Enhancing a Multi-agent System's Performance: From Implementation to Simulation Analysis

Fenintsoa Andriamasinoro, Remy Courdier
Applied Mathematics and Computer Science Research Institute
University of Reunion Island
{fandriam, courdier}@univ-reunion.fr

Eric Piquet
International Center for Agronomical Research and Development
Reunion Island
piquet@cirad.fr

Abstract

In a Multi-Agent System (MAS), agents preserve their proactivity and autonomy. Consequently, one associates an independent process with each of them. But when the simulation is only on a single mono-processor computer, the performance tends to be affected because computer resources are more limited. Unlike a system of multi-computers (where generally the real parallelism may be found), the management of these processes is more difficult in a mono-processor environment. We show in this paper that it is nonetheless possible to enhance the MAS performance in such an environment and discuss herein the way we do it. We determine the factors to be taken into account and refer to our GEAMAS platform in carrying out our investigation.

1. Introduction

In simulation runtime, agents in a multi-agent system need computer resources (processor, memory, etc.) to perform their goal. An agent is a computational entity, which acts on behalf of other entities in an autonomous fashion [3], and it performs its action with some level of pro-activity. Out of respect of this agent's autonomy, each of them theoretically possesses the processing resources to complete their respective tasks. Thus, agents independently run their own activity in parallel and may or may not interact with other ones.

The network of workstation is the best-distributed environment in which the parallelism is actual, especially when each agent in MAS is situated in each computer. So, in performance study, many works took on this architecture [2, 9, 11, inter alia]. But even though the architecture is interesting and increasing in number, multi-computer systems are either used for highly dimension projects, or unaffordable due to high cost. The single computer[1] usage remains the solution for many users running a simulation and/or measuring their MAS performance. However, works in this domain generally reflect simulation results but neither the duration nor the limit of the system on which simulation is performed.

The problem of the single computer is that its resources are more limited than ones of interconnected computers. This is a critical factor if we consider that agents independently perform their activities.

We then focused our work to the way of enhancing the performance of a system in such a single-computer environment. In particular, we consider the case of agent-based systems and the enhancement is focused on the simulation performance in such a system. For that, we refer to our multi-agent platform Geamas [6].

The remainder of this paper is organized as follows: Section 2 presents a more detailed account of the performance criteria with which we deal. Following this, Section 3 moves on to a detailed description of Geamas. Section 4 then gives an account of the underlying motivation behind this project followed in turn by the explanation of our enhancement procedure in Section 5. Section 6 presents the results of the work, and Section 7 concludes the paper.

2. The performance

The measurement of performance includes many variables: the number of tasks completed by agents resulting from their behavior [4], the number of agents the system can bear [12], etc. In this paper, we represent the enhancement as a) the minimization of the simulation duration while keeping the same simulation parameters, b) the ability of the system to manage the dynamics of the population size (i.e. the number of agents), c) the optimization of computer resources usage (e.g. avoiding

[1] In this paper, when we talk about a computer, it is one with a single processor. The multi-processor computer leads to another discussion in performance.

the overload of memory, processor, disk capacity), and d) the facility for users to interpret simulation results as well as the ability of the system to save and restore all simulation information. Here, information is divided into 3 categories: data (agents properties values), exchanged messages and structure (e.g. relationship between agents).

3. Architecture

3.1 The kernel GEAMAS

It contains 3 abstract levels of agents:

- the highest level contains MacroAgent, which has the total sight of the system;
- the lowest one is the MicroAgent, which corresponds to fine-grain agents. Interactions, micro-behaviors and evolution capabilities describe each agent.
- the MediumAgent is in an intermediate level. It is composed of a set of micro-agents that have a common characteristic or a common aim. The MediumAgent may be short-lived.

Agents in GEAMAS do not have a position in the environment. They can however be represented there by situated objects. In this manner, these agents can evolve independently of a given environment.

3.2 The timer agent

We emphasize the timer role because it is an important parameter in our measurement.

