Large-Scale Multiagent Platform Benchmarks[⋆]

Jose M. Such[⋆⋆], Juan M. Alberola, Luis Mulet[⋆⋆⋆], Agustin Espinosa
Ana Garcia-Fornes, and Vicent Botti

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València
Camí de Vera s/n 46022, València
Spain
{jsuch,jalberola,lmulet,aespinos,vbotti,agarcia}@dsic.upv.es

Abstract. When deploying large-scale multiagent systems, efficiency and scalability become important features that must be provided by the multiagent platform used to run them. These multiagent systems are complex, huge and are composed of several agents that have a great number of interactions with each other. Testing these systems on multiagent platforms is very helpful for platform developers to improve performance or for multiagent systems developers to ensure that the platform meets their requirements. Therefore, a set of benchmarks that can determine the behaviour of a multiagent platform when it is scaled up is necessary. This paper proposes four different benchmarks to achieve this goal.

Key words: Benchmarks and testbeds for Multiagent Platforms, Agent development tools and platforms

1 Introduction

As stated in [LMSW05], the next generation of computing system is likely to demand large numbers of interacting components, be they services, agents or otherwise. Current tools work well with limited numbers of agents, but are generally not yet suitable for the development of large-scale (and efficient) agent systems, nor do they offer development, management or monitoring facilities able to deal with large amounts of information or tune the behaviour of the system in such cases.

Within the area of multiagent systems (MAS), efficiency is a key issue. The efficiency of an agent system relies on both its design and the agent platform on which it runs. In the last few years, many researchers have focused on testing the performance of existing agent platforms [MSA$^+$06] , [CTG$^+$04], [Vrb03], [CFV03], [CACM], [BGNT], [LNW98], [Sha05].

[⋆] This work has been partially supported by the Spanish goverment and FEDER funds under TIN2005-03395 and TIN2006-14630-C03-01 projects.

[⋆⋆] Enjoying a grant of *Conselleria d'Empresa, Universitat i Ciència de la Generalitat Valenciana* (BFPI06/096).

[⋆⋆⋆] Enjoying a Spanish Ministry of Education and Science grant (AP2005-3731).

Most of these studies show poor performance in current agent platforms. However, these works ignore performance degradation when scaling up the systems, because they concentrate on showing multiagent platform performance when the MAS that is running on the platform has few agents and hosts. Only Chmiel et al. [CTG$^+$04] present multiagent platform tests using eight hosts, but they do not scale up any higher.

We propose a set of benchmarks for analyzing the most critical parameters that can be involved in platform performance degradation when platforms are launched in several hosts. These benchmarks can be applied to any multiagent platform because they are not designed for any specific one. We are currently applying these benchmarks to several platforms. In this paper, we apply the benchmarks to the Jade platform [BCPR03], [JAD] as an example of their use.

In this example, we try to get empirical results about the decrease of Jade performance when the Jade security add-on is introduced. By knowing the level of performance degradation when introducing additional components (like the security add-on) over a specific environment, the most suitable implementation of this platform can be chosen.

The rest of the article is organized as follows. Section 2 presents the main arguments for developing the proposed benchmarks. Section 3 provides an in-depth description of the benchmark design and Section 4 shows these benchmarks applied on the JADE platform. Finally, Section 5 presents some concluding remarks.

2 Purpose

Most multiagent platforms attempt to offer support for large applications that are composed of hundreds of agents running on numerous hosts. However, this cannot really be carried out by some platforms when trying to implement a large system. We have been working on different platforms, and we have verified their behaviours. Some of them offer very good behaviour when they are composed of few hosts and agents, but when these and other parameters such as message traffic are increased, their performance is completely different. Some platforms have problems when they are launched on several hosts. They crash when some hosts are overloaded with a large number of messages or when there are many agents running on several hosts.

There are actually some different problems when trying to implement a large application on a specific platform. Some previous works (as shown in section 1) present studies about multiagent platform performance. Nevertheless, these experiments are designed and tested on very few hosts. Since most of them do not scale up more than four hosts, we do not know what the platform response is when there is a large load, i.e., many hosts launching many agents. A common problem mentioned in some of these studies is the degradation of the performance of the platforms when parameters such as message traffic, number of agents, message size, etc. are increased.

