Scientific Programming 13 (2005) 1–14 1
IOS Press

Efficiency of JADE agent platform

Krzysztof Chmiel[a], Maciej Gawinecki[a], Pawel Kaczmarek[a], Michal Szymczak[a] and Marcin Paprzycki[b,c]
[a]*Department of Mathematics and Computer Science, Adam Mickiewicz University, Poznań, Poland*
[b]*Computer Science Department, Oklahoma State University, Tulsa, USA*
[c]*Computer Science, SWPS, Warsaw, Poland*

Abstract. Agent oriented programming is often claimed to become the next breakthrough in development and implementation of large-scale complex software systems. At the same time it is rather difficult to find successful applications of agent technology, in particular when large-scale systems are considered. The aim of this paper is to investigate if one of the possible limits could be the scalability of existing agent environments. For this purpose we have selected JADE agent platform and investigated its performance in a number of test-scenarios. Results of our experiments are presented and discussed.

1. Introduction

For a number of years, researchers promise that software agent technology is about to change the ways we construct software [2,3] as well as have a much broader impact on human-computer interactions [4,5]. Some of the principle areas that the software agent technology is expected to impact are [1–5]:

- development and maintenance of complex systems,
- resource management,
- delivery of personalized content,
- e-commerce on a large and small scale.

Obviously, this list is far from exhaustive, however, the very breadth and depth of these areas supports the claim that agent technology, if successful, can become the next "extreme event," leading to a substantial leap forward in a number of fields. The agent paradigm also promises to add a new dimension to our interaction with computers. Here, the promise of being able to deal with the information overload resulting from the exponential growth of information available on the Internet, which has been pledged in the influential work of P. Maes [5], is particularly tempting.

Unfortunately, as it is easy to see, more then 10 years after publication of [5], promises furnished there did not materialize (regardless of the rapidly increasing number of agent-focused conferences, workshops, publications, etc.). To the contrary, it is relatively difficult to point to successful large-scale implementations of agent systems (as understood in [1,2,5]). Moreover, what is particularly revealing, agent systems described in [5] as successful implementations of agents, for one reason or another, have never spread beyond the MIT Media Laboratory. There are many possible reasons for this fact and some of them have been discussed in [11,18], but there exists also a different possibility, and very simple one indeed.

The starting point for this paper was an exchange of messages in one of electronic discussion groups devoted to a particular agent platform. One of the participants described an e-commerce system under development. In this system a personal agent was to be devoted to (instantiated for) each user logged into the system. The question was therefore asked if it is possible to scale his system, implemented using that specific platform, to 500+ agents. The response, from someone who clearly was a practitioner of agent-system application development was that "this is a wrong way of looking into the problem and one should not expect realistically to scale an agent application to this number of agents." We found this response particularly fascinating as it contradicted one of the most basic tenets of agent system development that is constantly permeating the agent-system-literature. There, it is typically stated that an agent system should be designed exactly in such a way that each user should be served by his/her

ISSN 1058-9244/04/$17.00

"own" personal agent (see for instance [3,5]). More generally, it is claimed that when problem is decomposed to be implemented using software agents, then its well defined functionalities (e.g. agent deployment managers, sellers, database managers, watchers, buyers etc.) should be conceptualized and implemented as independent agents [3,18]. We had therefore to ask ourselves a question: is really the case that while agents are to become the breakthrough in development of software for large complex systems through their functional decomposition into agents, the currently existing agent technology cannot support implementation of large-scale software systems the way that they are supposed to be implemented?

For us, this was a question of agent system scalability. Proceedings with a literature review we have found that there exist a number of papers that discuss various aspects of this problem [6–10]. However, we have also realized that these papers contain an almost philosophical discussion attempting to answer the question "what is agent scalability?" With our research background in computational mathematics, we were interested in a more pragmatic route, where one starts form the basic assumption that "a good agent system is one that is implemented" (see also [11]). In the next step, such an implemented agent system has to be thoroughly tested to establish its performance characteristics for varying number of agents and/or messages etc. In this way we follow and expand work reported in [12,13]. While there, focus was only on agents exchanging messages, we decided to expand the area of interest. Therefore, we studied different scenarios involving messaging but also added tests of efficiency of agent creation and migration.

A methodological remark is in order. Since there exists no "benchmarking suite" to test performance of agent systems (similar to these found in scientific computing; in particular in computational linear algebra), questions about "what should be measured and why" remain unanswered. Scenarios proposed here were not designed to become the missing collection of benchmarks. Rather, we were interested in obtaining a broad understanding as to how our agent platform of choice (JADE) behaves when the number of messages and agents is increasing as well as obtaining some general assessment of efficiency of agent creation and migration. We believed that if results confirm robust scalability (of JADE) then one will no longer be able to make a case for implementing only limited size demonstrator systems. In this way we would strengthen arguments and research program put forward in the highly critical, but inspiring, work of Nwana and Ndumu [11].

To obtain such a general assessment of robustness of existing agent platforms, we have selected one of the best of them: JADE (version 3.1) [13,14] and "stress-tested" it in four scenarios, that can be divided into two groups: the first test studied JADEs message exchanging capabilities (results reported in the next section) and the remaining three of them were focused primarily on performance of agent creation and/or migration (results reported in Section 3). Work presented in this paper is an extension of results reported in [19]. Here, follow-up tests have been run for the tests that were carried over and a new test scenario has been introduced.

2. Performance of agent message exchanges

In this section we present results of a test that was aimed at messaging capabilities of JADE. This test differs from, but follows in spirit these reported in [12]. The main rationale is that in agent-based systems, when during design and implementation functionality is divided into agents (each agent is responsible for a particular sub-function of the system e.g. search agent, query agent, database wrapper agent etc.) it is typically assumed that these agents coordinate their actions (and communicate for all other purposes) by exchanging messages [3,18]. Assuming that a (very) large number of agents are to be used (which is a fundamental assumption made across this paper), a large number of messages have to be exchanged between them. We have therefore tried to establish the message-load robustness of the JADE agent platform. An additional test involving agent messaging combined with database access was reported in [19] and its results concur with our findings presented here.

2.1. Spamming test

Our messaging test is very simple and is designated to flood the system with *Agent Communication Language* (ACL) messages [16] (the ACL is the FIPA standardized language used for agent-agent communication). We have created a system with two types of agents: (1) *spammer* agents that send a large number of messages to (2) *user* agents that have to receive and "process" them. The current version of JADE supports only agent systems consisting of a single *platform* (environment in which all agents "exist," move and communicate) with multiple containers ("places" where agents reside and between which they move); agent migration between platforms is not permitted.