In GEAMAS, the timer is a particular agent representing and managing the time of the system. To distinguish it within the system, we call the other agents the *actors*.

The timer schedules some actions of actors, particularly those to be performed at a given time. The time is given either in a relative way (for instance, *within 30 minutes*) or in an absolute one (for instance, *at 4:00 p.m.*). It also contains a value d, evaluated in *milliseconds* (*ms*) computer time, and corresponding to 1 timeunit in simulation. d is called the *cycle duration*. It allows users to know the theoretical simulation duration D[2], by a simple function f. So, $D = f(d, timeUnit)$.

For instance, suppose $d = 10$ and the timeunit we take is one *minute*, the time we need to simulate 30 days is theoretically $D = (60 * 24 * 30 * 10)\ ms \approx 7\ minutes$.

To minimize D, d must be minimized. One may wonder why the d-value is not always set to *1*. Actually, this is the *perfect* value but there is a constraint which we will explore now.

Suppose that t is the current simulation time. In fact, the basic idea is that within the next d ms, each agent in the system has resources and performs 1 action (corresponding to the message at the top of their respective internal mailbox *MB* [1]). So, the constraint is that d must be adjusted to allow each thread of actors to perform one action.

At the time $t + d$, the timer increases and dispatches next agents' actions (if being) to be performed at this time, to the queue of each *MB*.

Thus, d depends on the number of agents. If d is too small, some agents may not have time to perform their action and so, the latter may be treated with delay. Here, the goal is precisely to avoid these delays. On the other hand, if d is too big, agents may do nothing during a certain time, so there is no optimization.

3.3 Agent activities

In GEAMAS, each agent corresponds to a thread[3] and then, agents run in pseudo-parallelism. In other words, $nbThreads > nbAgents$ where $1\text{agent} \leftrightarrow 1\text{ thread}$. Other threads are used for other components (for instance, to launch the GEAMAS kernel or the graphic user interface, etc.).

In Java implementation, we use the class *Thread*. We noted that Java gives a share of time for each thread to run and yield resources to others, depending on their priorities. The timer agent has, in GEAMAS, a higher priority than the others (because actors may stop their actions while timer always runs).

When the thread of one agent obtains resources, it performs its behavior. It consists of taking and treating the next event from its letter-box [1]. Afterward, resources are yielded to another one. The management of resources (so, the multi-threading) in agents' activities was, at this time, performed by Java (Figure 1).

At any time, a simulation may be suspended or stopped and the current resulting state analyzed.

3.4 Evaluation of GEAMAS performance

Description

The kernel performance can be evaluated by means of its applications. We could compare it to a mechanic (here, ourselves) who wants to test a car (the kernel) performance and calls a driver (any application) to carry out the task. The evaluation is made according to the car driver's scenario (e.g. running at high or low speed, carrying one or more passengers and luggage with him, etc.). Application is then like an interface allowing observers to assess the kernel state. In other words, without application, evaluation is very difficult. In this work, the performance of Geamas is measured by application scenarios (number of agents and situated objects, links and messages between agents, etc.). We use the BIOMAS application [5] for that. Actually, it is the

[2] the problem is exactly that this value D is never reached. The real duration is always very more greater than D.

[3] Another notion which makes confusion is that of process. We refer the reader, for instance to [8] for the distinction between the two concepts.

only one (driver) we can take to perform the test.

Biomas consists of exchanging animals' organic matter (OM) between farmers (which are agents in the model) to fertilize their crops (also agents in the model). These exchanges generally lead to many negotiations, decisions, and consequently, many exchanged messages, which may overload the system.

Note that only the state of Geamas is considered. Obviously, these modifications affect Biomas if we consider the observers' view. But here, Biomas is just used as an *entry point* to Geamas. Thus, when we will use another application in the future, we are sure that the result of this work is also valid for this new application.