Ascertaining how the obtained response times vary when these parameters are changed is very important in order to have a broad knowledge of the platform being used. With this knowledge, the best design for the multiagent system can be selected for specific platform, or a better platform could be selected if the one in use does not fulfil the minimal requirements. These measures could also be used by platform developers to test new versions or added functionalities to determine how differently the platform performs. This knowledge is very useful for platform users to be able to determine the loss of performance when adding features such as security components, service replication, etc. Moreover, these measures can help to configure the best parameters for any platform: number of hosts, services to be offered, agents running per host, etc.

To achieve these goals, we propose a set of benchmarks for testing multiagent platforms when developing a large-scale multiagent system. The benchmark set is general enough to be applied to any multiagent platform. These tests also allow us to compare some platforms to determine which one is the most scalable, which one is more susceptible to being overloaded, etc.

3 Benchmark Description

We have designed four benchmarks oriented to large-scale systems. Each benchmark is composed of a multiagent system running on the platform being tested. The result of each benchmark is the average round trip time (RTT) of each message, i.e., the time elapsed from when a sender agent sends a message until it receives the same message sent back by a receiver agent. The RTT obtained by each agent is an average of the RTT obtained by each message sent. The final RTT value of a benchmark execution is an average of the RTT obtained by each sender agent taking part in the experiment. The number of messages sent per sender agent is a parameter that the final user of the benchmark can configure in order to achieve the needed accuracy level. Another parameter that can be configured in the benchmark set is the message size in bytes.

Apart from sender agents and receiver agents, there is always a controller agent in each benchmark. This agent manages benchmark execution, synchronizing each sender agent when the benchmark is started. Therefore, all agents involved in the test begin at the same time. The controller agent also manages the finalization of the experiment, so that a new experiment can be run without being disturbed by the previous one. Furthermore, the controller agent runs alone in a host that is not used to perform the benchamrks. Therefore, when we state that a benchmark is run on N hosts, it means that the benchmark needs N+1 hosts to be performed.

3.1 Benchmark 1: Number of Hosts

The purpose of this benchmark is to measure the capacity of the analyzed platform to send messages to different agents placed in numerous hosts. the benchmark attempts to determine how the platform behaves when the number of hosts that it has to manage is increased. This test involves a lot of message exchanges between every pair of hosts in the platform.

The benchmark is launched with very few agents per host in order to prevent the overload of each host from influencing the results. If a host is overloaded due to the number of agents, the response time can increase without increasing the number of hosts. As a result, the number of hosts that the platform can support cannot be ascertained. Therefore, the number of agents per host stays the same, and only the number of hosts can be changed.

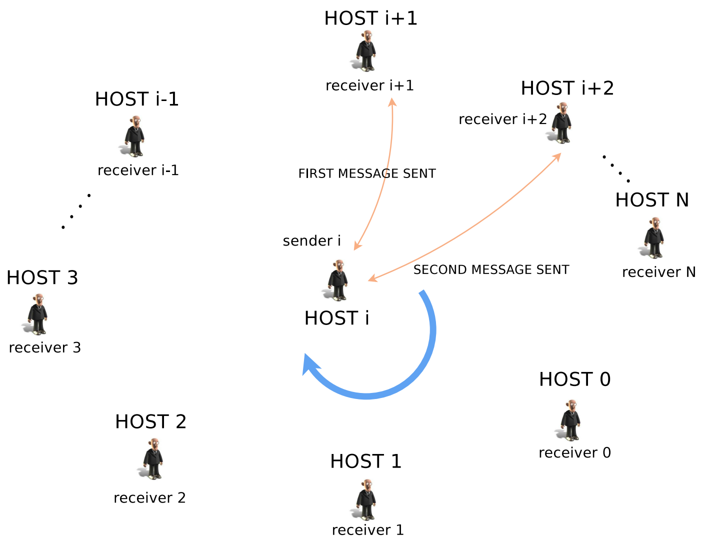

Fig. 1. Benchmark 1, Number of Hosts

This benchmark consists of a multiagent system composed of sender and receiver agents as shown in Figure 1. Agents are spread over several hosts of the platform, placing one receiver agent and one sender agent in each host. The number of hosts is a parameter to be set by the final user of the benchmark.