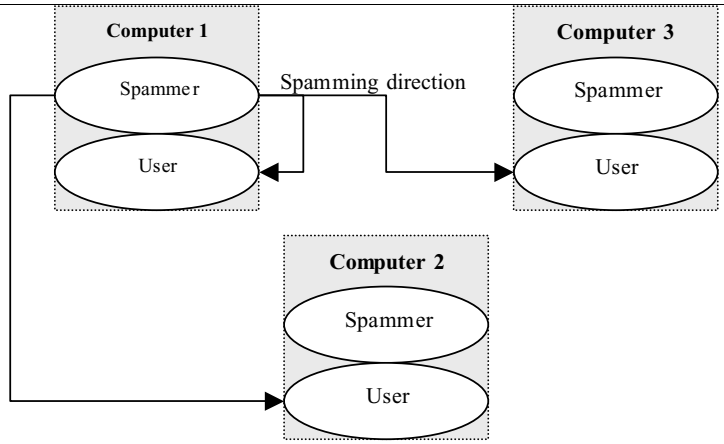

Fig. 1. Spamming scheme.

Since agent systems are to be developed in the context of networked computers we have developed our test in such a way that separate containers were located on separate machines (one container on one computer). In the initial test setup, each JADE container hosted a pair: one user agent and one *spammer* agent. At a given time all *spammer* agents started sending messages to all user agents (including these residing within their own containers/computers). The general scheme of interaction between three *spammer* agents and three user agents (located on three computers) is illustrated in Fig. 1. Here, a *spammer* agent from Container-1 (located on Computer 1) sends messages to user agents residing within Containers 1, 2 and 3 (located on Computers 1, 2 and 3). Similarly *spammer* agent residing within Container-2 (located on Computer 2) sends messages to *user* agents residing within containers 1, 2 and 3 etc. Note that, within the JADE platform, all posted messages are put in a receivers' message queue [14,15] and then they are processed by the receiver (see Fig. 2) and thus message spamming is rather expensive from the point of view of memory utilization.

To measure the performance we utilized a starter agent which initiated the spamming process and measured processing times. During the execution of the test, each spammer agent "broadcasted" a certain number of messages and the total time of this broadcast was measured. Separately the time of processing of all messages flooding the system (from the time that the start command was released, until the last agent completed processing of its last message) was also measured.

The first series of tests was performed on 8 Sun workstations, each with an UltraSparc III processor running at 300 MHz and 192 Mb of RAM. All these machines were Internet-connected through a Cisco switch with full backplane 100 Mbits/s transmission rate. We have used ACL messages with content consisting of 300 ASCII characters. A total of 5000 messages were sent by each spamming agent to each user agent. Experimental results are summarized in Table 1 and Fig. 3.

Table 1
Message sending and receiving times

Agent pairs	Spamming time (ms)	Receiving time (ms)
2	40034	87053
3	24440	141778
4	25128	217501
5	25217	313625
6	28843	448181
7	35164	634847
8	40624	821341

A number of observations can be made. (1) For the (relatively small) number of computers used in this test the total "spamming time" practically does not depend on the number of recipients (Fig. 3) and it can be related to the broadcast command used to send messages around. It is only after the total number of spamming agents becomes larger than 5 when the total spamming time starts to increase. (2) As the total number of agent pairs increases, the receiving time starts to increase immediately. (3) When each of 8 *spammers* sent 5,000*8 = 40,000 messages; resulting in a total of 320,000 messages flooding the system (with each message size being slightly more than 0.3 Kbytes – message and its ACL wrapper – totaling approximately 100 Mbytes of data), user agents were able to process them in no more than 14 minutes.

In the second series of experiments we have changed the setup in such a way that in each container located

Table 2
Message sending and receiving time; varying number of computers, message size and computer mixture

Number of containers and computers used	Message size (characters)	Spamming time (ms)	Receiving time (ms)
2 (2 pc)	200	1610	2954
2 (2 pc)	400	1266	2344
2 (2 pc)	800	1110	2141
4 (4 pc)	200	2782	7798
4 (4 pc)	400	1688	6517
4 (4 pc)	800	859	6579
8 (8 pc)	200	3985	26456
8 (8 pc)	400	2063	23284
8 (8 pc)	800	2157	23019
10 (8 pc + 2 sun)	200	21909	201380
10 (8 pc + 2 sun)	400	15096	200230
10 (8 pc + 2 sun)	800	31838	198812
12 (8 pc + 4 sun)	200	24530	290579
12 (8 pc + 4 sun)	400	23344	289528
12 (8 pc + 4 sun)	800	18719	291536
14 (8 pc + 6 sun)	200	26701	398910
14 (8 pc + 6 sun)	400	22437	393028
14 (8 pc + 6 sun)	800	30484	398046

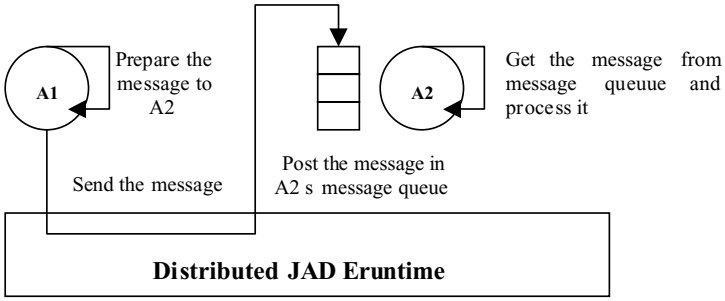

Fig. 2. JADE message processing scheme.

on each computer we have instantiated three spammer-user agent pairs (the remaining parts of the test scenario were unchanged); see Fig. 4. This time we have utilized 6 Sun workstations and 8 PC's with Pentium 4 processors running at 1.6 GHz, while each computer had 256 Mb of RAM. In our experiments we have varied the message size and experimented with homogeneous and heterogeneous environments. The results are summarized in Table 2.

The results are quite interesting. (1) Overall processing time does not seem to depend too strongly on the message size; at least in the range of messages that we have used. Here, we have conjectured that 800 ASCII characters represent a rather large message for a standard, communication oriented, message exchange. (2) This time we have run our experiments in batches, where sending messages of size 200 was followed by sending messages of size 400 and 800. We believe that this process explains in part the drop in message sending time (some form of process initialization takes place within JADE). We do not have, however any reasonable explanation for the further drop of message sending time between messages of size 400 and 800 that occurred in some cases. (3) Overall, in the largest case we were able to process: (3×14)*(3×14)*50 $= 135200$ messages; for a total of more than 108.2 Mb of data; in approximately 6 minutes. (4) This latter result, combined with these obtained in the first series of experiments confirms that *JADE scales very well when agent-to-agent messaging is concerned*.

3. Performance of agent creation and migration

The second group of tests was designated to test the support that JADE provides for creation and migration of a large number of agents. This is in response to potential of utilization of agents in implementation of sys-

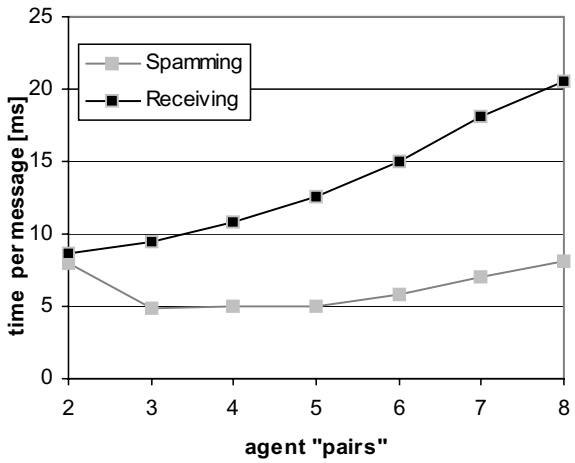

Fig. 3. Average message sending and receiving times for 2–8 machines (and thus spammer-user pairs); calculated by dividing total times by a number of sent messages.

tems in context of which agent mobility is considered to be particularly attractive.