Making simulation

The control and analysis of a simulation are carried out by the graphic user interface (GUI) of GEAMAS. This GUI makes it possible to display the evolution of the system. In particular, it is possible, as inputs, to create the environment and agent structures, to customize the value of the agent attributes, and to save and to restore a simulation at any time. As outputs, it is possible to display all messages in the whole system, to show the current value of agent properties and the graphical representation of their variation, and to follow the duration of the simulation

Simulation is made according to users' scenario, leading the kernel to make one more thread (i.e. one more *charge* for the system) for each new agent required by the application. Each created agent is an instance of one subclass of the (abstract) class *GeamasAgent* one of whose attributes is an instance of the class *Thread*. Figure 1 allows to more understanding the mechanism.

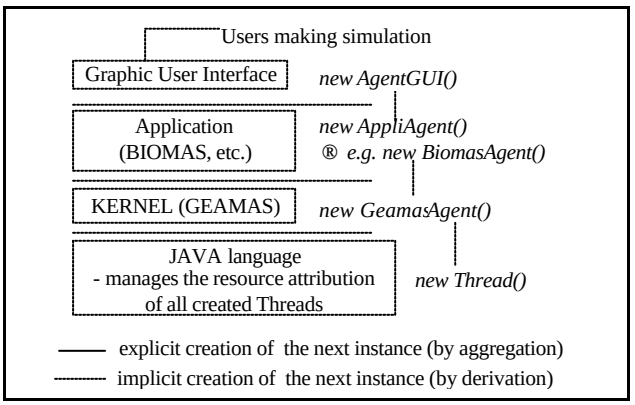

Figure 1: The creation of threads

After this general description of Geamas, we present next the motivation of this work.

4. Motivation of the work

4.1 Introduction

We were motivated to perform this work because we noted many problems when analyzing the BIOMAS simulation. More comprehension of the "problem" is illustrated in Section 4.2, which describes the situation one year ago (1999). To outline, we say that at this time and with this implementation technique, the kernel was able to support only around 25 agents in simulation while the Biomas application requires around 150 agents. So, improvement is needed so that the simulation result is suitable.

4.2 GEAMAS 1999

We are going to examine issues of BIOMAS during simulation. Problems can be decomposed into three parts:

1. the management of memory
2. the limit of the storage
3. the global performance.

Memory

In Table 1, we give some simulation examples in which we met memory problems. Presumably, the reasons were, on the one hand, the management of graphs and on the other hand, the management of messages.

About the graphs, it is not possible to plot them after simulation. The only possibility to follow a trend was to define ones before launching the simulation. Furthermore, they were not updated after some months of simulation (Simulation n°1). In the worst case, the system displayed error messages such as "out of memory" (Simulation n°2). In another case, the reopening of a saved simulation is impossible (Simulation n°3) when the number of opened graphs increases.

Moreover, even without graphs and without apparent problems, we always raised a problem of memory: the display of messages. The system failed as soon as we controlled the reading of the messages (Simulation n°4).

As a matter of fact, the degree of issues depended on the number of opened graphs, on the number of agents and on the duration of simulation.

Storage

In this second part, we are going to say that in GEAMAS 1999, each simulation was saved in only one file and only the current state of the simulation was able to be stored.

Performance

Management of memory, management of storage, inter alia were the reasons behind the weak performance.

For example, too many simulations with a cycle duration of 80ms failed when the number of agents in the system is beyond 50, and 6 months after the beginning of

simulation. We can point out that the duration of a cycle defines the actual duration of simulation. To put it in another way, for a cycle duration of 80ms, 12 hours were necessary to simulate 1 year. This value is inappropriate as we want to reach the optimal time defined by the function *f*, explained in Section 3.2.

Concerning the message monitoring, we noted that a lot of messages were lost when we decreased the cycle duration. Consequently, in Biomas, negotiations or exchanges were stopped. It might lead to false or spectacular results. As an example, there is the case where a negotiation of OM-exchange between farmers was completed but the delivery was not performed because the expected haulier did not receive the command message of transport (while this message was actually sent). In reality, the thread of the haulier did not obtain the computer resources to treat the message.

