Jurnal Teknologi

Full paper

Relaxing Synchronization Constraints in Distributed Agent-based Simulations

Omar Rihawi[a*], Yann Secq[a], Philippe Mathieu[a]

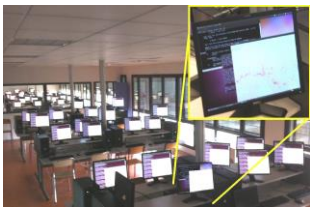
[a]*LIFL (CNRS UMR 8022), Université Lille1, 59650 Villeneuve d'Ascq, France*

*Corresponding author: omar.rihawi@lifl.fr

Article history

Received :11 September 2012
Received in revised form :
21 February 2013
Accepted :15 April 2013

Graphical abstract



Abstract

In the context of situated agents simulations, when the number of agents increases, the number of their interactions will be increased too. These growths leads to higher requirements in memory and computation power. When simulations involve millions of agents, it becomes necessary to distribute the simulator on a computer network. In this paper we study the impact of synchronization policies in such context. Our claim is that when millions of agents are used in a simulation, because observations of these complex systems is made at the population level, emergent properties at the macroscopic level should not be highly impacted if some failure appears at the microscopic level. This paper is focused on the study of the impact of synchronization relaxation in the context of large scale situated agents simulations. We evaluate the cost in performance of several synchronization policies and their impact on the macroscopic properties of simulations. To that aims, we study three different time management mechanisms and evaluate them on two multi-agent applications.

Keywords: Distributed multi-agent systems; distribute situated agent-based simulations; distributed architectures; synchronization policies; time management

1.0 INTRODUCTION

Multi-agent systems are made of autonomous entities (agents), which interact in an environment to achieve their own goals [1], producing emergent properties at the macroscopic level. When the number of agents or interactions grows to millions or billions, the simulation of such system requires important computation power and memory volume, which can be handled by distributing simulations over network. However, when working with this kind of simulations, the goal is not to observe millions of individual interactions (microscopic level), but to observe properties at macroscopic level. In some applications, we can even consider that some agents fail or cannot interact as fast as other agents, that should not be critical to the global simulation outcome. In other words, if we have a large scale situated multi-agent system, and some agents fail to interact, that should not affect the global behaviour of the system (macroscopic level).

To reach large scalability in such systems, the distributed computation over a computer network is required. That raise some problematics like: *time managemen*t and *synchronization.* This paper presents a first study of synchronization costs in performances and the impact of synchronization policies on the preservation of emergent macroscopic properties of situated multi-agent simulation.

The next section details the notion of time in a centralized and decentralized setting and introduce the three main synchronization policies that we have chosen for this study. The third section introduces the main concepts of multi-agent system. The fourth section details the platform that we have developed to experiment synchronization issues. The fifth section is the experimentation made on a prey-predator and capture the flag applications to benchmark the impact of synchronization policies on simulation outcomes. Last section is the conclusion.

2.0 NOTIONS OF TIME AND SYNCHRONIZATION

The word time is often defined as a non-spatial continuum in which events occur in apparently irreversible succession from the past through the present to the future. This transition from past events to events happening in the present is called the flow of time [2].

2.1 The Multiple Notion of Time and Time Steps

In a distributed simulation context, several notions of time are involved: user time, which is the real time, and simulated time, which is a set of small durations used to produce evolutions within

a simulation. This notion of simulated time is less linked to the flow of time and irreversibility than the property of ordering events in a sequence to guaranty causality between events. This notion of simulated time has been defined in a distributed context by Lamport [3] through a logical clock that induce a partial ordering of events, and has been refined as Logical Virtual Time (LVT) by Jefferson [4].

In multi-agent simulations, a common implementation to enable the simulation dynamic is to query all agents for their current action and to apply this set of actions. This round of talk defines a simulation step (simulation tick) or Time Step (TS). Because several actions are gathered within a time step, one can encounter conflicts between two or more actions, thus the simulator has to define tie-break rules for such situations. As illustrated in the prey-predator applications later in this paper, if two predators try to attack the same prey in the same simulation tick, a rule has to be given to define the outcome of such conflicting interaction.

In centralized multi-agent simulations, there is only one simulation time step that organize agents evaluation and that allows them to interact in a given period. In a distributed simulation, there is one logical clock per machine and the user time needed to handle a simulation step is not the same. In order to guaranty causality on all machines, we have to synchronize local time step within all machines. However, several policies to handle time step synchronization can be proposed as we will see in next sections.

The question that we are interested in is whether synchronization constraints can be relaxed without impacting the simulation outcome. Indeed, the balance between communication costs, performances and reliability is dependent on the application that is implemented. For example, if a simulation is used to generate an animation with a huge agent number, it should not be so important if some agents fail to interact, or if they do not interact as fast as other agents. However, in some other applications, like urban traffic simulations, we need reproducibility and reliability to ensure that all interactions between agents are fulfilled and also that performances are able to catch up with faster than real-time resolution.

2.2 Synchronization Policies

In this section, we explore three synchronization policies for distributed multi-agent simulations: strong synchronization, flexible synchronization and no synchronization. The main problem in a distributed setting is time management between machines [5] [6]. There are mainly two synchronization approaches in distributed systems: conservative (or synchronous) and optimistic (or asynchronous) synchronization [7], [8], [9]. However, we propose to divide synchronization policies into three main approaches: *strong synchronization*, *flexible synchronization* and *no synchronization*.

2.2.1 Strong Synchronization

This policy is simple: all machines are synchronized together in such a way that all local clocks are running at the same pace. Thus, the distributed simulator guaranty that all agents execute the same number of actions. To implement a strong synchronization, more messages have to be exchanged between machines, so communications costs are increased. This kind of conservative approach strictly avoid causality errors, but can introduce communication delays or deadlock problems.

2.2.2 Flexible Synchronization

The second policy allows machines to progress at different pace. One way to implement such flexibility is to use an optimistic (or asynchronous) synchronization, which allows machines to advance at different pace in simulated time. The main issue is to handle causality errors by detecting and recovering them through a rollback mechanism [9]. A rollback mechanism enforces temporal consistency by allowing a simulator to roll back previous events to reconstruct a previous state of the simulation. To enable this property, a simulator has to maintain a list of anti-messages that can undo side effects that have been produced by events evaluation. The gain of optimistic approaches is based on the fact that simulators should not roll back too often.