3.1. Agent migration

The first experiment was focused on pure agent migration and was mimicking a relay-race. As previously, a fixed number of containers were placed on separate computers. Each container constituted a "place" where agent runners "exchange batons." The "race" starts in JADE's "Main-Container" and participating agents move to 'Container-1" (on a different computer). There agents "pass the relay baton" by exchanging ACL messages with agents awaiting them and then stop and wait for the next time when they will become runners. In the meantime, the second group of agents (these that have received message-batons), proceeds to the "Container-2" (on the next computer). Then the process repeats.

In the first series of experiments the complete race consisted of 5 laps and ended when the last agent completed the last lap and reached the "Main-Container." Here, two tests were performed. In the first one, four runner teams were migrating within an increasing number of containers/computers.

For the homogeneous setup each container was placed on a Sun workstation (as described above). We have also used two PC's with a Pentium 120 MHz processor and 48 Mb of RAM to create a heterogeneous configuration consisting of 4 Suns and 2 PC's. In Figs 5 and 6 we depict the total migration time for four agent groups and increasing number of containers.

In the second test, an increasing number of runner teams migrated around four homogeneous computers/containers, or around a heterogeneous setup consisting of two Sun workstations and two PC's. Figures 7 and 8 illustrate the results of increasing the number of agent-teams while keeping the number of containers constant.

In the third series of experiments we have decided to substantially increase the number of agent teams. In Fig. 9 we present the total race time for the homogeneous environment consisting of 4 and 8 Sun workstations and 20, 40, . . ., 100 agent teams. The results for 4 workstations are very similar to these in the case of a small number of agents (as the number of agent teams increases, race time increases almost linearly). The situation changes considerably for 8 machines. Here, we were not able to complete the experiment for the largest case (100 agent teams) due to the "out of memory" error. This is likely to explain the faster increase of race time for more than 40 agent teams per machine. As memory becomes a scarce resource, total execution time increases more rapidly. Observe, however, that even in this case, for 40–80 agent teams, the increase of race time is almost linear again.

In the next series of experiments, we have scrutinized 20 runner-teams migrating around 8 Sun Workstations for an increasing number of laps. Results are summarized in Fig. 10 and, as expected, the race time is almost linear again.

In the final series of experiments we have used 4, 6 and 8 PC's with Pentium 4 processors running at 1,6 GHz, with 256 Mb of RAM each. We have experimented with 20, 40 and 60 runner-teams completing 3 laps. The results are summarized in Fig. 11; and, again, an almost linear behavior can be observed.

Overall, regardless of the test, we have observed behavior that, for all practical purposes, should be regarded as linear scaling (the case of running out of memory does not contradict this general assessment). Note also that JADE did not collapse even when a total of 640 agents (80 agent teams on 8 machines each) were residing and migrating in eight containers on eight computers with a relatively small amount of available memory (196 Mbytes).

3.2. Shop performance – agent flooding

In this scenario we wanted to test the performance of JADE when one of its nodes (a JADE container) was flooded by a mass of agents migrating into it from other machines within the platform. As an example of such a situation we have implemented a simplistic simulation of an e-commerce scenario. Here, one of

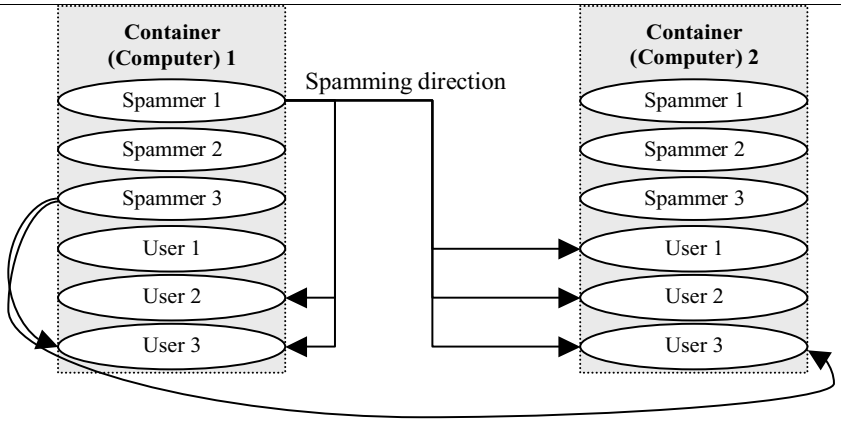

Fig. 4. Spamming scheme for second set of experiments.

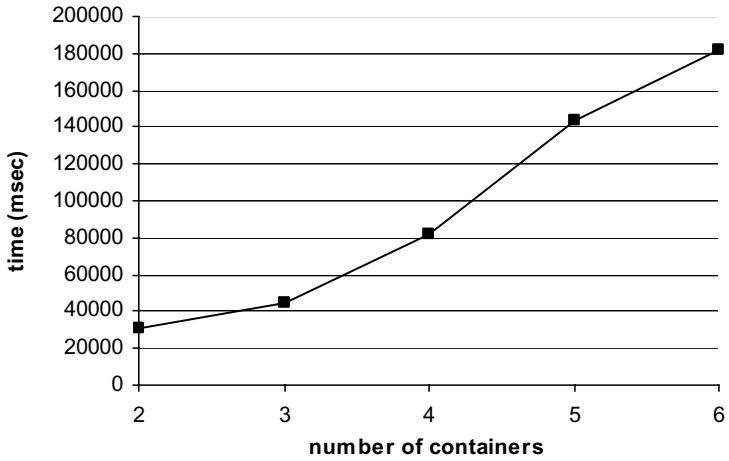

Fig. 5. Total migration time; heterogeneous environment; 4 agent teams; increasing number of containers.

the computers represented an e-shop. This machine (named *Server*) contained the *MotherShop* (*MS*) agent (sales manager) whose task was to create a *Seller* (*S*) agent for each *Client* (*C*) agent which visited the e-store and requested to be served. The *Seller* agent and the *Client* agent "negotiated" (exchanging ACL messages), about goods and prices (in our case the "negotiation" had the simplified form: "do you have beer?," "yes," "please give me one," "here you are," "thank you") and when the negotiations were completed, the *Seller* agent sent to the *MotherShop* agent the results of this process and self destructed. At the same time, the *Client* agent moved back to the container in which it was created to report to its *MotherClient* (*MC*) agent (agent which generated and sent it to the e-store) and after completing a report self-destruct. Here, the *MotherClient* agents were extremely simplistic. They just generated a given number of *Client* agents and sent all of them to "flood" the e-shop and then waited for their return. This overall schema of operation is similar to that described in more detail in [20] and is depicted in Fig. 12.