All sender agents behave in the same way, and they are labeled as $sender_0$ up to $sender_{N-1}$ (with N being the number of the hosts set by the user). The $sender_i$, which is the agent placed in host i, starts sending a message to $receiver_{i+1}$; it waits for the response of $receiver_{i+1}$ and calculates the RTT. Then, the $sender_i$ sends a message to $receiver_{i+2}$, and so on. When the $sender_i$ has sent a message to $receiver_{N-1}$, it sends its next message to $receiver_0$. After that, it sends a message to $receiver_1$, $receiver_{i+1}$, and so forth. The $sender_i$ keeps on sending messages in this way until it sends the number of messages chosen by the user.

Since some platforms do not carry out message sending in the same way when agents are placed in the same host than when they are placed in diffent hosts, the sender agent never sends messages to the receiver agent placed in the same host.

3.2 Benchmark 2: Massive Reception (1 receiver)

The second benchmark (Figure 2) has been designed to evaluate platform response in massive reception environments. The purpose of this test is to measure response capacity of the platform when a host has a high rate of message exchanges, i.e., when there are a lot of incoming and outgoing messages. Moreover, there are two cases: if this high message traffic is carried out by only one agent or if it is carried out by several agents. Benchmark 2 attempts to simulate a multiagent-based application where there is a single agent performing a service that is needed by a great number of agents.

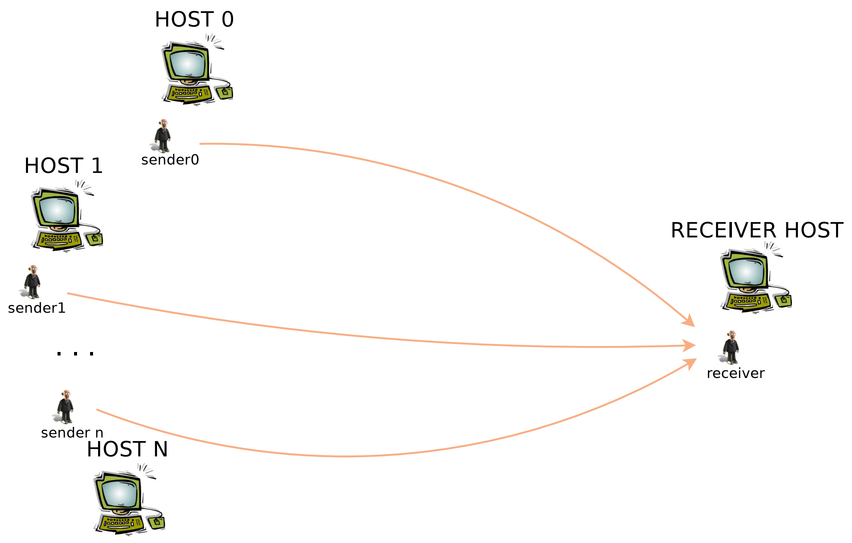

Fig. 2. Benchmark 2, Massive Reception (1 receiver)

In this benchmark, the platform is again distributed among various hosts. There is one host with a receiver agent, and a number of hosts with one sender agent in each one. All the sender agents send messages to the only receiver agent in this multiagent system. This receiver agent behaves like a ping agent, i.e., for every message that is received it obtains the sender agent that has sent the message and replies to it. Therefore, in this test, the receiver agent and the host where it is placed become overloaded. The number of hosts with a sender agent is a parameter to be set by the user.

3.3 Benchmark 3: Massive Reception (N receivers)

This benchmark complements the benchmark 2. In this experiment, we produce a lot of message traffic between one host and the rest of the hosts as in the previous experiment. In this one, however, we also evaluate whether distributing this message traffic among several receivers (which are placed in the same host) provides a better platform responses. This is done to determine whether the bottleneck when overloading an agent with a lot of message exchanges is caused by the message transport system of the platform or by the message queue management of the agent itself.

