* configuration file. Thus, the programmer is left in charge of manually creating the one or more *Platforms* and of distributing the environment and the agents among the *Containers* by means of the command-line JADE commands.

Communication in JADE follows an asynchronous message exchange approach where every agent has a mailbox within which the container introduces the messages sent by other agents. The messages have the format specified by the FIPA ACL language. Then, message passing is done through RMI when the agents share the same platform but are placed in different containers.

4. Test description

The goal of this work is to evaluate Jason as a plausible MAS framework for simulating interactive crowded scenes. We should notice that

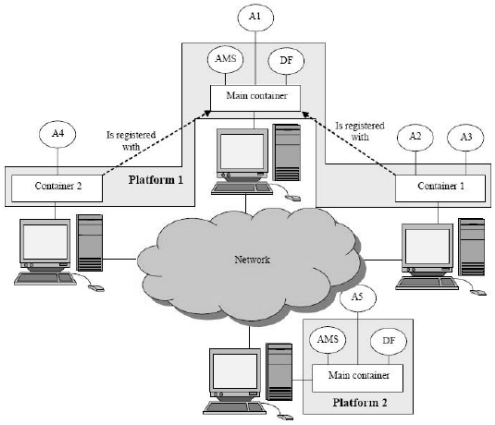

Figure 3: Overview of the JADE infraestructure [12]

normal communication procedures of software agents are far from the crowd ones. Crowd agents do not communicate among them often, but they need to act very frequently in a shared environment. Motion actions are the most frequent actions within a crowd and they must be executed carefully in order to ensure the world consistency.

Therefore, we have defined a locomotion testbed where a set of wanderer agents request movement actions to a grid-like single environment, which in turn replies with the result of the execution. Wanderer agents are written in AgenSpeak and they cyclically execute the following steps: (i) take start time, (ii) request a random movement to the enviroment, and (iii) take finish time. On the other hand, the environment executes each movement action in a synchronized manner to ensure the world consistency. This testbed is repeated for the three infrastructures being studied: Centralised, SACI and JADE. For the distributed versions we used two hosts, one devoted to run the environment and the other to execute all the wanderer agents. In the JADE case, a single platform has been created and the environment has been placed in the main container.

The performance evaluation carried out in section 5 measures the environment response time and the percentage of CPU utilization

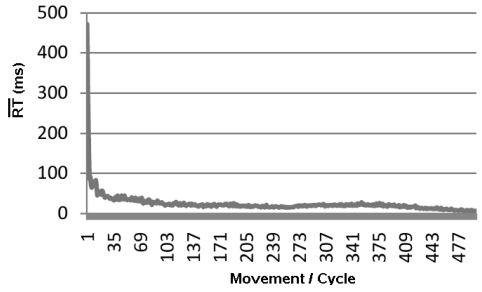

Figure 4: Response Time evolution

consumed while running the locomotion benchmark. We define this Response Time (RT) as the time elapsed between an agent asking for an action and receiving the reply from the environment. Although each agent performs 500 movements or cycles, we only consider the 100 mesurements in the middle to obtain the average response time ($\overline{RT}$). This interval reflects the system behavior at full load whereas the first and the last 200 cycles measurements are distorted due to the agent creation/destruction as demonstrated in figure 4.

In order to evaluate the scalability of Jason, we have increased the number of wanderer agents (N_{ags}). The size of the environment has been adapted accordingly so that the density remains around 40 %. In this paper, we show the results between 500 and 3500 agents since the system saturates beyond 3500 agents due to the JVM limitation in the number of threads.

5. Performance evaluation

The results shown in this section have been obtained using a PC cluster of 24 AMD Dual-Core Opteron processors running at 1.6 GHz with 4 GB of RAM and connected via Gigabit Ethernet. The cluster runs the 64-bit version of CentOS 5.1 and the Java implementation used is Sun JDK 1.6.

Table 1 shows the performance obtained for the test defined in section 4 in the Centralised architecture with PThA disabled and 50 threads in PThE. From left to right, the columns represent: the number of simulated agents (N_{ags}); the percentage of CPU utilization consumed; the average of the response time for each agent movement action ($\overline{RT}$); its corresponding standard deviation (σ_{RT}); and the total $Time$ consumed by the simulation.