Another flexible synchronization that can be proposed is ***time window synchronization***. With this approach, machines can progress at different pace but a global constraint is enforced such that the slowest and fastest machines do not have a time shift greater than the defined time window. Thus, a time window defines the worst spread in time steps that can happen between the slowest and fastest machine. With this window permission, machines can avoid some delays of strong synchronization but we have to check that this flexibility do not affect macroscopic behaviours outcome. Of course, with this approach, we can have situations where agents in different time steps can interact. These situations can be resolved through roll back mechanisms, or can be ignored if it is believed that the impact of time incoherency in some interactions is negligible in respect to the volume of interactions in the whole system. This is a strong hypothesis that will be studied through the two applications in next sections.

2.2.3 No Synchronization

The third and last policy is to simply drop synchronization between machines. It can be seen as a variation of the time window policy with an infinite window size. This approach exploit the available speed of all machines, however, we will see in section 4 obviously that this policy is not fitted for all applications.

To conclude, this section has presented the three synchronization policies that will be studied in following sections. With the synchronous approach (or strong synchronization), all machines guaranty the correct execution for all parts of the simulation but additional messages have to be exchanged and induce more delays for each time step. Whereas, in asynchronous approaches, machines take checkpoints independently without any synchronization among them. Unfortunately, because of the absence of synchronization, there is no guarantee that local time steps are the same. In order to get rid of these disruptions, machines have to roll back to older checkpoints.

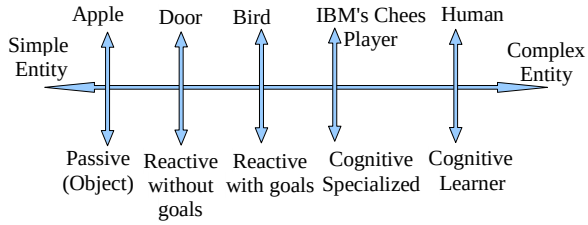
The problematic studied in this paper is thus to determine whether we are able to keep macroscopic behaviours emergence when relaxing synchronization constraints. This approach aim is to gain performance by reducing synchronization costs and to determine which kind of applications are robust with respect to these synchronization issues.

■3.0 AGENT AND ENVIRONMENT CONCEPTS

In MAS systems, agents and environment are the main concepts that allows entities to perceive and act on a common medium. We first describe what is an agent before detailing the importance of the environment in a situated multi-agent simulation.

An Intelligent Agent [1] is an autonomous entity which observes its environment and acts by following its own goals. Intelligent agents can use old knowledge or learn new one to achieve their goals. They can be very simple or very complex as we can see in table I. Agents can range from purely reactive agents (simple strategy) to cognitive agents (complex strategy) by involving abstract knowledge representation and planning systems.

Simplest agents can be passive entities like an apple. A more complex agent can be a door which is a reactive agent without goals. More complex, cognitive agents can learn and update their strategy. Agents are able to interpret their environment and try to achieve their own goals. Depending on the application, we can have a large number of agents like in a physical collision simulation (mainly made of reactive agents) or only one cognitive agent (chess player agent):



The environment in a multi-agent system can be considered as an agent container and an interaction mediator between agents. Different environment types can be used for different application types. In some applications, there is no spatial environment like in financial markets simulations ATOM [18]. However, we are mainly interesting in applications that have spatial environments.

In our study, we distinguish two main spacial environment types:

- Discrete Environments: like IODA [19],
- Continuous Environments: flocking model [16].

These environments types are deduced from the type of interactions that happens between agents. Depending on the application domain, the environment can enforce spatial constraints (soccer or collision simulations) or not (stock market simulations). When an environment has a spatial dimension, agents are embodied, so two agents cannot be in the same place at the same time. Within spatial environment, a distinction can be made between discrete and continuous environments. In a discrete context, the environment is made of a grid and agents move in this grid by swapping between environment's cells. In continuous environments, the space is represented by ranges and agents have floating positions. More details about environment properties can be founded in [1].

4.0 DISTRIBUTED AGENT-BASED SIMULATORS

To achieve large scale agent-based simulations, we believe that the distribution of a simulator on a computer network is necessary to reach a high number of agents and interactions. A simple distributed platform can be M machines with a communication layer that informs others about simulation changes. Each machine is able to build a partial view of the system, and with all other machines we have the global view of the system. However, there are different ways to distribute such simulations with its concepts (agents and environment). In this paper, we will focus more on time management and synchronization rather than the way of distribution.

4.1 State of the Art

Many platforms already exist in the domain of distributed large scale agent-based simulation: Repast [10], FLAME [11] (and FLAME-GPU implementation [12]), AglobeX Simulation [13], and DMASON [14] [15]. But, they do not support several synchronization policies.

Repast [10] provides components to build multi-agent simulations on a network with a shared middleware between machines, so simulator is free from all distribution considerations. Distribution through a middleware do not take into account specific optimizations that can be implemented to distribute multi-agent simulation. Another approach proposed by Repast is HLA, but it is focused on the coordination between different sequential simulation toolkit and is not really designed to gain speedup.

Other interesting works are D-MASON [6] and AglobeX [7] platforms. D-MASON is based on a master/workers approach, the master assigns a portion of the whole computation (like a set of agents) to each worker. Then, for each simulation step, each worker simulates the agents assigned and sends back the result of its computation to each interested worker. AglobeX[13] also has been built on the same mechanism and both platforms use simple models as application: AglobeX uses an airplanes applications while D-MASON uses a flocking model. These simple models do not produce complex interactions. For example in flocking model, birds will flock only by watching other birds and no interactions between two birds can explode the communication costs. However, in other classical models like prey-predator model, agents can produce complex or conflicting actions. For example, if two wolves want to eat the same sheep from different machines, then a protocol of agreement must be provided to resolve these actions.

To summarize, no platform can be considered as a test-bed of our works, as all works are working only on strong synchronization. Table 1 shows a comparison between all these platforms:

Table 1 Comparison between platforms

Platform	MaxNbOfAgents	MaxNbOfMachines	Model	Policy
Repast	68 billions	32000 cores	Triangles Model	Strong sync only
DMASON	10 millions	64	Boids	Strong sync only
FLAMEGPU	11000	GPU	Pedestrian Crowds	GPU-Strong Sync
AglobeX	6500	22 cores	Airplanes	Strong sync only
Our platform	20 millions	200 p2p-Machines	Prey-predator model	3 different capabilities

4.2 Testbed Description

To evaluate the impact of synchronization policies, we have developed a distributed simulator. This simulator is based on a set of machines (peer-to-peer network), that handle the simulation of a subset of the environment with its agents. Each machine is connected with all other machines, so the communication topology is a fully connected graph. The distributed simulator can be run in one of the three available synchronization policies: *strong*, *time window* and *no synchronization*.