In the first series of experiments, we have, again, experimented on the same network of Sun workstations. Each *MotherClient* agent was located within a different container located on a different machine, while the shop was also located in a separate container on a separate computer. Thus, in the largest reported case, 5 *MotherClient* agents were located on 5 machines, while the *MotherShop* was located on the sixth workstation. The experimental results (in milliseconds) of system processing 30, 40, 50 and 120 agents generated by 1, 2, . . ., 5 *MotherClient* agents (in the largest case a total of $5 \times 120 = 600$ agents flooding the system) are depicted in Fig. 13.

As previously, the processing time is almost linear. We have experimented also with a very large number

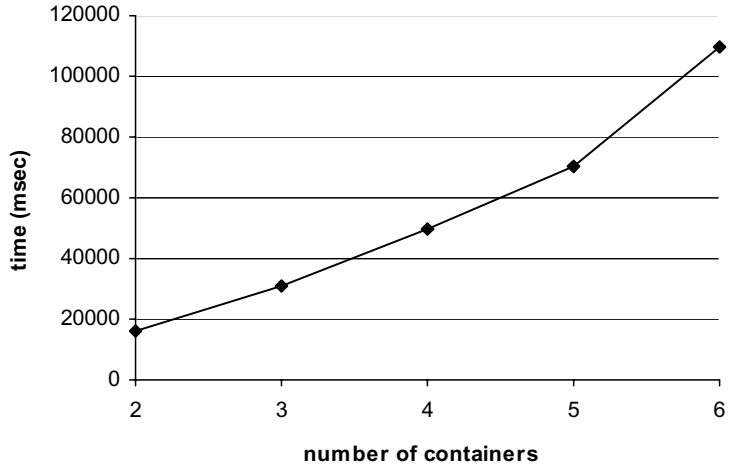

Fig. 6. Total migration time; homogeneous environment; 4 agent teams; increasing number of containers.

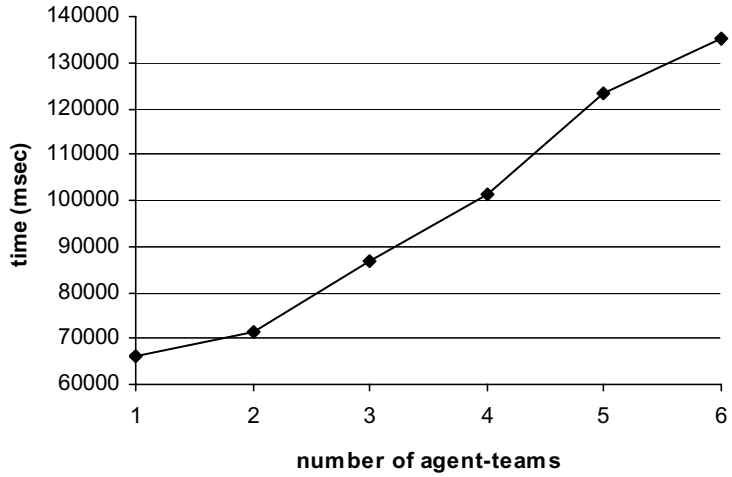

Fig. 7. Total migration time; heterogeneous environment; 2 Sun stations + 2 PCs; increasing number of agent-teams.

of agents in the system and we have found that the processing time was still approximately linear e.g. for 520 *Client* agents the processing time was approximately 75 seconds, while for 1020 *Client* agents time increased to approximately 136 seconds. However, we have also found that at approximately 1430 *Client* agents Java generated an "out of memory" exception. To verify the connection of an amount of RAM and numbers of agents we have made a test on a weaker configuration which consisted of PC's with 4 times smaller amount of available RAM (48 Mb). In this test the exception appeared after generating approximately 370 agents; again an almost linear relationship.

In the second series of experiments we have made our comparisons on two configurations consisting of PCs. The first configuration (named *weak*) consisted of the *MotherShop* agent residing within a container on a PII 350 MHz with 128 Mb RAM while the *MotherClient* agents resided on 4 PCs with 166 MHz Pentium processors and 48 Mb of RAM and 5 PCs with 350 MHz PII processors and 128 Mb RAM each. The second configuration (named *strong*) consisted of a homogeneous environment of PC's with Celeron 2.00 GHz processors and 256 Mb of RAM each. In Figs 14–16 we present the results for 10, 20 and 40 *Client* agents generated on 1, 2, . . ., 9 *MotherClient* agents.

In Fig. 16 we observe a sudden jump in processing time when $8 \times 40 = 320$ agents flooded the *weak* configuration system. Up to this moment (and in previous experiments- Figs 14 and 15) we observed an almost linear growth of processing time. We were able to establish that the jump in processing time was caused by

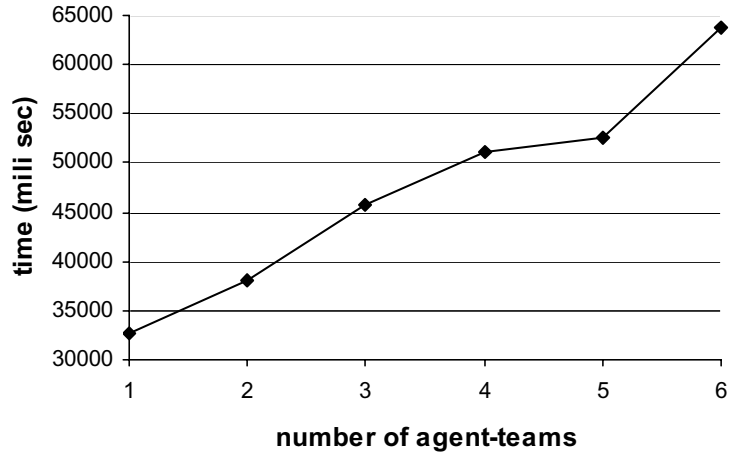

Fig. 8. Total migration time; homogeneous environment; 4 Sun stations; increasing number of agent-teams.

