Modelling and Simulation of Distributed Large Scale Situated Multi-Agent Systems

By

Omar RIHAWI

A thesis in the Department of Computer Science
Presented in Partial Fulfillment of the Requirements
For the Degree of

Doctor of Philosophy

Université Lille 1
03 Dec 2014

Approved by the Examining Committee:

Mrs.	Zahia Guessoum	Associate Professor (HDR), Université de Reims Champagne-Ardenne	(Referee)
Mr.	Laurent Vercouter	Professor, INSA Rouen	(Referee)
Mr.	Gregory Bonnet	Associate Professor (MCF), Université de Caen Basse-Normandie	(Examiner)
Mrs.	Sophie Tison	Professor, Université Lille 1 Sciences et Technologies	(Examiner)
Mr.	Yann Secq	Associate Professor (MCF), Université Lille 1 Sciences et Technologies	(Supervisor)
Mr.	Philippe Mathieu	Professor, Université Lille 1 Sciences et Technologies	(Supervisor)

UNIVERSITE LILLE 1 - SCIENCES ET TECHNOLOGIES
Laboratoire d'Informatique Fondamentale de Lille — UMR 8022
U.F.R. d'I.E.E.A. – Bat. M3 – 59655 VILLENEUVE D'ASCQ CEDEX
Tel. : +33 (0)3 28 77 85 41 – Fax : +33 (0)3 28 77 85 37 – email : direction@lifl.fr

Acknowledgements

First of all, I am grateful to **Allah** for establishing me to complete this work. *'No vision can grasp Him, but His grasp is over all visions. He is the Most Subtle and the Most Knowledgeable.'* [*AYAT al-Anaam 6:103, ALQURAN*].

Then, I would like to thank Prophet Muhammad (peace be upon him), the first and the greatest teacher for me. *'Philosopher, orator, apostle, legislator, warrior, conqueror of ideas, restorer of rational dogmas, of a cult without images; the founder of twenty terrestrial empires and of one spiritual empire, that is Muhammad. As regards all standards by which human greatness may be measured, we may well ask, is there any man greater than he?'* [*Alphonse de Lamartine (1790-1869) French poet and statesman*].

I wish to express my sincere thanks to the panel of expert examiners whom I defended for their questions and remarks that enriched this work. In detail, I wish to thank Mrs. Zahia Guessoum, Associate Professor (HDR) at Université de Reims Champagne-Ardenne, and Mr. Laurent Vercouter, Professor at INSA Rouen, for their reports that highlighted the ideas and the contributions of this work. Besides, I would like to thank Mr. Gregory Bonnet, Associate Professor (MCF) at Université de Caen Basse-Normandie, and Mrs. Sophie Tison, Professor at Université de Lille 1 Sciences et Technologies for accepting to examine my work.

I would like to express my sincere gratitude to my supervisors. Mr. Philippe Mathieu, Professor at Université de Lille 1 Sciences et Technologies and head of Research at SMAC team, for his help and for giving me the great honour to work my doctoral thesis in SMAC team.

Besides, I am deeply indebted to my supervisor, Mr. Yann Secq, Associate Professor (MCF) at Université de Lille 1 Sciences et Technologies, who has helped me to shape my research, and who has always been supportive and patient throughout the whole period of my study until the very last day before defense. I thank my supervisors for many fruitful discussions, ideas and their invaluable guidance and help during this research. I thank them also for their friendship and unconditional support. It has been a great privilege for me to work under their supervision.

I would like also to express my gratitude to all SMAC team members for endless support and encouragement since the very first day of my study: François, Yoann, Tony, Antoine, Sameh-Abdel-Naby, Iryna, Lisa, Fabien, Bruno, Sébastien, Maxime, Jean-Christophe, Patricia and Jean-Paul, and again my supervisors Yann and Philippe.

I would also like to thank all my colleagues at LIFL laboratory (now it is CRIStAL) for creating such an enjoyable working environment.

I owe a debt of gratitude to the financial support by Computer science department of Aleppo university.

Finally, I am particularly indebted to my family, without whom I would never have done anything. I would like to thank my mother, my father, my sisters and my brother who helped me to fulfill my dreams. Besides, I would like to thank my friends for all great days that we spent together.

Villeneuve d'Ascq, December 03, 2014 Omar Rihawi

Abstract

This thesis aims to design and implement a distributed simulator for large-scale situated-MAS applications of complex phenomena. Our goal is to support researchers by providing a software platform that can simulate very large-scale situated-MAS applications, which can simulate several millions of agents. When the number of agents and interactions reach such levels, it becomes impossible to use a single computer to execute the simulation. It is then necessary to distribute the simulation on a computer network; however, this can raise some issues: agents allocation, interactions between different agents from different machines, time management between machines, load-balancing, etc. When we distribute MAS simulation on different machines, agents must be separated between these machines because they have the most computational costs in the simulation. With this separation, agents should still be able to produce their normal behaviours. Our distributed simulator is able to cover all agents' perceptions during the simulation and allow all agents to interact normally. Moreover, with large-scale simulations the main observations are done on aggregated data, which are generally described as emerging properties at the macroscopic level. These emerging properties are induced by agents behaviours at the microscopic level.

In this thesis, we study two main aspects to distribute large-scale situated-MAS simulations. The first aspect is the distribution process or the efficient strategy that can be used to distribute MAS concepts (agents and environment) on a computer network. We propose two efficient distribution approaches: 1) *agents distribution* and 2) *environment distribution*.

The second aspect is the relaxation of synchronization constraints in order to speed up the execution of large-scale simulations. Relaxing this constraint can induce incoherent interactions, which do not exist in a synchronized context. But, in some applications that can not affect the macroscopic level.

Our experiments on different categories of situated-MAS applications show that some applications can be distributed efficiently in one distribution approach more than the other.

In addition, we have studied the impact of incoherent iterations on the

emerging behaviour of different applications, and we have evidenced situations in which unsynchronized simulations still produced the expected macroscopic behaviour. In other words, we have found that, for some applications, and for large-scale simulations with millions of agents, we can speed-up the execution time and we can highly reduce the communication costs by relaxing the synchronization between machines and keeping the macroscopic behaviour at the same time.

Résumé

Les systèmes multi-agents sont constitués d'entités autonomes qui interagissent avec leur environnement pour résoudre un objectif collectif. Les domaines d'application de ces systèmes vont de la résolution distribuée de problèmes à la simulation de phénomènes complexes. Ce type de simulation offre une granularité fine permettant d'exprimer les comportements à un niveau microscopique, c'est-à-dire individuel, et d'observer des phénomènes émergents au niveau macroscopique, c'est-à-dire de l'ensemble de la population. Néanmoins, lorsque le nombre d'agents et d'interactions augmentent, les ressources nécessaires en terme de puissance de calcul ou de capacité de stockage deviennent un facteur limitant.

Nous restreignons cette étude à la simulation d'agents situés dans un espace euclidien et dont les interactions ou les échanges de message ne peuvent se produire que lorsque deux agents sont suffisamment proches. Ce contexte correspond à la très grande majorité des applications réalisées dans le cadre de simulation d'agents situés comme la modélisation de traffic routier, d'écosystèmes humains ou biologiques ou encore de jeux vidéo.

Si l'on souhaite modéliser des systèmes contenant plusieurs centaine de milliers ou de millions d'agents, une puissance de calcul et de stockage importante devient nécessaire. Pour atteindre de telles simulations large échelle, distribuer le simulateur sur un réseau de machines est nécessaire, mais induit des problématiques de répartition de charge, de gestion du temps et de synchronisation entre les machines et de tolérance aux pannes.

En plus des problèmes de coûts de communication classiques dans le contexte d'applications distribuées, il faut prendre en compte des aspects plus spécifiques liés au fait que les agents sont situés dans un environnement. En effet, les dynamiques de déplacement des agents induites par leurs comportement affecte la répartition de la charge de calcul sur le réseau. Ces problématiques font l'objet de recherche actives, particulièrement sur la question du placement des agents sur un réseau de machines, notamment lorsque les agents sont mobiles. Dans ce contexte, la gestion du temps et de la synchronisation sont également des problèmes importants.

Le premier aspect de notre travail se concentre sur deux types de répartition de la charge de calcul : la première basée sur une répartition des agents en fonction de leur proximité, la seconde répartit les agents indépendamment de leur positionnement dans un objectif d'équilibrage de charge. Nous évaluons les performances de ces répartitions en les confrontant à des applications dont les dynamiques de déplacement sont très différentes, ce qui nous permet d'identifier plusieurs critères devant être pris en compte pour garantir des gains de performance lors de la distribution de simulations d'agents situés.

Le second aspect de notre travail étudie la problématique de la synchronisation des machines. En effet, à notre connaissance, tous les simulateurs existants fonctionnent sur la base d'une synchronisation forte entre les machines, ce qui garantit la causalité temporelle et la cohérence des calculs. Dans cette thèse, nous remettons en cause cette hypothèse en étudiant la relaxation de la contrainte de synchronisation. Le fait d'autoriser la progression d'agents dans différentes temporalités induit des interaction incohérentes, c'est-à-dire se produisant entre des agents n'appartenant pas au même pas de temps. La question qui se pose est alors de savoir si ces incohérences induisent une perte du phénomène émergent de la simulation et si ce n'est pas le cas, d'évaluer le gain en terme de performance du relâchement de cette contrainte. Il est d'ailleurs envisageable que pour certaines applications, des erreurs ou échecs d'interactions entre deux agents ne soient pas critiques pour le résultat global de la simulation. Afin d'étudier cette problématique, nous proposons deux politiques de synchronisation : la synchronisation forte classique et une forme de synchronisation reposant sur une fenêtre de temps bornée entre la machine la plus lente et la machine la plus rapide. Des applications de natures différentes sont exécutées avec ces différents mécanismes de synchronisation. Nous étudions dans cette thèse leur coût en performance ainsi que leur impact sur l'émergence des propriétés macroscopiques des simulations. Nous nous intéressons particulièrement au seuil critique d'interactions temporellement invalides qui entraînent un biais dans le résultat de la simulation.

Contents

Introduction

I.1 Motivations

There is a growing need to provide scientists with large-scale agent-based simulations to express, execute and analyse complex phenomena. When the number of agents or interactions grows to millions or billions, the simulation of such system requires important computational power and memory volume. The best way to achieve large-scale simulations is to distribute them over a computer network, that can speed up the simulation and improve the performance.

This thesis study different approaches to simulate large-scale distributed multi-agent simulations. This work is aimed to design a distributed multi-agent simulator in a high efficient way. The work of this thesis combines different aspects of *multi-agent systems* (MAS) and *distributed systems* to achieve large-scale simulations. The challenge was to modify the existing ideas of general distributed systems into new ideas which depend on the main concepts of multi-agent system, which are *agents* and *environment*. In our point of view, distributing multi-agent simulations should depend on these two concepts.

Thus, to build a distributed system for MAS simulations and optimize it for large-scale applications, we need to merge different aspects of two main domains (figure I.1):

1. The MAS domain: agents, environment, interactions, etc. All MAS concepts should be defined and implemented in a specific manner to achieve an efficient execution in a distributed environment. The distributed simulator should be able not only to distribute the computation and exchange messages between different machines, but also should respect agent technologies in execution and implementation: autonomously, mobility, etc. Thus, the distributed simulation should be built upon the main MAS concepts which are *agents* and *environment*.

2. The distributed system domain: computational load, communication,

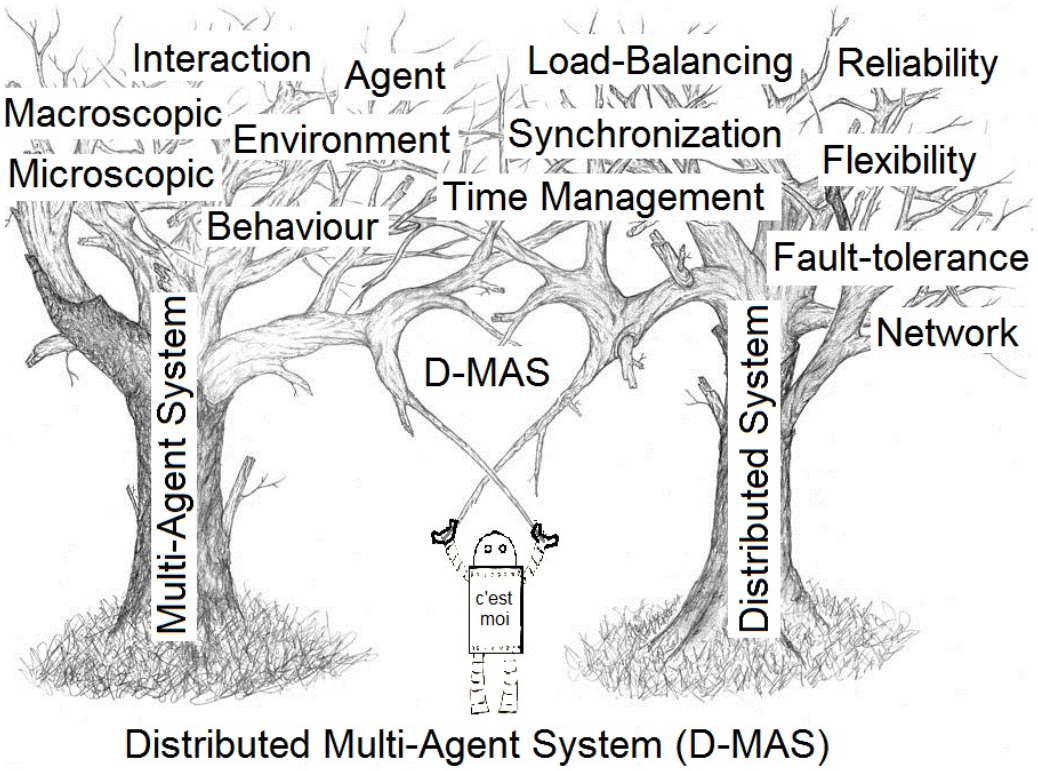

Figure I.1: For large-scale MAS simulations: D-MAS = MAS + Distributed System.

load balancing, synchronization, fault-tolerance, etc. The distributed-MAS platform should be structured to execute MAS simulation in an efficient way. The load between distributed machines should be balanced and should be based on agents and environment concepts. Communications and synchronization between machines should be implemented in a specific way to support interactions between agents.

This work proposes a distributed platform to implement large-scale situated-MAS simulations, and it is optimized to distribute agents and environment which are the main MAS concepts. To do that, we need to implement both domains in a specific way to respect each other. We need to implement MAS concepts on a distributed manner and in the same time we need to execute the distributed MAS simulation upon agent-based technology.

I.2 Objectives of the thesis

The primary goal of this thesis is to assist scientists with the necessary platform to implement, execute and analyze large-scale simulations of complex phenomena. Our research has achieved this goal by answering these questions:

- ***How to simulate large-scale multi-agent simulations?***
  To simulate a large number of agents or interactions that grows to millions or billions, the simulation needs high computational power and large memory volume. To reach such large scalability, we decided to choose the distributed system and to distribute the computation over a computer network.

- ***How to distribute situated-MAS concepts: agents and environment?***
  We can distribute MAS simulation with many ways, but not all of them can be in an efficient way. In this thesis, we suggested two efficient ways to distribute MAS simulation which derived from its main concepts: agent and environment.

- ***How to solve the agent placement problem in a distributed environment?***
  When we distribute MAS simulation between different machines, agents should be separated between these machines and they should still be able to produce their normal behaviours. To do that, the distributed simulator should cover all agents' perceptions during the simulation.

- ***How to manage interactions between different agents running on different machines?***
  When agents are located on different machines and some agents need to interact with others that exist on different machines, then we need to implement an agreement protocol between these machines to allow this kind of interactions.

- ***How to reduce communications costs in large-scale simulations while keeping the macroscopic behaviour?***
  With large-scale simulations, the goal is not to observe millions of individual interactions or microscopic level, but to observe properties at macroscopic level. If we 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 MAS

simulation with millions or billions of agents, and some agents fail to interact, that should not affect the global behaviour of the system or macroscopic level. In that case, we can relax the synchronization and reduce communication costs between machines to gain more speed in execution time at least for some applications.

These research questions lead us to develop a distributed-MAS simulator for large-scale applications.

I.3 Research contributions

With our simulator, the user can simulate his applications on a simple distributed peer-to-peer computing network in an efficient way. The user can configure his simulation easily to achieve his goals:

1. The user can distribute his application with two different distribution approaches. He can choose which one is the best for his application.

2. The user can optimize the simulator to reduce communication costs and network load by relaxing synchronization constraints between machines. Moreover, he can choose between three different synchronization policies to achieve his goal with a shorter execution time.

In this thesis, we are able to develop large-scale MAS simulations with different distribution approaches and with several synchronization policies. A comparison between different MAS applications shows that some applications are more fitted on some distribution types than others.

I.4 Thesis organization

This thesis begins with this general introduction (this chapter), then the body of the work (on three parts) and later the conclusion. The body of the thesis is divided into three main parts: the first part is the state of the art on two chapters, the second is our main work on two chapters and the third part is our experimental results on two chapters too.

Chapter 1 investigates the first part of the state of the art, which deals with multi-agent systems. We focus on situated-MAS simulations and its main concepts which are: environments, agents and interactions. First, we explore the beginning of these simulations types. We present different definitions of the agent technology and different environment notions. We explore three levels of behaviours: micro\meso\macroscopic behaviours. We present

some centralized-platforms in MAS domain. We summarize the notion of time and the time step or the *round of talk* between interacting agents in MAS simulation.

In chapter 2, we present the second part of the state of the art, which deals with distributed systems. A background of distributed systems is presented. We explore the essentials of distributed systems in: network relationships and topologies, load-balancing and time management, etc. Different synchronization mechanisms are then explored. We propose our main criteria to build an efficient large-scale MAS simulation: which are emanated from the basic concepts of MAS: *agents* and *environment*. Finally, we introduce and compare some existing platforms in the domain.

Chapter 3 defines our view of MAS and its main concepts. We deeply discuss how the distributed MAS simulation should be designed in our view. We first describe the importance of different distribution approaches. We present our understanding of agents and environment, and we distinguish between two categories of interactions. We compare different interaction mechanisms between centralized and distributed cases. Then, we detail different distribution approaches: *agents distribution* and *environment distribution*. Time management problems and synchronization policies are deeply studied to distribute MAS simulations efficiently and gain more performance. The question was 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. To study that, we propose three synchronization policies, two of them are flexible synchronizations that allow machines to avoid communications delays. Thus, in this chapter two distribution approaches and three synchronization policies have been presented.

In chapter 4, we describe our platform for large-scale distributed multi-agent systems. We present the different components of the platform by detailing the machines units that manage the main operations of the simulator in two different distribution cases. The basic platform layers are discussed deeply and different simulation states are introduced which are the *initial state* and the *running state*. We describe by UML sequence diagram the main steps that can be followed by different machines to distribute the simulation. After that, we study communication protocols and synchronization algorithms for three main synchronization policies. We explain the time step scenario between two machines to illustrate the differences between these policies. Finally, we describe our view of visualization in large-scale simulations.

The experimentations are divided into two chapters. In chapters 5 & 6, the results of different distribution approaches with several synchronization

policies on different MAS applications are presented. Chapter 5 presents our experiments on both distribution approaches to show that some applications are fitted more on one distribution approach than the other. First, we explore different categories of applications that can make our hypothesis clear. Then, we detail the two classical situated agent-based models, that we use in our experimentations. We measure the performance of the simulations with two criteria: *execution time* and *communication costs*. Our experiments show that one applications model is better with one distribution approach than the other. Then we explore more in details the two distribution approaches with different experiments.

In chapter 6, we study the performance costs of several synchronization policies and their impact on the properties of MAS simulations. We evaluate our simulator with two different situated MAS applications. We have studied three synchronization policies for distributed MAS simulations: *strong synchronization*, *time window synchronization* and *no synchronization* policies. We present how interactions are changed when we switch between these policies, and we study the instability of some application with different configurations when we relax synchronization constraints.

In chapter 7 the conclusion of the current work and several suggested ideas for the future works are presented.

In appendix A a summary of the work in French language is written.

Part I

State of the art

Chapter 1

Situated agent-based simulations

1.1 Introduction

In the last decades, simulations have been used by scientists to build and simulate many complex real-life phenomena. Some phenomena cannot be experienced directly and have to be simulated in order to test different hypothesis. Scientists usually use numerical approaches to study these phenomena. Those approaches normally depend on macro analysis of the phenomena which can be represented by some mathematical equations. This approach can provide models that are close to the complex phenomena but generally do not explain why the phenomena appears.

Since the last 80's, the *artificial intelligence* field has developed the notion of intelligent agents and multi-agent systems. In this context, agents perceive their environment and pursue their own goals, by cooperating or being in competition with other agents. With this approach, the phenomena is designed at the individual level. At this micro-level, interactions should produce emergent behaviours (or the macro-level), which reproduce the dynamic of complex phenomena. Contrary to mathematical approaches, multi-agent simulations provide clues on the main properties that lead to the emergence of the complex phenomena.

In the 90's, many scientists still used numerical approaches in their simulations, because artificial intelligence techniques and especially intelligent agents technology needs high computational power. With the growth of processor speed and memory volume, more scientists have begun to use agent based approaches.

Nowadays, the availability of huge computing power allows researchers

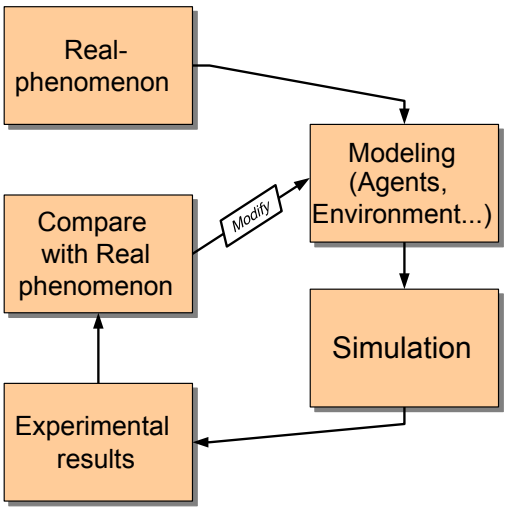

Figure 1.1: Simulation was first used in ecology to study a specific phenomenon.

to focus on individual micro modelling, instead of mathematical equations. Moreover, scientists needs have also grown, and simulations with millions of agents and interactions are still a challenge that need to be tackled.

This chapter investigates many concepts of multi-agent systems. We focus on situated-MAS simulations and their main concepts: environments, agents and interactions. First we explore the beginning of these types of simulations. Then, we present different definitions of agents and environment. We explore different levels of behaviours. Finally, we present the notion of time and time step in MAS simulations.

1.2 Situated-MAS simulations

Agent-based simulations have been used effectively in different domains, e.g, in ecology, robotics, etc. These types of applications were often called *Individual Based Models* (IBMs) [Grimm and Railsback, 2013] and were used in situations where individual variability of the agent was important. An example is population dynamics in ecology of prey-predator model where its agents behave identically [Judson, 1994]. To build such simulations, we start by modelling the structure, and then we run it to compare the result with the real phenomenon (see figure 1.1).

Multi-agent systems are increasingly needed in many scientific domains like: e-commerce, ambient intelligence [Vallee et al., 2005] or industrial do-

mains. One of these domains represents a family of real-world applications, where agents are explicitly placed in an environment (see figure 1.2). In these types of real-world applications like manufacturing control or transportation systems, situated agent-based simulations may be the most suitable solution. It is more efficient and flexible than other systems. Indeed, agent-based simulation provides a field to test agent behaviours (algorithms) under ideal conditions. Those algorithms that fail under ideal conditions can, in general, be eliminated in realization. Better agent behaviour evolution becomes possible, and many ideas can be tested before the real employment of the system.

Situated agent-based simulations are made of an environment that is populated with a set of localized agents which cooperate to solve a complex problem. Situated agents are autonomous entities that encapsulate their own behaviour and maintain their own state. They have local access to the environment (local vision), e.g, each agent is placed in a local context which it can perceive, act and interact with other agents.

The approach of situated multi-agent systems came from main ideas of *embodiment situatedness* and *emergence of intelligence* [Brooks, 1991]. In Brooks works, a situated agent does not use long-term planning to decide what action sequence should be executed. Instead, it selects actions on the basis of its position, the state of the world that it perceives and the limited internal state. In other words, the situated agent acts in the present time by its position and its state. The intelligence in a situated multi-agent system is produced from the interactions between different agents rather than from their individual capabilities.

Earlier works of MAS were done on robotics, Maes has introduced the basic mechanisms of a group of robots that coordinate through an environment a context of MAS simulations [Maes, 1990]. Wooldridge refers to some points of situated MAS simulation [Wooldridge, 2009]. That situated agents take into account only local current information and thus it is short-term-view. However in complex applications, it can be feasible or even useful for agents to collect global information or to have a long-term-view on the situation, if agents are more cognitive.

1.3 Main concepts of MAS simulations

In multi-agent systems: agents, environment and interactions are the main concepts that are derived from the application type. These concepts determine if the simulation can be situated or not, and if it can be large scale or not. First, we describe what is an agent and the importance of the environment,

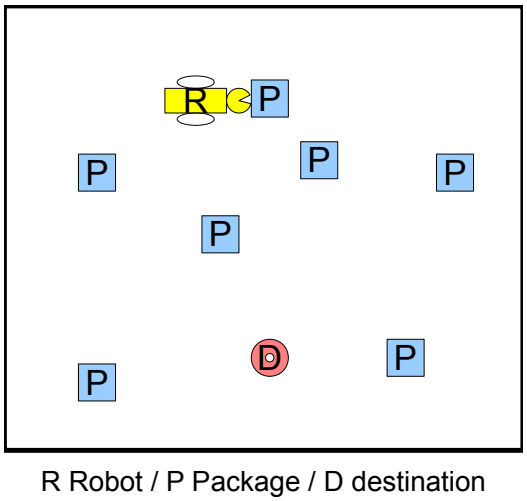

Figure 1.2: An example of situated MAS simulation: packages and destinations model [Weiss, 1999], later known as Packet-World [Weyns et al., 2005]. Agents are robot, packages and destinations. Robot has to deliver the packages to the correct destinations and must handle only one at a time.

then we detail interaction types.

1.3.1 Agent

The *intelligent agent* [Russell et al., 1996] is an autonomous entity which observes its environment and acts by following its own goals. However, there are different historical definitions of agents:

- *"An agent is an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices and commitments" [Shoham, 1993].*
- *"An entity is a software agent if and only if it communicates correctly in an agent communication language" [Genesereth and Ketchpel, 1994].*
- *"Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment, action to affect conditions in the environment and reasoning to interpret perceptions, solve problems, draw inferences and determine actions" [Hayes Roth, 1995].*
- *"A computer system that is situated in some environment and is capable of autonomous action in its environment in order to meet its design objectives" [Wooldridge, 2009].*

- *"An agent can be a physical or virtual entity that can act, perceive its environment and communicate with others. It is autonomous and has skills to achieve its goals and tendencies" [Ferber, 1999].*

Intelligent agents can use old knowledge or learn new one to achieve their own goals [Russell et al., 1996]. Agents can decompose their goals and tasks and give sub-tasks to other agents to achieve collective results. Agents can communicate with others by specific protocols or languages to improve the state of their coordination and to act coherently. Agents are able to interpret their environment and try to achieve their own goals.

They can be very simple or very complex as we can see in table 1.1. 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 agent can be cognitive agents which can learn and update their strategy. Agents can range from purely reactive agents (simple strategy) to cognitive agents (complex strategy) by involving abstract knowledge representation and planning systems. 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). Everything can be agent [Kubera et al., 2010].

Table 1.1: Agents can range from passive entities to complex proactive entities.

Passive (Object)	Reactive without goals	Reactive with goals	Cognitive specialized	Cognitive learner
Apple	Door	Bird	IBM's Chess player	Human

1.3.2 Environment

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.

Russell lists different dimensions and properties of MAS environments [Russell et al., 1996]:

- **Spatial** (discrete or continuous): the discrete or continuous environments come from the importance of a geographical space or the spatial interactions of situated agents. For example in packet-world, robot can move between blocks of a grid map or with floating position on the map (figure 1.2).

- **Observable** (fully or partially): if an agent has access to the complete state of the environment at each point in time, then the environment is fully observable. Otherwise, it is partially. Generally, in situated agent-based simulation each agent has only a partial vision of the environment.

- **Changeable** (static or dynamic): the environment is dynamic for an agent if the environment can change while an agent is acting. Otherwise, it is static.

However, some of these dimensions are related more on agents than the environment itself. In our study, we distinguish between three main environment types:

- Virtual environments: where there is no spatial dimensions like: a stock market simulation [Mathieu and Brandouy, 2010], reputation [Muller and Vercouter, 2005] and trust management problem in web Services [Bourdon et al., 2009].

- Discrete environments: where there are spatial dimensions on a grid. For example, a prey-predator simulation with IODA [Kubera et al., 2008].

- Continuous environments: where there are spatial dimensions with floating positions for all agents, like birds flocking in a sky [Cosenza et al., 2011].

The different environments types can be 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 and have positions on this environment. However, there are other kinds of applications where we have a mixed environments between spatial and nonspatial cases [Bonnet and Tessier, 2009].

Within a spatial environment, a distinction can be made between discrete and continuous environments. In a discrete context, the environment is made of a grid and its agents can move in this grid by swapping between its cells. In continuous environments, the space is represented by ranges and agents have floating positions. Virtual environments have no spatial aspect, their role is restricted to agent communication.

Everything in multi-agent systems, that is not inside an agent, can be considered as an environment of that agent. The environment represents the

space where all agents can move. For example, the environment represents the euclidean-space in which robots or agents can move. For market simulations, the mediator between sellers and buyers represent the environment where they can exchange information about their products.

Obviously, any agent can have a certain representation of its environment, even if it is virtual and it is not specialized. However in some types of applications, the environment is ignored as it is not so important, but this is not our case. We believe that any agent must be able to locate itself within an environment, then this environment can manage and define the capacity of interaction for each agent.

1.3.3 Interaction

An interaction is a change, a react or an interact on the state of one (or more) agent(s) caused by one (or more) agent(s) within a t period of time. There is a strong relation between interaction and time in any agent-based simulation. Each interaction has a period of time to be executed and scheduled with other interactions during the simulation.

However, some researchers investigate the notion of interactions and have different categorizations for them. In [Ngobye et al., 2010], there are 3 different types and priorities of MAS interactions according to its relation to other agents:

1. NI: No direct Interaction between agents.
2. SI: Simple Interactions between agents.
3. CI: Complex, or Conditional, or Collective Interactions between agents.

Other researchers divide interactions into basic behaviours and group behaviours [Mataric, 1994]:

1. There are basic behaviours: safe-wandering, following, surrounding, etc.
2. Interaction can consist of N other hidden-basic-interactions: combining basic behaviours like: $flocking = safe - wandering + aggregation + dispersion$ or $herding = safe - wandering + surrounding + flocking$

Some works try to design a general taxonomy of multi-agent interactions, for example in [Dyke Parunak et al., 2004] there are two types of interactions:

1. direct: from agent to agent

2. or indirect: through environmental variables.

However, we distinguish between two classes of interactions according to its execution complexity. First class is a simple interaction where agent does not need to modify another agent. The second class is the complex interaction where agent needs to modify a state of another agents. In the first class, actions can happen in the same time with the ability to overlap. For example two agents said '*do something!*' to third agent or two agents sent an answer of a question to third agent. Whereas in the second class, there are no possibilities to allow any kind of conflicting actions. For example, two wolf-agents can not eat the same sheep-agent at the same time.

1.4 From micro to macroscopic behaviours

To study a society in social science, people are divided into group [Yurdusev, 1993]: 1) *Micro-level group* which is the small and local community social unit in which everyone within that group knows everyone, like a small village. 2) *Meso-level group* which is intermediate group, and it is large enough to let some members of the group may never know about all other members, and may have still access to the leaders, but it is not so large group. Examples of meso-level are states. 3) *Macro-level group* is the larger population which operate at a national or a global level, and it determines the interactions outcomes of the sociality. Examples of macro-level are countries.

Agents in multi-agent system can be divided into different social units, where all agents can interact appropriately and cooperate successfully to meet their own goals. We can distinguish between different levels (see figure 1.3):

1. Micro-level (or low-level): agent and its interactions.

2. Meso-level (middle-level): collective or group of agents that can behave common interactions.

3. Macro-level (or high-level): view of all agents that behave as one global behaviour.

An example of such system is a traffic simulation [Sanderson et al., 2012], where there are vehicles in micro level, group of vehicle in meso level and the flow of traffic as macro level. Other examples are cosmological simulations, where there are several levels of agents as in [Torrel et al., 2007] with 4 levels. However, we can use micro, meso or macroscopic behaviours to build multi-agent systems with two main ways:

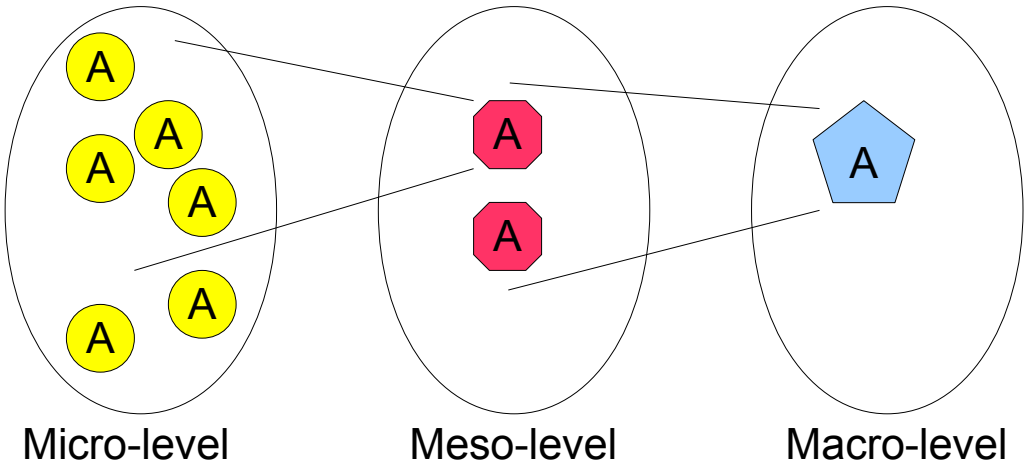

Figure 1.3: Micro meso macro: different levels of details in urban simulations [Navarro et al., 2013].