Figure 1 shows 4 machines executing a distributed agent-based simulation. Each machine consists of: a local simulator, a communication unit and an environment part with its agents. The local simulation is a top-manager layer in each machine, which manages all tasks like: interactions between agents, receive information from neighbourhood machines and local visualization. Communication unit manages connections links between machines for message exchange and informs local simulator about machines time step (TS).

In case of strong synchronization, each machine follows 7 main steps in each TS: 1) it sends information to all neighbours about the environment state near them. 2) Then it waits for new information from neighbours to inform local agents about the neighbours' environments. 3) After that, each machine asks local agents about their next desired interactions. 4) Then, it sends the external interactions, which are interactions between agents from different machines, to neighbours. 5) Also, each machine receives interactions from others. 6) And, the possible interactions should be applied. 7) Finally, each machine draws its local environment and it synchronizes to next time step.

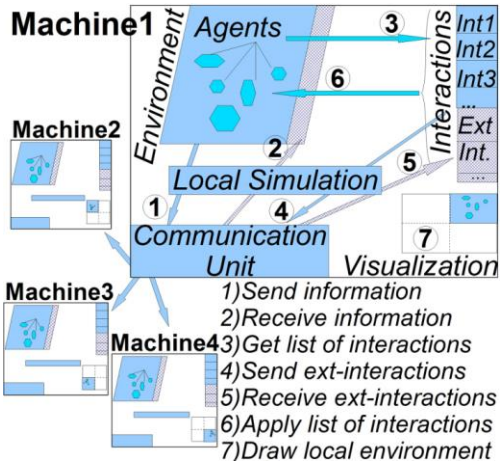


Figure 1 Description of a distributed agent-based simulation on 4 machines, each machine consist of: local simulation, communication unit and part of the environment with its agents. In strong synchronization: all steps (from 1 to 7) will be followed, whereas in flexible synchronization: receiving steps (like 2 and 5) can be passed to avoid communication delays

To illustrate the main loop executed by each machine, we have sketched two algorithms: algorithm 1 for *strong synchronization* and algorithm 2 for *flexible synchronizations*.

Algorithm 1 for *strong synchronization*:

1. Sync for zero-TimeStep
2. **While** Running **do**
3. //Exchange information
4. //Local Interactions
5. Get List Of Interactions From Our Agents
6. Solve List Of Conflict Interactions
7. Get List Of External Interactions
8. **While** Not All Machines Are Satisfied **do**
9. Send External Interactions
10. Receiving From Others External Interactions
11. Send Acceptance For Others' External Interactions
12. Receiving From Others Acceptance
13. **End while**
14. //Transferring Agents
15. Transfer Agents With Notifications
16. Receive Agents From Others
17. //Sync With Others
18. Receiving Messages
19. Sync With Others for next TimeStep
20. //Applying TimeStep's Interactions And Drawing
21. Applying Local Interactions
22. Drawing
23. **End While**

Algorithm 2 for *flexible synchronizations*:

1. Sync for zero-TimeStep to begin together
2. **While** Running **do**
3. //Send and Receive Information If Exist Without Wait
4. //Local Interactions
5. Get List Of Interactions From Our Agents
6. Solve List Of Conflict Interactions
7. Get List Of External Interactions
8. //Without any wait
9. Send External Interactions
10. Receiving External Interactions If Exist Without Wait
11. Add Possible Interactions To Local Collection
12. //Transferring Agents
13. Transfer Agents Without Notifications
14. Receive Agents From Others If Exist Without Wait
15. // W Time-Window-Sync With Others
16. Receiving Messages
17. Send Next TimeStep Notification
18. Sync With Others On W TimeSteps If We Reach It
19. //Applying TimeStep's Interactions And Drawing
20. Applying Local Interactions
21. Drawing
22. **End While**

The three synchronization policies have similar communication protocols with small differences. Algorithm 1 shows the states of a machine when it is running in *strong synchronization* mode. *Strong synchronization* algorithm has in

each communication state a notification, which is kind of replying and acceptance from other machines. Especially for last state of communication, all machines should be synced for next TS and they are suspended until other machines are ready for next TS.

Algorithm 2 shows the two other mechanisms: *time window* and *no synchronization*. In this algorithm, there is no notification for any communications states, except the last state which is for next TS. For *time window* policy, machine sends a notification of its current TS, and it checks if it has permission for next TS. Which is in our case, the difference between the local machine's TS and slowest machine's TS must be less than W Steps (W is a number of steps which can be determined by the user). Whereas, in case of *no synchronization*, machines send only a notification for next TS. It is similar to *time window*, but with an infinity window $W = \infty$.

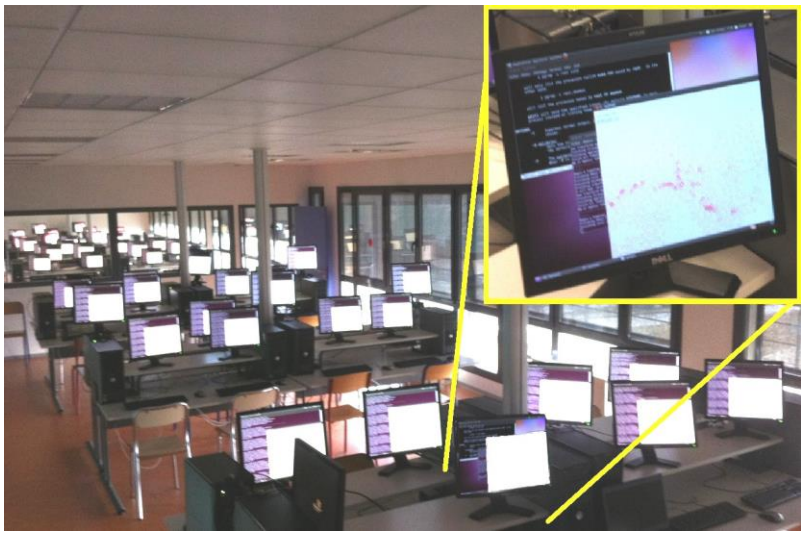


Figure 2 More than 5 million agents running on 50 machines

To evaluate the scalability of our testbed, we have implemented a simple flocking behaviour similar to Reynolds [16] and made it run on a network made of 50 machines, where each machine holds at the beginning of the simulation 100000 agents (Figure 2).