Table 1: Performance obtained in a Centralised infrastructure (PThA=0 and PThE=50).

N_{ags}	CPU **%**	$\overline{RT}$ **(ms)**	σ_{RT} **(ms)**	$Time$ **(ms)**
500	83	61	129	34528
1000	84	85	264	65510
1500	84	76	249	99815
2000	84	201	464	143170
2500	82	305	669	179427
3000	77	410	652	229985
3500	76	429	746	272099

Firstly, we focus on the response time since it is an important feature of crowd simulations that directly affects the degree of interactivity achieved. Generally, 250 ms/action is considered as a reference ratio (maximum period) for these domains [5]. Although this value is exceeded after 2K agents, this constraint is only satisfied for 500 agents due to the high standard deviation values. For instance, in the 1K agents simulation the 70 % of the actions are served in 85 $\pm$ 264 ms. That is, even though the average value (85 ms) indicates that many actions are served very fast, there are a few agents that must wait more than 250 ms for their actions to be executed. Besides, the percentage of CPU utilization decreases for higher number of agents. Both results point to a process that stops the system whenever it is executed, that is, the Java Garbage Collection. The garbage collector of the JVM stops execution of the agents too often, which prevents a more equilibrated RT distribution.

Table 2 evaluates the influence of the size of PThE. In this case, we have fixed the number of agents to a value of 1000. The average response times decrease when increasing the size of PThE, but the standard deviation shows huge values with an increasing slope and the percentage of CPU utilization decreases. These results show that, even though the agents

have more threads to act, this increases the amount of memory used by the JVM as well as the number of garbage collector invocations stopping the system. Therefore, it will be necessary to tune Java in order to improve this behavior.

Table 2: Influence of the number of threads in PThE in the Centralised infrastructure with N_{ags}=1000 and PThA=0.

N_{PThE}	CPU %	$\overline{RT}$ (ms)	σ_{RT} (ms)	$Time$ (ms)
100	85	119	351	66489
200	83	106	396	66958
300	84	121	547	68124
400	83	106	564	66701
500	83	88	608	67467
600	83	81	522	62341
700	81	76	560	62755
800	81	67	597	65393
900	81	93	672	64489
1000	81	78	601	65094

Although not included in this paper, we have also demonstrated that using the PThA makes the global agent cycle run slower due to the new thread competition introduced. Thus, it is only interesting when you want to run a lot of agents with limited resources.

For comparison purposes, we have executed the same locomotion benchmark over the two distributed infrastructures offered by Jason: SACI and JADE. For these simulations we used a simple distributed architecture composed by two hosts, the first one allocating the agents and the second one handling the environment. Tables 3 and 4 show the results obtained for SACI and JADE, respectively. These tables distinguish the percentage of CPU utilization required by both the host in charge of the agents (CPU_a) and the host in charge of the environment (CPU_e).

According with the $\overline{RT}$ column, both SACI and JADE show a linear response with the number of agents, although this response is slightly better for the SACI case. Furthermore, the standard deviation shown in both tables is considerably lower than the Centralised outcomes. However, the high values obtained for the RT are far from the interactive response time desired in crow simulation (i.e. 250 ms).

Table 3: Influence of the number of agents in the SACI infrastructure with PThA=0 and PThE=50.

N_{ags}	CPU_e %	CPU_a %	$\overline{RT}$ (ms)	σ_{RT} (ms)	$Time$ (ms)
100	32	36	100	11	162021
200	32	35	198	20	314212
300	32	35	299	31	468057
400	32	35	398	46	621113
500	31	35	503	52	780493
600	31	36	610	70	946934
700	31	36	707	93	1112253
800	31	35	810	111	1277237

Table 4: Influence of the number of agents in the JADE infrastructure with PThA=0 and PThE=50.

