

Synchronization Policies Impact in Distributed Agent-Based Simulation

Omar Rihawi, Yann Secq, Philippe Mathieu

▶ **To cite this version:**

Omar Rihawi, Yann Secq, Philippe Mathieu. Synchronization Policies Impact in Distributed Agent-Based Simulation. Distributed Computing and Artificial Intelligence (DCAI 2013), May 2013, SALAMANQUE, Spain. pp.19 - 26, 10.1007/978-3-319-00551-5_3 . hal-01115927

HAL Id: hal-01115927

https://inria.hal.science/hal-01115927

Submitted on 12 Feb 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Synchronization policies impact in distributed agent-based simulations

Omar RIHAWI, Yann SECQ and Philippe MATHIEU

Abstract When agents and interactions grow in a situated agent-based simulations, requirements in memory or computation power increase also. To be able to tackle simulation with millions of agents, distributing the simulator on a computer network is promising but raises issues related to time consistency and synchronization between machines. This paper study the cost in performances of several synchronization policies and their impact on macroscopic properties of simulations. To that aims, we study three different time management mechanisms and evaluate them on two multi-agent applications.

Key words: Distributed simulations, large scale multi-agent system, situated agent-based simulations, synchronization policies

1 Introduction

Multi-agent systems (MAS) are made of autonomous entities, that interact in an environment to achieve their own goals [11], producing emergent properties at the macroscopic level. To simulate millions of agents, we need a huge memory and computation. The goal in this kind of applications is not to observe individual interactions (microscopic level), but to observe population behaviours (macroscopic level). In some cases, if some agents fail or cannot interact as fast as other agents, it should not be critical to the global simulation outcome.

To achieve such scalability, one can distribute computations over a computer network. Nevertheless, in a distributed setting, problematics linked to time management and synchronization appear and have to be addressed. This paper presents a first study of synchronization costs in performances and the impact of synchronization policies on the preservation of emergent macroscopic properties.

LIFL (CNRS UMR 8022), Université Lille 1, France, First.LastName@univ-lille1.fr

The first section details the notion of simulated time and introduce three main synchronization policies. The second section introduces the platform that we have developed to experiment synchronization issues. The third and last section details experimentations made on two applications, prey-predator and capture the flag, to benchmark the impact of synchronization policies on simulation outcomes.

2 Time and synchronization in MAS simulation

The word *time* is often defined as a nonspatial continuum in which events occur in irreversible succession from the past through the present to the future (time flow notion [3]). In simulations, several notions of time are involved: user time (the real time) and simulated time, which is a set of small durations used to produce evolutions within a simulation. Simulated time has been defined by Lamport [7] through a logical clock that induce a partial ordering of events, and has been refined in distributed context as *Logical Virtual Time* (LVT) by Jefferson [5].

In MAS 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 as a simulation tick or *Time Step* (TS). In centralized MAS, there is only one TS that organize interactions between agents in a given period. Whereas, in a distributed MAS, there is one TS per machine which could not be the same on all. The main problem in such systems is time management between machines [12]. In order to guaranty causality, we have to synchronize TS between all machines. However, several policies of synchronization can be proposed. In distributed systems, there are mainly two approaches: *conservative* (or *synchronous*) and *optimistic* (or *asynchronous*) synchronization [8] [2].

We can define three synchronization policies for distributed MAS simulations: *1)* ***Strong Synchronization (SS)****:* in this policy: all machines TS are synchronized. This guaranty that all agents execute the same number of actions. But, the simulator needs more messages to synchronize machines. This conservative approach strictly avoids causality errors but introduce more delays. *2)* ***Flexible Synchronization****:* in this policy, machines are allowed to progress at different pace. As *optimistic* synchronization, which allows machines to advance in TS with detecting causality errors and recovering to previous states by rollback mechanism [4]. This mechanism rolls back by anti-event messages to reconstruct a previous state. But, if there are too much rollbacks, communication cost raises. *Time Window (TW)* synchronization is another flexible policy that can be proposed. In TW, machines can progress in TS until they reach the biggest possible difference W, between the slowest and fastest machines. With this window permission W, machines can avoid some delay of SS but in the same time, it can affect macroscopic behaviours outcome by allowing some agents in different TS to interact. These interactions can be ignored if their impact do not affect the simulation. This is a strong hypothesis that is studied in section 4. *3)* ***No Synchronization (NS)****:* last policy simply drops all synchronization