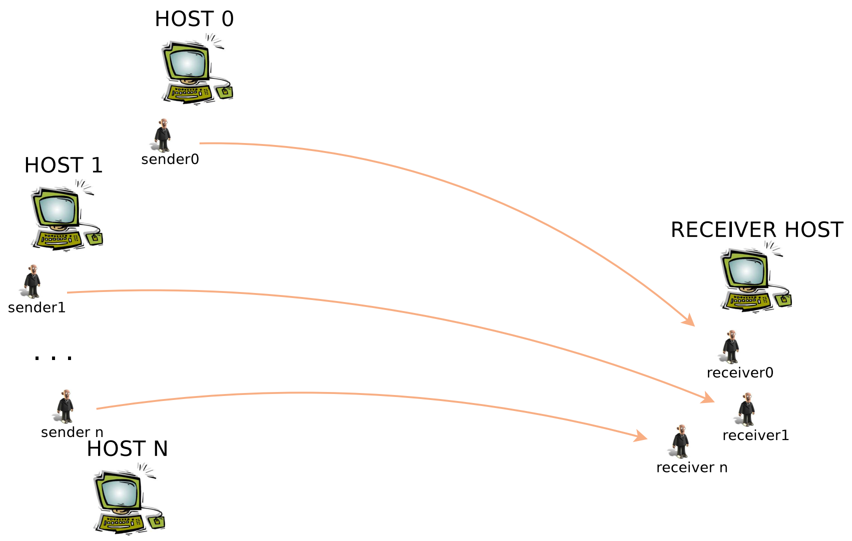

Fig. 3. Benchmark 3, Massive Reception (N receivers)

The multiagent system architecture designed for this benchmark is similar to benchmark 2. There is a host running a controller agent and a host running several receiver agents. Each of the remaining platform hosts is running a sender agent. The number of hosts running a sender agent must be provided by the user. Each sender agent is placed in a different host but the receiver agents are launched in the same host. Each sender agent has a specific receiver agent associated to it, i.e., each sender exchanges all its messages with a single receiver. Figure 3 shows this test.

3.4 Benchmark 4: Number of Agents per Host

The last experiment we propose is designed to check the scalability of the evaluated platform by increasing both the number of launched agents and the message exchanges between them. Not every platform design can offer support to several agents. In this sense, we propose increasing the number of launched agents in the same platform in order to observe its scalability and efficiency. Rather than designing idle agents, we use agent pairs that exchange messages with each other. The result of this experiment should show us how the platform performance evolves in large-scale systems.

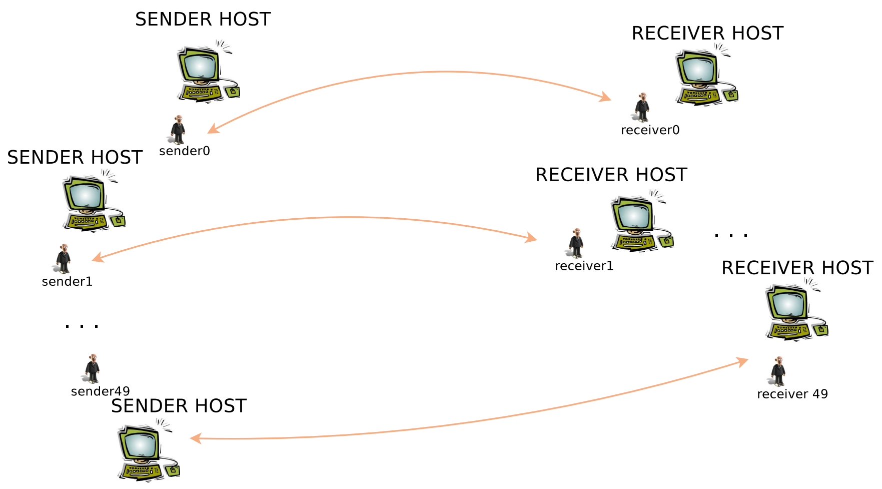

Fig. 4. Benchmark 4, Number of Agents per Host (one agent per host)