N_{ags}	CPU_e %	CPU_a %	$\overline{RT}$ (ms)	σ_{RT} (ms)	$Time$ (ms)
100	45	39	130	18	267926
200	45	39	258	31	513394
300	45	39	382	36	772402
400	45	39	521	46	1045591
500	44	39	641	61	1286356
600	45	39	783	77	1552234
700	45	39	907	78	1808992
800	44	39	1026	86	2048948

Additionally, tables 3 and 4 show very low percentages of CPU (CPU_a and CPU_e) utilization. These results suggest that the problem here comes from the communication platform in both infrastructures, which use RMI instead of method invocation to perform the message passing. In this way, when comparing the total time consumed with the centralised approach (table 2), the distributed simulation requires longer execution times to complete the same work. Therefore, we can conclude that the final performance is being seriously affected by the communication costs.

6. Conclusions and Future work

This paper studies the performance and scalability provided by Jason as a suitable framework to be used for large-scale crowd simulations. The results for the Centralised infrastructure show that the default running set-

tings do not allow the JVM to manage the memory appropriately. Therefore, the garbage collection pauses overly the execution of the agents and the standard deviation of the response time increases, thus preventing us from getting interactive times. For the distributed infrastructures, the results show the effects of the communication platform on this particular problem. Both SACI and JADE are based on RMI and result in a similar scalability degree (considering scalability as a function that associate the number of agents with the RT obtained). However, the distributed simulations require longer execution times and the performance (in terms of the response time) is seriously affected by the communication costs. Finally, the CPUs have never reached saturation due to both the number of agents waiting to gain access to the environment threads in PThE and the synchronized action code.

These conclusions allow us to address future work in the following directions. First, we should perform Java tuning in order to reduce the effect of the garbage collection when running centralised simulations over Jason. Then, a new P2P architecture can be proposed to distribute the problem, where each peer will be a computer executing a centralised version of the system (agents + environment) dealing with a smaller piece of the environment. In this case, spatial criteria should be used to distribute the agents to the computers, and dynamic algorithms will be required to keep the workload properly balanced during the simulation.

Acknowledgements

This work has been jointly supported by the Spanish MEC and the European Commission FEDER funds, under grants Consolider-Ingenio 2010 CSD2006-00046 and TIN2009-14475-C04-04. The authors greatly thank the efficient and kind support provided by the creators of Jason at any time.

References

[1] R. H. Bordini, J. F. Hübner, and M. Wooldrige. *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley, 2007.

[2] E. Cortese, F. Quarta, and G. Vitaglione. Scalability and performance of JADE message transport system. In *AAMAS Workshop on AgentCities*, 2002.

[3] T. Finin, J. Weber, G. Wiederhold, M. Genesereth, R. Fritzson, D. McKay, J. McGuire, R. Pelavin, S. Shapiro, and C. Beck. Specification of the KQML agent communication language. DARPA Knowledge Sharing Initiative External Interfaces Working Group., 1993.

[4] FIPA. FIPA (Foundation for Intelligent Physical Agents), 2009. Available at http://www.fipa.org/specs/fipa00023/.

[5] Miguel Lozano, Pedro Morillo, Juan Manuel Orduï¿½a, and Vicente 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.

[6] Luis Mulet, Jose M. Such, and Juan M. Alberola. Performance evaluation of open-source multiagent platforms. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1107–1109, New York, NY, USA, 2006. ACM.

[7] C. O'Sullivan, J. Cassell, H. Vilhjï¿½msson, J. Dingliana, S. Dobbyn, B. McNamee, C. Peters, and T. Giang. Levels of detail for crowds and groups. *Computer Graphics Forum*, 21(4):733–742, 2002.

[8] Nuria Pelechano, Jan M. Allbeck, and Norman I. Badler. Virtual crowds: Methods, simulation, and control. *Synthesis Lectures on Computer Graphics and Animation*, 3(1):1–176, 2008.

[9] Craig 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.

[10] Craig 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.

[11] J.S. Sichman and J. F. Hübner. SACI - Simple Agent Communication Infrastructure, 2009. Available at http://www.lti.pcs.usp.br/saci/.

[12] TILAB. JADE (Java Agent DEvelopment Framework), 2009. Available at http://jade.tilab.com/index.html.

[13] Adrien Treuille, Seth Cooper, and Zoran Popovic. Continuum crowds. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 1160–1168. ACM, 2006.

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