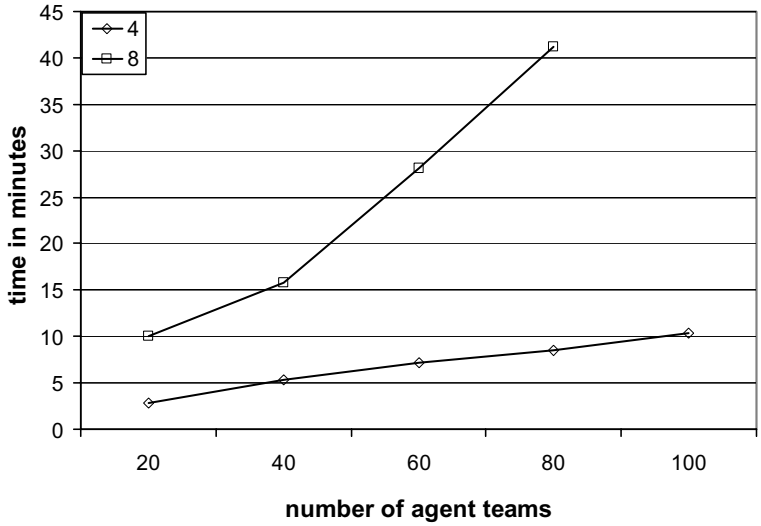

Fig. 9. Total migration time; 4 and 8 Sun workstations; large and increasing number of agent-teams.

near-exhaustion of memory available within the *weak* configuration (note that the *strong* configuration processed, without any problems, total of $40 \times 9 = 360$ agents). To establish limits of both configurations we have initiated experiments with 160 agents released by each *MotherClient* agent. Here we have clearly reached limits of what JADE can reliably process on our computers. The system simply started to crash randomly for large number of MotheClient agents. More precisely, on in the case of the weak configuration we were able to repeatedly and reliably complete work with up to 3 *MotherClient* agents (total of $3 \times 160 = 480$ agents). In the case of the strong configuration we were able to reliably complete work when up to 5 *MotherClient* agents flooded in the system (total of $5 \times 160 = 800$ agents).

Finally, it is rather revealing to compare the results obtained on the PC's running JADE on top of Windows with these obtained on Sun workstations running JADE on top of Solaris. We were able to process 1000+ agents on Sun workstations with 198 MB of RAM and only 800+ agents (of the same size etc.) on PC's with 256 MB of RAM.

3.3. Geospatial data conflation performance

The last scenario that we have experimented with originates from [13]. There an agent system devel-

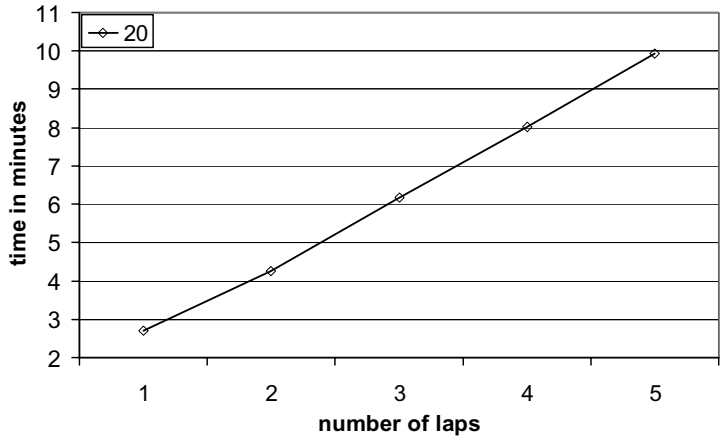

Fig. 10. Total migration time; 8 Sun workstations; 20 agent teams; increasing number of laps.

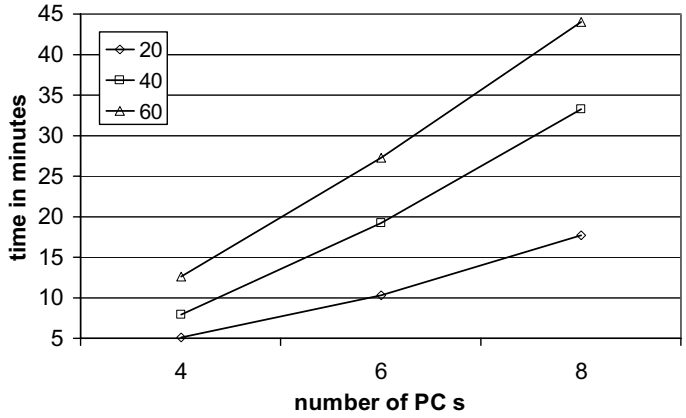

Fig. 11. Total migration time; homogeneous environment; 20, 40, 60 agent teams increasing number of PC's.

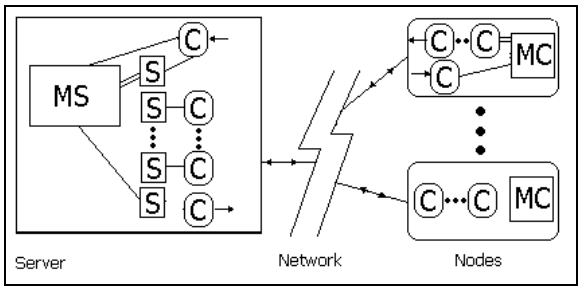

Fig. 12. Agent shop experiment; MS – MotherShop agent, S – Seller agent, MC – MotherClient agent, C – Client agent.

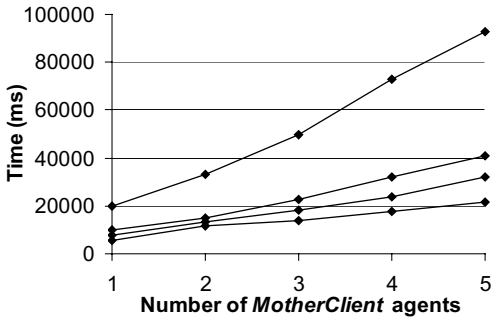

Fig. 13. Agent shop experiment; 30, 40, 50 and 120 agents generated by each MotherClient agent; increasing number of MotherClient agents.

oped to support conflation of geospatial data originating from multiple heterogeneous data sources was presented. An initial skeleton of system was implemented using Grasshopper agent environment [21] and tested vis-à-vis a client-server architecture. Since this is an interesting application of mobile software agents we have implemented, in JADE, its somewhat simplified version. Let us start form a very brief description of the application. For more details as well as background information reader should consult [13] and references

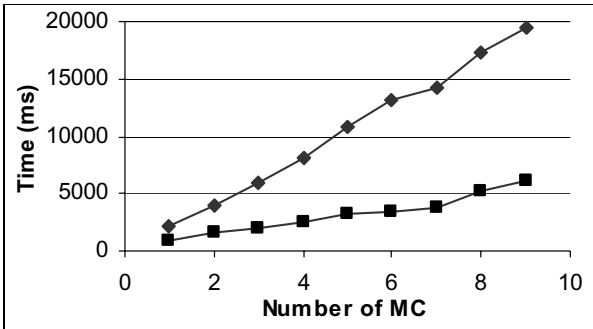

Fig. 14. Agent shop; 10 agents generated by each MotherClient agent; weak and strong configuration.