5.0 EXPERIMENTATIONS

In this section, we describe two applications that have been implemented and benchmarked to quantify the impact of the three proposed synchronization policies. It seems obvious that time inconsistency have not the same effect in all applications. For example, the simple boids application (Figure 2) can run without synchronization and still produce the emerging flocking behaviour. So, we want to determine with the following experimentations the impact of synchronization policies on the outcome of the simulation, more precisely on the conservation of the expected macroscopic behaviours.

5.1 Two Extrema Models

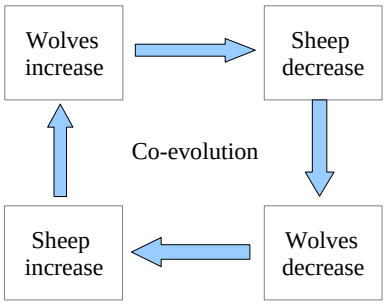
To study the synchronizations impact, we have implemented two applications. One is extremely affected by changing the policy of synchronization, while the other is extremely not affected.

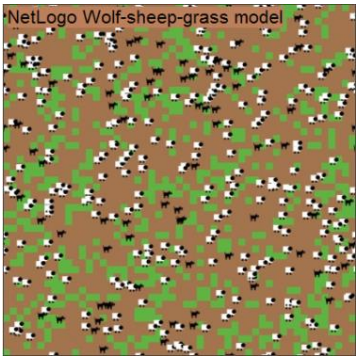
5.1.1 Prey-Predator (PP) or Lotka-Volterra Model

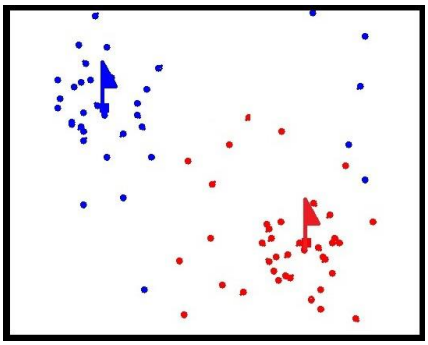
This model is a classical multi-agent application that involve two kind of agents, preys and predators. Both kinds reproduce themselves at a given rate, but predators seek and eat preys. If a predator does not find preys quickly enough, it dies of starvation. This application illustrates population co-evolution in a simplified ecosystem. An example of such model is the wolf-sheep-grass simulation proposed by Wilensky [17] that we have implemented in our test-bed. In this example, wolf tries to find and eat sheep, sheep searches for grass to eat and grass re-grows at a given rate. Wolves and sheep can have energy when they find something to eat, and then they can reproduce themselves.

In normal situation, the number of wolves and the number of sheep will be inversely proportional in some periods of the simulation and directly proportional in others. If the number of wolves increases, then they will eat more and more of sheep, and the number of sheep will decrease. Then the wolves will not find more sheep to eat and that will lead to decreasing in wolves energy and then decreasing in number of wolves. After that, the sheep will increase because there are no more wolves try to eat them and again the number of wolves will increase as there are more and more of sheep to eat.

However, if we lose all wolves or all sheep, then the model will be destroyed. That because, if we lose all sheep, then wolves cannot find any sheep to eat, and all wolves will be died. Again, if we lose all wolves, then the number of sheep will be increased to infinite as there are no wolves to eat them. So, all types of agents have to co-evolve to keep the model alive:







5.1.2 *Capture The Flag Model (CTF)*

This second model has been build to illustrate the fact that if a simulation outcome relies on timing issues, like population growth speed, then synchronization policies can introduce a bias. To achieve this goal, we propose the use of a simplified capture the flag application with two competing populations (or teams). For each team, we have two kind of agents: flag agents which produce new attackers at a given rate, and attacker agents which protect their flags or attack the other team (flags or attackers).

However, attack action is very simple: if an attacker agent from one team detects another agent from different team (attacker or flag), then it will try to reach that agent and destroy it (both agents should be dead). To enhance the stability of this model, we add defence behaviour for attacker agents to protect their flags, that can be by observing the number of team attackers around a team flag and see if this number is small (less than *N* for example), then the attacker agent will flock around the flag to protect it from any attack from the other team:

In both models, the macroscopic behaviour is considered as a stability measure of the model. For Prey-Predator (PP) model, the stability is to keep all populations in co-evolution during the simulation. That mean in all time steps, we should have wolves and sheep in the simulation, because if we lose one of these types, the model is destroyed. For Capture The Flag (CTF) model, the stability is to keep all flags of all teams alive and they produce more and more attackers. If a team loses their flags, then its agents should disappear, and the other team win.

5.2 Synchronization Policies Performances

We have executed experimentations on a network of machines with similar hardware. Most experimentations have run until 2 million time steps and the only parameter modified is the time window size. We have started with a time window with size = 0 *strong synchronization*, then 10 TS (between the fastest and slowest machine), 100, 1000, 10000, 100000 and finally *no synchronization* at all. Figure 3 Shows that *no-synchronization* policy always provides the lowest execution time because it is free from all communication delays. It shows that the simulation reach 10000 TS in 3 hours only with *no synchronization* policy. Whereas, it takes 6 hours (double time) in case of *strong synchronization*.

Consider that we have an emergency scenario of tsunami-town simulation which can be calculated faster with *no synchronization* policy than *strong synchronization*. Even, if some agents fail to interact, but we can get the main macroscopic behaviour. Then, we maybe save more lives in dangerous areas with *no synchronization* policy than *strong synchronization*. However, in some application like CTF, it can be unstable in case of *no synchronization*.

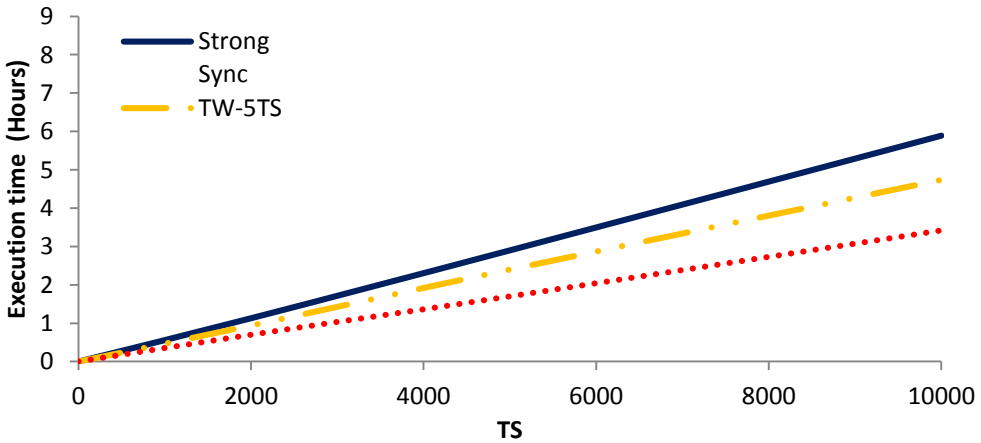


Figure 3 No-synchronization always gives the maximum speed

Table 2 Summary table with results of three synchronization policies for two models

Model	Strong-Sync	Time-Window	No-Sync
Wolf-Sheep-Grass WSG	Stable	Stable	Stable
Capture The Flags CTF	Stable	Stable if W < *n* time step	Unstable

Table 2 shows results of the stability of both models. The prey-predator model stay stable for long time (until 2 millions TS), for all experimentations, the co-evolution of prey-predator model has been preserved until we reach 2 million TS, even without any synchronization. Thus, this application is stable with all synchronization policies. Whereas CTF have been unstable in case of no synchronization and also time window synchronization after window size bigger than N (N depends on the initial configuration).