- Top-Down: from desired group behaviour to specific needed interactions at the level of the individual agents.
- Bottom-Up: from micro-level (agent capabilities) to achieve overall macro-level.

We can use both ways to build our simulations, first we define the desired macro-level behaviours that we need it from the simulation. Then, we define the needed low-level agents and their interactions. After that, we define agents and their interactions, when we implement the simulation, to achieve the global behaviour of the system.

1.5 MAS platforms overview

Many platforms and methodologies already exist in MAS domain: Swarm [Minar et al., 1996], Netlogo [Wilensky, 1997], Gama [Taillandier et al., 2010], Adelfe [Bernon et al., 2003], etc. However, we do not describe these platforms as they are not in our circle of interest (not distributed platforms). For more details, the survey of [Nikolai and Madey, 2009] analyses many MAS platforms.

The SMAC team has developed a centralized MAS platform, which is JEDI or *Java Environment for the Design of Agent Interactions*, that is based on IODA methodology [Kubera et al., 2011] to design MAS simulations by its interactions. This methodology has a different understanding of MAS simulations than others.

Source \ Target	∅	Grass	Sheep	Goat	Wolf
Grass	(Grow, p = 0)				
Sheep	(Move, p = 1)	(Eat, d = 0, p = 3)	(Procreate, d = 1, p = 2)		
Goat	(Move, p = 1)	(Eat, d = 0, p = 3)		(Procreate, d = 1, p = 2)	
Wolf	(Move, p = 1)		(Eat, d = 3, p = 3)	(Eat, d = 3, p = 4)	(Procreate, d = 1, p = 2)

Figure 1.4: IODA matrix [Kubera et al., 2011] of prey-predator model, each interaction has a priority (P) to be organized and a distance (D) to be triggered.

The *Interaction Oriented Design of Agent simulations* (IODA) [Kubera et al., 2011] is a methodology to design MAS simulations by focussing on interactions. Instead of defining only the agents of the simulation, IODA focuses on the notion of interactions between agents.

IODA designs agents with their interactions through the interaction matrix. This matrix can be automatically interpreted and executed by the simulator. For that, it is easy to compare interactions in the simulation and it is easy to change its priorities to define which interactions can be before or after.

In figure 1.4, an example of how IODA can define the matrix of prey-predator model, where each interaction has a source and target agents and has a priority to execute it. In addition to define each agent, IODA has to define each interaction by:

1. The source and the target agents of the interaction.
2. The priority of the interaction, so the highest priority interaction has to come before the lower priority interaction.
3. The distance to trigger the interaction.
4. The condition to trigger the interaction and the interaction itself.

The first three points are defined in the interaction matrix. Whereas, the last point must be defined outside the matrix.

1.6 Notion of time and time step in MAS

The word *Time* in language dictionaries is a non spatial continuum in which events occur in apparently irreversible succession from the past through the present to the future. Many researchers define time by events, for example

'*virtual time*' [Jefferson, 1985] and time ordering events in [Lamport, 1978]. In physic, the time came from the concept of moving objects, as Earth moves around itself to make duration, and so on... etc. Anyway, most of these definitions define time by duration which is happened by events, so it defined time by the time itself (duration). Moreover, some researcher reach to announce that time is not real, e.g '*The Unreality of Time*' [McTaggart, 1908].

However, it is not important for us to find a clear definition rather than to determine what we want from time and how we can simulate it. We, as a part of MAS community, want to determine what is time that we can use and how we can simulate it to make events.

1.6.1 Time step

In agent-based simulation, actions (or events) happen between agents in sequential order. At each time one agent should make one interaction only in one environment, or more than one interaction if these interactions on different environments. Thus, the time can be imagined as a series of sequential steps: *step 0, step 1, step 2, ... step n.* Each step has a duration, even if this duration is epsilon ϵ, but it is not zero. Because, different steps without durations must be in the same point in the time dimension and that make conflicting actions or implicit actions, which should not be happen in the simulation.

For us, the **Time Step** (TS) is the smallest time which used by the system to manage the fastest interaction from all agents' interactions on the system. In other words, a system with smaller time steps ($< \epsilon$ time) can reserve many empty time steps and no agent can produce any kind of interactions during these steps. Empty time steps are not useful and it is wasting time of the simulation without interactions.

1.6.2 Round of talk between agents

Each agent in MAS simulations behave to achieve its own goals. MAS simulation should use a fairness *updating scheme* to update the environment with agents behaviours and interactions, because different updating schemes may lead to different evolutions of the system [Fatès and Chevrier, 2010]. Sometimes, a desired behaviour of one agent can be interrupted by another behaviour of other agents. Thus, the simulation has to organize all interactions between agents in the system and must be fair among all these agents. Therefore, no agent has a chance to behave more than others or before others all the time.

That can be called *round of talk* between agents. Simulation gives all agents the same chance to interact one time only and prevents conflicting interactions between agents. This prevention can be made with sequential round of talk between agents, so the first agent needs an action can have it and other agents can not. For fairness, this sequential ordering can be changed randomly between agents after each round of talk. Then, all agents have their chance to be first in the ordering list. However, different methods can be implemented here.

1.7 Summary

In situated MAS simulation: agents, environment and interactions are the main concepts that are used to model an application. Different definitions of agent, interaction and environment can make different understandings of MAS simulations. That can lead to different methodologies and different implementations.

In our view, the agent is one solid component of properties (or states) and desires engine (or behaviours). Agent can explore part of the environment (other agents) and behave from its old state, and according to its desires engine, to reach a new desired state. The environment is an an agent container and an interaction mediator. The environment provides all information about all agents that exist in the simulation. Thus, for each agent there is its environment which is the other agents. We consider that any related changes in one or more of agents' states is an interaction.

In this chapter, We review the first part of our state of the art which is multi-agent system and especially the situated-MAS simulation. We presents the main principles of situated-MAS simulation. We review the most important definitions of MAS concepts: with a large variety of agents definitions, different environment types and different interactions categories. We shows the relation between micro\meso\macroscopic view and MAS modelling. We presents some centralized-platforms in MAS domain. We summarize the important of time in the situated-MAS simulation and the round of talk between interacting agents.

In the next chapter, we explore the second part of the state of the art which focuses on distributed large scale MAS simulators.

Chapter 2

Large scale distributed agent-based simulations

2.1 Introduction

In the last 80's, the artificial intelligence led to the appearance of intelligent agents technology in the computing domain. That technology needs high computational powers especially for large scale simulations. In some cases, the number of simulated entities in one simulation can be very large. The resources of one computer can not always ensure the execution of large scale simulations. A single machine computation can give us limitations when applying this kind of simulation to some phenomena. If we have a huge number of agents and even if these agents are very simple, one machine can not deal with it effectively. To run large scale simulation, we should distribute the simulator on a network of many computers to get the maximum computation capability.

Most of the existing MAS simulations toolkits available are either single machine simulations, that are not suited for large scale simulations, or they are driving from existing general distributed systems that are not necessary specialized for MAS concepts (agents or environment). For this reason, general distributed systems for multi-agent applications are sometimes used for deploying MAS simulation on general distributed resources. Thus, some platforms do not scale well for large MAS simulations due to wrong initial-load balancing of its agents or due to huge communication overheads.

Moreover, changing an existing MAS platform and transfer a single machine MAS simulation into a distributed environment is not an easy task. In addition, reaching the desired high performance with lowest communication costs by changing a general distributed system is not a good idea too. For

that, our work is focused on studying some problematics of current distributed MAS platforms, and find the missing aspects to distribute efficiently large scale MAS simulations.

This chapter is the second part of the state of the art, which deals with distributed systems. we define the distributed system and its main concepts in: network relationships and topologies, load-balancing and time management, etc. Different synchronization mechanisms are then explored. We propose our main criteria to build an efficient large scale MAS simulation: which are emanated from the basic concepts of MAS: *agents* and *environment*, which has different distribution approaches for different applications that can automatically balance the computational load during the simulation, which has different synchronization approaches to increase the performance while keeping the macroscopic behaviour, which are not customized for specific applications. Finally, we introduce and compare some existing platforms in the domain.

2.2 Distributed systems

There are several definitions and view points on what distributed systems are. Various types of distributed systems and applications have been developed, and many of them are being used extensively in the real world. A distributed system can be:

- *"a system in which hardware or software components located at networked computers communicate and coordinate their actions only by message passing" [Coulouris et al., 2005].*
- *"A collection of independent computers that appear to the users of the system as a single computer" [Tanenbaum and van Steen, 2008].*
- *"A distributed system is one on which I cannot get any work done because some machine I have never heard of has crashed" Leslie Lamport.*

The diversity of all these definitions reflects the huge number of challenges faced by designers to build a general distributed system.

Our study is focused on the way to distribute large scale multi-agent simulations and its problematics: network topologies, time management, agents migration between machines and load balancing. All these problematics are still under active researches, some works investigate the *agent placing* problem, n agents distributed on m machines [Miyata and Ishida, 2008] and what can happen when agents try to move between machines [Motshegwa and Schroeder, 2004]. However, our view of agent placing problem is not

restricted to agents only, but can also be related to simulation framework components (*meta-agent* in [Horling et al., 2004]), that could be distributed.

Time management is also an important problem that many researchers investigate. Several models have been proposed: a single global logical time step for the system or multiple time steps in each machine [Scerri et al., 2010] [Siebert et al., 2010]. Earlier works have been done on *Virtual Time* [Jefferson, 1985], which explain it for discrete event simulations.

2.2.1 Network relationships and topologies

Distributed systems are commonly used to support applications emerging in the areas of science, which commonly involve geographically distributed utilities in collaborative activities to solve large scale problems and require sharing of various resources such as computers, data, applications and scientific instruments.

Network relationships can be categorized into two main categories: control and physical relationships. The control relationship can be *client\server* or *peer-to-peer* (P2P). In *client\server* networks, all clients, that need accesses to some resources, must connect through the server. Whereas, in *peer-to-peer* networks all devices share their resources and are not controlled centrally. Moreover, P2P systems are often presented as a promising approach to build scalable distributed applications [Muller and Vercouter, 2005].

Network topology is a term used to described the physical relationship of network connections, which can be: *bus*, *ring*, *star*, *mesh* or *tree*. P2P can be either unstructured in physical relations or structured in one or more of network topologies.

However, there are various types of distributed computing systems from network view, or the connections between distributed computational units [Nadiminti et al., 2006]:

- A *cluster* is a dedicated group of interconnected computers that appears as a single powerful computer, generally used in high performance scientific engineering and business applications.
- A *grid* is a type of distributed system that enables coordinated sharing and aggregation of distributed, autonomous, heterogeneous resources based on users' requirements.
- A *Peer-to-Peer network (P2P)* which is a decentralised distributed system, that enables applications to be distributed over public networks. An example is distributed tasks between sharing computational resources [Jin et al., 2005].

In cluster or grid networks, the computational units should be in most of cases fully connected with a switch in star topology. That means, all computational units link to all others. Whereas, in P2P they can be in one or more (hybrid) of the five topologies.

2.2.2 Load balancing

Load balancing has been studied extensively in general distributed systems and different mechanisms has been proposed. In distributed systems, load-balancing is a method to keep the computational and communication loads balanced between all computational units. Load balancing is very important to build a useful distributed system, and especially to build a useful distributed MAS simulation for large scale applications, which has some specifications. Few researchers' works are done on the load-balancing aspect to get the spirit of agent systems in such a distributed simulation [Cosenza et al., 2011].

In general distributed systems, all computational units are often viewed as common resources, where users can submit their tasks through connected machines. Then, a load balancing mechanism can distribute the tasks among different machines to balance the workload, and then tasks can run until they are finished. New tasks may be added to some machines at any time by the users, and can be scheduled by the load balancing mechanism.

This scenario does not respect agent technology and can not be applied to large scale distributed multi-agent simulations. First, agent can be built from different tasks but it cannot be split into different machines or cannot be executed on different time steps during the simulation. Agents can still exist in the system unless the whole system is shut down or they decide to stop interacting with others. In addition, in most of MAS applications agents are specified and created at system start-up and in sometime there is a need to introduce more agents to the system. Moreover, agents can interact between themselves through variable communication forms which depends on interaction's nature. Whereas, the communication between the tasks in normal distributed systems are usually with static and fixed forms. Thus, large scale distributed multi-agent simulations need some specifications.

However, it is true that agent in MAS is one solid unit and it is difficult to split it into smaller parts. Agent could dramatically change its behaviours and then it could change the load during the simulation. This highly depends on the applications and agents types that are used. Anyway, changes in simulations can be categorized into two main categories:

1. Agent computation: which can increase or decrease during the simula-

tion. If we have a mixed list of agents between cognitive and passive agents in one machine, the load of calculation can be fine in the system. While, if we have all heavy cognitive agents in one machine and other machines hold passive agents or simple agents, then the load could not be the same between machines.

2. Agent communication: which can also increase or decrease during the simulation. If we have agents that need to communicate with other agents that exist in different machines, then communication costs can be high and the load of communication between two machines is more than others in the system. While, if the agents with heavy communications exist in the same machine, then the communication between machines should be similar and the load should be good.

However, according to the nature of agent's interactions the load balancing scheme could be changed during the simulation.

2.2.3 Fault tolerance

Fault tolerance is an important feature in distributed systems that enables these systems from operating normally in case of failure. Many techniques can be used for fault tolerance, one of them is snapshot (checkpointing) and backward recovery [Elnozahy et al., 2002]. Another technique is the *replication* of data or agents in case of multi-agent systems. To prevent failures during the simulations, some *critical agents* should be detected and replicated [Guessoum et al., 2003].

Moreover, in large scale simulations the fault-tolerance should have some specifications [Guessoum et al., 2005]. This is because, in these simulations the number of agents can be large and a dynamic approach of replication can be useful in that case [Guessoum et al., 2003]. DimaX [Faci et al., 2006] is an example of these platforms with dynamic *replication* technique, which is the result of the integration of DIMA multi-agent platform and DarX replication framework.

However, in situated MAS simulations for natural phenomena there are no *critical* or *non-critical* agents, because all agents are similar to each other. Thus, snapshot and backward recovery could be more suitable for situated MAS applications.

Table 2.1: OTS or MTS in agents level or in machines level.

Time	Agents level	Machines level
OTS	All agents in the same TS	All machines in the same TS
MTS	Window with N TS between fastest and slowest agents (maximum parallelism)	Window with N TS between fastest and slowest machines

2.3 Time management

The other main problem in distributed systems is *time management* between machines [Scerri et al., 2010; Siebert et al., 2010]. Many distributed systems try to define time by events, for example *virtual time* [Jefferson, 1985] and time ordering events [Lamport, 1978].

Anyway, if we use events in general distributed systems or interactions in distributed MAS simulations, we need to use a time step mechanism to manage both of them. This mechanism can be changed according to the application and user needs. In a distributed simulation, as we have different computational loads on different machines, then we could have multiple time steps.

2.3.1 Multiple time steps

In centralized multi-agent systems, we have one global time step which organizes all agents, and allows agents to interact in a given period of time -if and only if- this period of time is bigger than the interaction itself. However, for a distributed system two types of time step can be proposed:

1. *One global Time Step (OTS)*: strong synchronization between machines or agents

2. *Multi Time Steps (MTS)*: flexible synchronization between machines or agents

However, OTS or MTS can be used in any kind of MAS applications on more than one level: agents level or machines level (see table 2.1).

1. In agents level: OTS is reliable for all applications, whereas MTS can not be reliable for some application. An example of that is *Boids* simulation (from *Bird-oid*) [Reynolds, 1987], birds can keep the flocking

behaviour even if some birds are in different and incoherent time steps. Whereas, in other applications MTS could not keep the macroscopic behaviour.

2. In machines level: OTS means more communication costs between machines to keep all of them on the same time step. Whereas, MTS means that some machines can progress in the simulation more than others to avoid high communications. In agent-based simulation, MTS in machines level should lead to MTS in agent level too.

In distributed systems, OTS or MTS can be represented by synchronization approaches. There are mainly two synchronization approaches in general distributed systems: *conservative* (or *synchronous*) and *optimistic* (or *asynchronous*) synchronization.

2.3.2 Conservative synchronization

In *conservative* (or *synchronous*) mechanism [Logan and Theodoropoulos, 2001; Fujimoto, 2000], all machines are synchronized together in such a way that all local clocks are running at the same pace and the system should be on one global time step (OTS). Thus, the distributed simulator can guaranty that all agents execute the same number of actions. Of course, more messages should be exchanged between machines, and then communication costs are increased. This kind of conservative approach strictly avoids causality errors but can introduce high communication delays. In other words, all machines guaranty the correct execution for all parts of the simulation by additional messages that introduce more delays for each time step.

2.3.3 Optimistic synchronization

This second mechanism allows machines with MTS to progress at different pace with *optimistic* (or *asynchronous*) synchronization [Logan and Theodoropoulos, 2001; Fujimoto, 2000]. The main issue is to handle causality errors by detecting and recovering them through a rollback mechanism [Gupta et al., 2007]. 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.

With this approach, machines should take checkpoints independently without any synchronization among the others. Unfortunately, because of

the absence of synchronization, there is no guarantee that local time steps are the same in all machines, and for that reason some errors can be produced sometimes. In order to get rid of these disruptions, machines have to roll back to older checkpoints.

The gain of optimistic approaches is based on the fact that simulators should not roll back too often for small incoherence interactions. Moreover, it can be ignored if we believe that the impact of time incoherency in some interactions is negligible with respect to the volume of interactions in the whole system. Thus in large scale simulations, this flexibility should not affect macroscopic behaviours outcome and that is a strong hypothesis that will be studied through different applications in next chapters.

However, the balance between the performance, communication costs, and reliability depends on the application that the user needs to implement. For example, if we have a large scale simulation of an animation movie with large number of armies, it is not important if some agents fail to interact or interact slower than the others. Whereas, in other kind of applications, we have to get a maximum reliability to the system to get the reliable macroscopic behaviour.

2.4 Distributed agent-based platforms

Even if many platforms are able to distribute MAS simulations [Cabri et al., 2004; Jamali and Zhao, 2008], large scale situated-MAS simulations are still under active research. Moreover, some platforms are not fitted for situated applications. In our view, distributed MAS platforms must take into account the main concepts of MAS, which are *agents* and *environment*, to be efficient platforms. In other words, we consider that a distribution, that comes from MAS concepts, can make a good load-balancing for the simulation and that can increase the performance. Moreover, with different synchronization approaches we can increase the performance for any distributed agent-based simulation (Table 2.2).

However, modifying a good centralized agent-based simulation to be run on a distributed environment is not an easy task. For example, the execution mechanism of JEDI (IODA) is not fitted to be run in a distributed manner. The agent can change immediately the environment or other agents. In other words, the current mechanism in JEDI sequentially executes agents' interactions. In a distributed system, several agents can be running at the same time and can have conflicting actions. There is currently no method in

Table 2.2: The main criteria to compare platforms of large scale MAS simulations.

Criteria	Efficient D-MAS platform
MAS concepts	Emanated from the basic concepts of MAS: *agents* and *environment*
Computational load	Has different distribution approaches for different applications which can automatically balance the load during the simulation. Otherwise, different applications can upset the balance
Time management	Has different synchronization approaches to increase the performance while keeping the macroscopic behaviour
Application domain	not customized for specific applications
Scalability	Should reach a high number of agents

JEDI to manage such conflicts and the organizer in JEDI simulation should be changed completely to distribute MAS concepts.

There are also many platforms as *Java Agent DEvelopment Framework*, or *JADE* [Bellifemine et al., 2005], which allows the development of FIPA-compliant for multi-agent systems. JADE provides a set of interfaces to design agents in Java. This framework also provides a set of pre-defined behaviours like `Simple Behaviour`, `Cyclical Behaviour` or `Non Deterministic Behaviour`. However, JADE is more fitted to coordination and communication of non-situated agents systems. JADE does not provide simulation capabilities as time steps for interactions cycle between agents [Yoo and Glardon, 2009; Jinkai and Weihong, 2010].

2.4.1 D-MASON

An interesting work in the domain is *D-MASON* [Cordasco et al., 2011]. *D-MASON* is the distributed version of *Multi-Agent Simulator Of Neighbourhoods Networks* or *MASON* [Luke et al., 2005]. It is a Java-based library for agent-based simulations and geared towards speed. *D-MASON* is based on a master\workers machines (figure 2.1), the master assigns a part of the whole computation or a set of agents to each worker. Then, for each simulation step, each worker simulates the assigned agents and sends back the result of its computation to each interested worker. Many platforms have been built with this mechanism. However, most of these platforms use simple applications to illustrate their scaling (*D-MASON* uses flocking model). Those simple models cannot produce complex interactions. For example in flocking model,

birds will flock only by watching other birds. Thus, there are no interactions between two birds that can explode the communication costs in the system. However, in other classical models like prey-predator model, agents can produce complex or conflicting interactions. For example, two wolves want to eat the same sheep from different machines. In this situation a protocol of agreements must be provided to resolve these conflicting interactions.

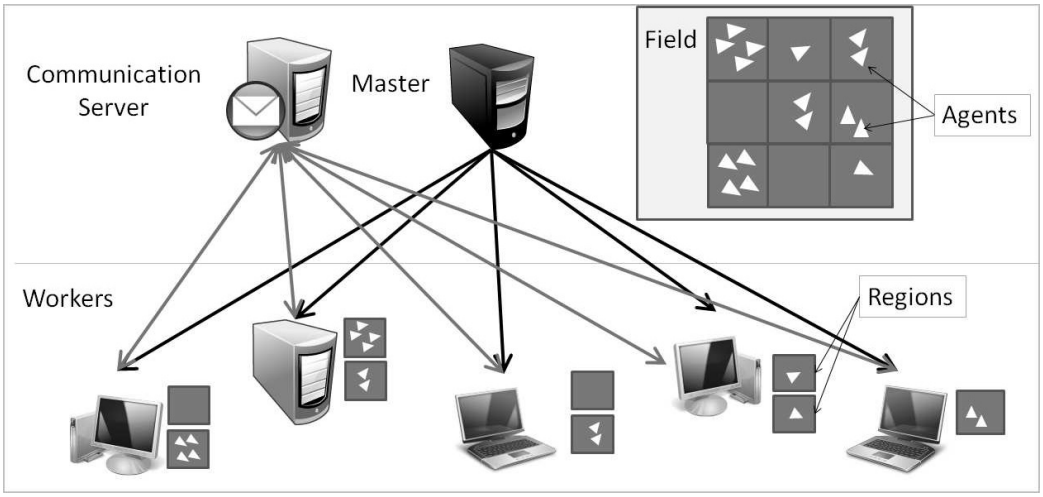

Figure 2.1: D-MASON [Cordasco et al., 2011]: different master\workers machines run a flocking model application.

2.4.2 AglobeX

Another platform is *AglobeX* [Šišlák et al., 2009]. This platform is very similar to *D-MASON* in its distribution mechanism. As *D-MASON*, *AglobeX* is based on a master\workers approach, the master assigns a portion of the whole computation or a group of agents to each worker. Then, for each simulation step, each worker simulates its local agents and sends back the result of its computation to each interested worker (figure 2.2). For that, *Aglobex* has been built with the same mechanism of *D-MASON*. However, both platforms use simple models, *AglobeX* uses airplanes model and *D-MASON* use flocking model. Those simple models cannot produce any conflicting interactions.

2.4.3 GOLEM

Another interesting platform is *Generalized Onto-Logical Environments for MAS* (*GOLEM*) [Bromuri and Stathis, 2009]. *GOLEM* uses Ambient Event

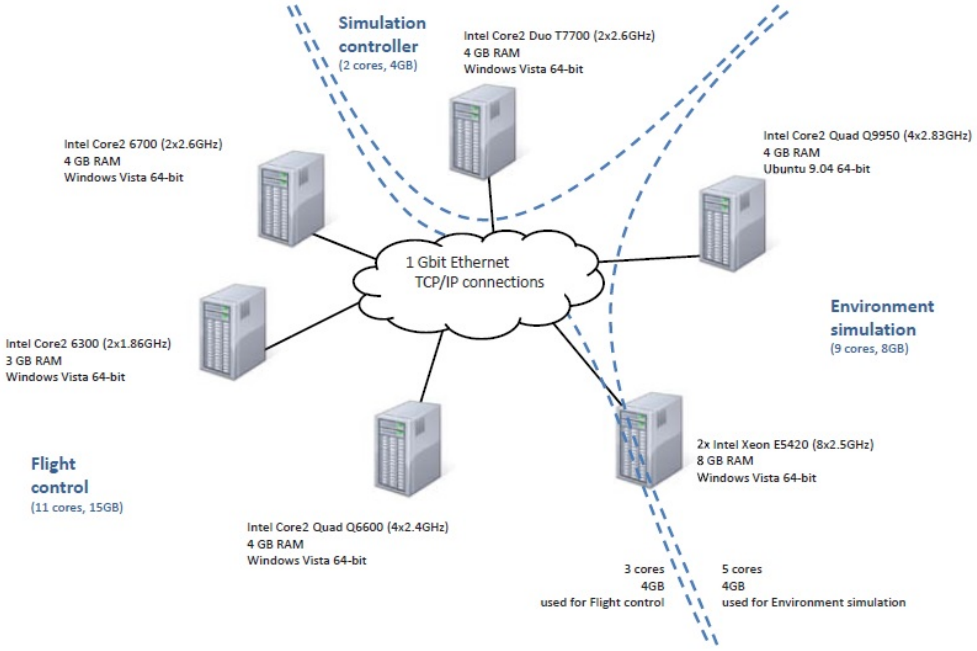

Figure 2.2: AglobeX [Šišlák et al., 2009]: different machines run an airplanes application.

Calculus language, which is a formalism to build cognitive agents. This platform consists of `Containers` which can be distributed between machines. Each `Container` can be composed of other `Containers` and this can complicate the execution hierarchy (see figure 2.3). This platform is more suited for cognitive agents than massively reactive agents and it does not guarantee high performance in case of massively multi-agent simulation [Bromuri and Stathis, 2008]. Also, it is not clear how they can distribute agents or environment on different `Containers`.

2.4.4 Repast

The *Recursive Porous Agent Simulation Toolkit*, or *Repast* [North et al., 2013], provides components for building multi-agent simulations. Its execution is based on events submitted by agents. Simulations relies on ticks, minutes of simulated time where actions are carried out. By default, the *Repast Symphony* platform does not have native tools for distributed simulations. There is another implementation of Repast with a Relogo specific language on parallel environment, which is *Repast for High Performance Computing* (or *Repast-HPC*) [Collier and North, 2012].

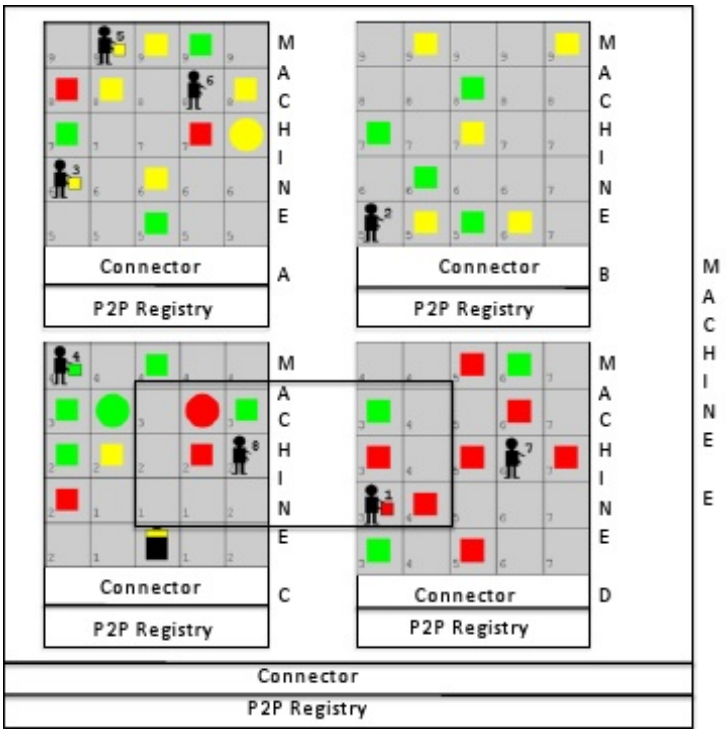

Figure 2.3: Packet-World application distributed on 5 machines with GOLEM [Bromuri and Stathis, 2009].

It is possible to launch a simulation on a network platform using a middleware, which provides a shared memory between machines. On one side, the main advantage in this solution lies in the fact that the platform is free from all distribution considerations. But on another side, this means that the distribution is made without any consideration of the platform: no optimization is made on the various components of the platform (including the run-time). Similarly, it is not possible to parametrize agents distribution on the computer network. Distribution through a middleware is ambiguous and does not have a clear distribution of the main concepts of MAS simulation (agents and environment).

Another version of *Repast* is *Repast-HLA* [Minson and Theodoropoulos, 2004] that is related to *High Level Architecture* (*HLA*) [Kuhl et al., 1999; Xiaoxia and Qiuhai, 2003], which is a more standard middleware, that has a strict hierarchical tree-oriented model. Communication between certain simulators is enabled by predefined gateways (figure 2.4). Thus, HLA can be considered as a centralized coordination approach to distributed simulation resources [Timm and Pawlaszczyk, 2005]. *HLA* clearly focuses on the coordination between different sequential simulation toolkits. Anyway, *HLA* has some limitations in complex systems and is not mainly designed to gain speed up [Davis and Moeller, 1999].

2.4.5 IBM Megaffic Simulator

Megaffic or *IBM Mega Traffic Simulator* is a large scale simulator for traffic application. This platform is built upon XAXIS or *X10-based Agent eXecutive Infrastructure for Simulation* [Suzumura and Kanezashi, 2012b,a], which is designed to handle large number of agents on distributed and parallel computing environments (HPC).

Its middleware is designed and implemented from X10 middleware [Ebcioglu et al., 2004], which is a parallel distributed programming language that IBM-Research is developing. X10 comes from a language called Partitioned Global Address Space (PGAS) and specific for large-scale traffic simulation, which was designed to test and experiment Tokyo road network data.

However, The agent programming model of XAXIS is derived from ZASE simulation platform (or *Zillions of Agents-based Simulation Environment*) [Yamamoto et al., 2008]. The negative points of this platform are the dependence on a specific language (PGAS) and the narrow application domain which is traffic simulation (figure 2.5).

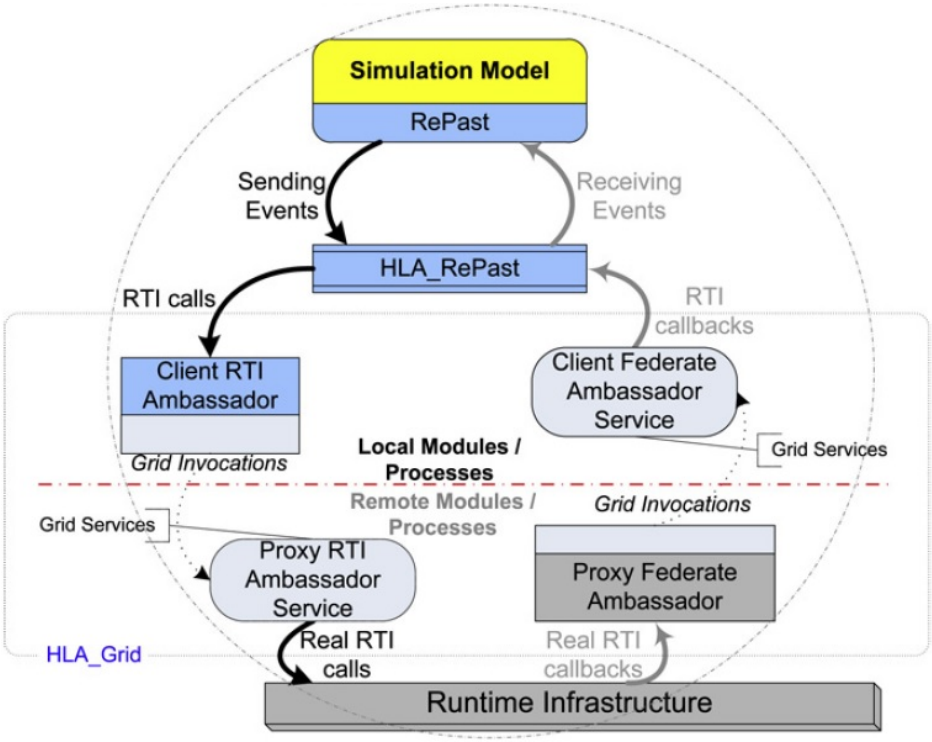

Figure 2.4: Repast with HLA architecture.