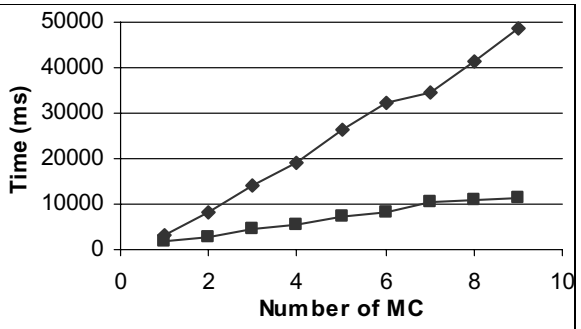

Fig. 15. Agent shop; 20 agents generated by each MotherClient agent; weak and strong configuration.

presented there. Here, Fig. 17 represents a high-level overview of the system, which consists of the following agents:

- *ROI agents* (*RA*): responsible for managing updates for a particular *Region of Interest* (*ROI*). For instance, the United Nation divides the world into ten such regions. *RAs* are static and remain connected to the central database during the entire process.
- *Conflation Manager* (*CM*): a static agent, located in the central database and is responsible for generating a conflation agent for each update request entry in the queue and initiating the conflation process.
- *Queue Manager* (*QM*): agent responsible for supervising a priority queue of updates generated by the *RAs*.
- *Conflation Agent* (*CA*): generated by the *CM*, is responsible for a single update request. *CAs* travel to the data repositories (feeder databases) to perform conflation in a round robin fashion (described below).
- *Query Agent* (*QA*): released from the central database by the *CAs* to gather information for conflation-related queries. Prior to the *CA*, *QAs* arrive in all pertinent databases (for each request a subset of all feeder databases), perform initial queries and post the result to be used in the conflation process by the *CA*. The *QA* cooperates with the Wrapper Agent (*WA*) to translate the data into the common data format of the system.
- *Change Detection Agent* (*CDA*): small and "unintelligent" agent that logs changes to the database, queries for necessary information and interact with the WAs to create an update object for transfer to the *RAs*.

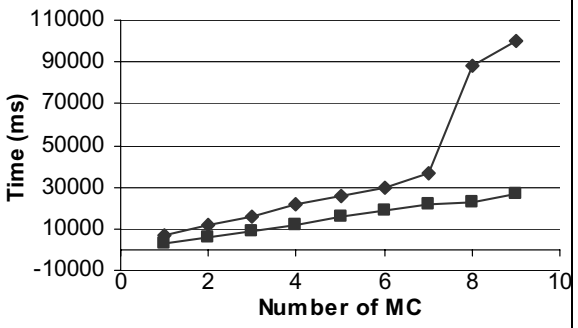

Fig. 16. Agent shop; 40 agents generated by each MotherClient agent; weak and strong configuration.

- *Wrapper Agent* (*WA*): responsible for translating different geospatial data formats to the common data model of the system.

Let us now describe a typical scenario of autonomous management of geospatial data. Here, we assume that initially data in the feeder databases is synchronized with the central database. Suppose now, from Fig. 17, that an update to an attribute value of a spatial feature is performed on the *DB1* database. The resident *CDA* notes the change and queries the database for the information needed for an update object, namely, the database identifier, the feature ID and type information, bounding box coordinates, and type of update performed (metadata, attribute, topology, geometry or unknown). This information is given to the *WA* to create the update object, which is then sent to the proper *RA* in the central database (flows 1, Fig. 17). At this moment, the *WA* can self-suspend, while the CDA returns to watch the database updates.

When the update object arrives at the central database, it is placed in a quad-tree that contains predefined region objects represented by the *RAs*. Each *RA* manages updates and when it determines that it is

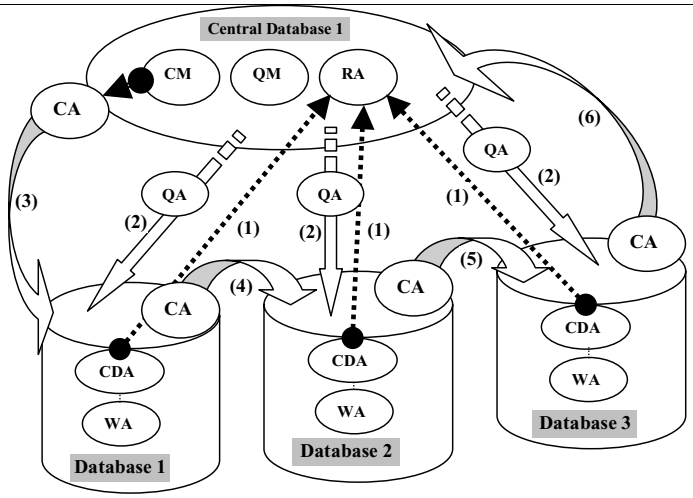

Fig. 17. The general multi-agent architecture of the conflation system.

time to check for updates, all the update objects are placed in a conflation queue managed by the *QM*, from where the *CM* agent removes the highest priority update request from the queue and initiates the conflation process by generating an appropriate *CA*. The *CA* generates *QAs* to retrieve data needed for performing conflation on that object. The *QAs* are sent in parallel to all the feeder databases in which a potential conflict with the update exists (flow2, Fig. 17), and begin the data extraction process. *QAs* communicate with the WA agents to translate the data to the common data format of the system and make it ready for the *CA* to start the conflation process.

The *CA* first moves to the database from which the update originated (flow 3, Fig. 17) and obtains the necessary data form the *QA* agent. The *CA* traverses all of the relevant databases, collecting the information and executing the conflation algorithm. It assembles the conflated data in a Round Robin fashion and then brings the results back to the central database for updates (flow 6, Fig. 17).

For the purpose of our test we have reduced the scope of the system. In particular, *CAs* were created directly by the *RA* and communicated with WAs and the RA without mediation of *CDAs*, *QM* and *QAs*. In Fig. 18 we present an output generated by the JADE provided Sniffer agent in the case of two regional databases, i.e. two WA agents named C1 and C2 residing in Containers 1 and 2. The illustration shows messages sent for two updates in local databases. First of them requires intervention of one *CA* and two are needed in the second case.

For our experiments we have used the above described Sun workstations and, in the second series of experiments, PC's with P3 processors running at 2G Hz and with 256 MB of RAM each and connected by a 100 Mbytes/s switch. In all cases, separate instances running MySQL database were used and separate JADE containers were located on separate machines. To test the performance of JADE we have set the *Central Database* on one of the computers. The remaining computers hosted "feeder databases." Then we have observed time of completion of a single round of system operation while increasing the total number of *Conflation Agents*. In the first experiment, we have used 7 and 8 Sun workstations (6 and 7 feeder databases) and were able to increase the total number of conflation agents to 250. For 300 conflation agents the system has ground to a halt and did not produce results within a couple of hours. The results are summarized in Fig. 19.

Interestingly, we observe here a similar behavior as we did for the experiments with spamming, where the early increase of "problem size" resulted in decrease of total completion time. We believe that we have run into the same problem as in the case reported in Section 2.