In next sections, we study first interactions effects between agents on the stable model (Wolf-Sheep-Grass WSG model) and see how interactions are impacted when we change the synchronization policies. Then, we study the instability of CTF model in details by exploring a biased initial configuration.

5.3 Interactions in Wolf-Sheep-Grass Model

As this model is stable for all synchronization policies, we study in details how interactions are impacted. Table 3 shows some facts on PP model in case of strong or no synchronization. It shows that all properties are similar in both synchronization policies, except Life-Circles. Life-Circle is the number of TS that will be taken by the population to return to the same previous state, or complete a phase of the co-evolution:

Table 3 Prey-predator model for two synchronization polices

Synchronization	Avg Life-circles	Max Nb of sheep	Max Nb of wolves	Min Nb of sheep	Min Nb of wolves	Max-age of sheep	Max-age of wolves
Strong-Sync	276.24 TS*	4281	1134	615	44	221 TS	153 TS*
No-Sync	270.43 TS*	3852	1142	612	53	215 TS	148 TS*

Figures 4 & 5 show the effects of Prey-predator model when synchronization policies changes. Both policies, strong synchronization and no synchronization have the same behaviour on prey-predator model, except that no synchronization is faster in life circles for its agents (prey and predators). That because in no synchronization, there are agents could fail to interact. For example, sheep cannot find grass to eat or wolves cannot find sheep to eat, then agents could have less energy and it will die to starvation. Thus the life-circles are a bit shorter in no synchronization than strong synchronization:

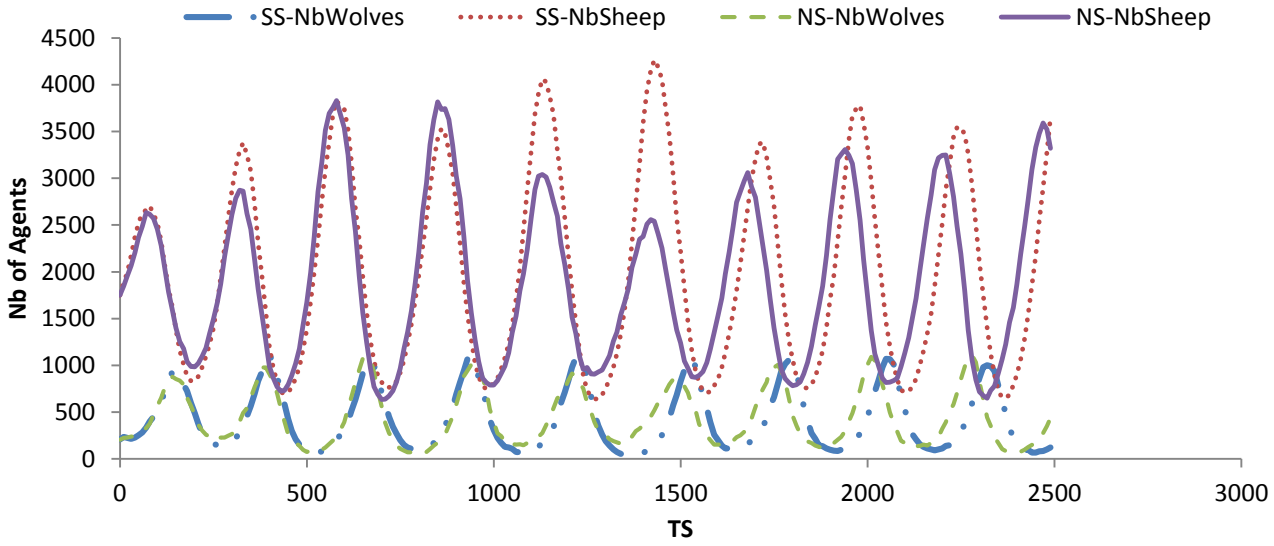


Figure 4 Both policies: strong-synchronization and no-Synchronization have the same behave on prey-predator model, but no-synchronization is faster in life-circles

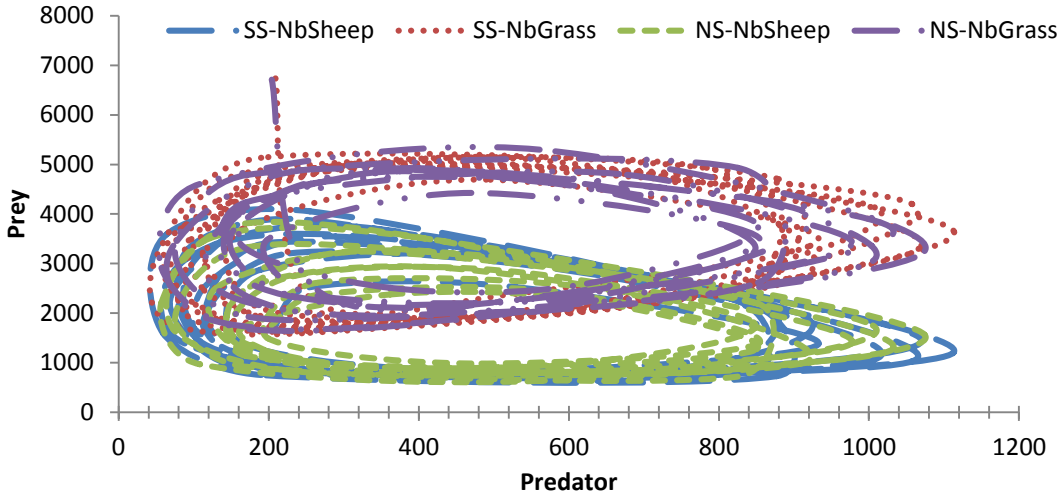


Figure 5 Life circles of agents in prey-predator model for both policies: SS and NS