Memory - Storage - Performance

To resume, we can write that GEAMAS version 1999 had a poor efficiency. Management of memory and that of storage were too complex. Besides, GEAMAS made the computer operating system unstable.

5. Enhancement

5.1 Multi-threading

In multi-threading management, relying on the multi-threading functions (we called *MTF*) of the development language is delicate because programmers do not necessarily know how the language libraries are designed and implemented. Consequently,

- there is no certainty about the equal obtainment of resources for each agent-thread. For instance, some agents may acquire resources twice before the latter are yielded to others; thus, some agents may either accomplish their goal quicker, or, in contrast, may be blocked because they are awaiting replies from other ones while the latter will never obtain resources (or will obtain them but no longer at the appropriate time);
- the way of implementing threads is unknown, in particular, about the algorithm speed, etc.
- the threads made by *MTF* generally run concurrently. However, in a multi-agent level, there would have to be no concurrence in processor attribution because each agent performs its activity in an independent way and then equally obtain the processor-time.

To deal with multi-threading, we decided in our work to manage the attribution of resources at the level of GEAMAS instead of the one of the Java language. In practice, we delegate greater control of agent resources to the MacroAgent level, which has a complete overview of the system.

With this aim, the MacroAgent creates a unique instance of the class *Thread* whose role is to allot to agents a shared time of execution of their (internal and external) behavior. This time corresponds to the processing of one message. After performing the last agent, and whose total time corresponds to one-cycle *D milliseconds* (*d* is adjusted so that $D \pounds d$), it returns to the first agent etc. This is the new (simulated) multi-threading we adopt (Figure 2).

The order of resources attribution is not important because agents are autonomous and can carry out its behavior independently of other ones, and regardless of roles and organization they may have in the MAS. The essential is that each agent obtains resources once (and only once) in a cycle.

The main advantages of this solution are that:

- the autonomy of our agents is always respected,
- the number of threads running concurrently in the system decreased significantly. Instead of *n* threads for *n* agents, we have now (exclusively for agents themselves) 1 thread for all actors and 1 thread for the timer. There are then *n-1* threads, which are removed, significantly increasing the system performance.
- there is a guarantee that each agent equally obtains computer resources and no agent receives them more than once in a cycle. Behavior of all agents in a cycle is executed in an atomic way. So, in BIOMAS case, if an agent sends an OM-offer, intended receptors can be sure to have time for considering it.
- when passing from one agent to another in behavior execution, there is no lost time due to the thread context management. In fact, this occurs only, for instance, when passing resources from the thread of actors to the one of the timer, but not between actors anymore.
- the number of agents does no longer affects the number of threads because all actors are "grouped" in a single thread.

To summarize, GEAMAS has taken the thread management in hand. The idea here is to group the previous respective thread of each agent in one single thread. We think that in general works about multi-agent implementation, this principle can be used: it considerably alleviates the system. The difference from one work to another may be the location of this thread which dispatches resources. In this work, we decide to put it in the MacroAgent as it has the total control of the system.

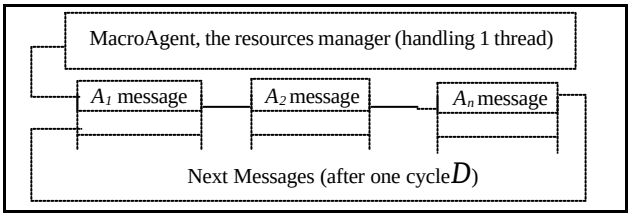

Figure 2: The simulation of multi-threading by the MacroAgent

5.2 Storage

Multi-threading management deals with only the runtime velocity. The main result is that simulation duration is largely decreased. Though, in their behavior, agents also have information to manipulate. When we see the previous state of GEAMAS, it shows that the system bugs after a long simulation time, the reason for this being the overload of the memory computer, particularly the Random Access Memory (RAM). Then, simulation fails or is over without enough information to be evaluated (see Table 1).