In the second series of experiments, we have used 1 through 7 PC's running feeder databases and increased the total number of conflation agents from 50 to 1000. The results are summarized in Fig. 20 and, again illustrate an almost linear scalability of the system. The only exception is when we run the system with one central database and one feeededr database (an extreme case). Here, we were not able to reliably complete execution of the system with more then 200 conflation agents. Since we were able to complete execution of the system with 1000 agents, we have decided to use 7 PC's running feeder databases and increase

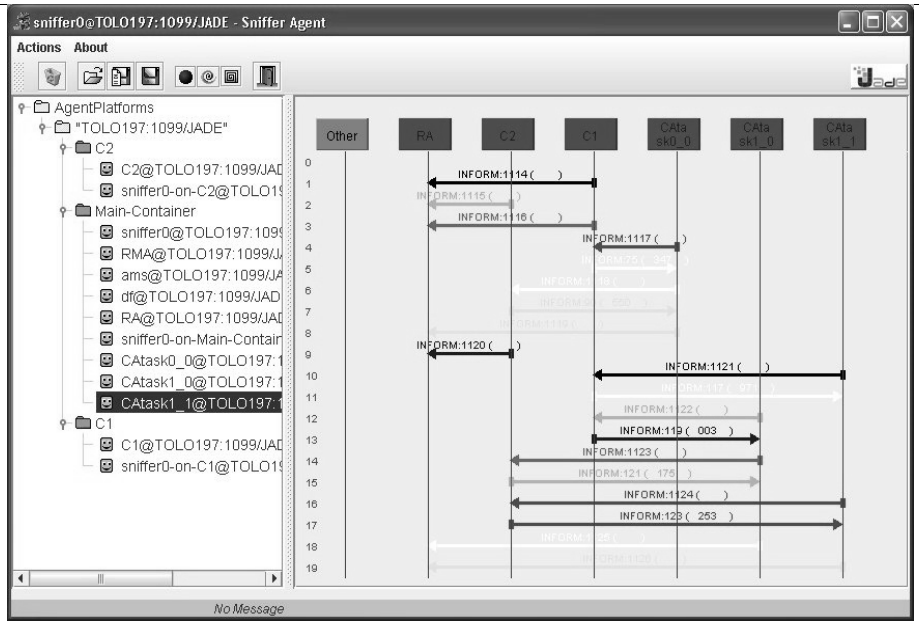

Fig. 18. Sniffer agent output. 1–2. local database registration; 3. notice of update in C1, RA creates CAtask0_0 agent; 4. this agent migrates to the container with C1and asks it for an update; 5. C1 queries its database and replies; 6–7. analogical actions of CAtask0_0, this time within container hosting C2; 8. CA returns to Main-Container and reports to RA, the latter performs insert query; 9. notice of update in C2, RA creates CAtask1_0 and CAtask1_1 agents; 10–19 analogical as in the case of actions of the CAtask0_0 agent.

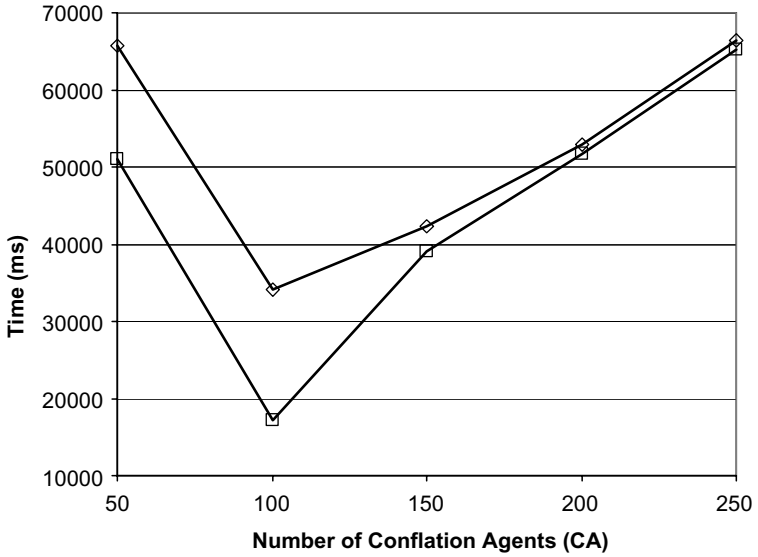

Fig. 19. Geospatial Data conflation, 7 and 8 Sun workstations, increasing number of conflation agents.

the total number of conflation agents until the system crashes. We were able to reliably complete one round of "pseudo-conflation" with up to 2000 agents. The results are illustrated in Fig. 21.

As we can see, even in a relatively complicated scenario, the system scales almost linearly. Furthermore, when 8 computers with a relatively large amount of memory are used, system scales up to 2000 agents. However, during our experiments we were able to find a strong relationship between the behavior of the system and the network support. When performing experiments on a larger network, where more than 60 com-

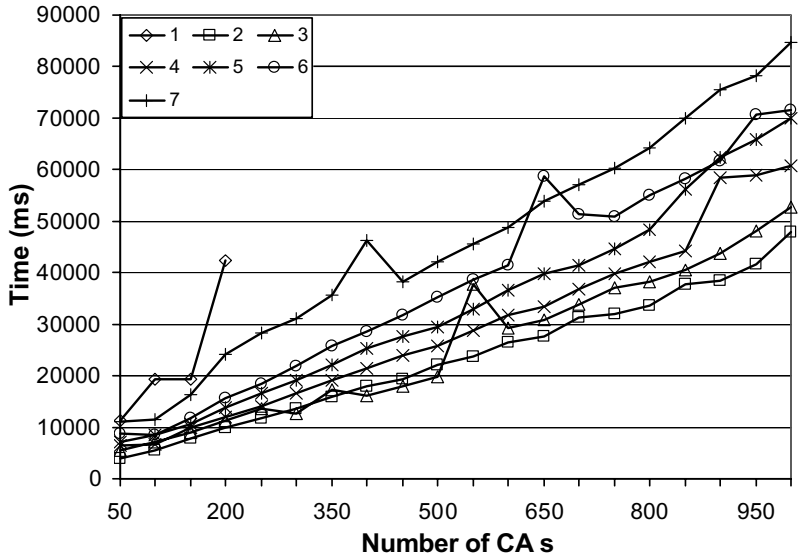

Fig. 20. Geospatial Data conflation, 1–7 PC's, increasing number of conflation agents.

puters were connected to a single 100 Mbytes/s switch, we were often unable to complete experiments with more than 200 agents. The error messages indicate that the Java RMI was incapable of properly handling operations in a highly congested network (local System Administrator indicated similar problems with connectivity even when running Seti@Home, which indicates the level to which the network was congested). After moving back to a smaller (and less utilized) laboratory, we were able to complete tests described above without any problems. Overall, this illustrates, again, that *as far as agent creation and mobility are concerned, JADE scales very well* and the only limitation come form Java and the hardware on which JADE is run.

4. Concluding remarks

The aim of our work was to follow and expand the experimental research presented in [12] (where JADE 2.5 was used). Here, we have used the, most recent, JADE 3.1 and in addition to messaging performed experiments focused on agent creation and migration. Our main goal was to establish if JADE can be used to as a tool supporting the research program put forward by Nwana and Ndumu in [11]. In other words, to find out if JADE can be used to develop and implement large agent-based software systems.