5.3.1 External Interactions

External interactions are interactions between agents that belong to different machines. We have measured the evolution of external interactions with different time windows ranging from 0 (strong synchronization), 2, 100 and until no synchronisation (all tests have been executed for 100000 TS). Figure 6 shows how much external interactions between agents are executed in the prey-predator mode (with 5000 agents for each family: sheep, wolves and grass). This graph shows that with strong synchronization we have a lot of external interactions, this is normal because information (about agents) can be sent and received by other machines. But, and this is important, when there is no synchronization or even for a small time window synchronization, external interactions are significantly reduced. Our explanation for that is even if we choose the time window of only one time step, information about agents are sent between machines, but they are not received in the corresponding time step. Thus external interactions are reduced significantly:

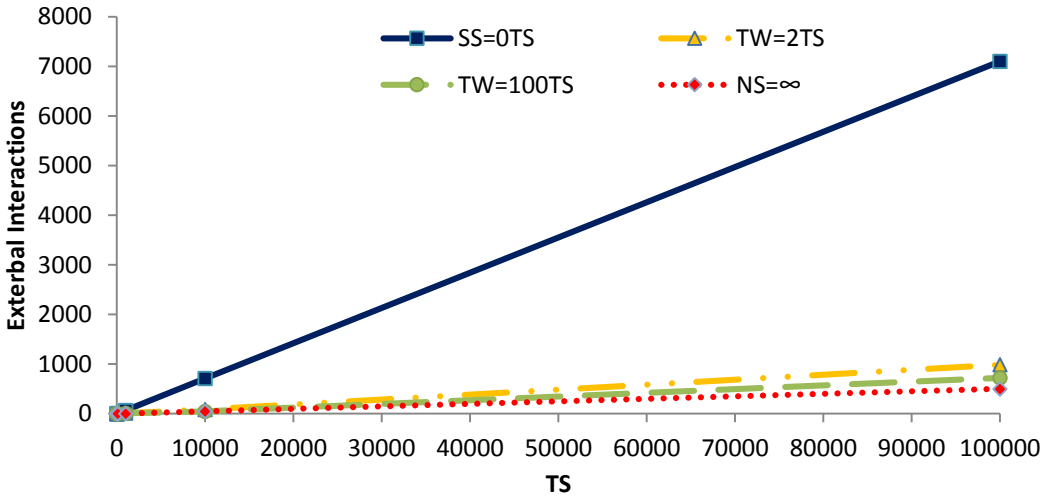


Figure 6 Prey-predator model, External-interactions with different Synchronization polices

5.3.2 Invalid Time Step Interactions

In this test, we study the interactions on prey-predator model, when machines are not on strong synchronization. We define invalid interaction as an interaction between two agents from different time step. Clearly, it represents interactions that are temporally incoherent. This situation can appear for example when two agents are coming from two different machines which are not on the same time step. Figures 7 shows that the percentage of invalid interactions increases when the W time window increase. However, the percentage of invalid interaction is less than 0.4 from the total number of interactions. In case of four machines we have a double percentage than two machines, that because agents can swap between 4 machines more than in case of two machines.

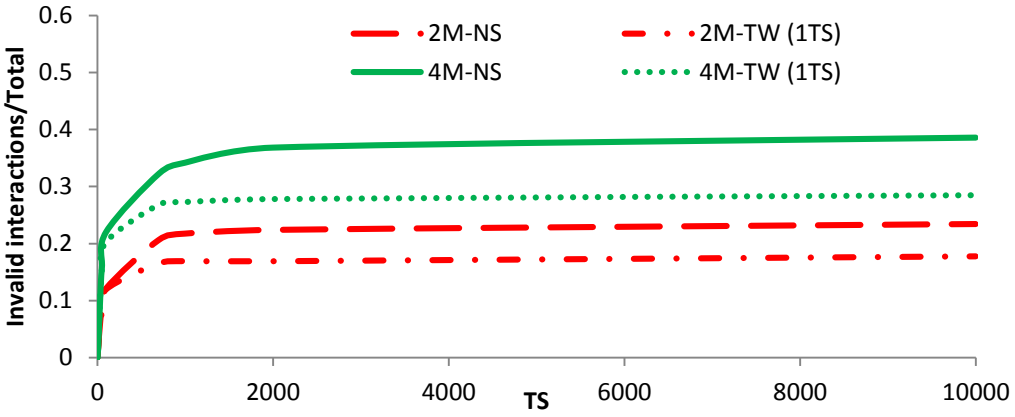


Figure 7 Invalid interactions between two or four machines in prey-predator model

5.4 Instability of CTF Model

As the previous experimentations, we have run these simulations on similar machines for 2 million TS and with a time window size evolving from 0 to infinity. The first initial configuration that has been explored was defined with one flag per machine. This configuration, nearly like the prey-predator model, is stable in respect to synchronization issues. If we choose another initial configuration, like 20 flags per machine, we can get different results. Table 4 shows that, for different sizes of time window bigger than 100000 the model is not always stable.

Table 4 Capture The Flags Model: 20 flags per machine

TimeWindowSize	0	10	100	1000	10000	100000	∞
CaptureTheFlags	Stable	Stable	Stable	Stable	Stable	Fail	Fail

*'Stable' Stable Model, 'Fail' Not Stable Model, '∞' means no synchronization

We have defined another initial configuration to evaluate the instability degradation that is induced by synchronization policies when machine load is not the same on all machines. This second experimentation runs on three machines: the first one contains all flags from the first population and the two others machines contain only half flags of the second population (Figure 8). The aim is to generate more load on one machine than others and to provoke an unstable model: the second population should always win because its attacker production will be higher (on two machines):

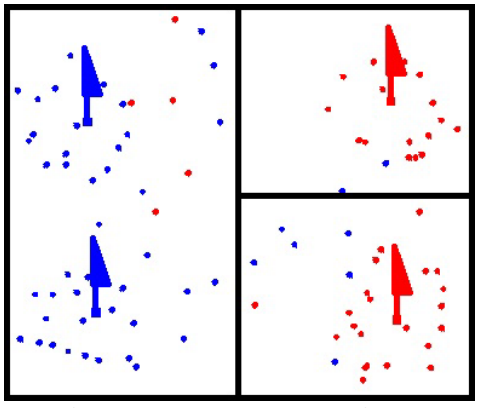


Figure 8 Two blue flags in one Machine and two red flags in two different machines