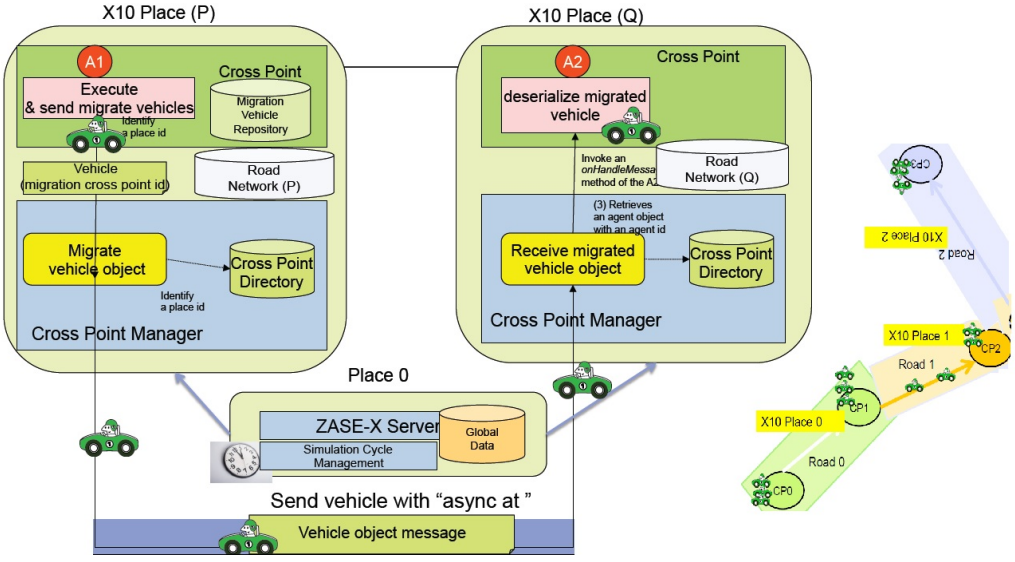

Figure 2.5: Megaffic simulator of traffic simulation: each road is represented by one machine.

2.4.6 FLAMEGPU

FLAMEGPU is an extension to the *FLexible Agent-based Modelling Environment* (or *FLAME*) framework for *Graphics Processing Unit* (GPU) [Richmond et al., 2009] (figure 2.6). *FLAMEGPU* merges between the X-Machines agent definition and GPU programming specifications. GPU are designed to process simple and large computations in parallel structures. It is clear that rendering a 3D environment with GPU is faster than using CPU, because GPU are better at performing repetitive tasks on large blocks of data than CPUs. For that, *FLAMEGPU* can visualize large amount of simple agents in real time as agent data is already located on the GPU hardware. But with complex computations, CPU is still better than GPU, especially if we have a cognitive agent with complex computations.

However, as GPU were designed in parallel structures to allow large blocks of data to be processed at one time. Then, the workload must be divided into small blocks to allow the parallel smaller processor units in GPU to process them successfully. That make the programming in GPU has some limitations.

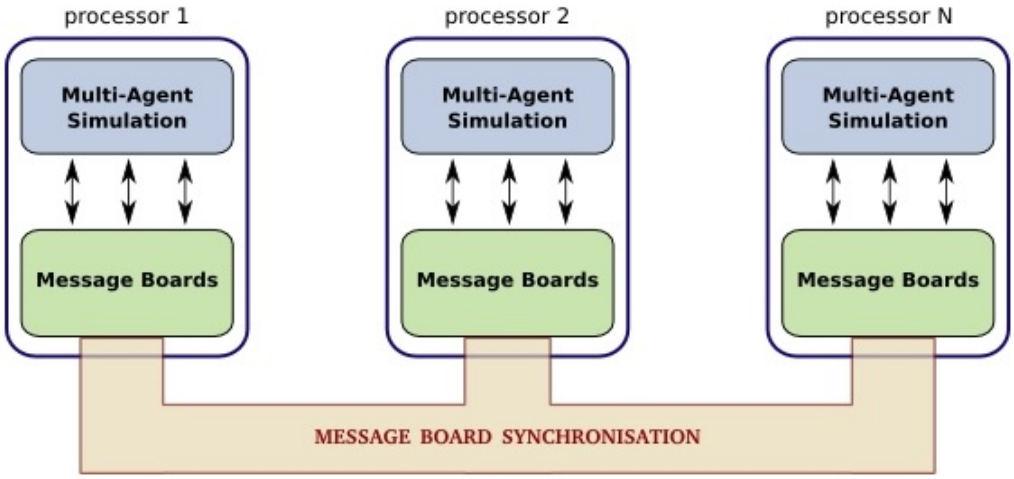

Figure 2.6: FLAME with distributed message-boards.

Even if a single GPU is able to simulate several thousands of agents, we still have a limitation to simulate several millions or billions of agents with a single GPU, even if these agents are very simple.

2.5 Summary

Even if many platforms are able to distribute some systems, large scale MAS simulations need some specifications and features. In this thesis, we focus on these features and put them under our research. Many platforms already exist on the domain, but most of them are not efficient for large scale applications. Table 2.3 shows a comparison between these platforms and their scalabilities of: the agents number, the machines number and the most commonly used applications to benchmark their works. Some of these platforms are scaled well, but with simple applications. Megaffic was built for traffic simulation only. All these platforms do not have different synchronization capabilities. And they do not have different distribution approaches which can be chosen by the user.

Distributed MAS simulators must take into account the main concepts of MAS, which are *agents* and *environment*, to increase the efficiency and the performance. We consider that the initial load-balancing and the time management are the most important features to increase the efficiency for any large scale distributed-MAS simulation.

This chapter introduced some concepts of distributed systems, which was the second part of the state of the art. We define different aspects of the distributed system like: network relationships and topologies, load-balancing and time management, etc. We explore different synchronization approaches. We proposed different criteria to build an efficient large scale MAS simulation. We believe that large scale MAS simulation should be emanated from the basic concepts of MAS: *agents* and *environment*. It should have different distribution approaches for different applications that can automatically balance the computational load during the simulation, otherwise it can upset the balance. It should have different synchronization approaches to increase the performance by avoiding some useless communications. Finally, it should not be customized for specific applications.

In the next two chapters 3 & 4, we present the second part of this thesis. Chapter 3 explores our understanding of large scale MAS simulators and the main ideas that should be taken into accounts to design and simulate large scale applications. Chapter 4 details our platform and its functionality.

Table 2.3: A comparison of large scale MAS simulators.

Platform	Scalability		
	Agents Nb	Machines Nb	Model type
Repast	68 billions	HPC-32000 cores	Triangles (simple)
DMASON	10 millions	64	Boids (simple)
AglobeX	6500	6 (22 cores)	Airplanes (simple)
GOLEM	5000	50	Packet-World (complex)
FLAMEGPU	11000	GPU	Pedestrian Crowds
Megaffic	10 millions	16 (192 cores)	Traffic simulation

Part II

Distributed MAS simulators

Chapter 3

Distributed-MAS: concepts and ideas

3.1 Introduction

Agent-based simulations are used by researchers to provide explanations about real life phenomena like flocking of birds, or to simulate population evolutions. These kinds of simulations are made of smaller entities called agents, which interact to produce emerging global patterns.

When the number of agents or interactions grows in large scale simulations, resources in computing costs and memory can exceed the capacity of a single computer to execute such simulations. For that, a distribution of MAS simulator over a computer network to divide the computations between different machines is our circle of interests.

In this chapter, we first describe our view of MAS concepts and the importance of different distribution approaches. We detail our understanding of agents and environment, and we distinguish between two categories of interactions. We compare different interactions mechanisms between centralized and distributed cases. Then, we detail different distributions approaches for MAS concepts, which depends on agents and environment. These distributions can adapt initially the loads between machines for some applications without any load-balancing mechanism. Finally, time management problems and synchronization policies are deeply studied to distribute MAS simulations efficiently and gain more performance.

3.2 The need of different distribution types

To reach a high number of agents and interactions in large scale MAS simulations, we believe that it is necessary to distribute these simulations over computer networks. To ensure performance gains when we distribute such simulations, it is necessary to take into account specific concepts (as agents & environment) more than the classical ones (as computation & data storage) in normal distributed systems. Our work presents these concepts: *agents* and *environment*, and their importance in large scale simulations.

We believe that the way, which we use to distribute MAS concepts, can give us more performance in some applications than others. In some applications, the environment is more important than agents and can play a major role in the distribution process, and an environment-based distribution can give us better performance in such applications. Whereas in other applications, the agent can be the most important thing, and a distribution related to agents instead of environment can give us better performance. However, communication costs also can be affected by the chosen distribution way. To study that, we propose different distribution approaches, which are related to MAS concepts.

Our aim is to reach a way that can help user to choose the best distribution approach for his application according to some general features.

3.3 MAS concepts between centralized and distributed approaches

In centralized approaches, MAS simulation can be implemented in different ways with similar performance. Whereas in distributed approaches, we need some specific techniques to avoid high communication costs, and make efficient distribution especially when a large number of agents interact. An efficient distribution means that we have to use all machines capabilities to calculate interactions of all agents. In other words, the loads between machines must be similar.

For example in a centralized approach, if there is a list of agents that want to interact, then, we can simply allow first agent to interact, then the second one, etc. One after another respectively until the last one. Because in one machine, the calculation have to be done sequentially. Whereas in a distributed approach, this scenario is not efficient, because agents are distributed between different machines. That means, some machines have to wait till other machines can finish their computations. To avoid such problems, we have to define MAS concepts in a specific way.

3.3.1 Agents

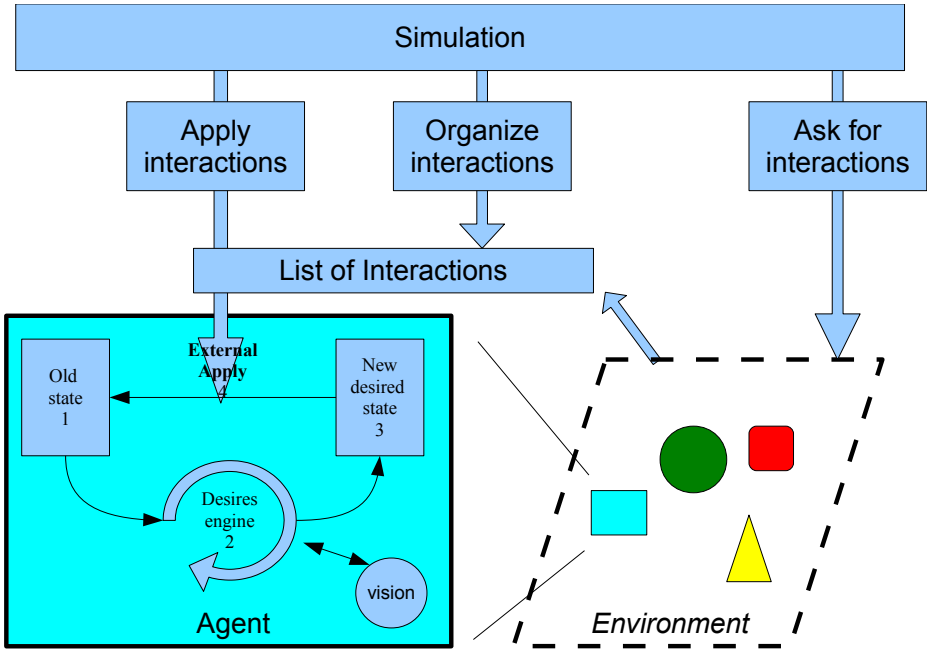

Figure 3.1: Agent model in MAS simulations. Agent consists of an old state, desires engine, vision and a new desired state. When MAS simulation asks each agent to interact, each agent behaves from its old state, and according to its desires engine with its vision, to a new desired state.

Far from the classical definition of agent as: an autonomous entity which observes its environment and acts by following its own goals [Russell et al., 1996], we describe here our view of an agent to implement it easily and effectively in a distributed system.

Figure 3.1 shows our view of agent that consists of properties (states) and behaviours (desires engine). In MAS simulation, agent behaves from its old state, and according to its desires engine, to a new desired state. That can be done if and only if the environment accepts this new state or other agents acknowledge that change. Then, we can apply the interactions or we can change the state of agent to its new desired state (see figure 3.1). To notice, a new desired state is not necessary an interaction, it can be a proposed interaction by the agent to be the new interaction. If we imagine that, all agents have a list of proposed interactions, then we can apply the possible ones only.

3.3.2 Environment

Agent's environment in our view is everything outside the agent. Environment is not only positions (x,y) of agents. But, it is also other agents or the information about the other agents. Environment of an agent should support this agent about the states of other agents. Environment should provide information about the possible interactions which can be applied during the simulation.

Environments can be classified in three main forms: *virtual environments*, *discrete environments* or *continuous environments*. In our view, *Continuous Environment* is more general than discrete or virtual ones. Because discrete environment can be inherited from continuous one by a division of it into zones on a grid. Virtual environment did not have any specification for the space and it can be implemented as continuous one too. However, we are not interested in virtual environment as our main focus on situated simulations.

Each application can have a continuous environment even if its agents can not float on the environment. In that case, environment has only information about other agents. However, information about agents must be transferred in efficient way to avoid high communication costs in distributed system.

3.3.3 Interactions

In chapter 1, we have seen different interactions categories from different researchers' works on different levels. However, we distinguish between two main categories of interactions according to its execution complexity:

1. **Simple interaction:** where agent does not need to modify another agent. This interaction allows actions to happen in the same time with ability to overlap, for example two agents said '*do something!*' to third agent and the both jobs can be done in the same time.

2. **Complex interaction:** where agent needs to modify a state of another agent. This category of interaction can be in some kind of simulations with physical representation. In this type, there are no possibilities to allow any kind of conflicting actions. For example, two agents (wolves) can not eat the same agent (sheep) in the same time. For that, this kind of interaction needs an organizer between agents to avoid any conflict situation.

3.3.4 Interactions organizer

There are different mechanisms to organize interactions between agents in centralized-MAS simulation. But not all of them are suitable in distributed-MAS simulation.

1) *First-Asked-First-Act* mechanism

In this mechanism, the first agent, which is asked to interact in each time step, should act its interaction or execute it. That means, agent can change its states directly, then the next agent, ..., until the last one sequentially. In the centralized approach, this is easy to implement and enough. But, this mechanism is not efficient in distributed approach because some machines have to wait the others. That means, high delays during the simulation and high communication costs between machines.

2) *Without-Asked-Group-Act* mechanism

This mechanism means that all agents can interact directly or change their states, then we should try to remove the conflicting actions, modify or abort them. This approach can use all machines capabilities in distributed systems. But it needs high communication cost between machines to abort conflicting interactions.

3) *Group-Asked-Group-Act* mechanism

This mechanism means that all agents can ask for interactions, and then the simulation can organize which agent must be first and which agent must be last, or which one can be re-asked for another interaction. Then, the simulator has a list of possible interactions to apply. This approach is more efficient in distributed systems because it benefits from all machines capabilities with lowest communication costs. But, it adds a new aspect to the simulation which is a judgement mechanism to manage conflicting interactions.

To make an efficient distributed simulation, we have chosen the *Group-Asked-Group-Act* mechanism. Because it benefits from all machines capabilities with lowest communication costs (see table 3.1). Each machine can ask its agents for interactions, then it prepares a list of possible interactions. If in this possible list there are interactions with other agents from other machines, then an agreement protocol should be started between machines

Table 3.1: Interactions organizer analysis between centralized and distributed approaches.

Organizer mechanism	Centralized approach	Distributed approach
1)	Easy to implement Sequential execution	More delays High communication costs
2)	Add abort mechanism	Less delays Parallel commutation Still high communication costs
3)	Adding judgement mechanism	Less delays Parallel commutation Less communication costs

to eliminate the conflicting interactions. Finally, each machine can apply its list of possible interactions.

3.4 Distribution types

To achieve large scale simulations, we need to distribute our simulator on a computer network to reach the maximum computation capability. Different distribution levels for MAS simulation can be tested: hardware level (devices and connections) or software level (MAS concepts).

In hardware level, the distribution depends more on the hardware devices (machines), or on the distribution way of: the data, the computation and the communication between different hardware devices. If we have some machines are faster in computations than others, and some others are better in communications, then the server\clients approach can be more suitable for the distribution. Server can distribute the work between different clients and communicate with them to give and get the simulation progress. Normally, the server needs more capabilities in memory and communication than others.

Alternative example can be more simple, if all machines have the same capabilities, which is the case of most existing network types within academic laboratories. Then, fully connected homogeneity machines can be used easily without server\clients approach, and machines can communicate in a specific way to reach the desired distribution. Definitely in this case, the load between machines must be fair, otherwise we could lose the homogeneity.

However and as we are more in MAS community, we are more interested in the second approach, which is the software approach. In our case, MAS concepts are agents and environment. The distribution can be changed

according to these concepts only. We can distinguish between different distribution types [Rihawi et al., 2014]: *agents distribution* and *environment distribution*. Each way is more suitable for some kinds of applications.

In our case, hardware distribution can be M homogeneity machines with a communication layer, that informs each machine about all changes in the system. It means that 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.

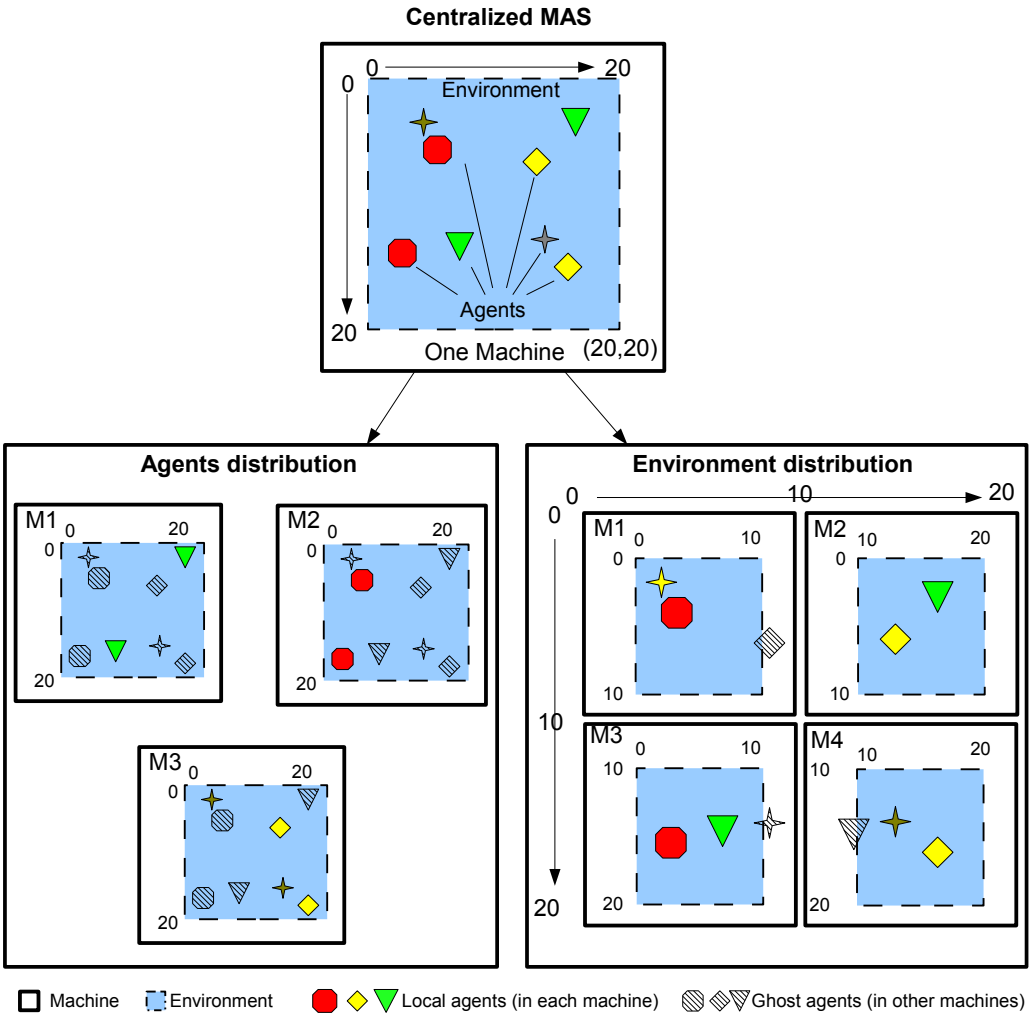

Figure 3.2: Two distribution types: *agents distribution* (lift) and *environment distribution* (right).

3.4.1 Agents distribution

An agent in multi-agent systems (MAS) is an atomic unit, which can not be divided into smaller parts. Even if we can consider it as a collection of different properties and states. Agents are the most important blocks on MAS simulations. Agent and its behaviours can determine the application type.

The distribution of agents can be simply by a division of the agents list between different machines or N agents for each machine.

Each machine handles N agents to execute their interactions and communicates with other machines to achieve agents' goals. The challenge with this approach is to inform all machines about all changes and actions between agents in the system. All that must be done with lowest communication costs. This type of distribution should be more suitable for cognitive agents, which require a high computation than simpler ones, which can have a lot of communications instead.

Depending on the application itself, the division of the agents list between different machines can be done by different ways:

1. Types division: all agents of the same type can be grouped together in the same machine (figure 3.3). For example in prey-predator model, all preys are in one machine and all predators on another machine.

2. Communications division: list of agents, which communicate more together, can be in the same machine (figure 3.3). For example in flocking model, all birds which near from each other can be in the same machines, so the communication can be minimized between machines.

3. Computations division: Cognitive agents, which have heavy computations, can be divided between machines to distribute the important computation.

4. Services division: list of agents which have the same services or the same computational needs. For example, if there are agents have the same and intensive mathematical computations in granular gas model, then these agents can be in the same machine.

5. Random division: If all agents are in similar computation and communication needs then the division simply can be randomly N agents for each machine.

Whatever the way that we distribute agents with, each machine has a list of agents and these agents should be able to interact with other agents in

local machine and with other agents on other machines. We call this type of interactions between agents from different machines as *External Interactions*

Definition 1 ***External Interaction****: is a complex interaction between two agents (or more) that exist on two different machines (or more) [Rihawi et al., 2013c].*

Then, we have to define a mechanism to handle these type of *External Interaction* (EI) that can be done with two main steps:

1. Information about agents between different machines must be updated at each time step, thus agents can see each other and interact together, and that can be done in different ways:

   (a) Send an information message about agents on other machines each time an agent needs it, which increases communication costs between machines.

   (b) Send all information about agents at each time step through one message, and then all agents, that exist on one machine, can have a vision of agents from other machines.

2. An agreement protocol must be able to apply these EIs. If and only if, these EIs are not conflicting with other interactions from other machines.

For the first step, each machine has its agents (local agents) and should be able to have information about other agents from other machines which can be called *Ghost Agents* (see figure 3.3).

Definition 2 ***Ghost Agent****: it is a copy of an agent's state (not a real agent), that reflects a real agent on another machine. This copy must be updated at each time step to allow all local agents to see and interact with that agent.*

3.4.2 Environment distribution

In some applications, the focus can be more on the environment itself instead of agents. Particularly for simulations with physical constraints, whose agents interact locally and probably are distributed over all environment. In this case, the environment can be divided between different machines, each machine holds a small part of the environment with agents that exist on this

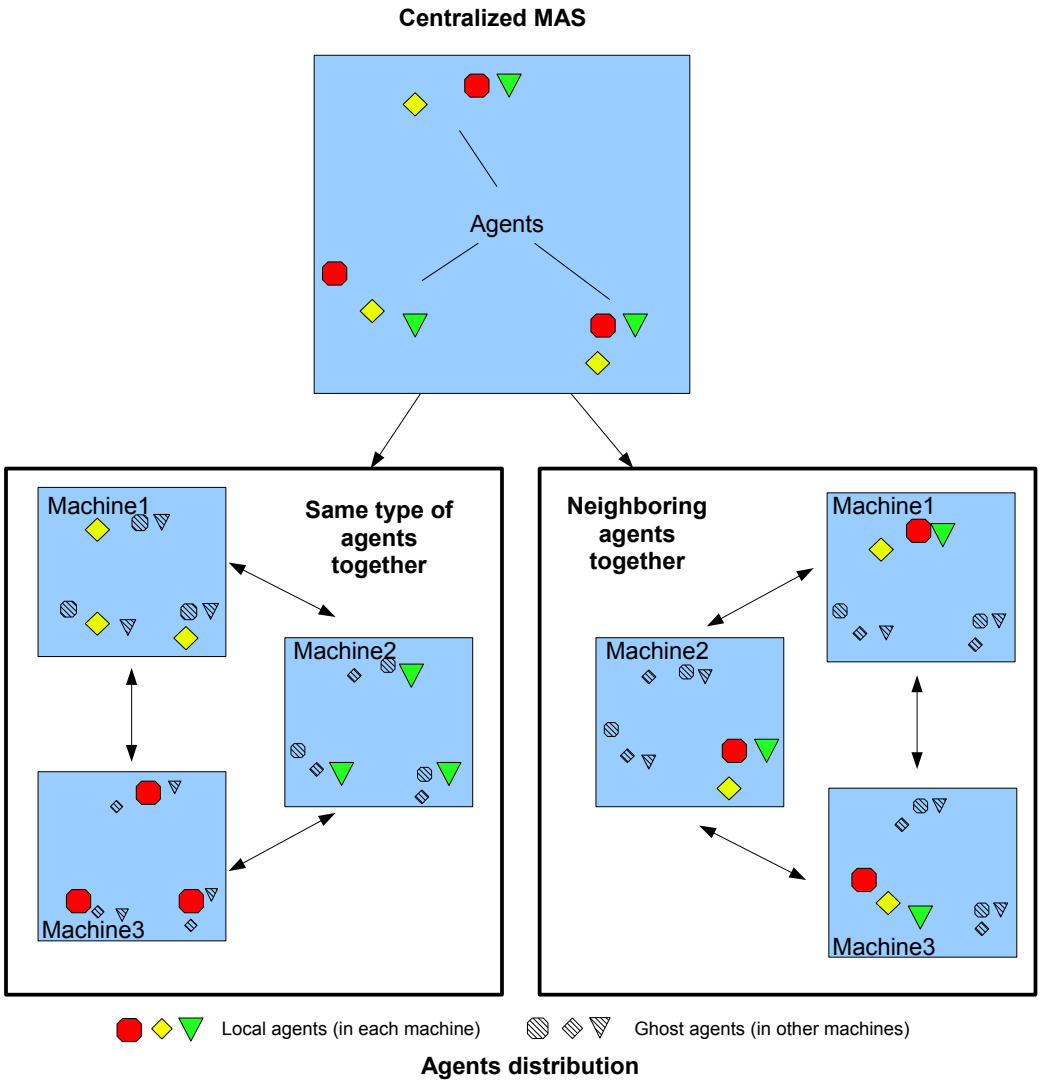

Figure 3.3: Agents distribution can be done with different ways: same type together (left) or neighbouring agents together (right).

part. Thus, each machine computes agents perceptions and all interactions that happens within its environment slice. In other words, each machine manages a part of the environment (see figure 3.2).

In that case, the challenge is how can we manage the transferred agents between machines or the agent placement problem [Miyata and Ishida, 2008]? And how can we deal with the interactions between agents in the edge-zones space or the overlapping problem?

Moreover, one of the most important issue is how we can inform other

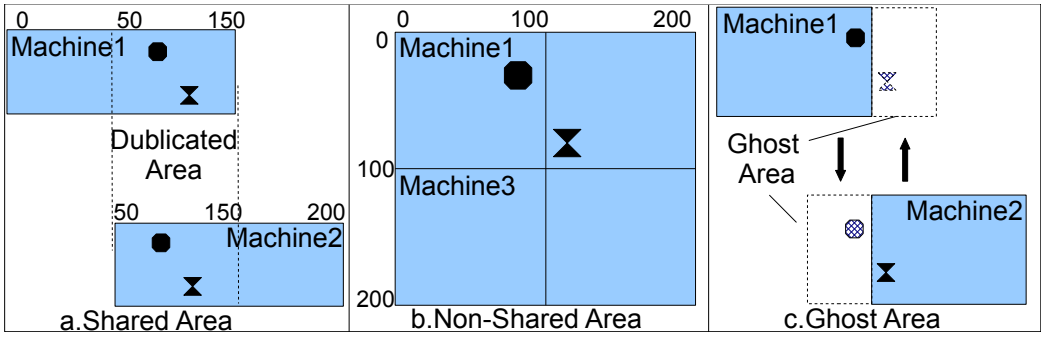

Figure 3.4: Three different ways to exchange information between machines: a.*shared area*, b.*non-shared area* and c.*ghost area*.

machines about all changes in the system. Information exchange between environments parts can be done in three different ways:

1. ***Shared area:*** is a common area between neighbours or between each two parts of the environment. These areas exist in two machines (or more), and must be synchronized at each time that some changes happen on one of the neighbours (as we can see in figure 3.4.a). This kind of area is very costly in communications, because it must be synchronized at each time step. In fact, this area is not useful when we want to collect information about our agents or the state of all environment's parts for visualization, because we must be sure that these shared areas have already been synchronized on all machines.

2. ***Non-shared area:*** in this case, the environment is separated into parts without a common area or shared area. Each agent exists in one part only, so that means in one machine only (one part on each machine). However, each time we have an agent close to the edge zone between neighbouring parts and whose vision span on others, we need to inform this agent about any changes that happen on these parts. That can be done by sending an information message to each agent close to the edge zone, but it would imply a high communication penalty.

   As an example, if we have an environment divided into two parts on two different machines as we can see in figure 3.4.b. In each machine, we have some agents that need information, because their visions cross the edge-zone to the other machine. For example, an agent $A1$ exists in *machine1* and needs information about another agent $A2$, which exists on *machine2* (see figure 3.4.b). Agent $A1$ wants to ask the *machine2* and receive an information message from it. Then and for the same

reason, other agent $A3$ from the same host *machine1* wants the same information about the same agent $A2$ from the *machine2* too. Thus, it asks *machine2* about the same information and receives a second message with the same information. In this case, the same information should be sent two times for only two agents. It is clear that, the non-shared area can explode the communication, especially if we have a large number of agents in the edge-zone (figure 3.4.b).

3. ***Ghost-area:*** this area is like the *non-shared area*, which consists of separated environment parts on different machines. But, a ghost area is added to each part, which represents a part of neighbours states as a ghost area (not a real area).

   Definition 3 ***Ghost Area****: it is a copy area (not a real area) that reflects a real area in the system, which exists on another machine. In other words, it is a copy of another environment part that exists on another machine and needs to be updated at each time step to allow local agents form seeing and interacting with other agents that belong to that area.*

   This area should be updated and informed with all changes by one message at each time step to allow local agents to interact. It is not like non-shared area which sends the same information in separated messages. Also, it is better than shared area, which duplicates the agents between machines. As we can see in figure 3.4.c, each machine has to receive *Ghost-areas* around it from other neighbours at the beginning of each time step, and also has to send *ghost-areas* for others too. *Ghost-area* approach is very near from 'ghost-data' which used with some researchers to visualize parallel simulation [Isenburg et al., 2010].

Independently of the chosen mechanism to exchange information, environment distribution is more suitable for situated agents whose positions have a normal distribution on the environment. In other words, each agent has an equal probability to exist on all special areas on the environment. That means, agents are diffused on the whole environment, and they do not aggregate on one place or they are not moving with large groups from one area to another. Because, if we have one large groups of agents in the same area or in the same part of the environment, that means huge computations should be done on one machine only, and then this distribution approach is not efficient.

3.5 Time management

To achieve large scale multi-agent simulations, we choose to distribute computations over a computer network. In distributed system, one of the most important problem is time management and synchronization. In addition to normal distributed system, in large scale multi-agent simulations the importance is to gain more performance by relaxing the synchronization with keeping the macroscopic behaviour.

This thesis is a first study of synchronization costs in performances and the impact of synchronization policies on the preservation of emergent macroscopic properties in large scale multi-agent simulations. To understand the important of synchronization, we detail the notion of time in a centralized and decentralized system and introduce the three main synchronization policies that we have studied. In later chapters, we can see in details experimentations made on some applications to benchmark the impact of synchronization policies about simulation outcomes.

As we mention before, the word *time* can be 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 [Gold, 2003].

In a distributed simulation context, several notions of time are involved: *user time*, which is the real time, and *simulated time* or a *time step* (TS), 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 by Lamport [Lamport, 1978] through a logical clock that induce a partial ordering of events, and has been refined as *Logical Virtual Time* (LVT) by Jefferson [Jefferson, 1985].

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 tick or time step (TS). Because several actions are gathered within a TS, one can encounter conflicts between two or more actions, thus the simulator has to define some rules for such situations. An example, in the prey-predator model, if two predators try to attack the same prey in the same TS, 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 allows them to interact in a given period of time. In a distributed simulation, there is one logical clock per

machine and the time needed to handle a TS is not the same between all machines. This is because, they can have different loads that comes from the differences of agents which they hold (see figure 3.5).

In order to guaranty causality on all machines, we have to synchronize local time step within all machines. Most of researchers' works are concerned with two main approaches as we mention before: *conservative* (or *synchronous*) and *optimistic* (or *asynchronous*) synchronization [Logan and Theodoropoulos, 2001], [Fujimoto, 2000], [Gupta et al., 2007]. Farther in this thesis, we define other types of synchronizations to handle time step simulation.