Our tests indicate that JADE is a very efficient environment limited only by the standard limitations of Java programming language, which is interpreted and executed in a Virtual Machine: processor speed, amount of available memory and speed of network connection. The environment itself does not introduce substantial overhead. Executing JADE on a relatively antiquated hardware (PC's with Pentium II processors running at 120 MHz with 48 Mbytes of RAM and workstations with UltraSparc III processors running at 300 MHz with 192 Mbytes of RAM) we were able to run experiments with thousands of agents effectively migrating among eight machines and communicating by exchanging tens of thousands of ACL messages. Furthermore, improvement of the "quality" of hardware resulted in immediate increase in the number of agents and messages that the system was able to accommodate (as well as a substantial reduction of processing time). Finally, and even more importantly, in all our experiments, any increase in the number of agents and/or messages resulted in a linear increase of processing time.

It has to be stressed that it does not really matter here how realistic or unrealistic our experimental scenarios were. Even if one is to argue that they are completely artificial, they still show how efficient and scalable JADE is. Therefore, there exists no excuse for agent researchers, but to start designing and implementing large software systems, consisting of hundreds of agents and study their behavior. We can do it already today and there is no reason to stop with demonstrator systems consisting of only a few agents. And we believe that this is very good news for the future of agent research.

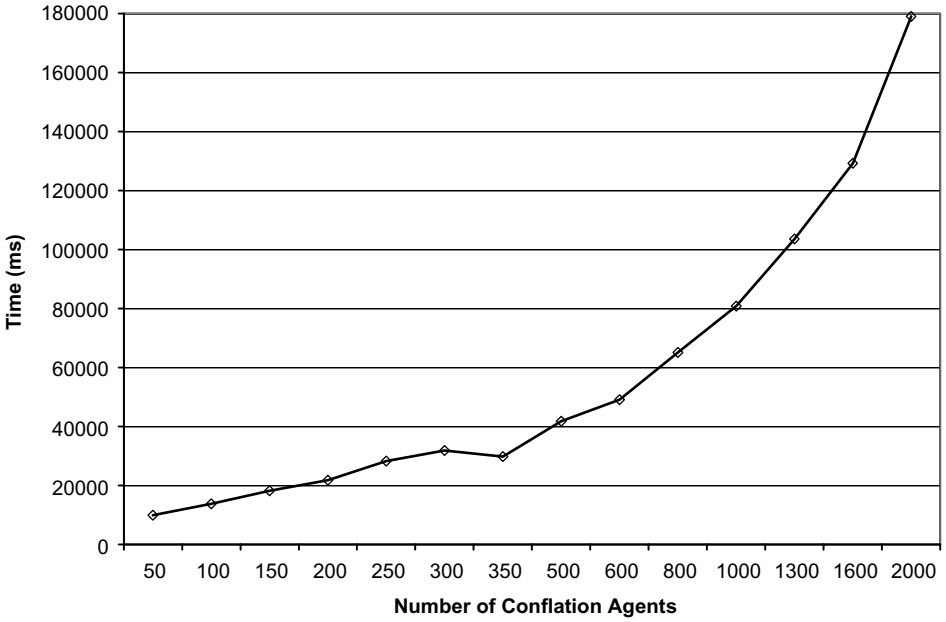

Fig. 21. Geospatial Data conflation, 8 PC's, increasing number of conflation agents.

References

[1] J. Hendler, *Is There an Intelligent Agent in Your Future*? Nature, http://www.nature.com, March 11th, 1999.

[2] M.L. Griss, *My Agent Will Call Your Agent . . . But Will It Respond*? Technical Report, Hewlett Packard, 1999, http://www.hpl.hp.com/techreports/1999/HPL-1999-159.pdf.

[3] N.R. Jennings, An agent-based approach for building complex software systems, *Communications of the ACM* **44**(4) (2001), 35–41.

[4] T. Berners-Lee, J. Hendler and O. Lassila, *The Semantic Web*, Scientific American, May, 2001, http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21.

[5] P. Maes, Agents that Reduce Work and Information Overload, *Communications of the ACM* **37**(7) (1994), 31–40.

[6] R. Deters, *Scalability & Multi-Agent Systems*, 2nd International Workshop Infrastructure for Agents, MAS and Scalable MAS. 5th Int. conference on Autonomous Agents, May-June 2001.

[7] N. Wijngaards, M. van Steen and F. Brazier, *On MAS Scalability*, Proc.2nd Int'l Workshop on Infrastructure for Agents, MAS and Scalable MAS. May 2001.

[8] P.J. Turner and N.R. Jennings, *Improving Scalability of Multi-Agent Systems*, Proc.1st Int'l Workshop Infrastructure for Scalable Multi-Agent Systems, June 2000.

[9] O.F. Rana and K. Stout, *What is Scalability in Multi-Agent Systems*, Autonomous Agents 2000, June 2000, ACM Press.

[10] L.C. Lee, H.S. Nwana, D.T. Ndumu and P. De Wilde, The stability, scalability and performance of multi-agent Systems, *BT Technology J.* **16**(3) (July 1998), 94.

[11] H. Nwana and D. Ndumu, A perspective on software agents research, *The Knowledge Engineering Review* **14**(2) (1999), 1–18.

[12] G. Vitaglione, F. Quarta and E. Cortese, *Scalability and Performance of JADE Message Transport System*, presented at AAMAS Workshop on AgentCities, Bologna, 16th July, 2002, http://sharon.cselt.it/projects/jade/papers/Final-ScalPerfMessJADE.pdf.

[13] S. Rahimi, J. Bjursell, D. Ali, M. Cobb and M. Paprzycki, *Preliminary Performance Evaluation Geospatial Data Conflation System*, Proceedings of The IEEE International Agent Technology (IEEE-IAT 2003), Halifax, Canada, 2003, 550–553.

[14] JADE: http://sharon.cselt.it/projects/jade/.

[15] G. Caire, *JADE Tutorial: JADE Programming for Beginners*, http://sharon.cselt.it/projects/jade/.

[16] Agent Communication Language Specification, http://www.fipa.org/repository/aclspecs.html

[17] M. Wooldridge, *An Introduction to MultiAgent Systems*, John Wiley & Sons, Chichester, UK, 2002.

[18] M. Paprzycki and A. Abraham, *Agent Systems Today; Methodological Considerations*, in: Proceedings of 2003 International Conference on Management of e-Commerce and e-Government, Jangxi Science and Technology Press, Nanchang, China, 416–421.

[19] K. Chmiel, D. Tomiak, M. Gawinecki, P. Karczmarek, M. Szymczak and M. Paprzycki, *Testing the Efficiency of JADE Agent Platform*, Proceedings of ISPDC 2004, IEEE CS Press, Los Angeles, 2004, 49–56.

[20] K. Chmiel, D. Czech and M. Paprzycki, Agent Technology in Modelling E-Commerce Processes; Sample Implementation, in: *Multimedia and Network Information Systems*, (Vol. 2), C. Danilowicz, ed., Wrocław University of Technology Press, 2004, pp. 13–22.

[21] http://www.grasshopper.de.