With this configuration, we ensure that the model is unstable, the same population always win. But, the convergence should be faster with more flexible synchronization policies, and fastest in case of no synchronization at all. If we define the Critical Time Step (CTS) as the necessary time step to completely destroy the model if no synchronization has been used. For capture the flag model, all flags of one population have been disappeared. Then, Figure 9 shows that the number of flags has a huge effect in the simulation, more flags mean less CTS to completely destroy the model. This CTS time depends on the initial configuration of the model, like the number of flags F and communication delays between machines:

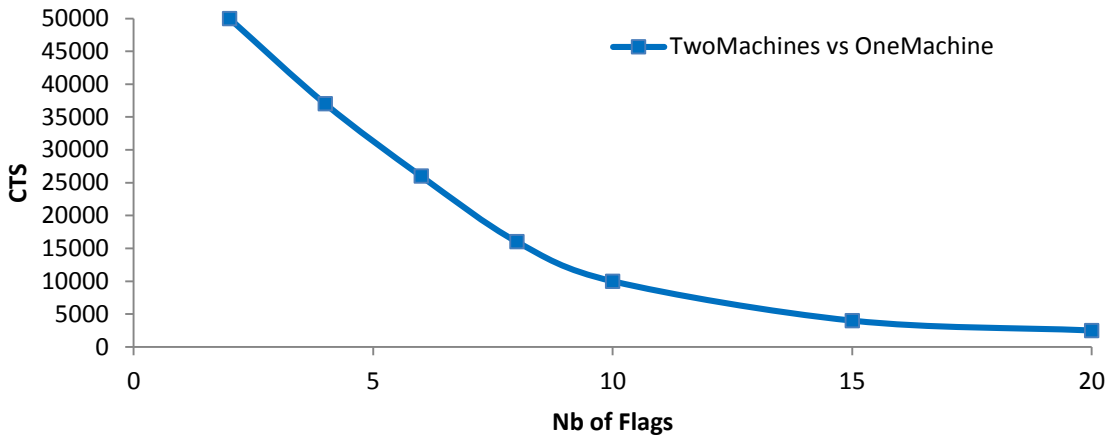


Figure 9 Capture The Flag Model: CTS vs Number of flags

As in Figure 9, the CTS will decrease by increasing the number of flags which have been used:

$$CTS = \frac{\alpha 1}{F}, \alpha 1 \text{ is constant}$$

However, CTS depends on the initial configuration and on machines itself which have been used.

If we define the *Time-Step to Destroy* (*TSD*) as the necessary TS to completely destroy the model if W time window has been used (*TSD = CTS if and only if* $W = \infty$ or *no synchronization*).

For this test, we study the time window size from 0 to infinity to determine how much time steps are necessary before one team disappear. Figure 10 shows that each configuration has a curve with different scale, which is reduced by increasing the time window W. It also shows that it is difficult to measure different initial configurations (different number of flags) as the curves do not have the same scales For that, we scale each curve to its CTS (as each configuration has its own CTS). Figure 11 visualizes results after scaling each curve in respect to its CTS. This figure shows that the Time Step to Destroy (TSD) decreases if the time window W increase.

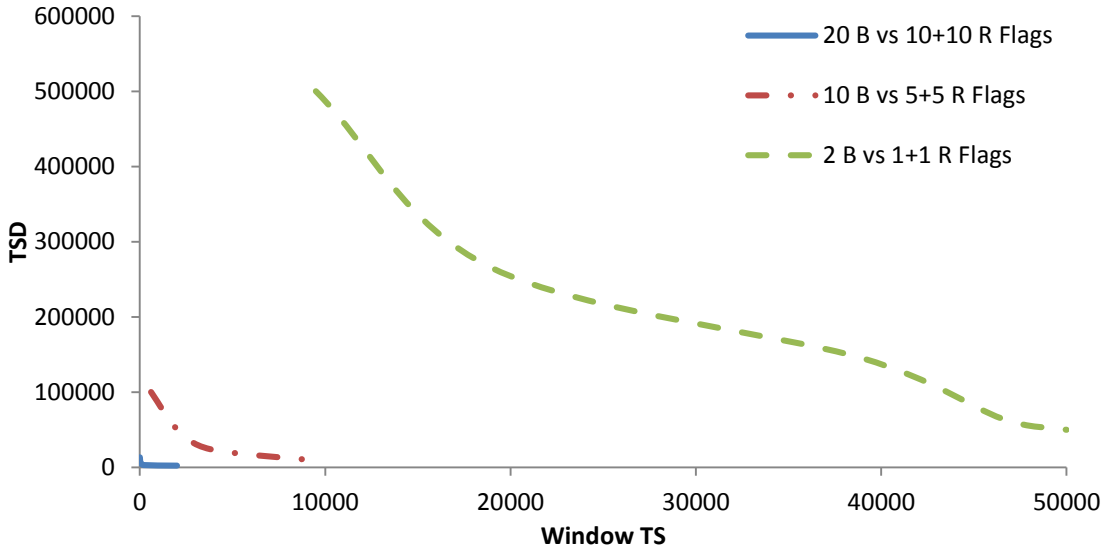


Figure 10 Different configurations of Capture The Flag model with *time window* synchronization

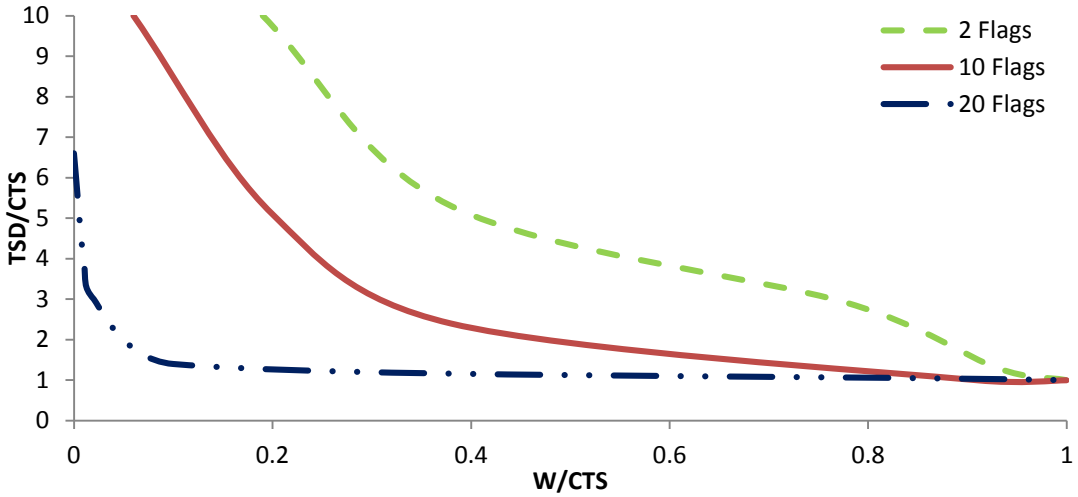


Figure 11 *TSD* for different configurations of CTF Model with different sizes of TW (*W*)