The question that we are interested in this thesis 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 number of agents, 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.

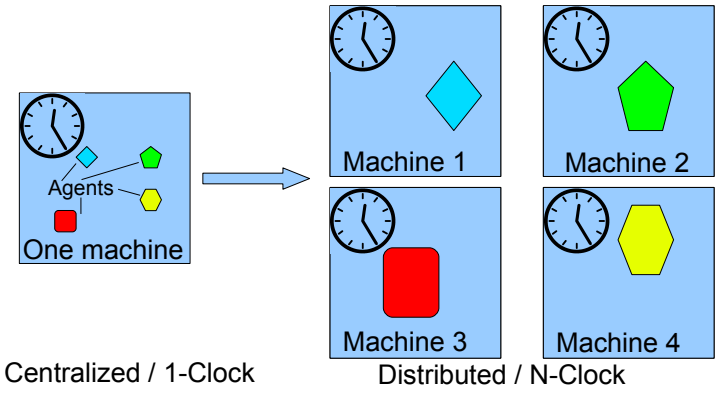

Figure 3.5: Multiple time steps in distributed approach.

3.6 Synchronization policies

As we mention before, time management between machines or synchronization is one of the most important thing in distributed systems. We divide synchronization policies into three main approaches for distributed multi-agent

simulations [Rihawi et al., 2013c,b]: *strong synchronization* (SS) policy, *time window synchronization* (TW) policy and *no synchronization* (NS) policy (figure 3.6).

3.6.1 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 should 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 high communication delays.

3.6.2 Time window synchronization

The second policy allows machines to progress at different pace, thus it is more flexible. It is similar as *optimistic* (or *asynchronous*) synchronization, which allows machines to advance at different pace in simulated time. But, it does not have a rollback mechanism [Gupta et al., 2007], which enforces the simulator to roll back into previous events to reconstruct a previous state of the simulation if there are errors in the system. Because, we don't want to loss the gain of flexibility by a probability of roll back too often during the simulation.

For that reason, we propose *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 without affecting macroscopic behaviours outcome in some cases.

Of course, with this approach, we can have situations where agents in different time steps can interact or some agents fail to interact. But in large scale simulations, that can be ignored. We believe 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 different applications in experimental chapters later.

3.6.3 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 exploits the available speed of all machines. Our experiment in large scale multi-agent simulations shows that machines, with homogeneous load computations in this policy, can have some and small differences in time steps. But, it is not for a long time and it can be swapped between machines. So, no machine has a greedy superiority all the time, and some applications can run for a long time. However, we will see later in some experiments that this policy is not suitable for all applications, but it can keep some applications from its outcome.

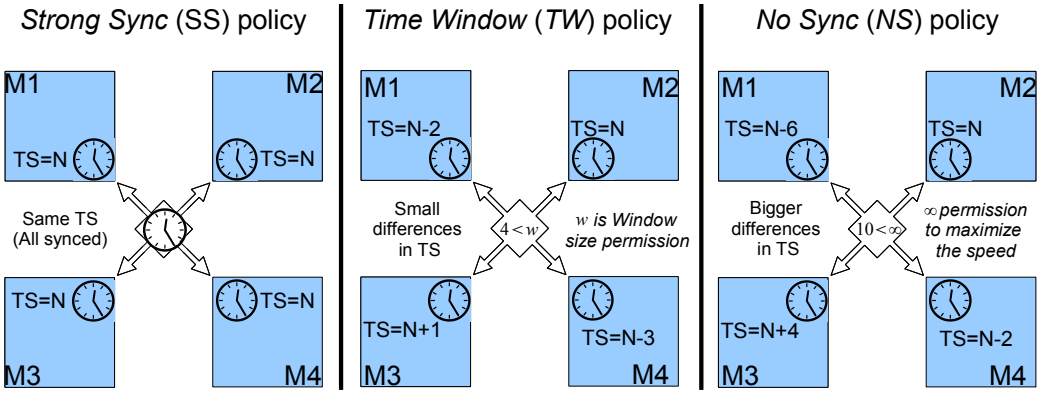

Figure 3.6: A distributed MAS simulation on 4 machines with three different synchronization policies: *strong synchronization*, *time window synchronization* and *no synchronization* policies.

To conclude, three synchronization policies (see figure 3.6) have been chosen to study large scale distributed multi-agent simulations. The problematic studied in this thesis is: can we determine whether we are able to keep macroscopic behaviours emergence when relaxing synchronization constraints or not? 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.

3.6.4 The influence of synchronization policies

In a centralized approach, there is no synchronization policies, as all agents are in one machine only and can easily communicate with other agents directly. Whereas in distributed system, agents from different machines want

to interact with others to achieve their own goals. These interactions should predetermine a mechanism to organize the time between different agents that interact from different machines, as agents are distributed between machines. Three policies can have different impact on the simulation (table 3.2).

Table 3.2: Analysis of synchronization policies.

	SS policy	TW policy	NS policy
Execution time	Slower	Faster	Fastest
Communication costs	High	Less	Less
External interactions	High	Less	Less
Microscopic behaviour	Reliable	Unreliable	Unreliable
Macroscopic behaviour	Reliable	Unreliable	Unreliable

Strong synchronization

In *Strong synchronization* (SS) policy all machines should progress together in the simulation. This policy is similar to centralized approach in outcomes, because all machines have to progress together, so all agents have the same chance to interact. But, the disadvantage of this policy is that all machines have to communicate a lot to progress one time step only, even if there are some time steps without external interactions between agents from different machines. This could be the reason for higher and useless communication costs.

No synchronization

In *no synchronization* (NS) policy all machines are free from all constraints to progress alone in the simulation. The advantage of this policy is that we can gain the maximum speed of all machines, because there are no useless communications between machines.

However, in this policy the disadvantage is that not all machines are able to get the same outcomes as in centralized approaches, because not all machines can progress together at the same time. Then, some agents have not the same chance to interact as others. That can be the reason to destroy the needed outcomes of some applications or the macroscopic behaviour of the simulation and make it useless. Generally, in large scale simulations with billions of agents we can ignore some incoherent or wrong interactions from a small number of agents.

Time window synchronization

To balance between both previous policies, *Time window synchronization* (TW) policy can be the solution. In this policy, machines can progress in the simulation with small and limited number of time steps. That can avoid some of useless communications between machines, and in the same time it can keep the outcomes of the simulation. That means for some applications, we can reduce the communications and increase the performance in the same time.

3.7 Summary

To execute large scale MAS simulations, resources in memory and computing costs can exceed the capacity of a single computer. Thus, distributing the computations can be the best solution for large scale simulations, which allow to divide computations between different machines. Moreover, in large scale situations we need to distribute the computations in a specific way to gain more performance. To do that, it is necessary to take into account specific concepts, which are agents and environment, more than the classical ones (computation and data storage) in normal distributed systems. Our work presents several criteria that should be considered to ensure performance gains when distributing such simulations over a computer network.

In this chapter, we first describe our view of MAS concepts and the importance of different distribution approaches. We detail our understanding of agents and environment, and we distinguish between two categories of interactions. We compare different interactions mechanisms between centralized and distributed cases. Then, we detail different distributions approaches for MAS concepts, which depends on agents and environment. That can adapt initially the loads between machines for some applications without any load-balancing mechanism. We propose two distribution types: *agents distribution* and *environment distribution*, which can give some applications better performance with one approach than the other.

Time management problems and synchronizations are deeply studied to distribute MAS simulations efficiently and gain more performance. Three synchronization policies has been proposed: *Strong synchronization* (SS) policy, *time window synchronization* (TW) policy and *No synchronization* (NS) policy. TW and NS are the flexible synchronization policies which allow machines from avoiding communications delays. The question was 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. In the next chapter, we detail our platform and the main operations in it for two distribution approaches and three synchronization policies.

Chapter 4

Distributed-MAS: platform description

4.1 Introduction

In the previous chapter, we have discussed different distribution types, which are: *agents distribution* and *environment distribution*. In addition, three synchronization policies have been proposed: *Strong synchronization* policy, *time window synchronization* policy and *No synchronization* policy.

To study the impact of these distributions and synchronizations policies on large scale MAS applications, we had to create our platform to make all our experiments.

In this chapter, we describe our platform for large scale distributed MAS simulations. We present the different components of the platform by detailing the machines units that manage the main operations of the simulator in the two cases: *agents distribution* and *environment distribution*. Then, the main platform layers are discussed deeply and different simulation states are introduced. After that, we study communication protocols and synchronization algorithms for three main synchronization policies, which are: *strong synchronization*, *time window synchronization* and *no synchronization* policies. We explain the time step scenario between two machines to illustrate the differences between these policies. Finally, we describe our view of visualization in large scale simulations.

4.2 Machine units

Figures 4.1 and 4.2 show machine units for the two different distribution types that we have chosen: the *environment distribution* with ghost-area

approach and the *agents distribution*. We build these two distribution types in our framework to distribute MAS simulation, and we give the user the capability to choose which type of distribution he wants.

In our work, we choose a simple communication layer between machines, where each machine can connect directly to all other machines. Each machine makes a part of calculation during each time-step and communicates with others to build a complete global simulation view (figure 4.1).

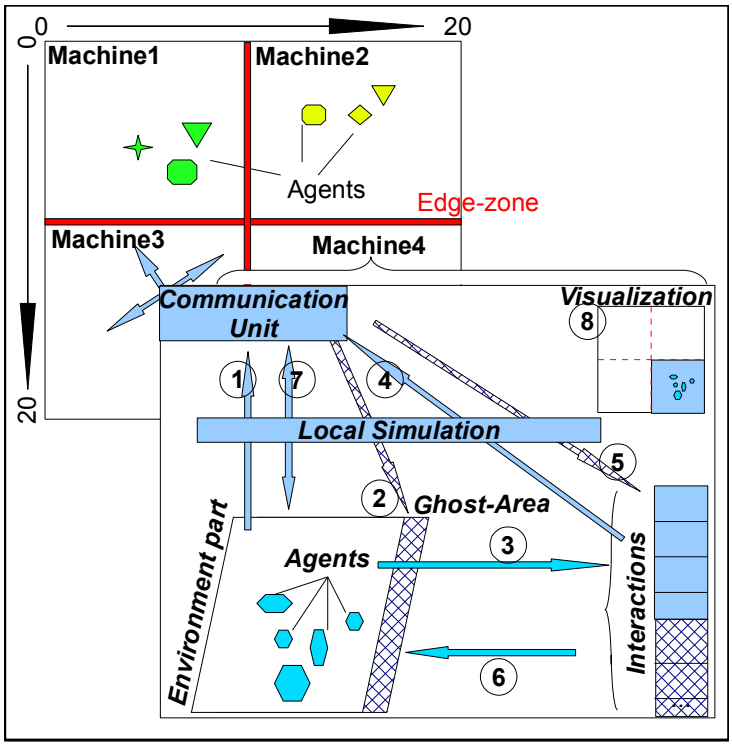

Figure 4.1: Machine units in *environment distribution* approach: 1-2)send-receive ghost area 3)get interactions 4-5)send-receive external interactions 6)apply interactions 7)transfer agents 8)visualization.

4.2.1 Environment distribution case

Environment distribution means that the environment is divided into different parts on different machines. Each machine has a part of the environment with its agents. Figure 4.1 shows all machine's units: communication unit, local simulation unit and visualization unit.

Communication unit manages all communications with other machines and all messages should be passed through it. Visualization unit has the re-

sponsibility to read and visualize local environment part with its local agents. Local simulation unit manages all local operations inside the machine.

At the beginning of the simulation, local simulation unit should send information about ghost areas to other machines through the communication unit, and it should receive ghost areas from others too.

After that, local simulation unit should ask each local agent about its preferred interaction and create two lists of interactions: one for internal interactions and the other for external interactions. The internal interaction list is the list of interactions between local agents in this machine. Whereas, external interactions list is the interactions between one of the local agent in this machine with an agent from ghost areas which exist on other machines.

Then, the local simulation unit can send to and receive from other machines the external interactions through the communication unit. Once we have a list of possible interactions, the local simulation unit can apply this list.

Then, local simulation unit should transfer any agent that move out of the local environment part to other parts or to other machines and also should receive any agent that wants to enter this local part from other machines too. Finally, visualization unit can visualize all local agents in local environment part.

4.2.2 Agents distribution case

Agents distribution means that agents are divided into different parts on different machines with the same environment. However, each machine has a list of agents that can interact locally with other local agents or can interact with other agents from other machines through *ghost agents*.

Figure 4.2 shows all machine's units of agents distribution. It is similar to environment distribution case: communication unit, local simulation unit and visualization unit. But, it has small differences in data structures, where we can replace *ghost areas* with *ghost agents*.

As in environment distribution case, the communication unit manages all communications with other machines and all messages should be passed through it. Visualization unit has the responsibility to read and visualize local agents. Other works inside the local machine can be managed by the local simulation unit.

At the beginning of the simulation, local simulation unit should send information about local agents to the other machines through the communication unit, and it should also receive information about other agents from the other machines as *ghost agents*.

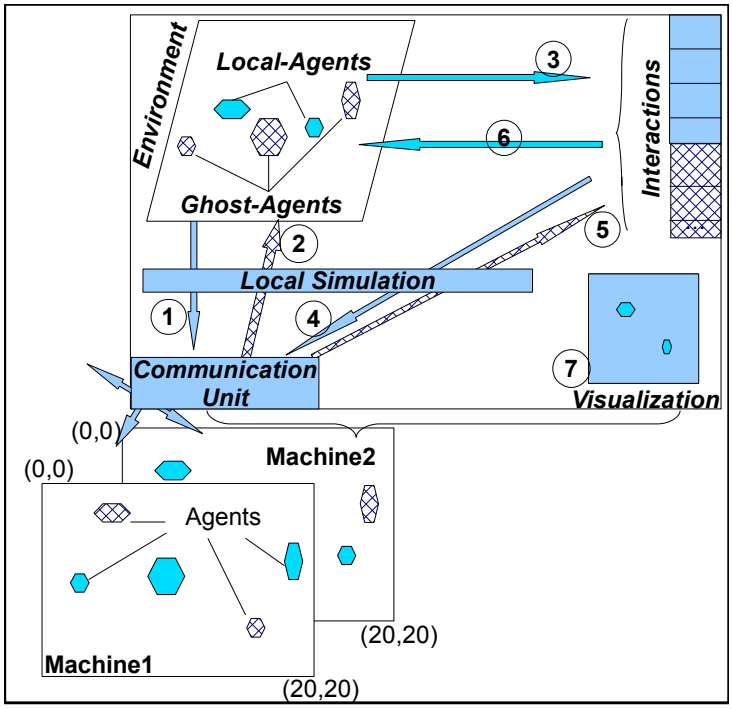

Figure 4.2: Machine units in *agents distribution* approach: 1-2)send-receive ghost agent 3)get interactions 4-5)send-receive external interactions 6)apply interactions 7)visualization.

After that, the local simulation unit should ask each local agent about its preferred interaction and then create two lists of interactions: one for internal interactions and the other for external interactions. The internal interaction list is the list of interactions between local agents in this machine. Whereas, external interactions list is the interactions between one of the local agent in this machine with ghost agents or with other agents from other machines.

Then, the local simulation unit can send to and receive from other machines the external interactions through the communication unit. Once we have a list of possible interactions, the local simulation unit can apply them. Finally, the visualization unit can visualize all local agents as in environment distribution case.

To conclude, the only difference between agents distribution and environment distribution is the transferred agents. In the environment distribution state, there are transferred agents, because agents can swap between environment parts or can move from one position on one environment part to another position on another part. That means, agents can move between

machines and should be transferred between them.

Whereas in agents distribution state, all machines have the same environment with a list of agents, that can move in the same machine to all positions on the environment. For that, there are no transferred agents between machines in case of agents distribution, except if there is a load balancing mechanism that needs to transfer agents between machines to balance the load at some time steps.

4.3 Platform layers

Our platform can be scaled to N machines. Each machine can communicate with other machines directly. Every machine has an ID (IP address or name), which is unique and different from others' IDs. Thus, each message can be sent to a destination ID or a destination machine.

Each machine holds a simulation part that consists of three main layers: *communication layer*, *distributed simulator layer* and *application layer* (figure 4.3).

4.3.1 Communication layer

The first layer is the *communication layer*, which has the responsibility to build and establish connections between machines. At the beginning of the simulation, this layer should send a connection request to each machine through its ID. Once all connections have been established, the simulation can be started.

This layer should be able to send and receive messages between machines. This communication is done in an asynchronous way and machines should not wait to send or receive messages. All messages are registered in two main lists: *Inbox* and *Outbox* lists (see figure 4.3). *Inbox* list for incoming messages and *Outbox* list for messages that are ready to be sent. Each message has the destination ID of the destination machine with a stamp of the current simulation time step.

4.3.2 Distributed simulator layer

The second layer is *distributed simulator layer*, which distributes MAS simulation and manages all distributed parts. It provides configuration to decentralize the environment and it should associate agents to the environment or a slice of it. It is located between two layers: the *communication layer* and

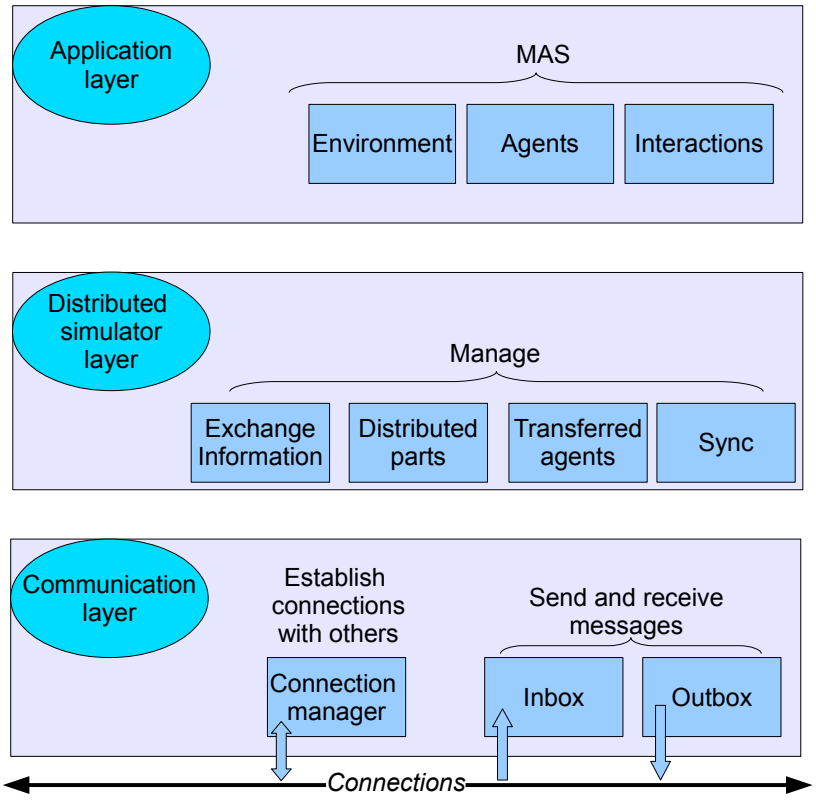

Figure 4.3: Three framework layers in our platform: application layer, distributed simulator layer and communication layer. Communication layer deals with network connections and messages between machines. distributed simulator layer deals with all distributed operations. Application layer deals with MAS application from agents to environment.

the *application layer*. All messages of this layer should be passed through the *communication layer* to be sent to their destinations on other machines.

At the beginning of the simulation, this layer is responsible of all distributed operations. It should create all distributed parts and it should create maps for MAS concepts (agents and environment) of the neighbouring machines. For example: the map for positions ranges of other environment parts on other machines, the map for agents locations on other machines, the map for exchanging information with neighbouring machines... etc.

This layer should also transfer agents between machines if it is necessary or if there are moving agents between different positions ranges on different machines. This layer should update information about other environment parts or other agents on other machines each time step.

Moreover, this layer should manage external interactions with others machines, which are interactions between local agents in the local *application layer* with other agents on other application layers or on other machines. In addition to that, this layer sends synchronization messages to other machines at the end of each time step to notify others that this machine is ready for next time step.

4.3.3 Application layer

The third layer is the MAS level layer with agents, environment and application domain definitions. In this layer, the user can define the notion of agent in different levels, for example: physical level (agent's body handling), social level (agent communication), mental level (knowledge and action selection mechanism).

These levels of interactions can be determined by user needs, figures 4.4 and 4.5 show examples of defining agents and environment in the application layer. The user can define different types of agents according to his application, and must inherit them from the agent abstract class. The user must define all possible interactions in his applications and should define a judgement mechanism to prevent agents from acting a conflicting interaction. If there are any external interactions, this layer should pass these external interactions to the *distributed simulator layer* to manage their acceptance.

4.4 Platform configuration

Before the simulator can be started, the user should configure his application to get a better performance. That can be done through a configuration file, which allows users to determine the properties of the simulation, if it runs with *environment distribution* or with *agents distribution*.

In case of environment distribution (see figure 4.6), the user can divide the environment into different slices on different machines through the configuration file. The configuration file consists of different lines: the header line as first line and environment parts lines. In the header line (first line), the user define the type of his simulation and all properties like: ED or environment distribution case, with or without GUI, the synchronization policy, with or without log-file, etc.

Each following line should define one environment part with its initial agents and with its machine. That means, the configuration file has lines of the machine ID, the machine name or IP, the environment slice ranges and the initial agents which is contained within that slice (see figure 4.6). For

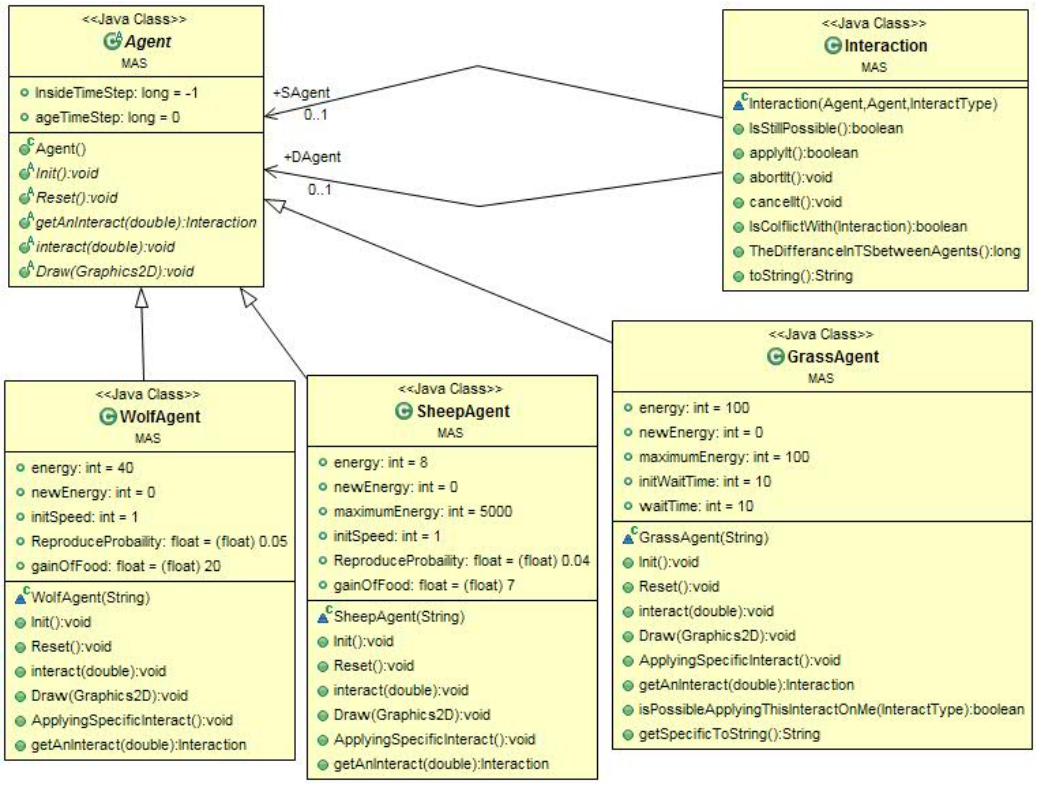

Figure 4.4: Agents UML class diagram for prey-predator model. Each agent type has to define a list of abstract functions to allow simulation from lunching the interaction process. The user can create many types of agents according to his application, and all these types must be inherited from the agent abstract class.

example: ID number (0), machine's name or IP (M1), positions range of this part ($x0$ $y0$ $x1$ $y1$ = 0 0 10 10), then number with an agent type (50 wolf 2000 sheep 3000 grass, etc.).

In case of agents distribution (see figure 4.6), the user can divide the initial agents into different lists on different machines through the configuration file. Again, the configuration file consists of different lines: the header line and machines lines. In first line or the header line, the user can define the type of his simulation and all properties like: AD or agents distribution case, with or without GUI, the synchronization policy, with or without log-file, etc.

Then, each following line of the configuration file can define one machine with its initial agents and with the same environment positions range for each machine. That means, the configuration file has lines of the machine ID, the machine name or IP, the main environment positions range that agents can

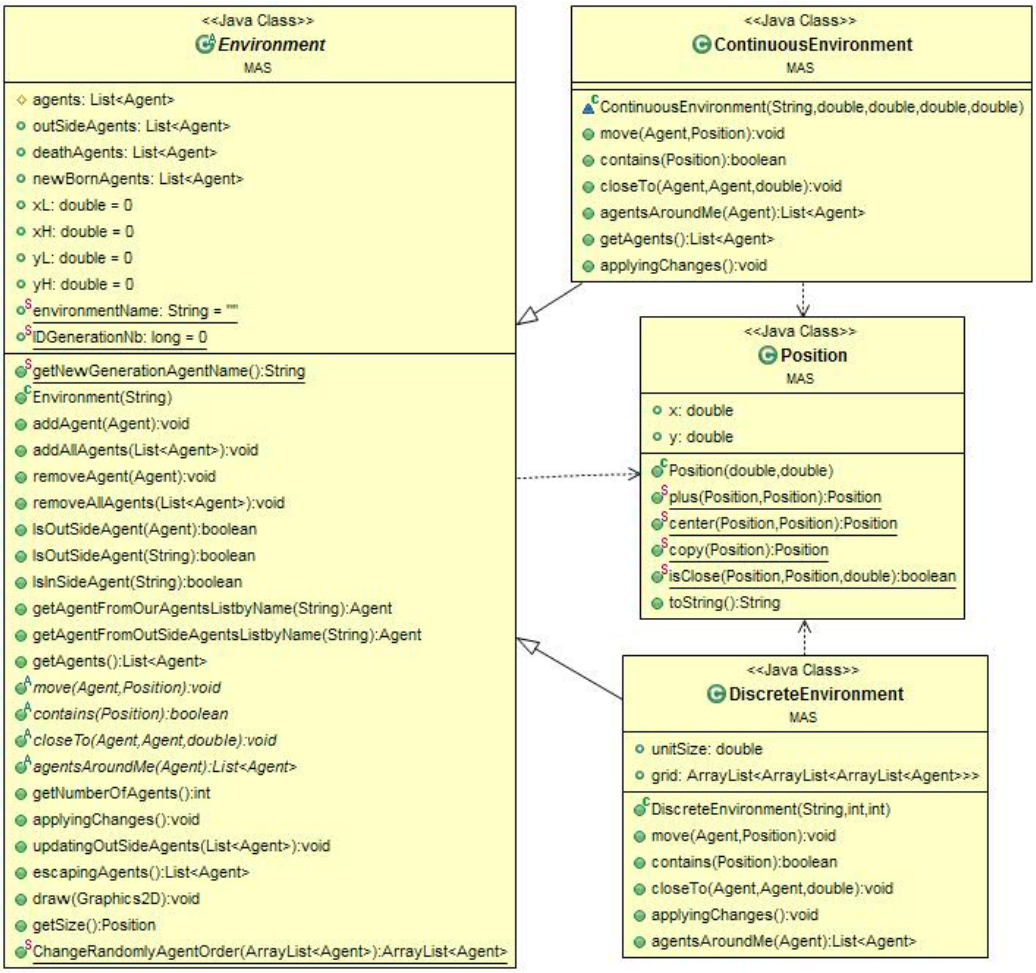

Figure 4.5: Environment UML class diagram. The user can choose the environment type that is more suitable for his application.

move through it and the initial agents which is contained within each machine (see figure 4.6). For example: ID number (0), machine's name or Ip (M1), positions range of the environment at all ($x0$ $y0$ $x1$ $y1$ = 0 0 20 20), then number with an agent type (50 wolf 2000 sheep 3000 grass, etc.). If the user need to configure large scale simulation (more than 100 machines), a small script can be used to generate 100 lines with all necessary information for each machine (number of agents, agents types, etc.).

Environment distribution file:

```
line1:ED WITHGUI SYNC WITHLOGFILE ...
line2:ID=0 m1 0 0 10 10 #NbAgent1=50 #Type1=wolf #NbAgent2
#Type2 ...
line3:ID=1 m2 10 0 20 10 50 wolf 2000 sheep 3000 grass
line4:ID=2 m3 0 10 10 20 50 wolf 2000 sheep 3000 grass
line5:ID=3 m4 10 10 20 20 50 wolf 2000 sheep 3000 grass
```

Agent distribution file:

```
line1:AD WITHOUTGUI NOSYNC WITHOUTLOGFILE ...
line2:ID=0 m1 0 0 20 20 #NbAgent1=50 #Type1=wolf #NbAgent2
#Type2 ...
line3:ID=1 m2 0 0 20 20 150 wolf 2000 sheep
line4:ID=2 m3 0 0 20 20 6000 sheep
line5:ID=3 m4 0 0 20 20 4000 grass
```

Figure 4.6: Two examples of configuration files for two different distribution types. User can configure in the first line all platform parameters, and in next lines he should define ID number, a name (or IP), environment ranges and agents that should be created on each machine (in each line).

4.5 Simulation states

Our simulation is divided into different parts, each part is handled by one machine. Each simulation part manages one environment part with its agents in case of *environment distribution* (figure 4.7) and manage a list of agents in case of *agents distribution*.

Each simulation has to follow two main states: *initial state* and *running state*. In the initial state, the simulation should use the configuration file to prepare all platform layers from the communication layer, to the distributed simulator layer until the application layer. The running state should run in one of distribution types: *environment distribution* type or *agents distribution* type. It can run also with one of three available synchronization policies: *strong*, *time window* and *no synchronization*.

4.5.1 Initializing state

The first step, which has to be done by the user, is to define the configuration file of the simulation. If the user chooses environment distribution for his application, the environment must be divided into different parts on different machines. Whereas in case of agents distribution, the user can divide only

the list of agents between different machines (as in figure 4.6).

Each machine has a part of the simulation and must be able to communicate with other machines: firstly to exchange information between machines, secondly to transfer moving agents to other parts (or other machines) and finally to synchronize with other machines.

Our framework depends on the configuration file to initialize the simulation. Each machine should get other machines' IDs and names (or IP addresses) from the configuration file to establish the communication layer. Then, each machine can collect the needed information from the configuration file about the local environment and local agents to prepare the suitable data structures.

In case of environment distribution, each machine reads the positions ranges on the environment which represent the part of environment that each machine must manage and the initial agents that exist on that part. In case of agents distribution, each machine can prepare only the initial agents list, because the positions range on the environment must be the same on all machines.

After each machine has initialized the communication layer and the data structures for the simulation, it can send a zero synchronization message to other machines to announce that it is ready for the the running state. With that synchronization, all machines should be able to begin the running state together.

4.5.2 Running state

After the initialization of all parts in all machines, each machine has to begin the running loop or what can be called a time step of the simulation. At the beginning of the time step, each machine can collect information about neighbouring machines or information about other agents on neighbouring machines, which are *ghost-area* information in case of environment distribution and *ghost-agents* information in case of agents distribution.

Then, after the simulator has sent and received the necessary information, the simulator can start by asking each agent about its next interaction. Each machine has to exchange external interactions with other machines to get a list of possible interactions. Once we have a list of possible interactions, each machine can apply this list to its local agents.

After applying the possible interactions and in case of environment distribution, each machine can transfer any agent that move out from the local environment part to another part or to another machine. It means that, this agent want to move from local machine to a neighbouring machine. In this case, the machine should transfer this agent to its new position on the other

machine and also it should be able to receive any agent that wants to enter the local part from other machines too. Whereas, in case of agents distribution, there are no transferring between machines, as all machines run on the same positions range.

The final step in the time step is the synchronization. Each machine has to send a synchronization message to other machines to announce that it is ready for next time step. The synchronization can be done with three different policies: *strong*, *time window* and *no synchronization*. After the synchronization messages all machines should be able to begin the next time step.

4.6 Communication protocols

Our platform has been developed with three synchronization policies: *strong synchronization*, *time window synchronization* and *no synchronization* policies. The simulator is distributed on a set of machines. Each machine should deal with a part of the simulation. Each machine is connected to 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.

Figure 4.7 shows 4 machines that execute a distributed agent-based simulation with the environment distribution approach. Each machine consists of: local simulation, communication unit and part of the environment with its agents. Local simulation is a top-manager layer in each machine, which manages all tasks: from interactions between agents, receiving information from neighbouring machines and local visualization. Communication unit manages the connections between machines to exchange messages and informs local simulation about other machines' time steps (TSs).