We overcome this problem by swapping on disk data which has been used the least recently. The system must be capable of keeping and providing, all the information users need, on-line or off-line, and without taking into account the simulation duration. The swapping method is now the best way to extend memory capabilities.

	N° Simula-tion	Duration of cycle	Simulated duration / Duration of simulation	Agent number	Graph number	Issues
1999	1	50ms	6 months / 4h	6	3	Not updated
	2	50ms	3 months / 2h	8	6	Out of memory
	3	50ms	3 months / 2h	46	15	Not opened after saved
	4	80ms	9 months / 9h	46	0	Bug after seen messages

Table 1. Some simulations states and met problems, on 1999

5.3 Simulation making

We also had to improve the manner whereby simulation is made in GEAMAS. Indeed, for users, making an important simulation manually is, tedious because there are generally many agents to input into the system while most of them generally have different properties. Then, properties of each created agent must be readjusted. Hypothetically, if there were around 150 agents in the system (that is our objective), it would be very difficult to manually change the properties of each of them.

The idea is then to put all agents' properties in a database, and to connect the database itself to the MAS, allowing the latter to retrieve agents' properties automatically. For that, we use the JDBC method [10]. The connection procedure is also justified by the fact that users (out of multi-agent domain) may prefer to input their information via a Database Management System (with which they are more familiar) rather than manipulating a particular MAS. Moreover, using JDBC is appropriate because it is an Application Program Interface which does not consider the real feature of the database to which it connects.

6. Improvement results: GEAMAS 2000

GEAMAS has gained some stability with these enhancements. We are now going to present what was changed for the users of BIOMAS according to memory, storage and performance.

Memory

The most efficient use of memory is using a buffer file. Indeed, with this system, the RAM is emptied regularly.

Consequently, data will now be accessible after simulation by graph. Besides, all messages of the overall system can be written in a particular file. Thus, simulator users can easily access them by a simple text editor instead of the message window of the simulator (whose capabilities are very limited). These two last ways introduce an increase in efficiency of the management of memory.

Storage

Now, all the information may be taken out according to users' request because information storage can be performed along the simulation (unlike the situation described in Section 4.2 GEAMAS 1999.Storage). Three files are written: a file which contains the structure of simulation; another which contains the data for plotting graphs; and one which contains all messages of the simulation.

Instead of displaying messages with the property editor windows, we can also save all messages of each agent in a text file.

Performance

Within just one year, performances grow significantly due to several reasons: management of the multi-threading, the memory and the storage.

In BIOMAS case for instance, the duration of cycle d has dropped down to 20ms. Currently, running 1 month takes 15 minutes instead of 1 hour. Sometimes, d can even reach 1 millisecond. Furthermore, we can run and save 200 agents over 5 years, without bugs and without illogical results. Another indicator of performance is the message reading. All messages are read and treated without apparent delay by the system. Consequently, all requests of all agents are treated, and as example from Biomas, one same negotiation between farmer agents can be treated in less than one day. On the whole, the general simulation state is improved (Table 2).

Memory - storage - performance

We now present in Table 2, different simulation resulting from the improvement. We can see that we do not find notable problems any more.

	N° Simulation	Duration of cycle	Simulated duration / Duration of simulation	Agent number	Graph number	Issues
2000	5	1ms	10 months / 7min	44	Entirely accessible	No problem found
	6	1ms	10 months / 7min	19	Entirely accessible	No problem found
	7	20ms	10 months / 2h	195	Entirely accessible	No problem found
	8	1ms	60 months / 45min	195	Entirely accessible	No problem found

Table 2. Some simulations states and met problems, on 2000

7. Conclusion and future work

To conclude, GEAMAS (owing to the BIOMAS test) is today a multi-agents system that can be used more satisfactorily. Despite a certain heaviness, GEAMAS is stable with better management of memory and with a good management of storage. Consequently, the performance is now sufficient to simulate, at BIOMAS level, many agronomical scenarios.