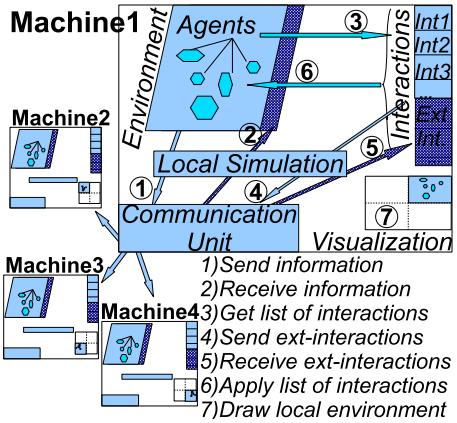


Fig. 1 Architecture of our testbed



Fig. 2 Five million agents on 50 machines

between machines. It is like TW policy with ∞ window size. This policy exploits the speed of all machines, but it is obviously not fitted for all applications.

To summarise, this paper studies whether synchronization constraints can be relaxed without impacting macroscopic behaviours emergence.

3 Testbed to benchmark synchronization policies

To evaluate synchronization policies, we have developed a distributed simulator. Each machine handles a subset of the environment with its agents. The simulator can be run with one of these policies: *strong*, *time window* or *no* synchronization.

Figure 1 shows 4 machines that execute a distributed agent-based simulation. Each machine has: *local simulation*, *communication unit* and *part of the environment* with its agents. *Local simulation* is a top-manager layer, which manages all tasks: from interactions between agents, receiving information from neighbourhood machines and local visualization. *Communication unit* manages exchange messages between machines and informs local simulation about machines TS. In each TS, machines follow 7 main steps: 1) it sends information to all neighbours about the environment state near them. 2) it waits for new information from others to inform local agents about the neighbours' environments. 3) each machine asks local agents about their next desired actions. 4) it sends the external interactions, which are interactions between agents from different machines, to others. 5) each machine receives interactions from others. 6) possible interactions are applied. 7) finally, local visualization and machines go to next TS. These synchronization policies have similar communication protocols with small differences. *Strong Synchronization* (SS) has in each communication state a notification, which requires an acknowledgement from others. Especially for the last state of communication, each machine should be synced for next TS and suspended until all others are ready. In case of *Time Window*

policy, each machine sends a notification of its current TS, and checks if it has permission to switch to next TS. If the difference between its TS and slowest machine TS is less than W number of steps (W is user defined) then it can progress to next TS. Whereas, *No Synchronization* (NS) policy, machines can progress to next TS without any notification.

To evaluate our testbed scalability, we have implemented a flocking behaviour similar to Reynolds [10] and run it on 50 machines, each machine holding 100000 agents (Figure 2). Many platforms already exist in the domain of distributed agent-based simulation: `Repast` [9], `FLAMEGPU` [6], `AglobeX` [13], and `DMASON` [1]. But all these platforms rely only on strong synchronization. Some platforms distribute the simulation on a network using a middleware, which provides a shared memory between Java Virtual Machines. So, it is free from all distribution considerations. Others are working with master/slave approaches with only strong synchronization mechanism through the server. So, no platform can be considered as a testbed for our work. Table 1 shows a comparison between all these platforms.

Table 1 Comparison between platforms

Platform	NbOfAgents	NbOfMachines	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 testbed	100 millions	200 machines	Prey-predator	SS, TW and NS

4 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 boids application can run without synchronization and still produce the emerging flocking behaviour. Thus, we want to determine with the following experimentations the impact of synchronization policies on simulations, more precisely on the expected macroscopic behaviour. To study synchronization impacts, we have implemented one application that is extremely affected by synchronization issues, while the other is not:

1) The prey-predator (PP) model is a classical multi-agent application that involve two kind of agents: preys and predators. Both kind reproduces themselves at a given pace, but predators seek and eat preys. If a predator do 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 [14].