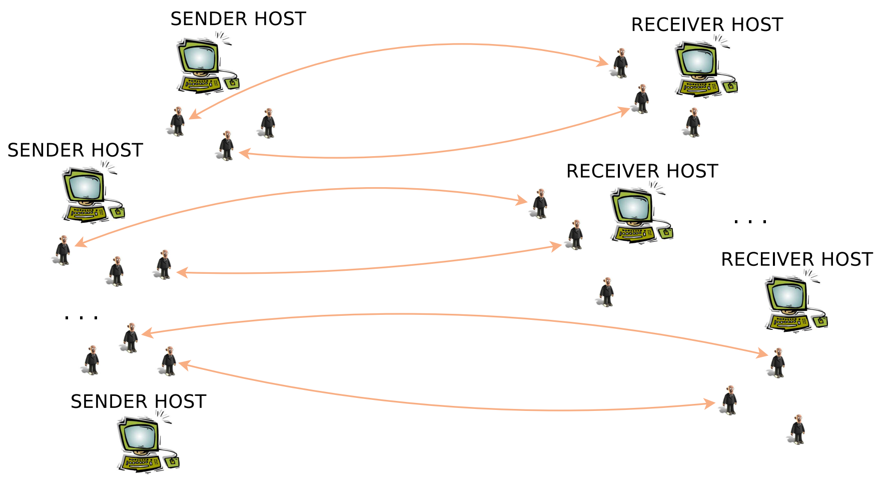

Fig. 5. Benchmark 4, Number of Agents per Host (several agents per host)

This benchmark design is a little different from the other three. The parameters needed are the number of platform hosts and the number of agents per host. We divide the number of hosts into two equal groups: the first half are for launching sender agents and the second half are for launching receiver agents. Each sender agent has a single receiver agent associated to it to exchange its messages.

Figures 4 and 5 show the benchmark when launching one agent per host and when launching several agents per host, respectively.

4 Benchmark Application

This section illustrates the use of the benchmark set. The aim is to evaluate the overhead added to the Jade platform when the security add-on (called Jade-S in this paper) is used to sign the messages exchanged.

The signature process is known to have a performance penalty. Because of this, we want to know the perfomance and scalability degradation when signatures are used. By comparing the standard version with the secure version, the influence of adding this security add-on can be determined empirically.

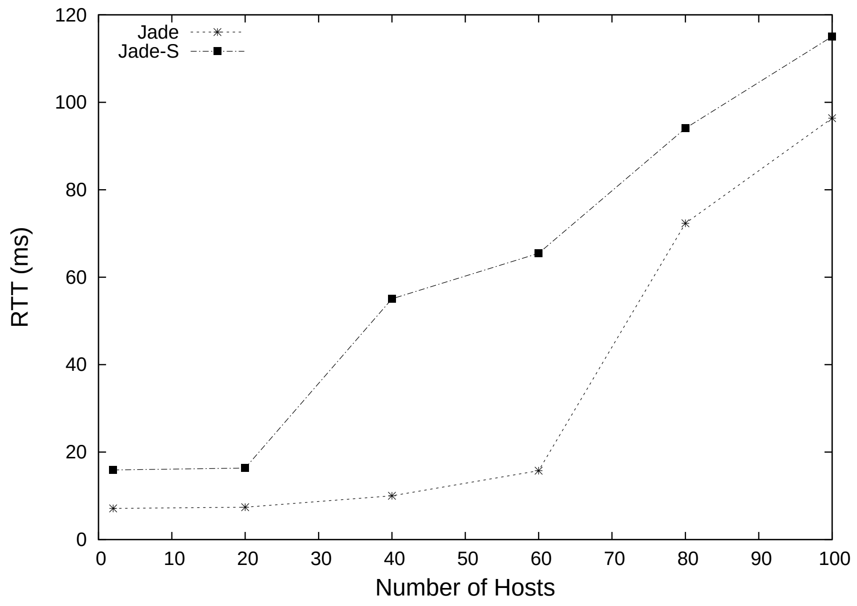

Fig. 6. Benchmark 1: Number of Hosts

We tested Jade and Jade-S on 100 PCs at our university laboratories. These PCs are connected with each other by a Fast Ethernet. Each PC has an AMD processor of 2.8 GHz and 512 MB of RAM Memory. We used Sun JDK 1.5, Jade version 3.4, and the security add-on for Jade version 1.6.