In case of strong synchronization, each machine follows 11 main stages in each time step:

1. First, it should send the necessary information to all neighbouring machines about the environment state near them. In this example (environment distribution), it can send ghost areas to other machines.

2. Then, it waits for a new information from neighbours to inform local agents about other parts of environments (or ghost area).

3. After each machine has received its necessary information, it can ask local agents about their next desired interactions.

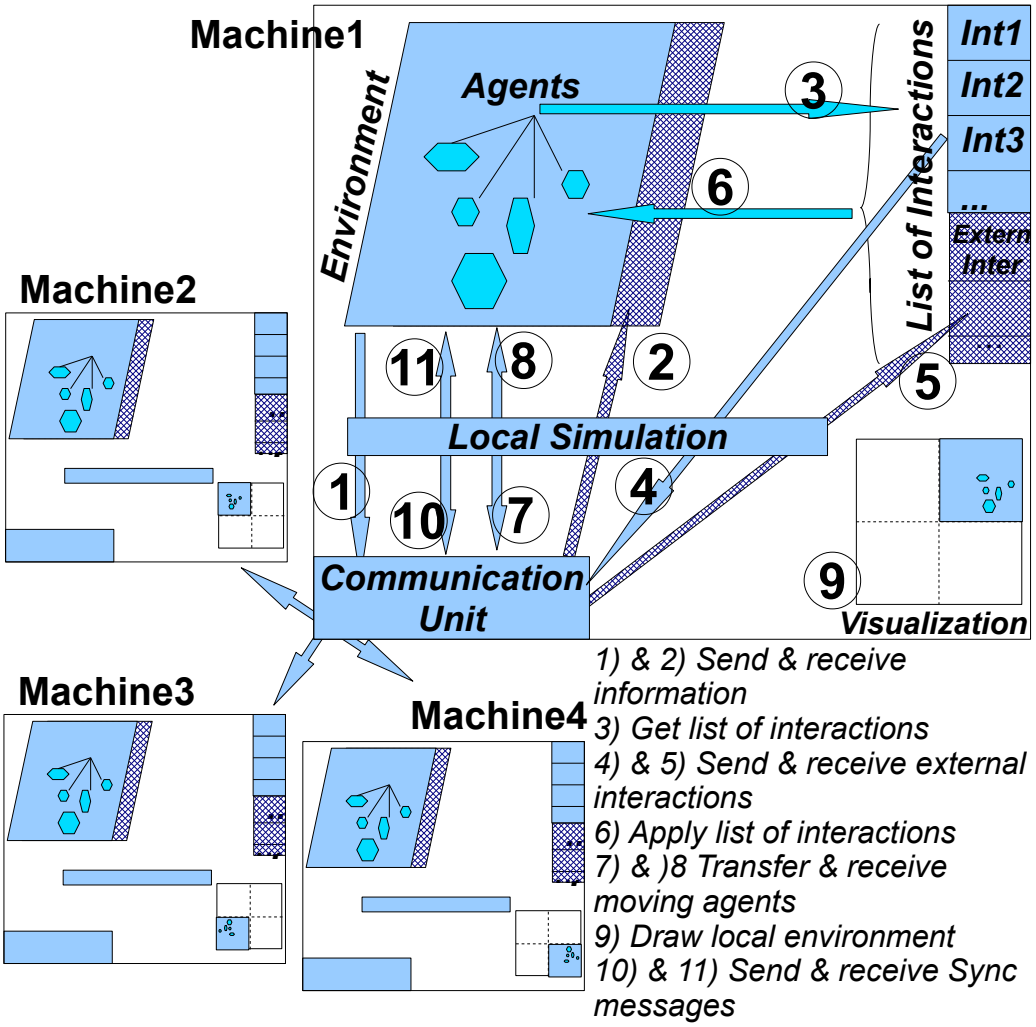

Figure 4.7: Description of a distributed agent-based simulation on 4 machines with *environment distribution* approach, each machine consists of: local simulation, communication unit and part of the environment with its agents. In strong synchronization: all steps (from 1 to 11) should be followed, whereas in flexible synchronization: receiving steps (2, 5, 8 and 11) could be skipped to avoid communication delays.

4. Then, it can send the external interactions, which are interactions between agents from different machines, to its destination machine for getting the acceptance.

5. Also, each machine receives external interactions from other machines and sends acceptances of possible ones.

6. After that, each machine can apply the list of possible interactions on local agents.

7. Then and only in case of environment distribution, each machine should search and then transfer the local agent that wants to go out of the local environment part to another environment part or to another machine. That, if this agent has a new position out of the local environment part.

8. Each machine can receive new transferred agents from other machines.

9. Then, each machine draws its local environment with its agents.

10. After that, each machine sends synchronization messages to all other machines to inform that it is ready for the next time step.

11. Finally, each machine should wait to receive also the synchronization messages from all other machines to be sure that all these machines are ready for the next time step.

Whereas in cases of the other two policies, the 11 steps can be modified to avoid any wait in the loop. To understand more the time step in one machine, The figure 4.8 illustrates the process through UML sequence diagram.

4.7 Synchronization algorithms

To illustrate the main execution loop for each synchronization policy, we have sketched their algorithms: algorithm 1 for *strong synchronization* and algorithm 2 for two flexible synchronizations: *time window synchronization* and *no synchronization* policies.

The three synchronization policies have similar communication protocols with small differences. Algorithm 1 shows the states of one machine when it is running in *strong synchronization* (SS) policy. Strong synchronization algorithm has in each communication state a notification, which is a kind of replying or acceptance from other machines. Especially for last state of communication, all machines should be synchronized for next time step, and they are suspended until other machines are ready for it. That mean, no machine can swap to next time step until all other machines are ready to do it.

Algorithm 2 shows the two other mechanisms: *time window* (TW) and *no synchronization* (NS) policies. In this algorithm, there are no notification for any communications states, except the last state which is for the next time step. For TW policy, each machine sends a notification of its current

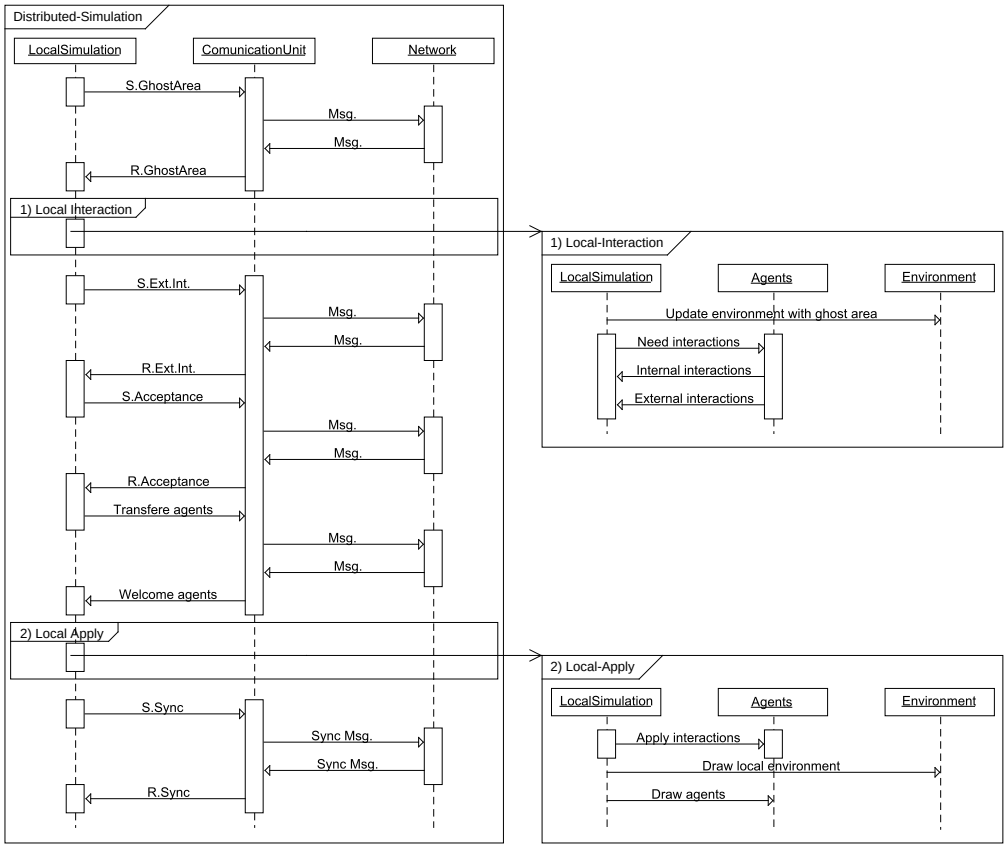

Figure 4.8: UML sequence diagram of local machine operations for one time step. First, each machine should send and receive *ghost area* information. Then, each machine should get interactions from its agents and divide them into two lists: *external* list and *internal* list. After that, each machine should send and receive external lists through the communication unit to and from other machines. Each machine can send and receive transferred agents, and then it can apply the list of possible interactions. Finally, each machine draw its local part and send synchronization messages to other machines.

time step, and it checks if it has permission for next time step. Which is in our case, the difference between the local machine's time step and slowest machine's time step, that is recorded, must be less than a W steps. W is a number of steps which can be defined by the user. Whereas, in case of NS policy, machines send only a notification for next time step and then begin of it directly. Thus, it is similar to TW policy, but with an infinity window $W = \infty$ of steps.

```
1  sync for zero-TS;
2  while Running do
3  |   exchange information with others;
   |   /* //Local interactions                                    */
4  |   get interactions from local agents;
5  |   solve conflicted interactions;
6  |   get external interactions list;
7  |   while not all machines are satisfied do
8  |   |   send external interactions;
9  |   |   receiving from others external interactions;
10 |   |   send acceptance For others' external interactions;
11 |   |   receiving from others acceptance;
12 |   end
   |   /* //Transferring agents                                   */
13 |   transfer agents with notifications;
14 |   receive agents from others;
   |   /* //Applying TS's interactions and drawing               */
15 |   applying interactions;
16 |   drawing;
   |   /* //Sync with others                                      */
17 |   receiving messages;
18 |   sync with others for next TS;
19 end
```

Algorithm 1: Environment distribution with strong synchronization policy.

4.8 Example of a time step execution

To explain the dynamic of a simulation time step, we illustrate it with an example of two machines in case of environment distribution (figure 4.9):

If we consider that we have two environment parts in two different machines (*machine1* and *machine2*). Each machine has its agents which exist within its environment part. Each machine begins with the same time step ($TS = 0$), which is represented by the first sync-line in figure 4.9.

Then, each machine sends to all other machines the *ghost-areas* information. This information about agents and environment area that near the edge zone between machines with positions depth of N for example. N is a depth of ghost area which can be determined by the user or by the largest vision depth of agents. After that, each machine has to receive the *ghost area*

```
1  sync for zero-TS to begin together;
2  while Running do
3  |  send information To Others;
4  |  receive information if exist without wait;
   |  /* //Local interactions                                    */
5  |  get interactions from local agents;
6  |  solve conflicted interactions;
7  |  get external interactions list;
   |  /* //Without any wait                                      */
8  |  send external interactions;
9  |  receiving from others external interactions if exist without wait;
10 |  add possible interactions to local collection;
   |  /* //Transferring agents                                   */
11 |  transfer agents without notifications;
12 |  receive agents from others if exist without wait;
   |  /* //Applying TS's interactions and drawing               */
13 |  applying interactions;
14 |  drawing;
   |  /* // W Time-Window-Sync with others                       */
15 |  receiving messages;
16 |  send next TS notification;
17 |  sync with others on W TS if machine reaches it;
18 end
```

Algorithm 2: Environment distribution with flexible synchronization policy

around it from other machine.

Later, each machine asks its agents about their interactions according to the *ghost area* information. Then, it divides these interactions into two lists: *internal interactions* and *external interactions*. The *external list* is related to the *ghost area* or interactions with neighbours agents from other machines. As it is external, we need to ask about agreement from others to this list, that for avoiding any conflict interactions between machines. As an example, two wolves want to eat the same sheep in prey predator model.

Then, each machine can send external interactions list and receive also from other their external interactions list. Later, machines can send and receive again acceptances for external lists. This loop can still re-run, until all machines are satisfied for all interactions between their agents.

After that, each machine has a list of possible interactions and it can

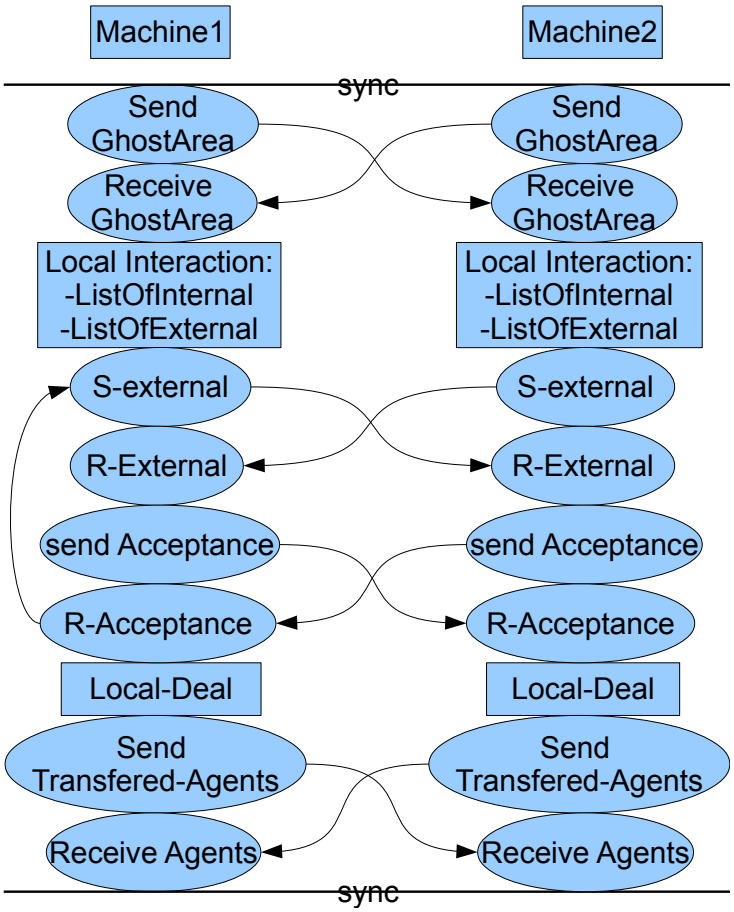

Figure 4.9: A scenario of a time step between two machines in case of environment distribution.

directly deal with it. It can apply it to its local agents. Then, each machine checks if there are any agent that wants to move from its part to another environment part. If it is the case, agents are transferred to the new part or the new machine. Finally, all machines are ready to sync for next time step.

In case of agents distribution, the scenario of a time step is similar to environment distribution scenario but with replacing *ghost area* information by information about other agents or *ghost agents* that exist on other machines. In addition to that, there is no transferring between machines as all machines are on the same positions range or the same environment, except if we need to activate a load-balancing mechanism.

4.9 Visualizations in large scale simulations

The visualization of large scale simulations in some applications is very important. It can be done by two approaches: either real time visualization during the simulation or offline rendering after the simulation.

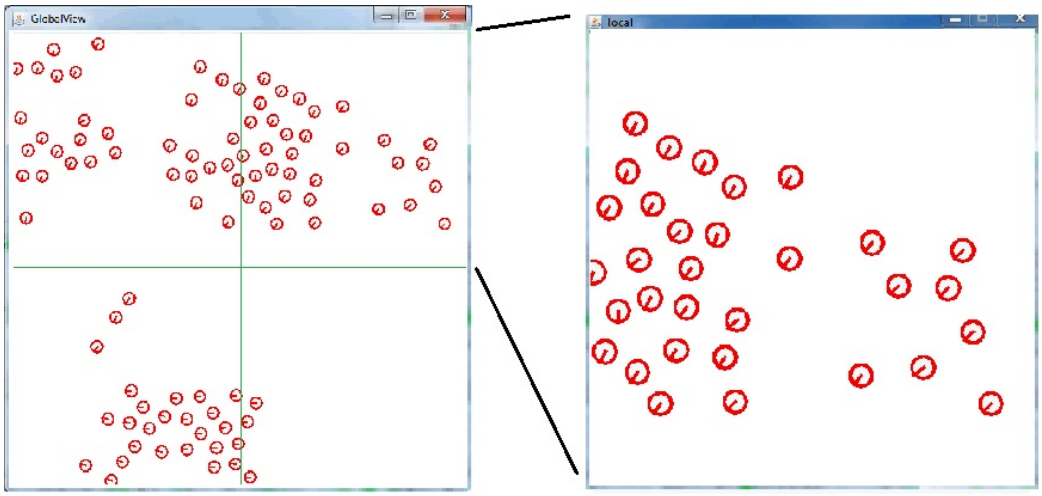

Figure 4.10: Visualization of large scale simulation between global and local visions.

The real time visualization approach can be built with two main ways: 1) a local visualization for each machine with the local agents and the local environment part and 2) a global visualization of all agents and the whole environment. The first way is the easiest way as each machine has all necessary information that it needs to draw its local part of the simulation. The second way can be done easily by collecting all necessary information to one machine, and then draw the whole simulation, but this solution in large scale simulations can have some limitations. We can optimize the global visualization of a large scale simulation by collecting images from each machine with a user needed resolution and then we can attach all images together in one global view of the whole simulation.

The second approach, which is rendering a scene after finishing the simulation, needs large amounts of memory to register all necessary information about agents during all simulation time steps, and that in large scale situations with billions of agents and with millions of time steps is very difficult.

4.10 Summary

In large scale distributed MAS simulations with millions or billions of agents, the distribution process can play a major role in the performance. Different distributions approaches can give us differences in the performance of some applications. For that, we have proposed two distribution types:*agents distribution* and *environment distribution*. In additions, we have proposed three synchronization policies to increase the performance in these kinds of large scale MAS simulations. To study the impact of these distributions and synchronizations policies on large scale applications, we have created our platform from the base to make all necessary experiments later (in next chapters).

In this chapter, we have described our platform for large scale distributed MAS simulations. We have presented the different components of the platform. We have detailed the machines units, which are: *communication unit*, *local simulation* unit and *visualization unit*, in the two cases: *agents distribution* and *environment distribution*. Then, the main platform layers have been discussed deeply and different simulation states have been introduced. After that, we have studied communication protocols and synchronization algorithms for three main synchronization policies, which are: *strong synchronization*, *time window synchronization* and *no synchronization* policies. We have explained the time step scenario between two machines to illustrate the differences between these policies. Finally, we have described our view of visualization in large scale simulations.

In the next part, we present our experiments with our platform on different application categories, these experiments are divided between two chapters (5 & 6). In chapter 5, we study the impact of two distributions types on different applications. Then, we study the three policies with different application categories in chapter 6.

Part III

Experimentations

Chapter 5

Effective distribution of situated multi-agent simulations

5.1 Introduction

In previous chapters, we have explored different distributions approaches according to the main concepts of MAS, which are *agents distribution* and *environment distribution*. We believe that in large scale simulation some applications can be simulated efficiently in one approach better than the other. To examine this hypothesis, this chapter focuses on experimentations with different categories of applications for large scale simulations. Our developed platform can handle both distributions approaches.

In following sections, we present our experiments on both distribution types. First, we explore different categories of applications that can make our hypothesis clear. Then, we explore two classical situated agent-based models, that we use in our experimentations, which are flocking model and prey-predator model. We measure the performance of the simulations with two criteria: *execution time* and *communication costs*. Our experiments show that one applications model is better with one distribution approach than the other. Then we explore more in details the two distributions approaches with different experiments.

Figure 5.1 shows our simulator on 50 machines that run boids or flocking behaviour of birds [Reynolds, 1987]. Each machine holds 100000 agents at the beginning of the simulation, so there are 5 million agents in total.

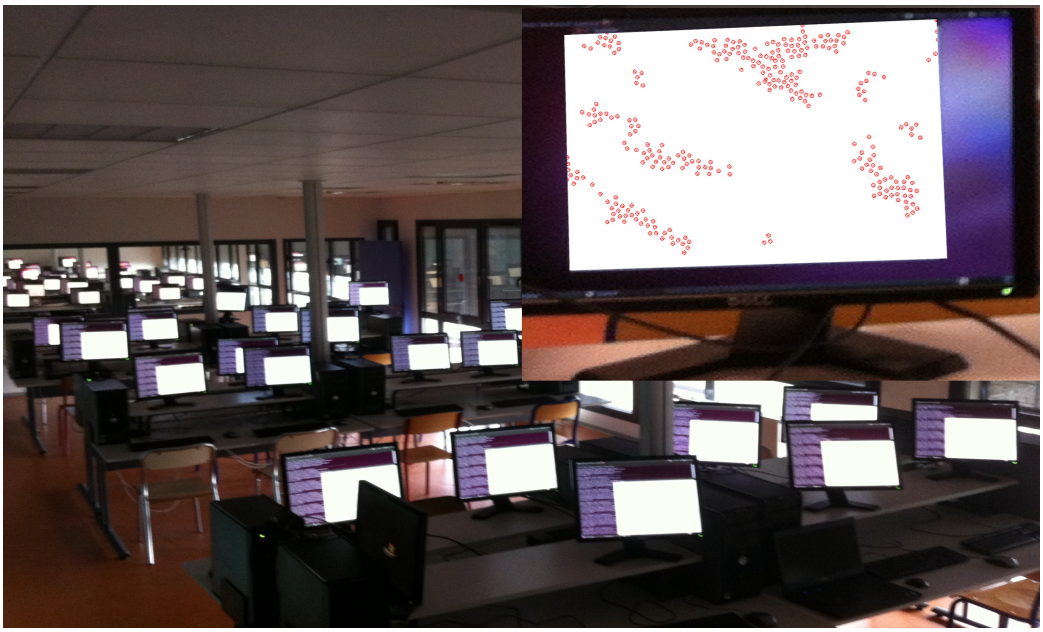

Figure 5.1: Five million agents on 50 machines for flocking behaviours.

5.2 Distributing MAS simulations efficiently

As we have seen before, we propose two interesting ways to distribute a simulation, and we believe that each one has some specifications and better performances on some kind of applications. Each application also may have some basic properties that determine the best distribution type for it.

Table 5.1 shows a comparison between two applications categories. Different applications can have different features, which come from agents' features themselves. For example, agent's life-cycle can be short (small number of time steps) or long during the simulation. Agents can move on small area or large area, and that can change the load of computations in the simulation. Agents can exist on the whole environment or can be aggregated on a part of it.

The user should be able to choose the distribution type that is more suitable for his application according to agents' features. For example, the short life-cycle and reproducing could make the agent distribution approach useless for some applications, because computation can be aggregated in one machine only. Whereas, the aggregation and the long life-cycle could make the environment distribution approach useless too for other applications, because computation can be aggregated in one machine too. For that, one distribution type can be the best solution for some applications than the

other.

Table 5.1: Applications can be in different categories according to agents' features.

Agent features	Category-1	Category-2	...
Life-cycle	Short	Long	...
Movement	Small area	Large area	...
Positioning	Everywhere	Aggregation	...
Reproducing	Yes	No	...

In the following sections, we describe the distribution of two classical multi-agent applications: flocking behaviour of birds and prey-predator models. Then, we show with our experiments that one of them is better in one distribution type, and the other model is better for the other type.

5.2.1 Flocking behaviour model

This application illustrates a steering behaviour that is commonly observed with birds or fish [Reynolds, 1987], which evolves in groups. In this model, there is only one kind of agent (e.g. bird), which can move forward with a group of other nearby agents within its perception range. Normally after some iterations in this simulation, groups of agents are emerging and after some times, there is only one big group of birds that moves smoothly together. On the environment's view, there is heterogeneous distributions, or some parts of the sky hold a lot of birds, while others are less filled (see figure 5.2).

In the environment distribution case, we can imagine that the load can be switched from one machine to another when a group of birds moves from one environment part to another. In this model, the best solution to distribute this model should be the *agent distribution*, so each machine has the same number of agents (birds) and thus the same computational cost.

Agents in flocking behaviour model

In this model there is only one kind of agent: *bird*. Each bird has its perception range which helps to find nearest set of other birds and then flies in the same direction as this set. Normally, bird try to fly near other agents. But, it can not fly too close or too far from the centre of that group which is the nearest set of birds.

At the beginning of the simulation, all birds are initialized and distributed randomly in the sky or on the whole environment with different directions.

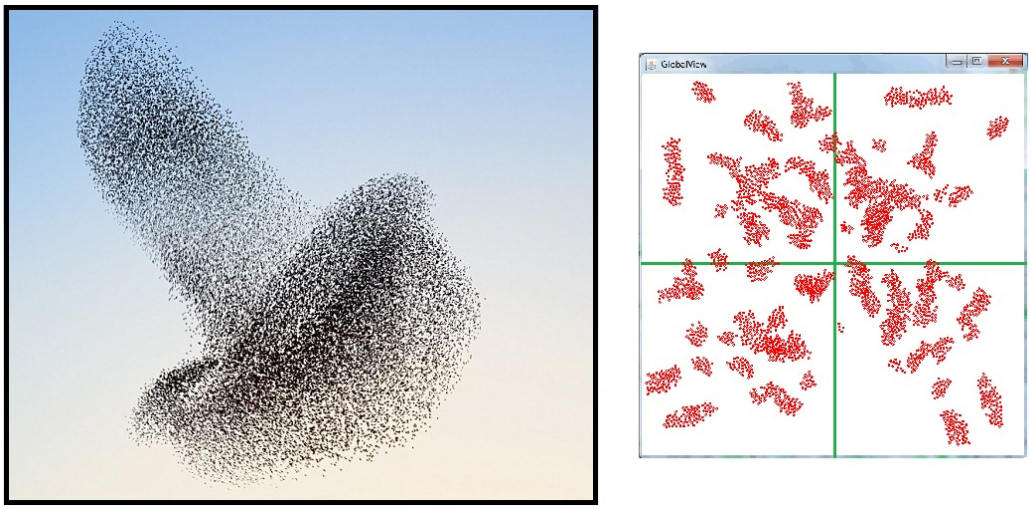

Figure 5.2: Flocking behaviour model [Reynolds, 1987] (left) and a demo of 4 machines that runs this model (right).

After some iterations, groups of agents are emerging and after some times, there is only one big group of birds which move smoothly together. Then, there are no homogeneous distributions in the sky (environment), some parts of the sky hold a lot of birds, while others are less filled.

Agents' properties

All birds have similar individual properties like:

1. *Vision*: birds can see other nearby birds in their perception range.

2. *Moving*: birds can move or fly with a direction. This flying comes from the balance of three main rules: *cohesion*, *alignment* and *separation* (see figure 5.3):

   (a) *Cohesion*: steer towards the average position of nearby birds in its vision.

   (b) *Alignment*: steer towards average heading of nearby birds in its vision.

   (c) *Separation*: steer to avoid crowding with nearby birds in its vision.

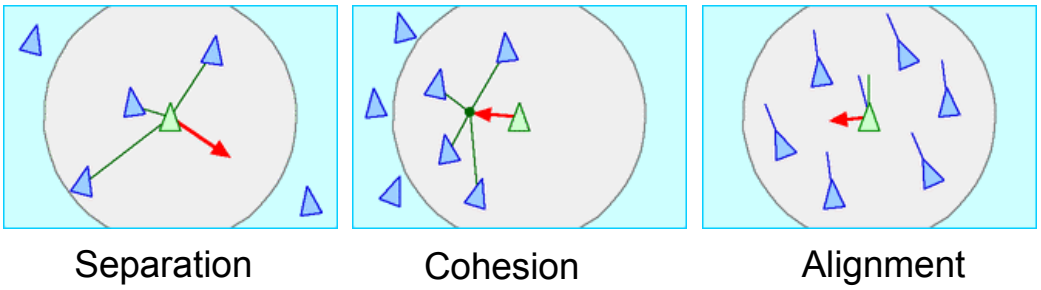

Figure 5.3: In flocking behaviour model: three main rules create the steering behaviour [Reynolds, 1987]: *cohesion*, *alignment* and *separation*.

Macroscopic phenomena

The macroscopic phenomena in flocking behaviour is very simple. Initially, the simulation should create each bird with a random position and a random direction on the sky. Then, all birds try to get close to other birds in their perceptions, then and after some iterations some birds should be together in larger groups. That means, after N time steps most of birds fly together in one and big group.

5.2.2 Prey-predator model

The second model is more complex than FB model, which is prey-predator (PP) model or Lotka-Volterra model. This model is a classical MAS model that uses agents with goals. A predator is an organism that eats another organism which is the prey. For example of predator and prey, we can simulate the co-evolution of wolves and sheep. Predators and preys evolve together, the prey is a part of the predator's environment, and the predator dies if it does not get enough food (or preys). Additionally, the predator is a part of the prey's environment, and the prey dies if it is eaten by the predator. The fastest predators in the environment are able to catch food and eat, then they survive and reproduce to make up more predators. The fastest preys are able to escape from predators, then they survive and reproduce more preys to keep the population. The two populations should evolve together to keep the ecosystem alive.

An example of such model is the wolf-sheep-grass simulation (figure 5.4) that proposed by Wilensky [Wilensky, 1997], which has been implemented in our framework for our experimentations. In this example, a wolf-agent tries to find and eat a sheep-agent, a sheep-agent searches for grass-agents to eat and grass-agents re-grows at a given rate. Wolves and sheep can have energy

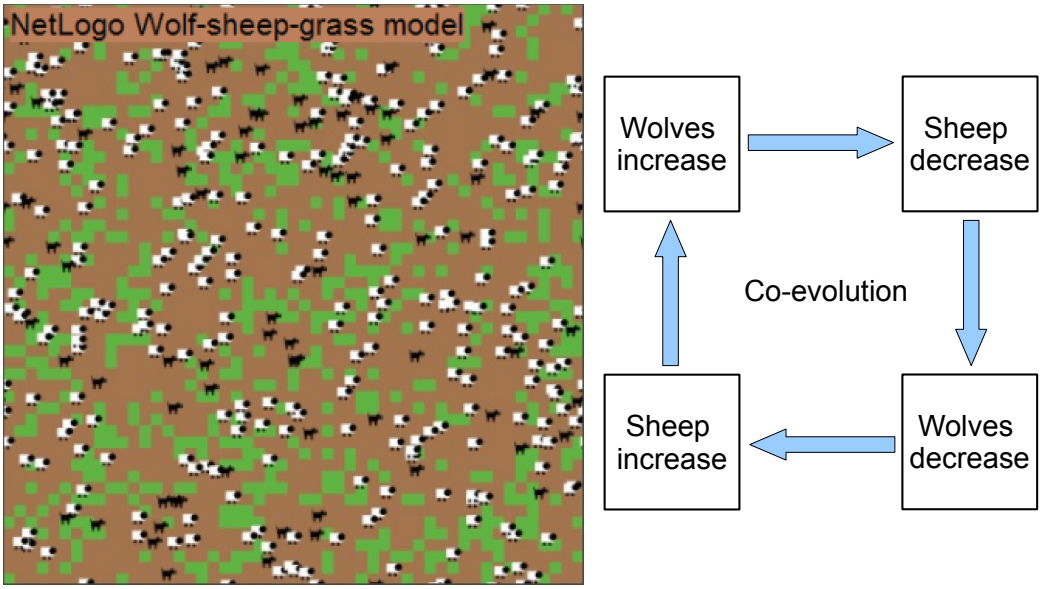

Figure 5.4: Wolf\sheep\grass model as prey-predator ecosystem in *Netlogo* [Wilensky, 1997] (left). Wolves and sheep co-evolve together to keep the ecosystem alive (right).

when they find something to eat, and then they can reproduce themselves.

Agents in prey-predator model

In this model there are three types of agents: *wolf*, *sheep* and *grass*. Some agents are preys (grass), others are predators (wolves) and the rest are preys and predators together (sheep).

1. *Wolf:* has a vision range and an energy to keep it alive and movable during the simulation. This energy is reduced at each time step, and if it becomes zero, then this wolf should die or disappear from the system. Wolf can get more energy by hunting more of its preys which are sheep. Wolf can produce other wolves at a constant rate (every N TS).

2. *Sheep:* has also a vision range and an energy that keep agent alive and movable. As the wolf, this energy is reduced over time and if it is zero, then the agent dies or disappears. Sheep can get more energy by eating more grass, and it dies if a wolf hunts it as well. Sheep can produce other sheep at a constant rate (every M TS).

3. *Grass:* can grow over time. It should give sheep energy if it has been eaten by a one of them. As well as others, grass can re-grow at a given

rate (every K TS).

Agents' properties

Agents have similar and different individual properties like:

1. *Move*: wolf and sheep can move to reach the food (preys). Grass can not move.
2. *Vision*: wolves can see sheep and seek to hunt one of them. Sheep can see grass and seek to eat them, and it can see wolves and should go away from them to escape.
3. *Eat*: wolf and sheep can eat to get more energy.
4. *Power*: wolf and sheep use energy to move and eat, so they lose energy over time but they can increase it by hunting more preys.
5. *Disappear*: wolf dies if its energy is zero. Sheep dies if its energy is zero too or it is eaten by a wolf. Grass can disappear if it is eaten by a sheep.
6. *Reproduce*: wolves and sheep can re-produce themselves at a given rate. Grass can re-grow after N TS.