According to the Figure 11:

$$TSD/CTS = \frac{\alpha 2}{\mathrm{W/CTS}}, \alpha 2 \text{ is constant}$$

Then:

$$TSD = \frac{\alpha 2 \times \mathrm{CTS}^2}{\mathrm{W}}$$

Again, each curve has more flags, will be destroyed with smaller TS, we can replace:

$$CTF = \frac{\alpha 1}{\mathrm{F}}$$

Then:

$$TSD = \frac{\alpha}{\mathrm{W} \times \mathrm{F}^2}, \alpha = \alpha 1 \times \alpha 2^2 \text{ is constant}$$

That means, TSD decreases if the flags number increases or the time window W has been increased too. Figure 11 shows also that, for all configurations the model stay stable for a small time window. According to this figure and for all configurations, we can divide each curve into two main parts. First with W ranging from 0 to 30% of CTS, the model stay stable for a long time. So we can give permissions of advancing in time step between different machines until 30% of its critical time-step, and in this part all curves have a strong effect to the time window. The second part with W bigger than 30% of CTS, in this part the TSD is decreased slowly according to W time window.

6.0 CONCLUSION

To simulate millions or billions of interacting agents, we have to distribute our agent based simulator in order to scale it on network machines. A safe approach consists in splitting the environment into smaller parts and using a strong synchronization policy, but it implies a high cost in message exchanges and execution time.

This paper has explored a relaxation of this constraint to speed up execution time and has identified applications where this relaxation do not degrade simulations outcome. We have studied three synchronization policies for distributed multi-agent simulations: *strong synchronization*, *time window synchronization* and *no synchronization*. Experimentations show that some applications, like prey-predator model, stay stable with any synchronization policy. Whereas in others models, like capture the flags, it can be strongly affected by changing these policies. We have studied how interactions are changed when we switch the synchronization policies on prey predator model and we have explored in details the instability of capture the flags model when a biased initial configuration is used.

Experimentations presented in this paper are a first step, we have to experiment other kind of applications to illustrate synchronization policies impacts and see if results presented in this paper are suitable for other applications. For example, an emergency scenario of tsunami-town simulation which can be calculated faster with *no synchronization* policy than *strong synchronization*. Even, if some agents fail to interact, but we can get the main macroscopic behaviour, and we may be able to save more lives in such dangerous situation with *no synchronization* policy than *strong synchronization*.

References:

[1] Russell, S. J., Norvig, P., Candy, J. F., Malik, J. M., Edwards, D. D. 1996. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[2] Gold, T. 2003. Why Time Flows: The Physics of Past & Future. *Daedalus.* 132(2): 37–40.

[3] Lamport, L. 1978. Ti Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM 21*. 558–565.

[4] Jefferson, D. R. 1985. Virtual Time. *ACM Trans*. 7: 404–425.

[5] Scerri, D., Drogoul, A., Hickmott, S., Padgham, L. 2010. An Architecture for Modular Distributed Simulation with Agent-based Models. In: AAMAS '10: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, Richland, SC, International Foundation for Autonomous Agents and Multiagent Systems. 541–548.

[6] Siebert, J., Ciarletta, L., Chevrier, V. 2010. Agents and Artefacts for Multiple Models Co-evolution: Building Complex System Simulation as a Set Of Interacting Models. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 -Volume 1. AAMAS '10, International Foundation for Autonomous Agents and Multiagent Systems. 509–516.

[7] Logan, B., Theodoropoulos, G. 2001. The Distributed Simulation of Multiagent Systems. Proceedings of the IEEE. 89(2): 174–185.

[8] Fujimoto, R. 2000. Parallel and Distributed Simulation Systems. *Wiley Series on Parallel ad Distributed Computing.* Wiley.

[9] Gupta, B., Rahimi, S., Yang, Y. 2007. A Novel Roll-back Mechanism for Performance Enhancement of Asynchronous Checkpointing and Recovery. *Informatica, Slovenia*. 31(1): 1–13.

[10] Minson, R., Theodoropoulos, G. K. 2004 Distributing Repast Agent-Based Simulations with Hla. In: *In European Simulation Interoperability Workshop*. 04–046.

[11] Kiran, M., Richmond, P., Holcombe, M., Chin, L.S., Worth, D., Greenough, C. 2010. Flame: Simulating Large Populations of Agents on Parallel Hardware Architectures. In: AAMAS '10: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, Richland, SC, International Foundation for Autonomous Agents and Multiagent Systems. 1633–1636.

[12] Karmakharm, T., Richmond, P., Romano, D. 2010. Agent-based Large Scale Simulation of PedestriansWith Adaptive Realistic Navigation Vector Fields. In: *Theory and Practice of Computer Graphics*. 67–74

[13] Šišlák, D., Volf, P., Jakob, M., Pěchouček, M. 2009. Distributed Platform For Large-scale Agent-based Simulations. In: *Agents for Games and Simulations, Springer-Verlag, Berlin*. 16–32

[14] Cordasco, G., Rosario, D.C., Ada, M., Dario, M., Vittorio, S., Carmine, S. 2011. A Framework for Distributing Agent-based Simulations. In: In Proc. of The International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms. HeteroPar'11, Bordeaux, France.

[15] Cosenza, B., Cordasco, G., De Chiara, R., Scarano, V. 2011. Distributed Load Balancing for Parallel Agent-based Simulations. In: Parallel, Distributed and Network-Based Processing (PDP), 19th Euromicro International Conference on.

[16] Reynolds, C. 1999. Steering Behaviors for Autonomous Characters.

[17] Wilensky, U. 1997. Netlogo Wolf-Sheep Predation Model, Center For Connected Learning And Computer-Based Modeling, Northwestern University, Evanston, Il.

[18] P. Mathieu and O. Brandouy. 2010. A Generic Architecture for Realistic Simulations of Complex Financial Dynamics. In Advances in Practical Applications of Agents and Multiagent Systems, 8th International conference on Practical Applications of Agents and Multi-Agents Systems (PAAMS'2010). Springer. 185–197.

[19] Y. Kubera, P. Mathieu, and S. Picault. 2008. Interaction-oriented Agent Simulations: From Theory to Implementation. In Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08). IOS Press. 383–387.