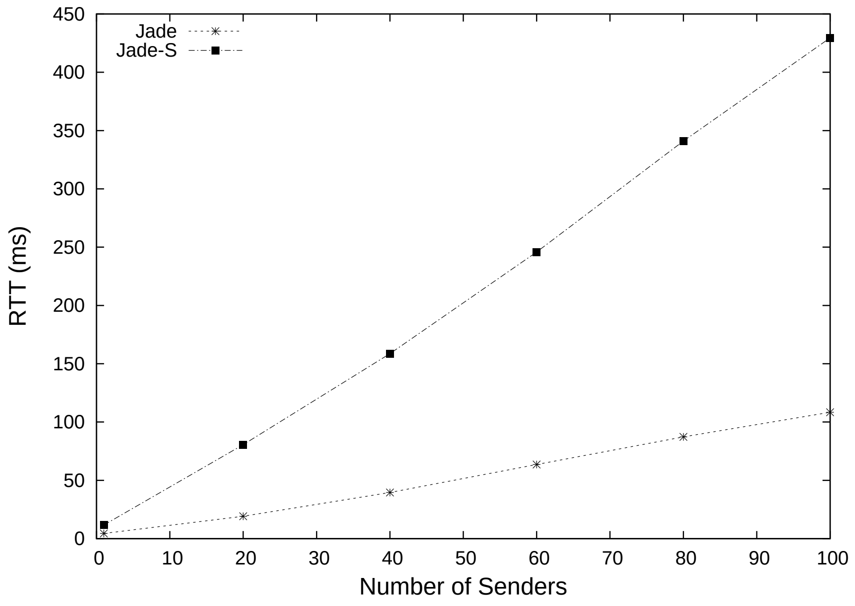

Fig. 7. Benchmark 2: Massive Reception (1 receiver)

For all four benchmarks, the number of sent messages per sender agent was set to 1000 messages, and the size of the exchanged messages was set to 10 bytes. We had previously checked that these parameters provided us with results that were sufficiently accurate.

Figure 6 shows the behaviour of the Jade and Jade-S platforms when benchmark 1 was launched. The number of hosts was increased in each experiment, from 2 up to 100. As one expected, Jade-S performed worse than Jade. An extra overload was included due to the message signature. However, this overload does not seem to be related to the number of hosts. Therefore, if fewer agents are located in the same host, and all platform hosts have similar message traffic, it can be concluded that the overload introduced when signing messages remains nearly constant.

The results obtained for benchmark 2 are depicted in Figure 7. We tested the platform using this benchmark when there is only one sender host. We also increased the number of sender hosts up to 100 (20 more sender hosts each time). This test shows that the higher the number of sender agents, the larger the difference between Jade and Jade-S. This is due to the fact that there is only one receiver in the platform. This receiver agent has to carry out the signature process when replying to all the sender agents in each experiment. As a result,