Macroscopic phenomena

In normal situation, the number of wolves and the number of sheep can be inversely proportional in some periods of the simulation and can be directly proportional in others, or co-evolution of both populations. If the number of wolves increases, then they can eat more of sheep, and the number of sheep should decrease (see figure 5.4). Then, the wolves could not find more sheep to eat, and that leads to decreasing in wolves energies and then decreasing in the number of wolves. After that, the sheep can increase, because there are no more wolves that try to eat them, and again the number of wolves should increase as there are more and more preys to be eaten (see figures 5.4 and 5.5).

Figures 5.6 and 5.7 show the balance between the preys and predators to keep the ecosystem alive. Sheep and wolves must exist in the model during the simulation. If we lose all wolves or all sheep, then the model is destroyed. This is because, if we lose all sheep, then wolves can not find any sheep to eat, and all wolves should be died. Again, if we lose all wolves, then the number of sheep could be increased to infinity as there are no predators to

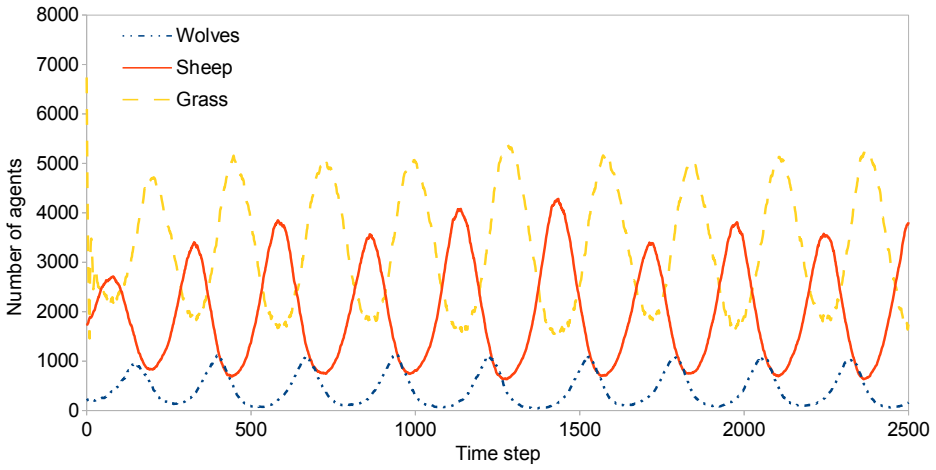

Figure 5.5: In prey-predator model, the agents number changes over time by being ups and downs. The numbers of preys and predators increase and decrease during the simulation as a cosine function.

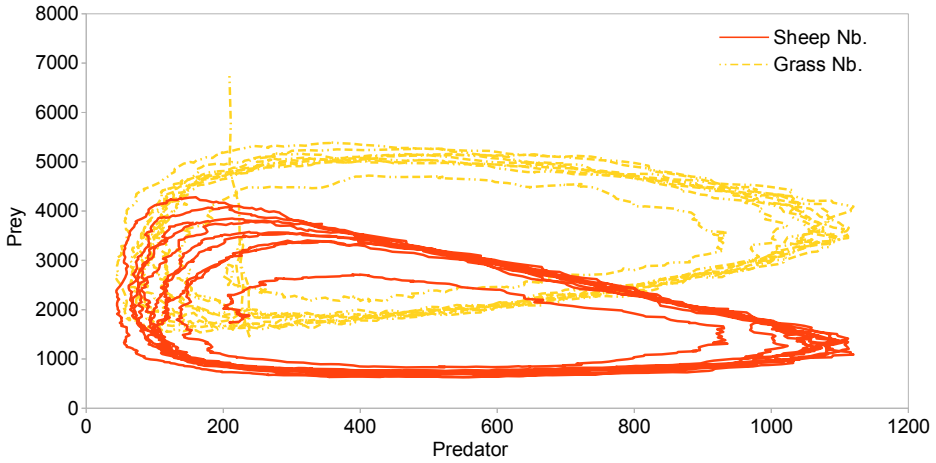

Figure 5.6: Numbers of preys to predators in the simulation. The number of grass to sheep and sheep to wolves.

attack them. Thus, all types of agents have to co-evolve to keep the model alive. For more details about the model, we add the table 5.2 with some statistical details that we gathered from our experimentations.

These two applications have been chosen because they imply distinct population dynamics. Indeed, in a prey-predator model, preys and predators can move during the simulation but they are homogeneously distributed over

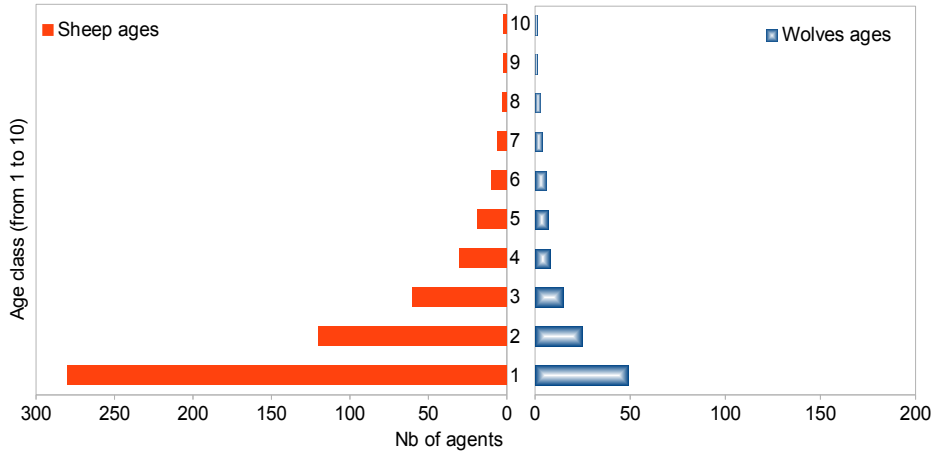

Figure 5.7: Age pyramid in prey-predator model, ages are classified to 10 classes (from 1 to 10): class-1 is the number of new agents and class-10 is the number of oldest agents.

Table 5.2: Prey-predator model statistics with initial 2500 agents.

Average life-cycles	Min/Max sheep Nb.	Min/Max wolves Nb.	Max-age of sheep	Max-age of wolves
276.24 TS	615/4281	44/1134	221 TS	153 TS

whole the environment. While in the flocking model, even if the distribution is homogeneous at the beginning of the simulation, flocks rapidly emerge together. Then and after some iterations, there should be only one main flock that appears in the simulation. It means that these two applications illustrate the trade-off that has to be made while we distribute the load between machines in a computer network, and the type of distribution can give really different results.

5.3 Experimentations mechanism

In this section, we evaluate the performance of our distributed simulator (D-MAS[1]) on two applications, which are *flocking behaviour* (FB) and *prey-predator* (PP) models, with two distributions: *environment distribution* and *agents distribution*. Most of experiments have been done with different parameters like: the environment size, the number of machines and the number

[1]D-MAS simulator is available at `https://sites.google.com/site/omarrihawi/resources/D-MAS-software`

of agents.

We use two main criteria to measure the performance:

- The *execution time*: the classical criteria which is mostly used by most of researchers to measure the performance of any platform. It is the needed duration time to execute a time step of the simulation. The performance should be better if this value is lower.
- The *communications cost*: we use this criteria to study the differences between machines in communication level. It helps us to quantify the quality of communications by measuring the numbers and volumes of messages.

5.4 Experimentations description

Most of the following experiments have been done on homogeneous hardware: 1) with laboratory machines[2] and 2) with clusters of grid computing (*Grid5000*[3]). In these experiments, several initial configurations have been used:

- We are able to simulate a large scale experiments on 50 machines with 5 million agents (see figure 5.1). But, some experiments are ranging from 1 up to 16 machines to facilitate the measuring.
- At the beginning of the simulation, the number of agents that is usually used for prey-predator (PP) model is 5000 agents per machine and for flocking behaviour (FB) model is 10000 agents per machine. That means, if we have one experience with 2, 4, 8 and 16 machines, we use $5000 \times 16 = 80000$ agents for 16, 8, 4 and 2 machines to make a reasonable comparison between machines in the same experiment.
- For the agent distribution, agents are divided equally between machines.
- For the environment distribution, environment is divided equally between machines as a grid as possible.
- Perceptions of all agents are small with respect to the environment slices that are used, and the size of *ghost areas* has been chosen as the maximum agent perception to avoid any deprivation of any agent.

[2]Intel-R CoreTM2 Duo CPU E8400 3.00GHz, memory 4GB and 100Mb connection
[3]Intel Xeon E5620 2.4GHz, memory 16GB and 1Gb connection *www.Grid5000.fr*

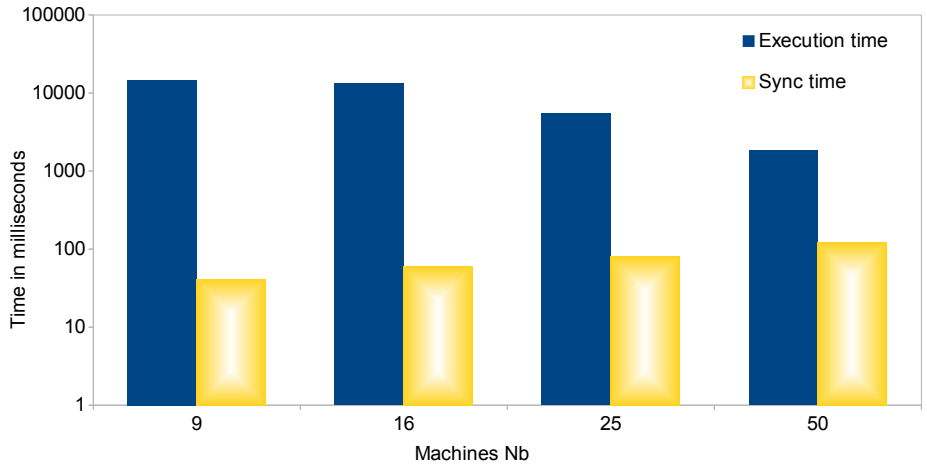

Figure 5.8: Time step delay and communication delay in prey-predator model with *environment distribution* approach.

5.5 Scaling the platform to 50 machines

We have tested a *prey-predator* (PP) model with initial 250000 agents distributed on 9, 16, 25 and 50 machines with the environment distribution approach. Figure 5.8 shows that the execution time is significantly reduced when more machines are used. In this figure, the first column represents the whole execution time, while the second one is the synchronization delay. The figure shows that with a small number of machines, the synchronization delay is small and the execution time is large as the number of agents is large too. Whereas, if the number of agents is small and the number of machines is large (in case of 50 machines), the synchronization delay has to be large too, but the execution time is small. It is clear that, it is not efficient to distribute a small number of agents on a large number of machines or a large number of agents on a small number of machines.

5.6 Efficient distribution of MAS applications

Next experimentations evaluate the performance of our distributed simulator with two types of distribution: *environment distribution* and *agents distribution*, and on two models: *flocking* and *prey-predator*.

Figures 5.9 & 5.10 show results for two types of distributions for each application. Figure 5.9 shows that in flocking model and in case of agents distribution the performance is better than in case of environment distribution. Because, in case of environment distribution there could be many large

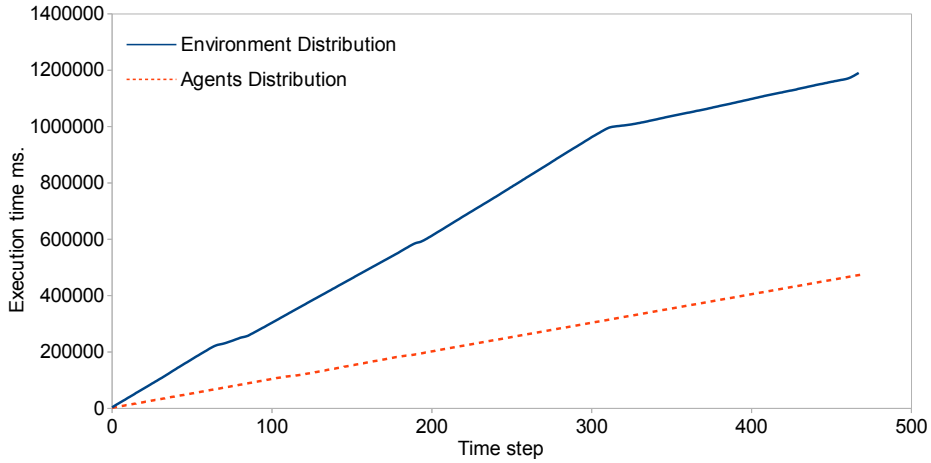

Figure 5.9: Total execution time of flocking model with two different distribution types. *Agent distribution* shows better performance than *environment distribution*.

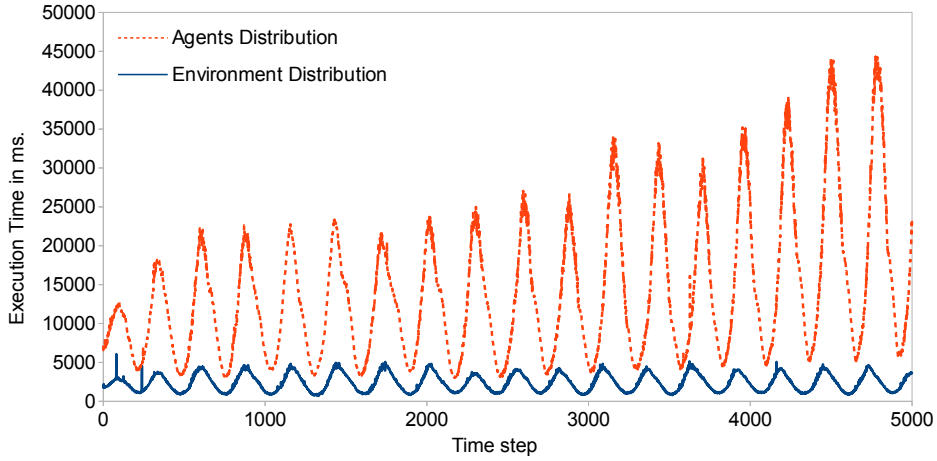

Figure 5.10: Execution time of prey-predator model with two different distribution types. *Environment distribution* shows better performance than *agent distribution*. The execution time has a cosine form, because preys and predators numbers have been changing during the simulation.

groups of birds that fly together and swap between machines. That should increase the execution time as there is more load of birds on one machine than others. Whereas, in case of agent distribution we have the same number of agents on each machine and the execution time should be the same for all.

Figure 5.10 shows that the prey-predator model has a completely opposite situation than flocking model. The execution time is better in case of

environment distribution type than agent one, because there are no large groups of agents on the same environment part and swap between machines as in flocking model. In prey-predator model, agents can reproduce and die during the simulation at each time step, for that the execution time looks like a cosine function as the number of agents is reduced and increased during the simulation on the same machine.

To summarize, environment distribution type is better for prey-predator model than agents distribution one, whereas agents distribution type is better for flocking model than environment one.

5.7 Communication costs evaluation

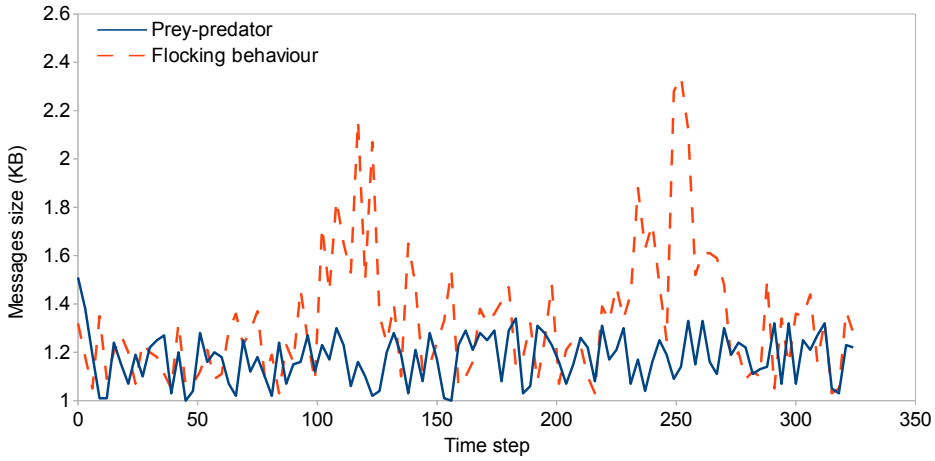

Figure 5.11: Communication costs of both models with environment distribution approach. The prey-predator model is more stable in messages volumes. Whereas, the flocking model is not stable and has some peaks from time to time.

In this experimentation, we evaluate the volume of exchanged messages in both applications. We test the flocking and prey-predator models on two machines with environment distribution approach. Figure 5.11 shows that the flocking model has important variations in messages volume, while prey-predator model did not have such peaks and is more stable. These differences come from agents' behaviours in both model. In flocking model, we may have a large number of birds moving from one machine to another. That means there are bigger messages between machines to transfer these large group of agents and then more communication costs too. Whereas in prey-predator model, it is not the case because agents do not have aggregation

behaviours like in the flocking model, so the model is more stable in the size of exchanged messages between machines. The figure demonstrates that in distributed situated-MAS simulations with spatial environments, speed up is highly related to the dynamic of agents movements and agents models.

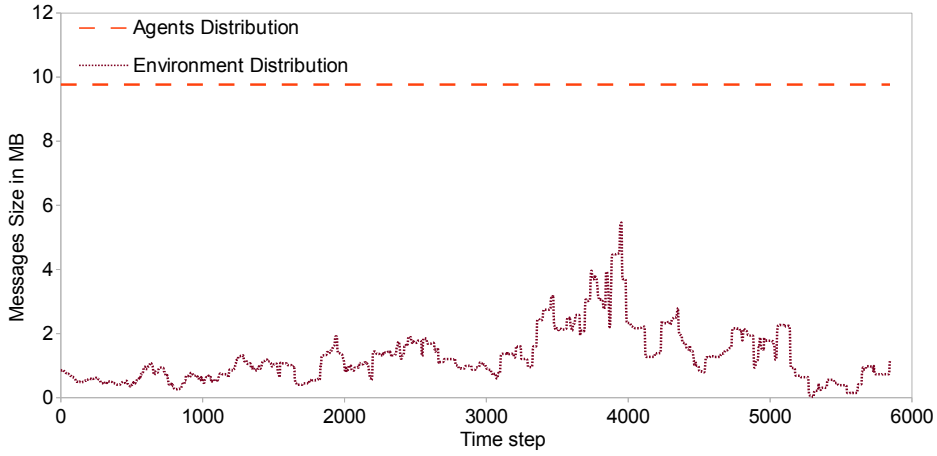

Figure 5.12: Messages volume of flocking simulation with two distribution approaches. *Agent distribution* shows more stability than *environment distribution*.

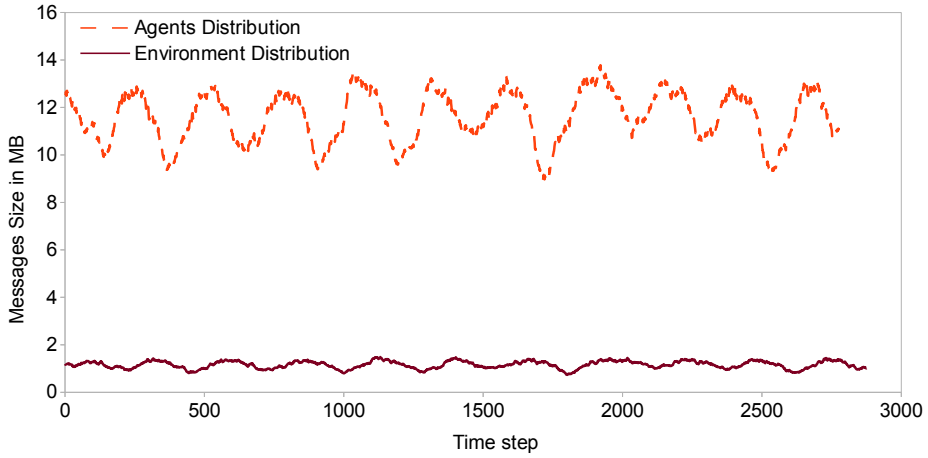

Figure 5.13: Messages volume of prey-predator simulation with two distribution approaches. *Environment distribution* shows more stability than *agent distribution*.

To detail more the flocking behaviour, we test communication costs on both distribution types, figure 5.12 shows that the communication costs are

larger in agents distribution type than environment one but more stable. This is because in flocking with agent distribution, the number of agents is fixed in all machines. Then, the information, which have to be sent between machines, have the same size. Whereas in environment distribution, some birds can be together in one big group on a part of the sky or on one machine, and other machines may have less birds. That should make the communication cost lower but less stable in environment distribution.

However, in prey-predator model the stability exists in environment distribution and not in agent distribution (see figure 5.13). This is because in prey-predator model, there are agents that reproduce and die during the simulation in the same machine at each time step. Thus in agent distribution, it can make more (or less) charge of agents in one machine than the others, and the messages size can be changed during the simulation to look like a cosine function and that should make it less stable in agent distribution type than environment one.

5.8 When should we use each type of distribution?

Table 5.3: A comparison between two models with two distribution types.

	Agent distribution	Environment distribution
PP execution time	Not efficient	Efficient
FB execution time	Efficient	Not efficient
PP communication cost	Not stable	Stable
FB communication cost	Stable	Not stable

Table 5.3 shows a comparison between the two models (PP & FB) with two distribution types. Environment distribution type is better in execution time for prey-predator model than flocking model, whereas agents distribution type is better for flocking model than prey-predator model.

However, for communication costs, environment distribution is more stable for prey-predator model than agent distribution, whereas agents distribution is more stable for flocking model than environment one. To summarize, some distribution types are more suitable for some applications than others.

To analyse the results in details, we try to extract some general features from both models, then the user can choose which distribution type is more suitable for his application type (see table 5.4). In prey-predator model,

agent' life-cycle is short (small number of TS) and all agents exist overall the environment. Whereas in flocking model, it is completely the opposite situation, agent' life-cycle is long and agents are aggregated during the simulation in one place only.

To conclude, the short life-cycle and the reproduction could make the agent distribution approach a bad solution for prey-predator model, because computation can be aggregated in one machine only. Whereas, the aggregation and the long life-cycle could make the environment distribution approach a bad solution for flocking model, because computation can be aggregated in one machine only too. For that, the agents distribution is the best solution for flocking model. Whereas for prey-predator model, it is the opposite.

Table 5.4: Analysis of agent's features between two models: prey-predator and flocking.

Agent features	Prey-predator	Flocking
Life-cycle	Short	Long
Movement	Small area	Large area
Positioning	Everywhere	Aggregation
Reproducing	Yes	No

5.9 Environment distribution experiments

We have studied in details the *environment distribution* (ED) with *ghost area* approach on prey-predator model. Our experiments have run to measure: the execution time, the communication cost and the depth of ghost area.

5.9.1 Execution time

Figure 5.14 shows that the execution time reduces significantly when more machines are used. We have tested prey-predator model with 42000 agents distributed on 1, 2, 4 ... machines. The figure shows that, with two machines the performance is six times faster than using only one machine. This is because, the complexity of interactions between agents in a complex model like prey-predator model is not linear, even if we have a communication between two machines. However, if the number of machines is increased, the communications also between machines will be increased too and we should not observe such speed-up in the execution time.

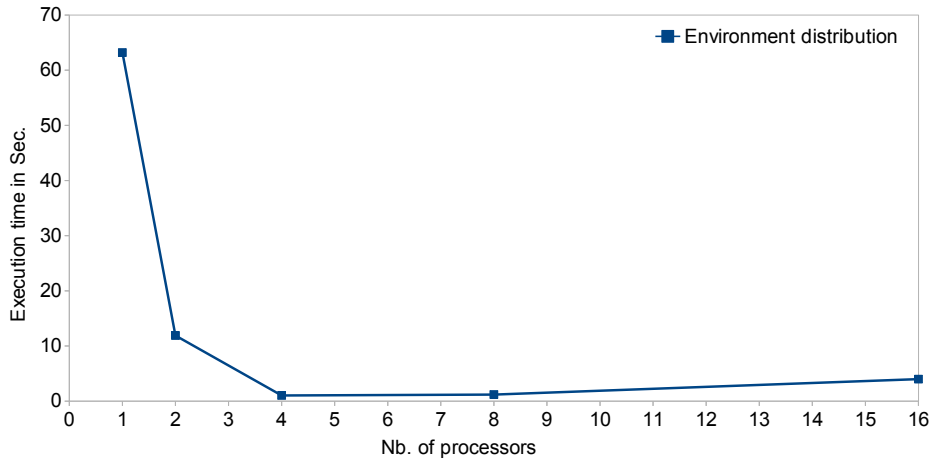

Figure 5.14: Time step calculation delay for prey-predator model with 42000 agents on 1, 2, 4, 8 or 16 machines. More machines can reduce the execution time, but more extra machines can make it worse, because it can increase the communication delays.

5.9.2 Communication delay

The figure 5.15 shows that, communications delay is increased when the number of machines is increased. The first column represents interaction delay, while the second one is the whole execution time with the communications delay, which are separated slowly.

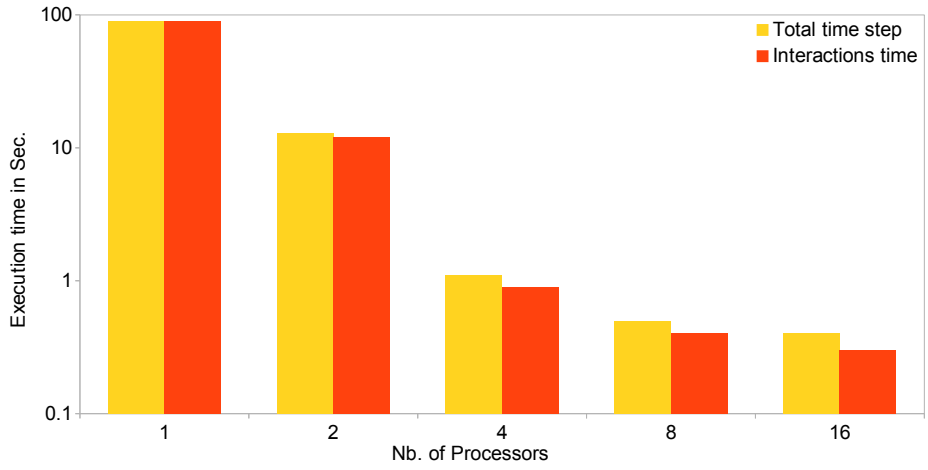

Figure 5.15: A log view of the total time step delay and the interactions delay.

5.9.3 Ghost area experiments

In this experiment, we launch 3x3 machines for flocking model with 9, 900, 9000, 90000 and 900000 birds. Birds are flocking between all 9 machines and with different depths of *ghost area*: 10, 50 and 100 units, unit is the size of the smallest agent in the simulation. However, we run the simulation until 100 units only, because all agents' visions are less than 100 units. It is clear that it is not useful to launch the simulation with more than 100 units. Otherwise, it should make high communication cost for nothing.

Figure 5.16 shows that the delay has increased by increasing the number of agents. It also shows that *ghost area* with different depths has the same effect on the simulation. That can be explained by the communication protocol, which needs only one message to send all information about edge-zones to other machines.

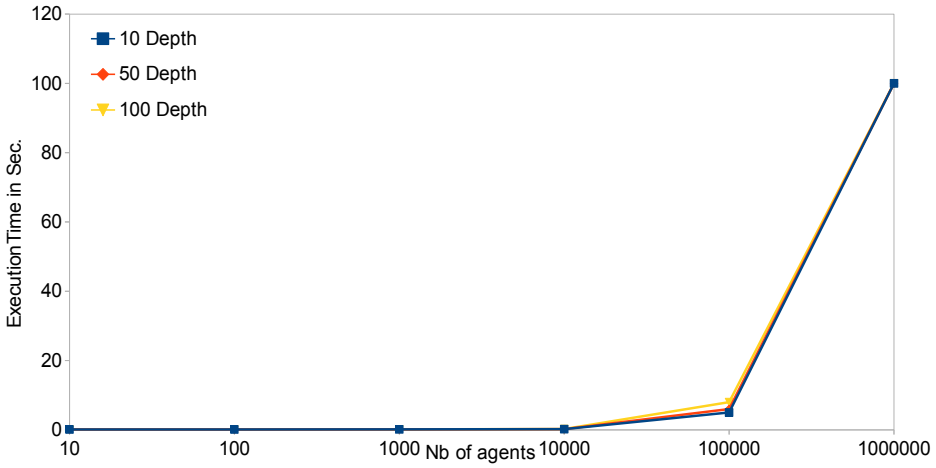

Figure 5.16: Time step with different depths of *ghost area*. As all necessary information about *ghost area* should be sent with one message only, the simulation with different *ghost area* depths can have close results.

5.10 Agents distribution experiments

We have seen before that the prey-predator simulation is better in *environment distribution* approach than *agents distribution* one. To explore more that, figure 5.17 shows an experiment on 4 machines (m1, m2, m3 & m4) for prey-predator model in both approaches. The initial load for both approaches has been exactly the same for all machines, then after N time step the load is calculated in each machine. The figure 5.17 shows that the load between

machines can be changed in case of *agents distribution* more than *environment distribution*. That can affect the balance between machines, as in all machines there are some agents die and re-produce during the simulation.

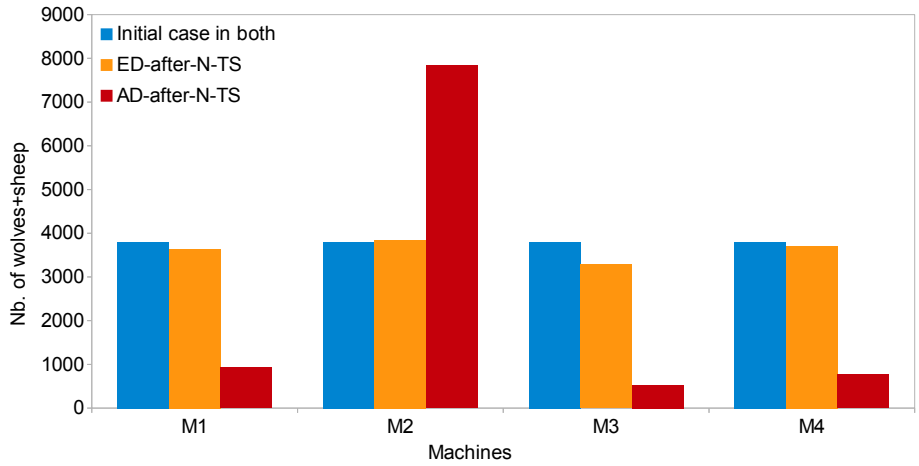

Figure 5.17: Prey-predator model after N time step between *agent distribution* and *environment distribution*. At the initial state for both approaches, all machines have the same computational load. After N time step, *environment distribution* can keep similar loads between machines. Whereas, *agent distribution* can not and some machines have more loads than the others.

The best solution to re-balance the loads between machines is by adding a *load balancing* mechanism, to re-load machines with the missing agents at each N time step. But, even with this solution, the simulation should be parametrized carefully to avoid high communications.

Figure 5.18 shows experiments on prey-predator model with a *load balancing* mechanism. At each N time step, this mechanism transfers the mission numbers of agents between machines to re-load the balance in the simulation. If the simulation make more balance on the load, then more communications delays can be produced and that can affect the performance.

Now, we experiment again the prey-predator model with both distribution approaches on 9 machines, but with a load balancing-mechanisms for *agent distribution*. Figure 5.19 shows that *environment distribution* approach is better than *agent distribution* approach in prey-predator model even with load balancing mechanism. Our explanation for that, the communication cost of external interactions is higher in *agent distribution* approach than *environment distribution* one. Because in *agent distribution*, all machines can have external interactions with all other machines, whereas in *environment distribution* each machine can have external interactions with its neighbour-

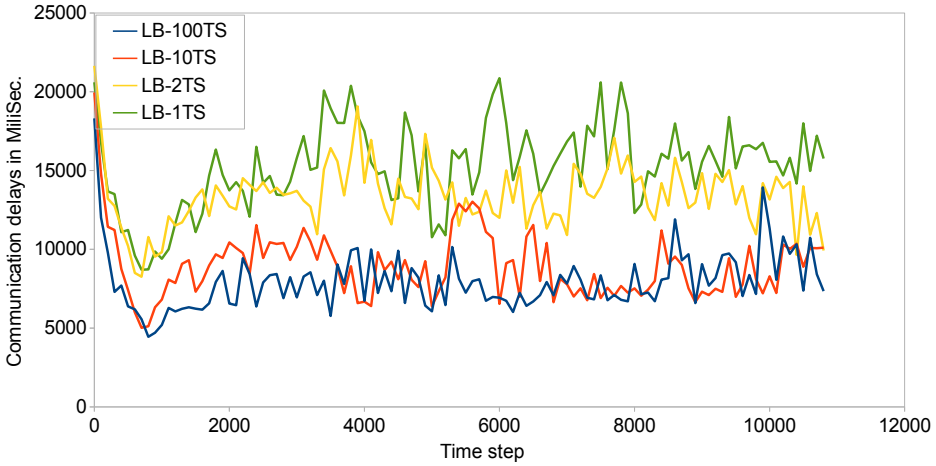

Figure 5.18: Communication delays in prey-predator model with different simulations that apply load-balancing at each N time step (LB- N TS).

ing machines only.