2) Capture the flag (CTF) model has been developed to illustrate the fact that if a simulation outcome relies on timing issues (population growth speed in this case), thus 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. For each population, we have two kind of agents: flag agents that produce new attackers at a given rate, and attacker agent that protect their flags or attack the other population. When two agents attack each other, they both die.

4.1 Synchronization performance

We have executed experimentations on a LAN network where all machines have similar hardware. Most experimentations have run until 2 millions TS and the only parameter modified is W, the time window size, $W = 0$ *Strong Synchronization* (SS), then 10, 100, 1000, 10000, 100000 until ∞ or *No Synchronization* (NS). Table 2 shows that *No Synchronization* (NS) always gives the maximum speed, because it is free from all synchronization issues. But, in some application like CTF, the simulation outcome is biased.

Table 2 Prey-predator on 4 machines: *No Synchronization* always performs better

Time Steps	2000	4000	6000	8000	10000
SS execution time (hours)	1	2.5	3.5	4.7	6
NS execution time (hours)	0.8	1.4	2	2.7	3.4

A comparative test of both models stability (PP and CTF) shows that, PP model stays stable for more than 2 Millions TSs. For all experimentations, the co-evolution of PP has been preserved until we reach 2 millions TSs, even without any synchronization. Thus, this application is stable with respect to synchronization policies, stable meaning here that all types of agent still exist in the model. Whereas, CTF model has been unstable after W is bigger than N a number of steps ($N \geq 0$ depends on the initial configuration).

In next sections, we study invalid, internal and external interactions on the stable model to see how these interactions are changed when we switch synchronization policies. Then, we study CTF instability in details by exploring initial configuration biases.

4.2 Interactions effects on prey-predator (PP) model

External-Interactions (EI) are interactions happening between agents that belong to different machines. Figure 3 shows EI between agents which are executed in PP

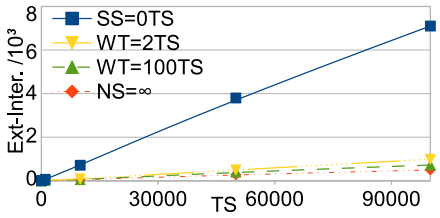


Fig. 3 External interactions in PP model with different synchronization polices.

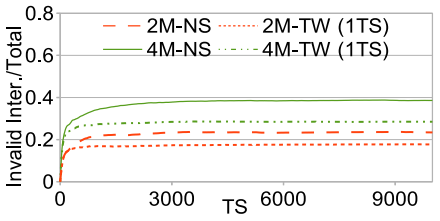


Fig. 4 Invalid interactions between 2 & 4 machines in PP model.

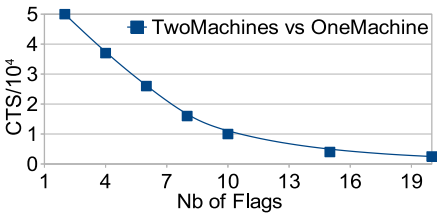


Fig. 5 CTS vs Number of flags in CTF model.

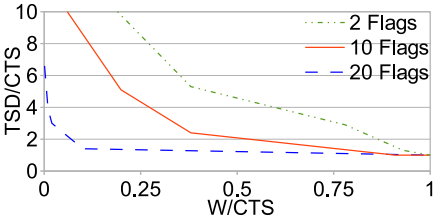


Fig. 6 *TSD* for different configurations of CTF Model with different size of TW (W).

model (with 5000 agents of wolf, sheep and grass), with size of TW: $W = 0$ or SS, 2, 100 and until NS or ∞ (all tests until TS = 100000). Results show that for SS we have a lot of EI, this is normal because information about agents can be sent and received by other machines. However, NS or TW significantly reduce EI, even with small window size. It can be explained because the information about agents is always sent to others machines, but it is not received from others within the corresponding time step. Thus, there is not a lot of EI because they are dropped because of differences between time steps. **Invalid-Time-Step Interactions** happens between two agents coming from different machines that are not in the same TS. Figures 4 shows that invalid interactions percentage increase with time window size (W). However, this percentage is less than 40% of total interactions. In case of 4 machines, we have doubled the percentage of two machines, which is because agents can swap between 4 machines more often than in two. Again, NS gives more invalid interactions than TW, because there are more differences between machines TS so information are dropped.