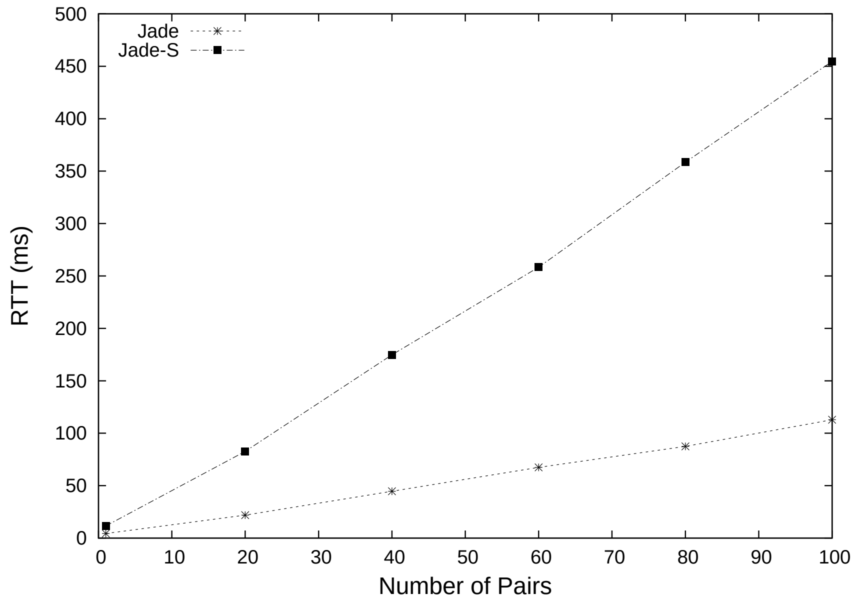

Fig. 8. Benchmark 3: Massive Reception (N receivers)

if a large-scale multiagent system with an agent offering a popular service is being deployed, signing messages could introduce a great overload making overall performance of the multiagent system worse.

Figure 8 shows the response of Jade and Jade-S platforms for benchmark 3. We started the experiment with 1 sender host (plus 1 receiver host) and then increased them 20 at a time up to 100. We increased the number of receiver agents according to the number of sender agents. When this benchmark was launched in a Jade platform, the results obtained were similar to the ones obtained when benchmark 2 was launched. This is due to the way that Jade implements communication among all the platform hosts. The bottleneck created when an agent is overloaded by a lot of message exchange is caused by the message transport system and not by the way the message queue is managed by the agent itself. As mentioned in section 2, we are currently testing other platforms, and we obtain different results when benchmarks 2 and 3 are launched on these platforms.

Finally, Figure 9 depicts the results obtained when running benchmark 4. In order to measure the scalability degree of each platform, we propose launching one agent per host and increasing the number of agents up to 20 per host. As in benchmark 1, the overload introduced by signing messages is nearly constant and does not depend on the number of agents per host. Thus, it can be observed that the overload is almost the same when there is one agent per host (100 agents in the multiagent system) and when there are 20 agents per host (2000 agents in the multiagent system).

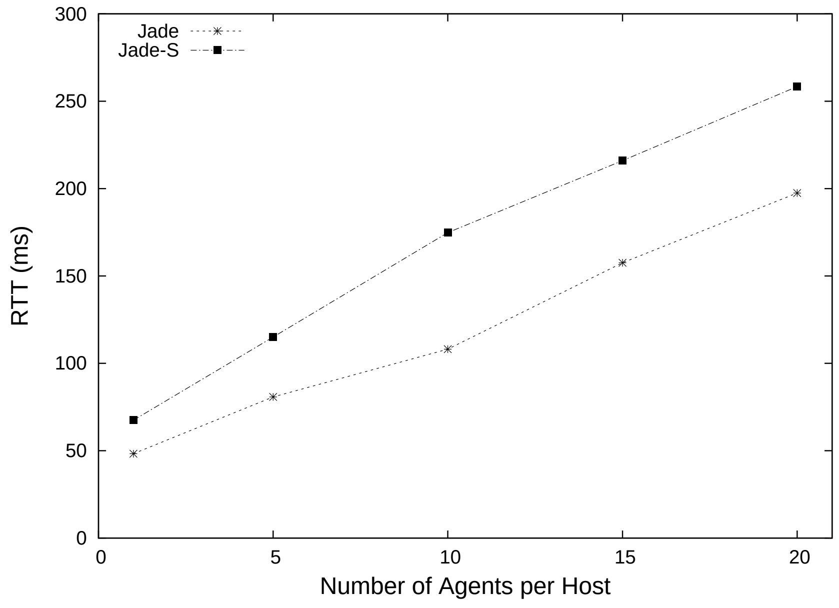

Fig. 9. Benchmark 4: Number of Agents per Host

When the results obtained in our four benchamrks for Jade and Jade-S platforms are analyzed, it is easily observed that, if the multiagent system being secured with digital signatures is distributed in a good way, the extra overload is constant. Therefore, there cannot be an agent whose services are more frequently requested than the others. If this occurs (as shown in the results for benchmarks 2 and 3), when the number of requesting agents is increased, performance will be worse.

5 Conclusions

In this paper, we have presented a set of benchmarks for testing large-scale multiagent platforms.