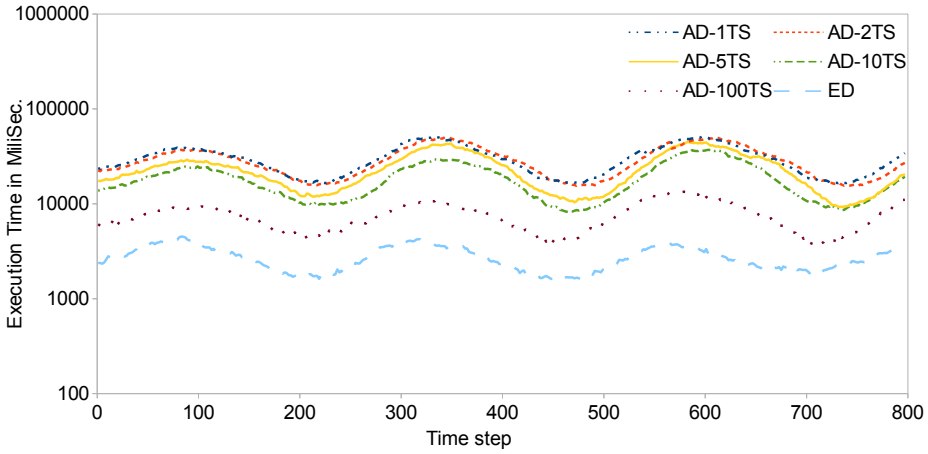

Figure 5.19: In prey-predator model and even with load balancing mechanism for *agent distribution*, the *environment distribution* shows a high performance.

5.11 Summary

In this work, we have proposed two distributions approaches for large scale situated-MAS simulations: *environment distribution* and *agents distribution*.

We have evaluated them with two different applications, which are the flocking of birds and prey-predator models. In case of *environment distribution*, the simulated environment is divided into different partitions on different machines, and then each partition is allocated on one machine only. During the simulation, each machine communicates with its neighbouring machines to collect the needed information about the surrounding areas or *ghost areas*. In case of *agents distribution*, the simulated environment is the same for all machines and agents are divided into different lists between different machines. During the simulation, each machine communicates with others to collect the needed information about other agents or *ghost agents* which exist on other machines.

Experimental results show that the proposed distributions types have better performances in some models than others (table 5.3). For example, prey-predator model has better performance in the execution time than flocking model when we distribute the environment. Whereas, agents distribution type is better for flocking model.

To conclude, the user has to chose the distribution type that can be more suitable for his application according to agents' features (table 5.1). For example, if the agent moves in a small area, it is allocated everywhere in the environment and its life-cycle is short, then the *environment distribution* is recommended. Whereas, if the agent life-cycle is long and agents are aggregated in some parts of the environment, then the *agent distribution* is recommended.

In the next chapter, we present the second part of our experimental results on large scale simulations, which relates to synchronization issues.

Chapter 6

Relaxing synchronization constraints in large scale simulations

6.1 Introduction

To simulate millions or billions of agents and interactions, we have to distribute our MAS simulator in order to scale it on different machines. A safe approach is to use a **strong synchronization** policy, but it implies a high cost in communications and longer execution times.

However, when we are working with large scale simulation with situated-MAS applications, the goal is not to observe millions of individual interactions or microscopic level, but to observe properties at macroscopic level. In some applications, we can even consider that if some agents fail or can not interact as fast as other agents can, then that should not be critical to the global simulation outcome. This work presents a first study of synchronization costs in performances and the impact of different synchronization policies on the preservation of emergent macroscopic properties.

This chapter studies the performance costs of several synchronization policies and their impact on the properties of simulations. We evaluate our simulator on two situated-MAS applications: prey-predator and capture the flag. We have studied three synchronization policies for distributed MAS simulations: *strong synchronization*, *time window synchronization* and *no synchronization* policies (as detailed in chapters 3 & 4). We have studied the changing that can be happen in interactions when we switch between these policies, and we have studied the instability of some application with different configurations when we relax synchronization constraints.

6.2 Time in large scale simulations

The *time* is the most important notion in our real-life. *Time* organizes all events that occur from the past through the present to the future. Without *Time*, all events would be merged together and there will not be a law for the occurrence of the real-life stuffs.

As we mentioned before, there are several notions of time: the user time (or real time) and the simulated time (or time step). Time step (TS) is a set of small durations used to produce evolutions in a simulation.

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 a time step (TS) in the simulation. Normally, the simulation must prevent any conflict between two or more actions in any TS. For example, in the prey-predator applications, if two predators try to attack the same prey in the same TS, a rule has to be given to define the outcome of such conflicting interaction.

In centralized MAS simulations, there is only one simulation time step that organize agents evaluations and that allows them to interact in a given period. In distributed simulations, there is one logical clock per machine and the needed time to handle a TS is not the same on all machines. 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 synchronizations have been proposed in chapter 3.

The question that we are interested in this work 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 large number of agents, it should not be so important if some agents fail to interact, or if they can not interact as fast as other agents. However, in other applications, like urban traffic simulations, we need reproducibility and reliability to ensure that all interactions between agents are fulfilled.

6.3 Synchronization policies

The main issue in distributed systems is the time management between machines [Scerri et al., 2010; Siebert et al., 2010]. There are mainly two synchronization approaches in distributed systems: *conservative* (or *synchronous*) and *optimistic* (or *asynchronous*) synchronizations [Logan and Theodoropoulos, 2001; Gupta et al., 2007; Fujimoto, 2000].

However, and as we mentioned in chapter 3, we have explored three synchronization policies for distributed MAS simulations [Rihawi et al., 2013c]: *Strong synchronization* (SS), *time window synchronization* (TW) and *No synchronization* (NS) policies.

6.3.1 Strong synchronization policy

In SS policy, 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. This policy is similar to centralized approach in outcomes, because all machines have to progress together, and then all agents have the same chance to interact.

To implement SS policy, more messages have to be exchanged between machines, and then communications costs are increased. This kind of conservative approach strictly avoid causality errors, but it can produce communication delays. Thus, the disadvantage of this policy is that all machines have to communicate a lot to progress one time step only, even if there are some time steps without external interactions between agents from different machines. This could be the reason for higher and useless communication costs.

6.3.2 Time window synchronization policy

In TW policy, machines can progress in the simulation with small and limited number of time steps. Thus, a W time window defines the worst spread in time steps that can happen between the slowest and fastest machines. 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.

However, with this approach of course we can have situations where agents in different time steps can interact. These situations can be ignored in large scale simulations if the impact of time incoherency in some interactions is negligible to the high volume of interactions in the whole system. This is a strong hypothesis that is studied through the two applications in this chapter.

6.3.3 No synchronization policy

In NS policy, machines can simply drop the synchronization. It can be seen as a variation of the time window policy with an infinite window size. This approach uses the available speed of all machines, because all machines can be free from all constraints to progress alone in the simulation independently.

The advantage of this policy is that we can gain the maximum speed in all machines, because there are no useless communications between machines.

However, with this policy the disadvantage is that some machines are unable to get the same outcomes as in the centralized approach, because not all machines progress together at the same pace, and then some agents have not the same chance to interact as others. That can be the reason to lose the macroscopic behaviour in some applications, and that can make the simulation useless. We will see later in experiments that this policy is not fitted for all applications.

To conclude, we have presented the three synchronization policies that are used in following experiments. With 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, with other approaches machines progress independently with a relaxed synchronization.

The problematic studied is: whether we are able to keep macroscopic behaviours emergence when relaxing synchronization constraints. The question is: can we gain performance by reducing synchronization costs in some kind of applications?

6.4 Experiments on two extrema models

In our experiments, we choose two applications that have been implemented and benchmarked to quantify the impact of the three proposed synchronization policies. It seems obvious that time inconsistencies do not have the same effect in all applications. For example, the simple 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 the outcome of the simulation, more precisely on the conservation of the expected macroscopic behaviour.

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

6.4.1 Prey-predator model

Prey-predator (PP) model or Lotka-Volterra model is a classical multi-agent application that involve two kind of agents, preys and predators. Both kinds reproduce themselves at a given pace, and predators seek and eat preys. If

a predator do not find preys quickly enough, it can die of starvation. This application illustrates population co-evolution in a simplified ecosystem. An example of such model is the wolf-sheep simulation proposed by Wilensky [Wilensky, 1997], which we have implemented in our testbed. We have detailed this model in the previous chapter.

6.4.2 Capture the flag model

This 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* (CTF) application with two competing populations or two 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).

The attack action is simple: if an attacker agent from one team detects another agent from different team (attacker or flag), then it tries to reach that agent to destroy it and simply both agents die.

To enhance the stability of this model, we add defence behaviour in attacker agents to protect their flags. This ability can be done by observing the number of team attackers around a team flag. If this number is small (less than N for example), then the attacker agents have to flock around that flag to protect it from the other enemy team.

Agents in capture the flag model

In this model, there are two types of agents: *flag* and *attacker* which can belong to different teams like: *red* and *blue* teams (see figure 6.1):

1. *Attacker:* this agent has a vision range and can move. It can defend team flags or attack any enemy agent which is an attacker or a flag from another team. If it finds an attacker from the enemy team, then it should seek to its position and destroy it (both agents die). If this agent find other team flags, then it can seek to its position and destroy it too.

2. *Flag:* this agent can not move and can generate more attackers at a given rate. This agent dies if an attacker from different team attacks it.

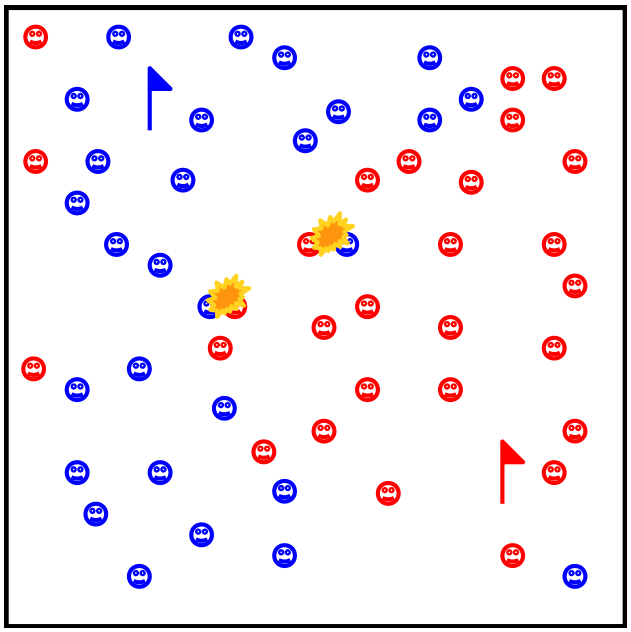

Figure 6.1: Two teams in capture the flag model, agents can be: flags or attackers. The first team, that loses all its flags, is the loser.

Agents' properties

Agents have similar and different individual properties like:

1. *Move*: attacker can move to defend its flags or attack other agents from other teams (flags or attackers). Flags can not move.

2. *Vision*: attacker can see other agents around it. Flags can not.

3. *Attack*: attacker can attack other agents. Flags can not.

4. *Defence*: attacker can use its vision to check if there are enough team's attackers around the team's flag, if it is not the case then it flies around it to protect it from enemies.

5. *Disappear*: attacker disappears if it attacks other agents from another team, or if it is attacked by another attacker from another team. Flag should disappear if it is attacked by an enemy attacker from other teams.

6. *Produce*: Flag generates new attackers at a given rate.

Macroscopic phenomena

The macroscopic phenomena in this model is very simple. With the same number of flags for each team, we should have a balance between all teams during the simulation. All flags must be alive and they can produce more attackers. If one team lose all its flags, then it should lose all its attackers too, and then the game is over. The idea of this model is that: if all teams have the same number of flags then the balance must exist between teams and the model is stable. Otherwise, the simulation can not keep in progress and it becomes unstable.

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 of agents, the model is destroyed. For *Capture The Flag* (CTF) model, the stability is to keep all flags of all teams alive so they can generate more and more attackers. If one team loses its flags, then its agents should disappear, and the other team should win.

6.5 Experiments on synchronization policies

We have executed our experimentations on similar machines (similar hardware). Most experimentations have run until 2 million time steps and the only parameter modified is the time window size to relax the synchronization. We have started with a time window $W = 0$ (SS policy), then 10, 100, 1000, 10000, 100000 and ∞ or NS policy. All experiments run with *environment distribution* approach, because prey-predator model is more fitted for this approach than the other (as we have seen in the previous chapter).

Figure 6.2 shows that NS policy always gains the maximum speed in prey-predator model. This is because, it is free from all communication delays. But, in other application like CTF, it can be unstable in case of NS policy because one of the population could disappear.

Table 6.1 shows results of stability of both models. The prey-predator model stay stable for a long time (until 2 million TS) for all experimentations. The co-evolution of preys and predators has been preserved until we reach 2 million time steps, even without any synchronization. However, in this model the life-cycle of agents is 300 time step only as we have seen in the previous chapter. Thus, for more than 300 time step (until 2 million TS) the model should reproduce the same behaviour, and the wolves and sheep

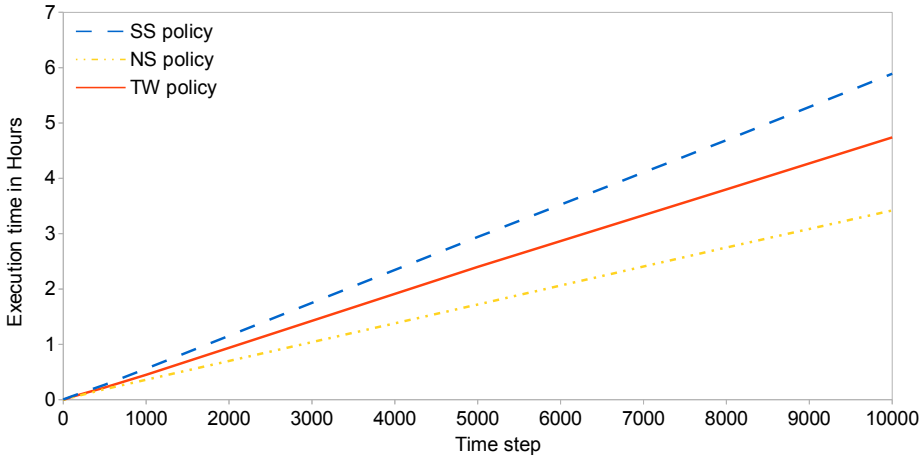

Figure 6.2: In prey-predator model, the evaluation of execution time shows that NS policy always gives the maximum speed and it can preserve the macroscopic behaviour. SS policy has more delays in each time step, because it has more communications in each time step.

should co-evaluate to ∞ TS. In other words, this application is stable for all synchronization policies, stable means here that all populations still exist and co-evaluate in the model. Whereas, CTF model in some cases is not stable after N TS.

Table 6.1: Summary table with results for two models with three synchronization policies until 2 million TS. Prey-predator model can stay stable for a long time with all policies. Whereas, the capture the flag model could not be stable after N TS.

	SS policy	TW policy	NS policy
PP model	Stable	Stable	Stable
CTF model	Stable	Stable less N TS	Unstable

6.5.1 The gain from relaxing synchronization

Figure 6.3 shows that with large number of agents (large scale) we can gain more and more from relaxing the synchronization. With prey-predator model, if we increase the number of agents, we get more gain from *NS* policy.

In the following sections, we examine the impact on the interactions between agents on the stable model (PP model), and see how interactions can be

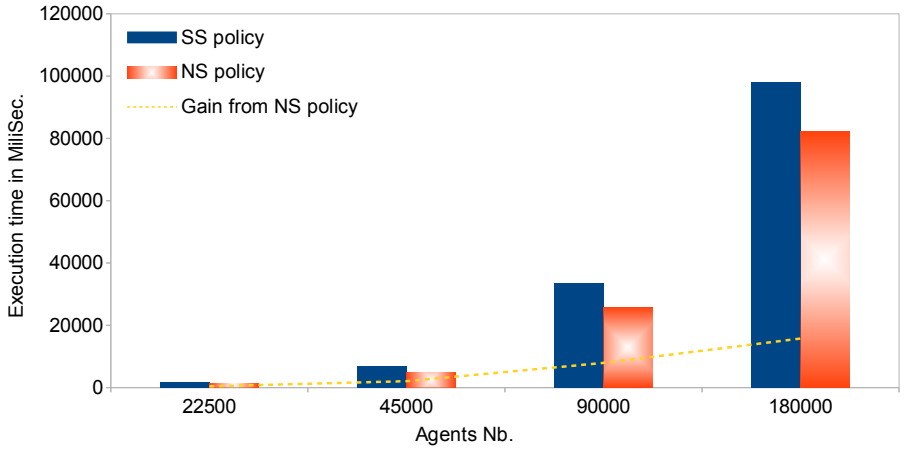

Figure 6.3: The gain from *NS* policy with different number of agents, with high number of agents we gain more than 20 seconds in each time step as average.

changed when we change the synchronization policies. Then, we study the instability of CTF model in details through a bias configuration.

6.5.2 Policies impact on interactions

External interactions in PP model

This experimentation study the *External Interactions* (EI), which are interactions between agents from different machines. It is clear that, when we use different synchronization policies, the exchanged information between machines can be affected and then the probability of EI occurrence should be affected too.

We test different time window sizes: from 0 (SS), 2, 100 and until *NS* or ∞ and each test was until 100000 TS. Figure 6.4 shows the numbers of EI, which are executed in the prey-predator model with different policies for 5000 agents of sheep, wolves and grasses between two machines. Results show that for *strong synchronization* (SS) policy we have a lot of EI, this is normal because the information about agents can be sent and received between machines safely. But, for *no synchronization* (NS) or *Time window* (TW) policies the external interactions are reduced significantly less than the SS policy, even if we have small size of TW like W=2 or 1 time step only.

Our explanation of that, even if we choose one TS only as window permission, then machines can advance in TS for one step only. Then, the information about agents can be sent from machines to another, but it may

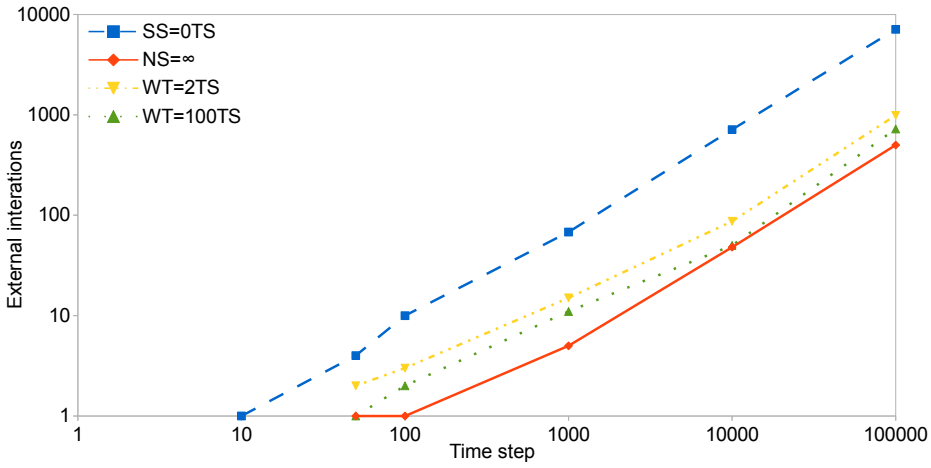

Figure 6.4: The external interactions with different synchronization polices in prey-predator model. SS policy has the highest number of external interactions because all machines progress together during the simulation. Whereas, TW policy has a small number of interactions, even with a W window permission of TW policy is only one time step.

not be received from others. Because machines can advance in the simulation even in one TS, then the number of external interactions becomes small.

Invalid time step interactions in PP model

In this test, we study the interactions on prey-predator model, when machines are not on strong synchronization. We define *Invalid-TS Interactions* (ITSI): as an interaction between two agents with different *time step* (TS) from two different machines or on the same machine. For example, two agents from two different machines which are not on the same TS if one machine has been faster than the other in simulation progress.

Figures 6.5, 6.6 and 6.7 show that the percentage of ITSI is increased when the W window permission of TW policy is increased too. However, the percentage of ITSI is less than 0.4 from the total number of interactions. In cases of 4 (or 8) machines, we have a double percentage than 2 machines only. In fact, in case of 2 machines we have two edge zones that allow agents to swap between machines. Whereas, in cases of 4 (or 8) machines we have 4 edge zones that allow agents to swap between machines. In other words, agents can swap between machines more in cases of 4 (or 8) machines than in case of 2 machines only, for that ITSI is double in case of 4 (or 8) machines.

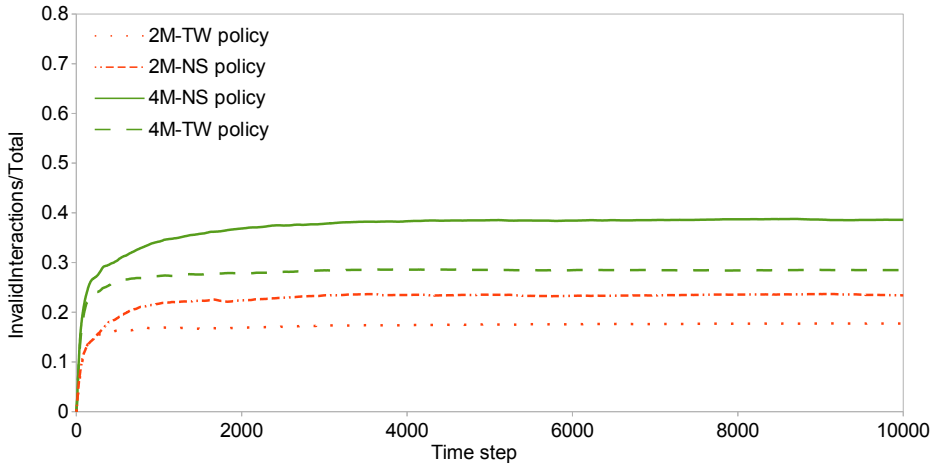

Figure 6.5: *Invalid-TS Interactions* between 2 and 4 machines in prey-predator model. The percentage of these interactions is higher in case of NS policy. In cases of 4 machines, we have a double percentage than 2 machines only. This is because, agents have 2 edge-zones only to swap in case of 2 machines and 4 edge-zones to swap in case of 4 machines.

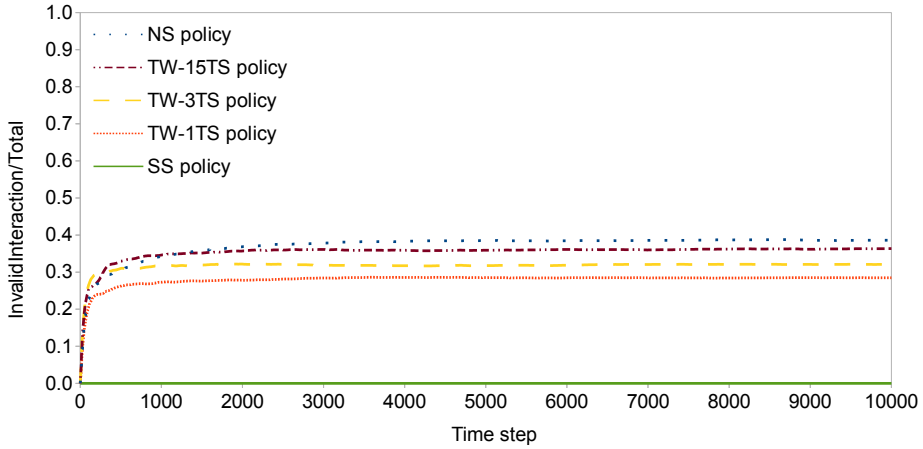

Figure 6.6: *Invalid-TS Interactions* between 4 machines in prey-predator model. The percentage of these interactions is increased when the W window permission of TW policy is increased.

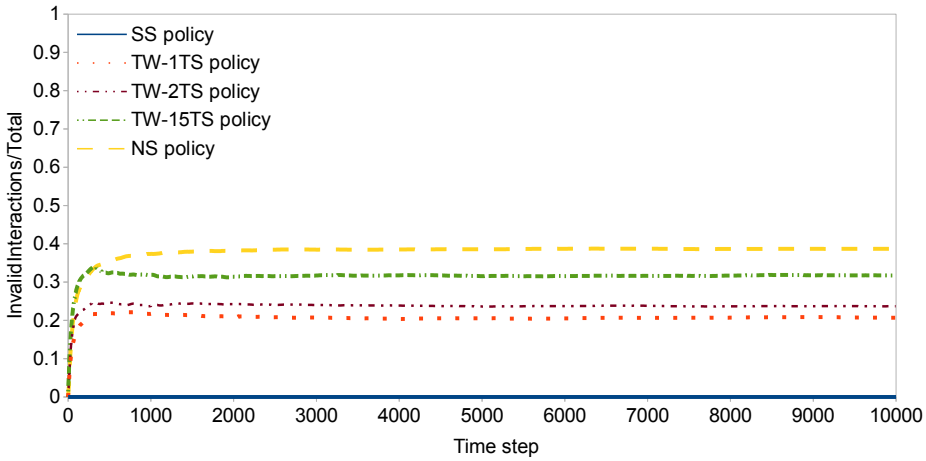

Figure 6.7: *Invalid-TS Interactions* between 8 machines in prey-predator model. In case of 8 or 4 machines (figure 6.6) we have the same percentage of these interactions. This is because, in both situations agents have 4 edge-zones to swap between machines.

Table 6.2: Capture the flag model: 20 flags per machine. Experiments show that for a W window permission of TW policy more than N time step the model could not be stable.

W window time	0	10	100..10000	100000	∞
CTF model	Stable	Stable	Stables	Unstable	Unstable

6.5.3 Instability of capture the flag model

As the previous experimentations, we have run capture the flag model with the same machines context for 2 million time steps 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 PP model, is stable in all synchronization policies. If we chose another initial configuration, like 20 flags per machine, we can get different results. Table 6.2 shows that, for different sizes of time window bigger than 100000 the model could not be stable, stable means here that the flags of all populations are still alive.

To study the instability of capture the flag model we have defined another initial configuration. It evaluates the degradation that can be induced by synchronization policies when computational loads are not the same on all machines. This configuration allows simulation to be run on three machines:

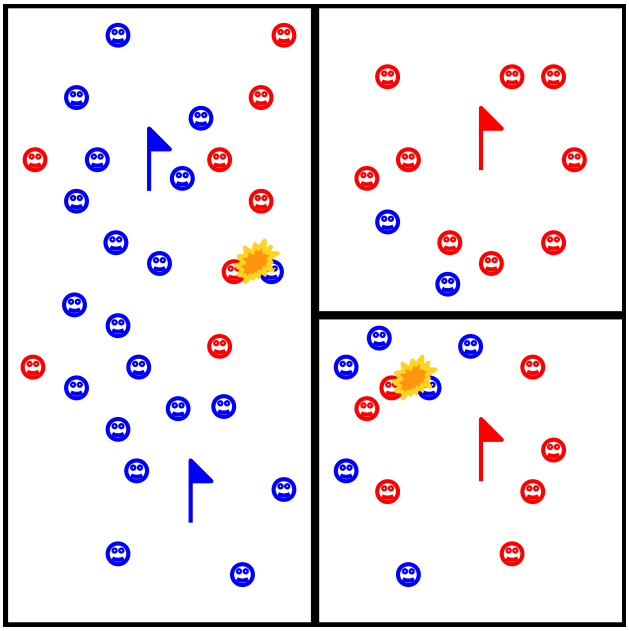

Figure 6.8: A bias configuration of capture the flag model. Two blue flags in one machine and two red flags in two different machines.

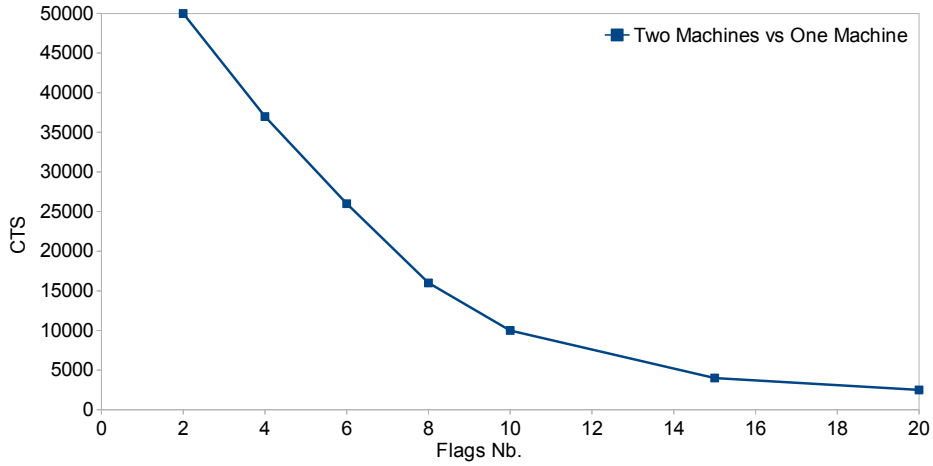

Figure 6.9: In the same bias configuration of capture the flag model, *Critical Time Step* (CTS) decreases if the number of flags increases.

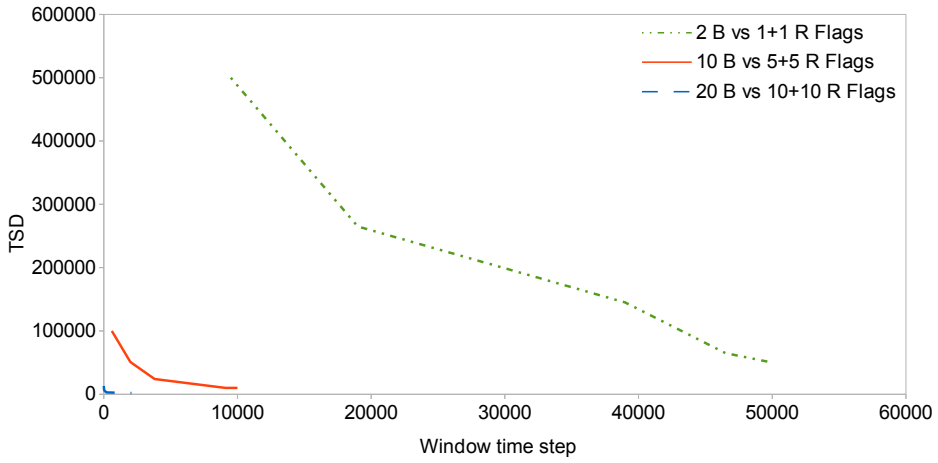

Figure 6.10: In the same bias configuration of capture the flag model, *TimeStep-to-Destroy* (TSD) decreases if the W window size permission increases.

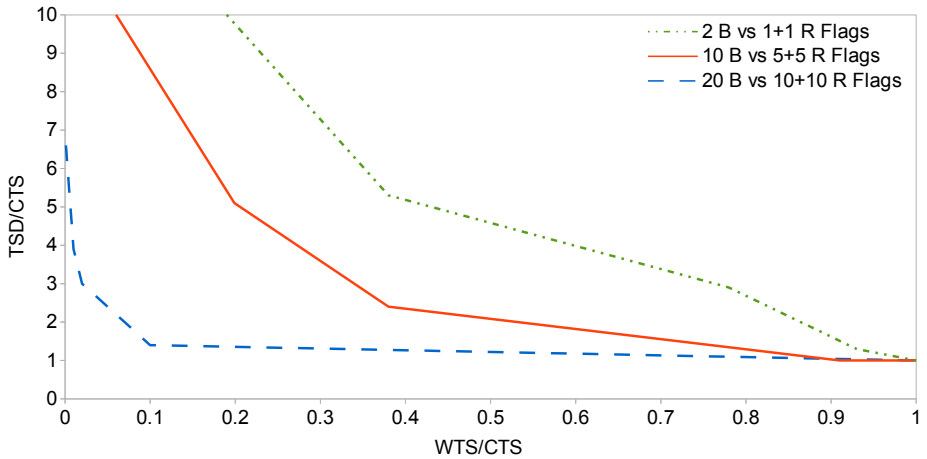

Figure 6.11: Same results of figure 6.10 after adjusting each curve to its CTS. That can help to measure different curves with a common scale.

the first one contains all flags from the first population (or team) and the two other machines contain only half flags of the second population in each (Figure 6.8). The aim is to have a more load on one machine than the others, and then it produces an unstable model. The second population (or team) should always win, because its attacker production should be higher as it is allocated on two machines. With this configuration, we ensure that the model is unstable. But, the convergence should be faster with more flexible synchronization policies, and fastest in case of NS policy.

If we define the *Critical Time Step* (CTS) as the necessary *time step* (TS) to completely destroy the model if *NS* policy has been used. In capture the flag model, if all flags of one population have disappeared, then the model is destroyed. Figure 6.9 shows the relation between the number of flags and the CTS, more flags mean less CTS to completely destroy the model. This CTS can depend on the initial configuration of the model, like the number of flags F and communication delays between machines.

As in figure 6.9, the CTS decreases by increasing the number of flags which have been used, $CTS = \frac{\alpha_1}{F}$, α_1 is a constant that depends on the initial configuration and it depends also on the machines which have been used during the simulation.

Now, if we run our simulation on the same configuration but with another policy which is TW policy, and with time window sizes from 0 to infinity. We can calculate the number of TS, that the simulation needs to stay in progress until it is completely destroyed with that time window. Then, we define the *TimeStep-to-Destroy* (TSD): the necessary TS to completely destroy the model if W window permission of TW policy has been used.

Figure 6.10 shows TSD test. Each configuration has a curve with different scale, but all are reduced by increasing the W window permission of TW policy. However, it is difficult to measure different initial configurations with different number of flags as each curve has not the same scale. For that, we normalize each curve to its CTS because each configuration has its CTS.