Enhancing the performance of a MAS in a single computer is not an easy task because of the limitation of computer resources: we use a mono-processor computer. We cope with allocating these resources to many autonomous agents, where each is supposed to have its own life activity independently of others. So, the constraint with the work was to respect this principle while computer resources are very limited.

According to our work, it is better for MAS programmers to carry out the management of the performance themselves, rather than relying on the basic libraries of the implementation language.

In addition to this, the other aspect to which we referred was users' requirements: ease in using the system with an acceptable simulation-period as well as obtaining maximum information.

Experiment analysis shows that our method leads to better results. Indeed, comparing the simulator we had one year ago to the one we have now, many differences may be noted: a shorter simulation duration, ease of use and interpretation at user level, support of a greater number of agents, etc. We can now make simulation with around 200 agents without notable problems, with a system initially designed for around 25 agents. We now plan to optimize the management of the multi-threading. On the one hand, we plan to keep the fact that the multi-threading control is handled by GEAMAS, not by Java (this work). On the other hand, we think of returning to the thread possibilities, by assigning one thread per agent, except that unlike the previous situation (many threads in competition), only 1 actor-agent thread is running in the system. All the other ones are suspended by the MacroAgent, and after a certain time, are notified to have the processor.

Improving the performance of the language itself will be needed. Actually, in Java domain, the language we used, many works to improve language performance provide promising results [7, 13, inter alia]. Our next work will also take into account all of these possibilities.

Acknowledgement

We would like to thank Kari Stuckey, Ed Hallet, Andy Tait and François Guerrin for sharing their knowledge and improving this paper.

References

[1] Andriamasinoro F. and Courdier R. A model of virtual competition between remote agents. *In proceedings of International ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce* 2000.

[2] Bivens A., Gao L., Hulber M. F. and Szymanski B. K. Agent-Based Network Monitoring, *In Proceedings of 3rd International Conference on Autonomous Agents*, 1999.

[3] Ferber J., *Les systèmes multi-agents*, Inter-Editions, 1997.

[4] Goldberg D. and Mataric M.J. Interference as a Tool for Designing and Evaluating Multi-Robot Controllers, *In Proceedings of 4th AAAI Conference in Artificial Intelligence*, 1997.

[5] Guerrin F., Courdier R., Calderoni S., Paillat JM., Soulié JC. and Vally JD. Conception d'un modèle multi-agents pour la gestion des effluents d'élevage à l'échelle d'une localité rurale. *In Proceedings of 6th JFIADSMA'98*, 1998.

[6] Marcenac P., Courdier R., Calderoni S. Zooming on a MultiAgent Simulation System: from the Conceptual Architecture to the Interaction Protocol. *In Proceedings of 3rd International Conference on MASs (ICMAS-98)*, 1998

[7] NewHall T. and Miller B. Performance measurement in Dynamically Compiled Java Executions. *In Association for Computing Machinery (ACM) Workshop on Java for High-Performance Network Computing*, 1999.

[8] Niemeyer P. and Peck J. *Exploring Java*, 2nd Edition. O'Reilly Edition, 1997.

[9] Rana O. F.. Performance Management of Mobile Agent Systems. *In Proceedings of 4th International Conference on Autonomous Agents*, 2000.

[10] Reese G. *Database Programming with JDBC and JAVA*. Edition O'Reilly, 1997.

[11] Rubinstein M. G. and Duarte O. C. 1999. Evaluating TradeOffs of Mobile Agents in Performance Management in Mobile Agents. *In Networking and Information Systems*, 1999

[12] Turner P.J. and Jennings N.R. 2000. Improving the scalability of Multi-agent System. *In Proceedings of 4th International Conference on Autonomous Agents*, 2000.

[13] Yelick K., Semenzato L., Pike G., Miyamoto C., Liblit B., Krishnamurthy A., Hilfinger P., Graham S., Gay D., Colella P. and Aiken A. Titanium: A high-Performance Java Dialect. *In Association for Computing Machinery (ACM) Workshop on Java for High-Performance Network Computing.*, 1998.