This set of benchmarks can be helpful for MAS developers to:

- Choose a suitable platform according to their needs.
- Set up options for the selected platform.
- Design the multiagent system (number of hosts, service distribution, etc.).
- Predict multiagent system performance on newer versions of the platform used.

It can also be helpful for multiagent platform developers to:

- Choose between implementation alternatives.
- Evaluate performance.

– Detect implementation errors (overflows, unexpected behaviours, etc.).
– Detect bottlenecks in extreme situations to improve platform implementation.

The main purpose of this paper is to provide a test bed for analyzing the most critical parameters when a a large-scale multiagent system is implemented on a multiagent platform. The way a multiagent platform implements its messaging service is key issue for the behaviour of a large-scale multiagent platform, mainly due to the huge number of interactions in applications of this kind. Our previous work in evaluating multiagent platform performance has allowed us to identify the main parameters that can affect message service performance so that extreme configurations (many agents, many hosts, etc.) can be tested when launching the benchmark set.

The design is general enough to be applied to any platform. We only define the interactions among the agents and the number of agents and hosts in each experiment. We do not focus on any specific platform API or any agent model. Moreover, we do not measure efficiency taking into account agent-internal processes, so, a future work will include not only message sending but also processing messages at high level, i.e. internal reasoning, joint planning, etc.

Finally, an application of this set of benchmarks to Jade and Jade-S is shown as an example. When message traffic is distributed in the same way along the platform hosts (benchmarks 1 and 4), the overload introduced by Jade-S is nearly constant. However, when this message traffic is focused on one host (benchmarks 2 and 3), the more agents there are taking part in the experiment, the greater the overload introduced by Jade-S with respect to Jade. Therefore, the results obtained may be helpful when migrating a MAS from Jade to Jade-S, allowing developers to ascertain how many hosts may be added and/or how the agents should be distributed among the hosts in order to reach the same level of performance.

References

[BCPR03] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa. Jade a white paper. *Telecom Italia EXP magazine*, 3(3):6–19, 2003.

[BGNT] K. Burbeck, D. Garpe, and S. Nadjm-Tehrani. Scale-up and performance studies of three agent platforms. In *IPCCC 2004*.

[CACM] D. Camacho, R. Aler, C. Castro, and J. M. Molina. Performance evaluation of zeus, jade, and skeletonagent frameworks. In *Systems, Man and Cybernetics, 2002 IEEE International Conference on.*

[CFV03] E. Cortese, F.Quarta, and G. Vitaglione. Scalability and performance of jade message transport system. *EXP*, 3:52–65, 2003.

[CTG$^+$04] Krzysztof Chmiel, Dominik Tomiak, Maciej Gawinecki, Pawel Karczmarek, Michal Szymczak, and Marcin Paprzycki. Testing the efficiency of jade agent platform. In *ISPDC '04: Proceedings of the Third International Symposium on Parallel and Distributed Computing/Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (ISPDC/HeteroPar'04)*, pages 49–56, Washington, DC, USA, 2004. IEEE Computer Society.

[JAD] Jade. http://jade.tilab.com.

[LMSW05] M. Luck, P. McBurney, O. Shehory, and S. Willmott. Agent technology roadmap. http://www.agentlink.org/roadmap/, 2005.

[LNW98] L. C. Lee, D. T. Ndumu, and P. De Wilde. The stability, scalability and performance of multi-agent systems. *BT Technology Journal*, 16:94–103, 1998.

[MSA$^+$06] L. Mulet, J. M. Such, J. M. Alberola, V. Botti, A. Espinosa, A. Garcia-Fornes, and A. Terrasa. Performance evaluation of open-source multiagent platforms. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-06)*, volume 1, pages 1107–1109. Association for Computing Machinery, Inc. (ACM Press), 2006.

[Sha05] Elhadi Shakshuki. A methodology for evaluating agent toolkits. In *ITCC '05: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume I*, pages 391–396, Washington, DC, USA, 2005. IEEE Computer Society.

[Vrb03] P. Vrba. Java-based agent platform evaluation. In *Proceedings of the HoloMAS 2003*, pages 47–58, 2003.