4.3 Capture The Flag model instability

As previous experimentations, simulations run on two machines for 2 millions TS and with a TW evolving between 0 to ∞. The first configuration is defined with one flag per machine. This configuration, nearly like the prey predator model, is stable with respect to synchronization issues. If we choose another configuration, like 20

flags per machine, we get different results. For TW size $W > 10000$ the model is no more stable and all flags from one population are captured.

We have defined another initial configuration to evaluate synchronization policies when machines loads are not the same. We have used three machines: the first contains all flags from the first population and two other machines contain half flags of the second population in each. The aim is to have more load on one machine, so the model becomes unstable: the second population should always win because its attacker production will be higher. The convergence should be faster with more flexible synchronization policies, and fastest without synchronization.

We define the *Critical Time Step* (CTS) as the necessary TS to completely destroy the model if no synchronization has been used. Figure 5 shows that CTS will be decreased by increasing the number of flags, from that $CTS = \frac{\alpha_1}{F}$, α_1 is constant which depends on the initial configuration and machines which have been used. We define the *Time-Step to Destroy* (TSD) as the necessary TS to completely destroy the model if W has been used ($TSD = CTS$ *if and only if* $W = \infty$ or NS). Figure 6 visualizes results after scaling each configuration to its CTS (as each one has its own CTS). This figure shows that the TSD decrease if W increase, then and according to the figure: $TSD/CTS = \frac{\alpha_2}{W/CTS}$, α_2 is constant. If we replace $CTS = \frac{\alpha_1}{F}$, then $TSD = \frac{\alpha}{W \times F^2}$, α is constant. That means, TSD decrease if the number of flags increase or W has been increased too.

Figure 6 also shows that for all configurations the model stay stable for a small TW. However, and in all configurations, we can divide each curve into two main parts. The first with W from 0 to 30% of CTS, the model stay stable for a long TSs. That mean, we can give permissions of advancing in TS between different machines until 30% of its CTS, and in this part W strongly affect all curves. The second part with W bigger than 30% of CTS, the TSD decreases slowly according to W.

5 Conclusion

To simulate billions of interacting agents, we have to distribute the simulator. 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, emergency scenario of town simulation can be calculated faster with *No Synchronization* policy than *Strong Synchronization*.

References

[1] Cordasco G, Rosario DC, Ada M, Dario M, Vittorio S, Carmine S (2011) A framework for distributing agent-based simulations. In: Proc. of HeteroPar2011, Springer Berlin Heidelberg
[2] Fujimoto R (2000) Parallel and distributed simulation systems. Wiley
[3] Gold T (2003) Why time flows: The physics of past & future. Daedalus 132(2):pp. 37–40
[4] Gupta B, Rahimi S, Yang Y (2007) A novel roll-back mechanism for performance enhancement of asynchronous checkpointing and recovery. Informatica (Slovenia) 31:1–13
[5] Jefferson DR (1985) Virtual time. ACM Trans Program Lang Syst 7:404–425
[6] Karmakharm T, Richmond P, Romano D (2010) Agent-based large scale simulation of pedestrians with adaptive realistic navigation vector fields. In: Theory and Practice of Computer Graphics, pp 67–74
[7] Lamport L (1978) Ti clocks, and the ordering of events in a distributed system. Commun ACM 21:558–565
[8] Logan B, Theodoropoulos G (2001) The distributed simulation of multiagent systems. Proceedings of the IEEE 89(2):174 –185
[9] Minson R, Theodoropoulos GK (2004) Distributing repast agent-based simulations with hla. In: Euro. Simulation Interoperability Workshop, pp 04–046
[10] Reynolds C (1999) Steering behaviors for autonomous characters
[11] Russell SJ, Norvig P (1996) Artificial intelligence: a modern approach. Prentice-Hall
[12] Scerri D, Drogoul A, Hickmott S, Padgham L (2010) An architecture for modular distributed simulation with agent-based models. In: AAMAS'10 Proceedings., pp 541–548
[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, pp 16–32
[14] Wilensky U (1997) Netlogo wolf-sheep predation model