Figure 6.11 visualizes results after adjusting each measured curve, which are on different scales, to a common scale which is its CTS. This Figure 6.11 shows that the *TimeStep-to-Destroy* (TSD) decreases if the W time window increases, and according to the figure 6.11: $TSD/CTS = \frac{\alpha_2}{W/CTS}$, α_2 is constant. Then $TSD = \frac{\alpha_2 \times CTS^2}{W}$. Again, each curve has more flags, should be destroyed with smaller TS, we can replace $CTS = \frac{\alpha_1}{F}$. Then $TSD = \frac{\alpha_2 \times {\frac{\alpha_1}{F}}^2}{W}$. Then $TSD = \frac{\alpha_2 \times \alpha_1^2}{W \times F^2}$. Then $TSD = \frac{\alpha}{W \times F^2}$, $\alpha = \alpha_2 \times \alpha_1^2$ is constant. That means, TSD decreases if the number of flags is increased or W time window is increased too.

Figure 6.11 shows also that for all configurations the model stay stable for

a long time with a small W time window size. According to figure 6.11 and in all configurations, we can divide each curve into two mainly parts. First with W time window 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 W time window. The second part with W bigger than 30% of CTS, in this part the TSD is decreased slowly according to the W time window.

6.6 Summary

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 can be by splitting the environment into smaller parts and using a strong synchronization policy, but it implies a high cost in exchanging messages and the execution time. This chapter has explored a relaxation of this constraint to speed up the execution time and has identified applications where this relaxation do not degrade simulations outcome.

In this chapter, we have experimented the three synchronization policies in our distributed MAS platform: *strong synchronization* policy, *time window synchronization* policy and *no synchronization* policy (as detailed in chapters 3 & 4). We study the performance of several synchronization policies and their impact on the properties of situated applications. We evaluate simulations on two situated MAS applications: prey-predator and capture the flag.

Experimentations show that some applications, like prey-predator model, stay stable with any synchronization policy. Whereas in other models, like capture the flags model, 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 with relaxing synchronizations when a bias initial configuration is used.

To conclude, in large scale situated MAS simulations we can run many applications with high speed execution by relaxing the synchronizations and keeping the macroscopic behaviour in the same time. Even with critical applications as capture the flag model, the relaxing synchronization can be possible with a small number of time window (less than 30% of CTS as figure 6.11).

Part IV

Conclusion & Future works

Chapter 7

Conclusion

7.1 Thesis summary

The main goal of this thesis is to study strategies that can be applied to enable simulations of large scale situated multi-agents systems. This type of simulations relies on the modelling of interactions that happens between agents in order to reproduce the emergence of complex phenomena at the population scale. This approach allows to understand better the modelled phenomena, but has a cost in term of computation and memory. Thus, to be able to execute large scale simulations that involve millions of agents and interactions, it becomes necessary to distribute computations over a computer network. This thesis has studied several hypothesis to reach such scale and has led to the implementation of a distributed simulator to experiment these hypothesis.

To ensure performance gains when we execute such simulations in a distributed system, it is necessary to take into account specific concepts of multi-agent simulations, which are agents, environment and population dynamics. More specifically than classical ones in distributed systems, which are computation, data storage and communication costs. Our work presents several criteria that should be considered to ensure performance gains when distributing such simulations over a computer network.

Our work relies on different aspects of two large domains: multi-agent system (MAS) and distributed system. From the MAS point of view this work has allowed us to provide a large scale multi-agent simulator:

- that is autonomously, because agents can interact and move between different machines during the simulation.
- that is dynamic, because agents can be added dynamically during the simulation.

- and it is one block simulation, because each agent can see all other agents in its perception as in one simulation.

From the distributed system point of view this work has allowed us to build a distributed large scale simulation:

- that is initially load balanced, because there is an ability to configure the simulation at the beginning for each application to balance initially the load.

- that is high-scalable because there is an ability to scale it into high number of distributed machines.

- and flexible, because there is an ability to relax the synchronization and optimize the performance.

In this thesis, we describe our view of MAS concepts and the importance of different distribution approaches. These different distributions can give us better performance in some applications than others. We propose two main distribution types: *agents distribution* and *environment distribution*. *Agents distribution* depends on non spatial criteria to distribute agents between different machines. Whereas, *environment distribution* relies precisely on spatial properties to divide the computation.

In some applications, the environment is more important than agents and can play a major role in the simulation. Thus, *environment distribution* gives us better performance in execution time than other approaches. Whereas in other applications, agents can be the most important aspect, and a distribution related to agents (or *agents distribution*) can give us better performance in execution time.

Distributing simulations over a computer network raises some problems in time management between different machines, which can be handled through a strong synchronization between machines. However, the goal of a large scale MAS simulation is not to observe millions of individual interactions or microscopic level, but to observe properties at macroscopic level. In other words, if we have a large scale situated-MAS simulation, and some agents fail to interact, that should not affect the global behaviour of the system. Using a safe approach (as *strong synchronization* policy) in large scale MAS simulations can be possible, but it implies a high cost in exchanged messages and then in execution time too. This work has explored a relaxation of this constraint to speed up execution time and has identified different applications where this flexibility do not degrade simulations outcome. In addition

to different distribution types, our simulator has three main synchronization policies: *strong synchronization*, *time window synchronization* and *no synchronization* policies.

In this thesis, we have detailed machine units for different distribution types. The main platform layers were discussed and different simulation states were introduced. We have studied different communication protocols for different three main synchronization policies. We have explained the time step between two machines to clarify the work protocol of our simulator. We have implemented the proposed protocol to allow machines to communicate between each other for building a large scale distributed MAS simulation. The main technical problem to be resolved was interactions between two agents or more from different machines, which has been solved with an agreement protocol.

In the experiments, we have evaluated the two types of distributions with two different categories of applications, which are the flocking of birds and prey-predator models. In case of environment distribution, the simulated environment is divided into different partitions on different machines (one partition on one machine). During the simulation, each machine communicates with its neighbouring machines to collect the needed information about the neighbouring areas or *ghost areas*. In case of agents distribution, the simulated environment is the same for all machines and agents should be divided into different lists between different machines. During the simulation, each machine should communicate with others to collect the needed information about other agents or *ghost agents* which exist on other machines.

Experimental results show that the proposed distributions approaches have better performances in some models than others. For example, prey-predator model has better performance in execution time than flocking model when we distribute it by environment distribution approach. Whereas, agents distribution approach is better for flocking model.

Other experiments have been done on three synchronization policies for distributed MAS simulations: *strong synchronization*, *time window synchronization* and *no synchronization* policies. Experimentations show that some applications, like prey-predator model, stay stable with any synchronization policy. Whereas in other models, like capture the flags, it can be strongly affected by changing these policies. We have studied how interactions can be 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.

The main thesis contributions are:

- the exploration of the main properties of *multi-agent systems* and *dis-*

tributed systems domains,

- the study of different distribution approaches for large scale MAS simulations,
- the exploration of different synchronization policies,
- and the experimentation of large scale situated applications with different configurations.

In our point of view, we have achieved our objectives because:

1. we successfully built an efficient distributed simulator that was specially designed for large scale situated agent-based simulations,
2. our simulator can be configured and optimized to run in any peer-to-peer computer network,
3. agent models that are written in JAVA can be integrated easily in our simulator.

7.2 Future works

Even if our actual simulator implementation provides an interesting framework, there are still open issues that should be addressed:

- we can investigate more the two different distribution approaches and we can in the future merge or mix the two approaches in one hybrid approach that can be a good solution for other categories of applications. Moreover, we can experiment other applications with different synchronization policies to illustrate their impacts and see if the results presented in this work are suitable for them. We can explore more the external interactions and the agreement protocol that can help non acting agents to be re-asked for another valid interactions. We can also increase the scalability of our framework. We currently reach near 5 million of agents and we plan to distribute 10 million of agents in less than one minute for one time step.
- we can explore a real large scale application: as 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.

- we will try to find other platforms that have similar implementation in distribution approaches or synchronization policies to make a logical comparison between our framework with others by re-implementing the same applications to other platforms and then make the logical comparison.

- fault tolerance is one important feature in distributed systems that enables these systems from operating normally in case of failure. Many techniques can be used for fault tolerance, one of them is the replication of data or agents in case of multi-agent systems. In large scale situations this approach can have some limitations and need some specifications. This can be one of our interesting domains.

- the visualization of large scale simulation can be built with two main ways: 1) a local visualization for each machine with the local agents and the local part of the environment and 2) a global visualization of all agents and whole environment. The first way is the easy way as each machine has the information that it needs to draw its local part of the simulation. The second way can be done easily by collect all necessary information to one machine and draw the whole MAS simulation, but this solution in large scale simulation can have some limitations. We can in the future optimize the global visualization of large scale simulation by collecting images from each machine with a user needed resolution and then we can attach all images together in one global view of the whole simulation.

Annexe A

French chapter

A.1 Introduction

Les systèmes multi-agents sont constitués d'entités autonomes qui interagissent avec leur environnement pour résoudre un objectif collectif [Russell et al., 1996]. Les domaines d'application de ces systèmes vont de la résolution distribuée de problèmes à la simulation de phénomènes complexes. Ce type de simulation offre une granularité fine permettant d'exprimer les comportements à un niveau microscopique, c'est-à-dire individuel, et d'observer des phénomènes émergents au niveau macroscopique, c'est-à-dire de l'ensemble de la population. Néanmoins, lorsque le nombre d'agents et d'interactions augmentent, les ressources nécessaires en terme de puissance de calcul ou de capacité de stockage deviennent un facteur limitant.

Nous restreignons cette étude à la simulation d'agents situés dans un espace euclidien et dont les interactions ou les échanges de message ne peuvent se produire que lorsque deux agents sont suffisamment proches. Ce contexte correspond à la très grande majorité des applications réalisées dans le cadre de simulation d'agents situés comme la modélisation de traffic routier, d'écosystèmes humains ou biologiques ou encore de jeux vidéo.

Si l'on souhaite modéliser des systèmes contenant plusieurs centaine de milliers ou de millions d'agents, une puissance de calcul et de stockage importante devient nécessaire. Pour atteindre de telles simulations large échelle, distribuer le simulateur sur un réseau de machines est nécessaire, mais induit des problématiques de répartition de charge, de gestion du temps et de synchronisation entre les machines et de tolérance aux pannes.

En plus des problèmes de coûts de communication classiques dans le contexte d'applications distribuées, il faut prendre en compte des aspects plus spécifiques liés au fait que les agents sont situés dans un environne-

ment. En effet, les dynamiques de déplacement des agents induites par leurs comportement affecte la répartition de la charge de calcul sur le réseau. Ces problématiques font l'objet de recherche actives, particulièrement sur la question du placement des agents sur un réseau de machines [Miyata and Ishida, 2008], notamment lorsque les agents sont mobiles [Motshegwa and Schroeder, 2004]. Dans ce contexte, la gestion du temps et de la synchronisation sont également des problèmes importants. Plusieurs modèles de synchronisation ont été proposé, avec une gestion du temps sur les différentes machines [Scerri et al., 2010] [Siebert et al., 2010], notamment dans le contexte de simulations à événements discrets [Jefferson, 1985]. Cependant, bien que certains de ces travaux permettent d'assouplir les mécanismes de synchronisation, ils n'explorent pas le compromis entre reproductibilité et gain de performance.

Le premier aspect de notre travail se concentre sur deux types de répartition de la charge de calcul : la première basée sur une répartition des agents en fonction de leur proximité, la seconde répartit les agents indépendamment de leur positionnement dans un objectif d'équilibrage de charge. Nous évaluons les performances de ces répartitions en les confrontant à des applications dont les dynamiques de déplacement sont très différentes, ce qui nous permet d'identifier plusieurs critères devant être pris en compte pour garantir des gains de performance lors de la distribution de simulations d'agents situés.

Le second aspect de notre travail étudie la problématique de la synchronisation des machines. En effet, à notre connaissance, tous les simulateurs existants fonctionnent sur la base d'une synchronisation forte entre les machines, ce qui garantit la causalité temporelle et la cohérence des calculs. Dans cette thèse, nous remettons en cause cette hypothèse en étudiant la relaxation de la contrainte de synchronisation. Le fait d'autoriser la progression d'agents dans différentes temporalités induit des interaction incohérentes, c'est-à-dire se produisant entre des agents n'appartenant pas au même pas de temps. La question qui se pose est alors de savoir si ces incohérences induisent une perte du phénomène émergent de la simulation et si ce n'est pas le cas, d'évaluer le gain en terme de performance du relâchement de cette contrainte. Il est d'ailleurs envisageable que pour certaines applications, des erreurs ou échecs d'interactions entre deux agents ne soient pas critiques pour le résultat global de la simulation. Afin d'étudier cette problématique, nous proposons deux politiques de synchronisation : la synchronisation forte classique et une forme de synchronisation reposant sur une fenêtre de temps bornée entre la machine la plus lente et la machine la plus rapide. Des applications de natures différentes sont exécutées avec ces différents mécanismes de synchronisation. Nous étudions dans cette thèse leur coût en performance ainsi que leur impact sur l'émergence des propriétés macroscopiques des simulations. Nous

TABLE A.1 – Principales plateformes de simulation distribuée.

Platforme	Scalabilité		
	Agents	Nœuds	Application
Repast	68 billions	HPC-32000 cores	Triangles (simple)
DMASON	10 millions	64	Boids (simple)
AglobeX	6500	6 (22 cores)	Simulation de avions (simple)
GOLEM	5000	50	Monde de colis (complex)
FLAMEGPU	11000	GPU	Foule de piétons
Megaffic	10 millions	16 (192 cores)	Simulation de embouteillage

nous intéressons particulièrement au seuil critique d'interactions temporellement invalides qui entraînent un biais dans le résultat de la simulation.

A.2 Etat de l'art

Il existe déjà plusieurs platformes dans le domaine des simulations réparties : `REPAST` [North et al., 2013], `FLAME` [Kiran et al., 2010], `FLAME-GPU` [Karmakharm et al., 2010], `AglobeX` [Šišlák et al., 2009] et `D-MASON` [Cordasco et al., 2011] [Cosenza et al., 2011]. Néanmoins, toutes ces plateformes reposent sur une mechanism de destribution seulment et une politique de synchronisation (forte). Selon les implémentations, cette synchronisation est gérée au travers d'un intergiciel, d'un machine virtuelle dédiée ou au travers de mémoire partagée entre machines virtuelles. Certaines utilisent plutôt une approche orientée maître/esclave avec synchronisation forte au niveau du maître. Nous ne pouvions donc pas aisément utiliser l'une d'entre elles pour définir et étudier d'autres formes de synchronisation et distribution. La table A.1 donne une synthèse des principales caractéristiques de ces plateformes, particulièrement en ce qui concerne la montée en charge.

A.3 Distribuer l'environnement ou les agents

Pour simuler un grand nombre d'agents, nous proposons de distribuer le calcul sur un réseau pour maximiser l'évaluation parallèle des agents. Cette répartition peut s'effectuer de plusieurs manière, mais induit nécessairement des

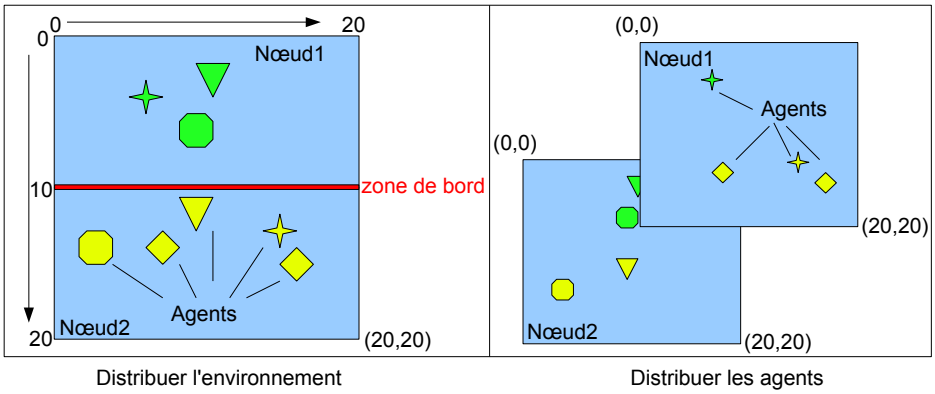

FIGURE A.1 – Deux types de distribution : l'environnement (gauche) ou les agents (droit).

coûts de communication et des problématiques de synchronisation. Nous proposons dans cette section trois types de répartition possible qui nous semblent pertinents pour la simulation d'agents situés : la répartition de l'environnement, la répartition des agents ou une solution hybride exploitant ces deux types de répartition.

A.3.1 Distribuer les agents

Cette première approche consiste à répartir équitablement les agents sur les différents nœuds comme l'illustre la figure A.1. Chaque nœuds a la responsabilité d'un ensemble d'agents et les gère sur l'ensemble de l'environnement. Comme des interactions peuvent se produire entre des agents répartis sur des nœuds différents, un mécanisme de diffusion de l'état des différents nœuds doit être instauré pour que le calcul des interactions puisse être réalisé au sein de chaque nœuds. Ce coût élevé de transmission des états devra être compensé par un volume de calcul suffisant au niveau de l'évaluation des comportements des agents.

A.3.2 Distribuer l'environnement

Cette deuxième approche consiste à distribuer l'environnement, ce qui revient à découper ce dernier en différentes sections, chacune d'entre elles étant affectée à un nœuds. Chaque nœuds a alors la responsabilité d'évaluer les agents se trouvant dans cette portion de l'environnement comme l'illustre la figure A.1.

La difficulté avec cette approche concerne la gestion des interactions des

agents se trouvant aux frontières des sections de l'environnement ainsi que la mobilité des agents entre les nœuds. En effet, lorsqu'un agent sort de l'espace géré par le nœuds l'hébergeant, il doit être transféré sur un autre nœuds.

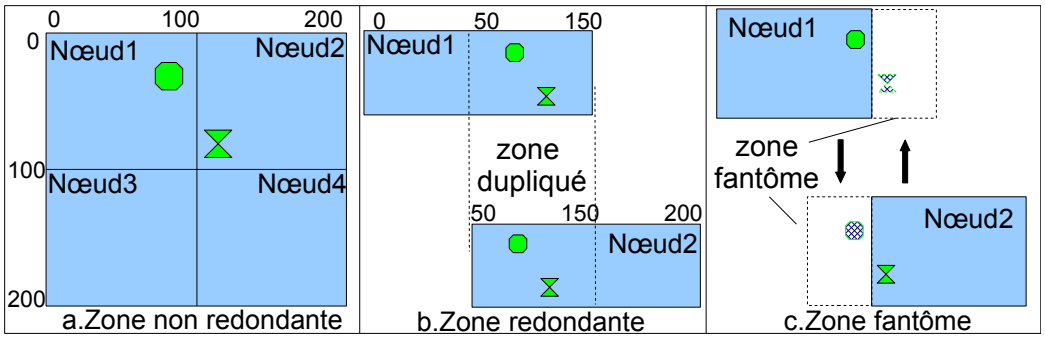

FIGURE A.2 – Zone redondante, zone non redondante et zone fantôme.

Plusieurs approches sont envisageables pour effectuer les échanges d'informations entre les nœuds détenant des sections adjacentes de l'environnement initial :

1. une ***zone redondante*** est une portion de l'environnement qui est gérée simultanément par deux nœuds. Cette approche nécessite une synchronisation forte des deux nœuds afin que chaque modification effectuée par l'un des nœuds dans cette zone soit reproduit dans l'autre nœuds (figure A.2). Il est délicat d'implémenter cette approche sans un coût prohibitif au niveau des communications pour maintenir la cohérence de l'ensemble des zones partagées.

2. une ***zone non redondante*** est une portion de l'environnement gérée exclusivement par un nœuds. Chaque agent ne peut ainsi appartenir qu'à l'un des nœuds, cependant son comportement peut être dépendant de perceptions dépassant le nœuds courant et nécessiter l'obtention d'informations de nœuds adjacents. Dans le cas d'interactions conflictuelles, cela implique qu'un message soit envoyé pour la phase de perception, puis un message pour la vérification de la possibilité de réalisation de l'interaction.

3. une ***zone fantôme*** correspond à une copie d'une partie d'une zone adjacente. Ce concept est similaire à la notion de *donnée fantôme* introduite par Isenburg dans [Isenburg et al., 2010]. Au début d'un pas de simulation, chaque nœuds envoie une copie des agents se trouvant à proximité de la frontière de la portion d'environnement qu'il gère.

Après cette étape, chaque nœuds dispose ainsi d'une zone de perception étendue permettant aux agents proches des frontières de détecter les agents situés sur d'autres nœuds et de les intégrer ainsi dans leur prise de décision.

La répartition de l'environnement est sûrement plus pertinente pour les simulations où la répartition des agents est homogène car dans ce cas, la charge de calcul se trouve spontanément répartie sur les nœuds du réseau.

A.4 Temps et synchronisation

La notion de temps est souvent définie comme un continuum non spatial dans lequel une succession d'événements se produisent de manière irréversible [Gold, 2003]. Dans une simulation informatique, plusieurs notions de temps se superposent : le temps de l'utilisateur (ie. le temps *réel*) et le temps de la simulation, qui est modélisé par une succession d'instants de courte durée permettant de réaliser le déroulement de l'évolution de la simulation (ie. souvent appelé le déroulement *pas de temps*). Cette notion de temps a été définie plus formellement par Lamport [Lamport, 1978], comme une horloge logique définissant un ordre partiel sur les événements qui se produisent. Une extension appelée *Logic Virtual Time*, LVT, a été introduite plus tard par Jefferson [Jefferson, 1985] dans le cadre des recherches sur les simulateurs à événements discrets distribués (*time warp system*).

Une implémentation classique de la notion de temps dans les simulations multi-agents consiste à définir un tour de parole pour récupérer les actions que les agents souhaitent effectuer et à les appliquer ensuite. Ce tour de parole correspond à une étape atomique de progression temporelle dans l'évolution de la simulation. Nous appellerons ce tour de parole, un *pas de temps* (PT) dans la suite de ce chapitre. Dans une simulation centralisée, il n'y a qu'un pas de temps qui organise les interactions entre l'ensemble des agents dans le laps de temps fixé. Par contre, dans une simulation répartie, chaque machine dispose de son propre pas de temps et il devient alors nécessaire de tous les synchroniser[Scerri et al., 2010] [Siebert et al., 2010]. Pour garantir les relations de causalité, une synchronisation forte de l'ensemble des pas de temps est possible mais induit un coût important.

Deux approches ont été proposées dans le contexte des systèmes distribués pour gérer cette synchronisation : les approches conservatives ou synchrones, impliquant une progression simultanée de tous les pas de temps, et les approches optimistes ou asynchrones[Logan and Theodoropoulos, 2001] [Fujimoto, 2000], qui permettent à certains pas de temps de progresser plus

rapidement que d'autres, mais en instituant un mécanisme de retour en arrière (*rollback*) en cas d'incohérence temporelle [Gupta et al., 2007]). Les approches synchrones constituent la principale approche utilisée dans les plateformes actuelles de simulation multi-agents distribuées. Les approches asynchrones permettent de maximiser les performances en augmentant le parallélisme, mais leur apport s'écroule lorsque de trop nombreux *roolbacks* se produisent.

Dans notre travail [Rihawi et al., 2013a], nous étudions deux types de synchronisation : une politique conservative, appelée *synchronisation forte*, et une politique autorisant l'apparition d'incohérences temporelles, appelée *synchronisation flexible*. Nous n'avons pas évalué d'approche optimiste car dans un système multi-agent contenant un très grand nombre d'agents la gestion de la journalisation de leur état est difficilement réalisable.

A.4.1 Synchronisation Forte

Dans la politique de synchronisation forte (SF), l'ensemble des pas de temps des différentes machines sont synchronisés à l'issue de chaque tour de parole. Cela garantit que tous les agents du système exécutent le même nombre d'actions et qu'ils évoluent dans le même pas de temps. Il n'y a donc pas d'incohérence temporelle possible, c'est-à-dire d'interaction se produisant entre des agents n'appartenant pas au même pas de temps. L'intérêt de cette approche est la garantir la causalité, mais cela implique des coûts de communication important et les performances du système sont limitées par la machine la plus lente du réseau.

A.4.2 Synchronisation avec une Fenêtre Temporelle

Avec la politique de synchronisation reposant sur une fenêtre temporelle (SFT), un écart fixe entre le pas de temps de la machine la plus chargée et celui de la machine la moins chargée est autorisé. Ceci implique que durant la simulation, différentes machines peuvent être dans des pas de temps différents.

Ainsi, il est possible que des interactions entre des agents n'appartenant pas au même pas de temps se produisent, ce qui introduit des incohérences temporelles. C'est l'hypothèse forte que nous faisons dans ce travail et que nous évaluons par les différentes expérimentations de la section A.9.

Il est important de remarquer que la notion de machine la plus rapide ou la plus lente n'est pas absolue et qu'elle évolue au cours du temps en fonction de la charge provoquée par les dynamiques de déplacement ou de génération et destruction d'agents. Il est ainsi possible pour une fenêtre suffisamment

large de ne pas atteindre l'écart maximal autorisé par la fenêtre temporelle, c'est dans ce type de situation que l'apport d'une fenêtre de temps est le plus sensible.

Finalement, on peut remarquer que si l'on fixe une taille de fenêtre infinie, on se ramène à un système dans lequel il n'y a plus aucune synchronisation entre les machines. Cette situation limite au maximum les coûts de communication et maximise les performances mais n'est évidemment pas exploitable pour toutes les applications car le nombre d'interactions invalides explosent.

Dans la suite de ce chapitre, la question à laquelle nous donnons des réponses est la suivante : dans quelle mesure les contraintes de synchronisation peuvent elles être affaiblies sans impacter significativement les propriétés observées au niveau global de la simulation.

A.5 Description de la plateforme

Pour évaluer les deux types de répartition décrits dans la section précédente, nous avons développé un simulateur pouvant s'exécuter selon ces deux modes. Le protocole de communication choisi est un graphe totalement connecté permettant à chaque nœuds de communiquer directement avec les autres nœuds présents sur le réseau. Lors de chaque pas de temps, chaque nœuds effectue une résolution locale et les informations sont ensuite transmises à un client particulier pour la visualisation globale (figures A.3 et A.4).

A.5.1 Processus de simulation

Notre simulateur repose sur l'interaction de nœuds de calcul, chaque nœuds étant affecté à une machine du réseau. En fonction du mode de répartition, chaque nœuds gère soit une portion de l'environnement avec les agents qu'elle contient (figure A.3), soit un sous-ensemble des agents situés en n'importe quel point de l'environnement (figure A.4). Le simulateur est structuré en trois niveaux : une couche de communication qui établit les connections entre les nœuds et effectue les envois/réceptions de messages, une couche de simulation qui gère le tour de parole au niveau des agents et effectue les transferts d'information en amont et aval d'un pas de simulation et d'une couche applicative qui définit les comportements des agents et les interactions qu'ils peuvent réaliser.

Initialisation de la simulation

Si l'utilisateur choisit la répartition de l'environnement, la première étape consiste à diviser l'environnement en plusieurs portions et à les répartir sur

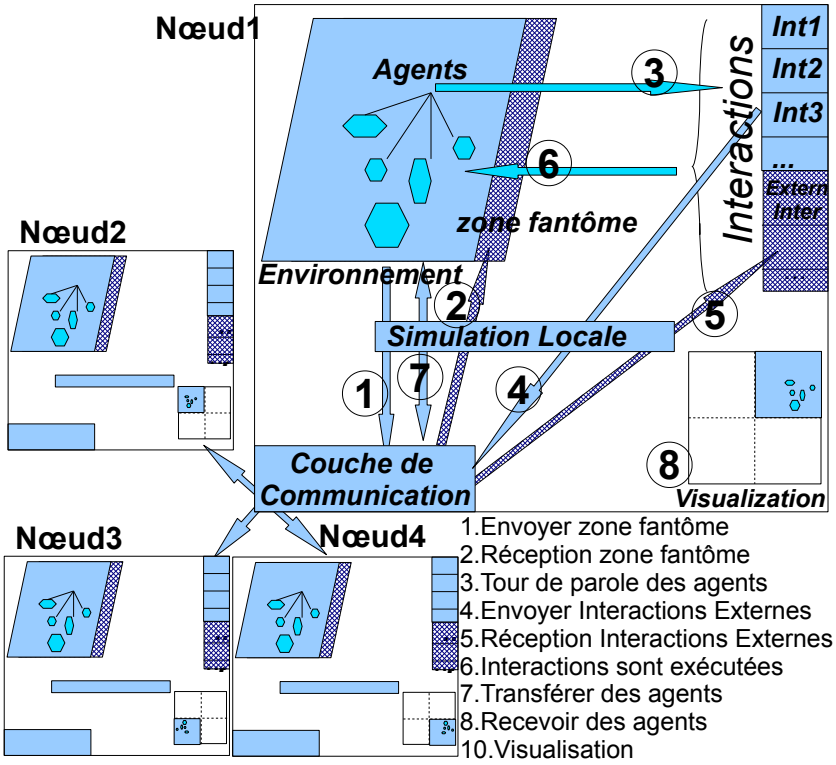

FIGURE A.3 – Protocole suivi par le simulateur en mode *environnement distribué*.

les différents nœuds. Ainsi, chaque nœuds possède une portion de l'environnement et peut communiquer avec les autres nœuds pour échanger les zones fantômes et effectuer les migrations d'agents. Un fichier de configuration définit de manière statique la partition de l'environnement et l'affectation des différentes portions sur les différents nœuds, ainsi que les agents à instancier au démarrage de la simulation. Ensuite, chaque nœuds initialise la couche de communication et établit les connections avec l'ensemble des autres nœuds. Dans le cas de la répartition des agents, l'utilisateur ne doit définir dans le fichier de configuration que la répartition des agents sur les différents nœuds, l'environnement global étant partagé par l'ensemble des nœuds comme l'illustre la figure A.1.

Exécution de la simulation

Après la phase d'initialisation, chaque nœuds récupère les informations des autres nœuds, ce qui correspond à l'échange des zones fantômes adjacentes pour la répartition de l'environnement, et aux agents des autres nœuds pour

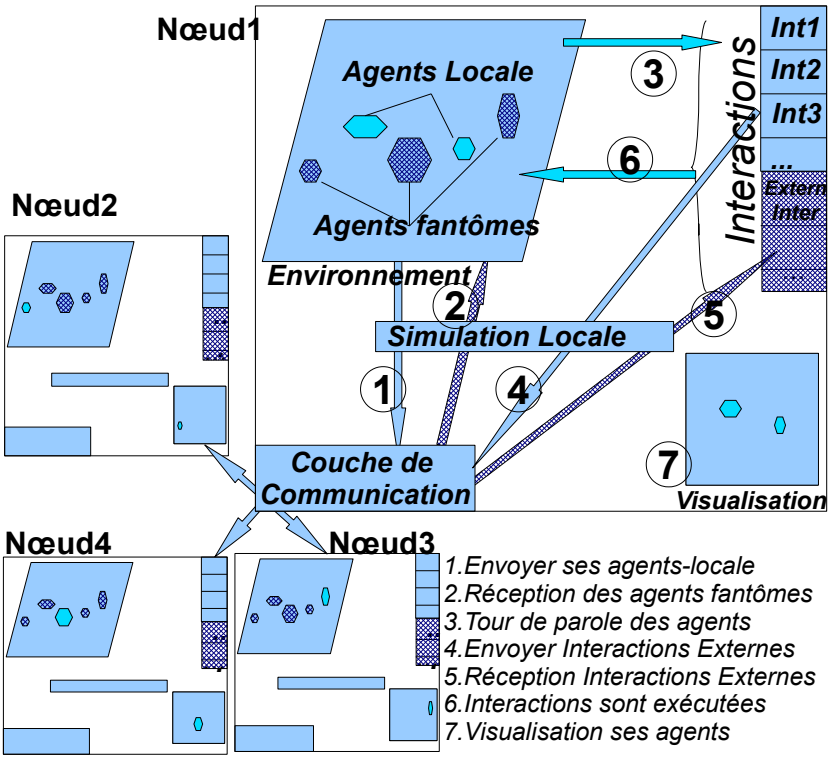

FIGURE A.4 – Protocole suivi par le simulateur en mode *agents distribués*.

la répartition des agents. Puis, le simulateur effectue le tour de parole des agents qu'il gère. Cependant, dans le cas de la répartition de l'environnement, si des agents effectuent un déplacement les positionnant en dehors de la portion d'environnement géré par leur nœuds, alors ils sont transférés aux nœuds concernés. Par contre, pour la répartition des agents, une telle mobilité n'est pas possible et les agents ne peuvent quitter le nœuds sur lequel ils ont été créé, puisque tous les nœuds gèrent le même environnement.

A.5.2 Dynamique d'un pas de simulation

Pour expliquer la dynamique d'un pas de simulation, le schéma A.5 détaille le protocole d'interaction se produisant entre deux nœuds. Supposons que nous ayons deux nœuds sur lesquels est réparti l'environnement en deux sections égales. Au début du pas de simulation, chaque machine transmet les zones fantômes aux nœuds adjacents et réciproquement réceptionne les zones fantômes de ses voisins. Ensuite, chaque nœuds enclenche le tour de parole au niveau des agents en distinguant les interactions internes au nœuds (ie. n'impliquant que des agents présent sur ce nœuds) des interactions externes (ie.

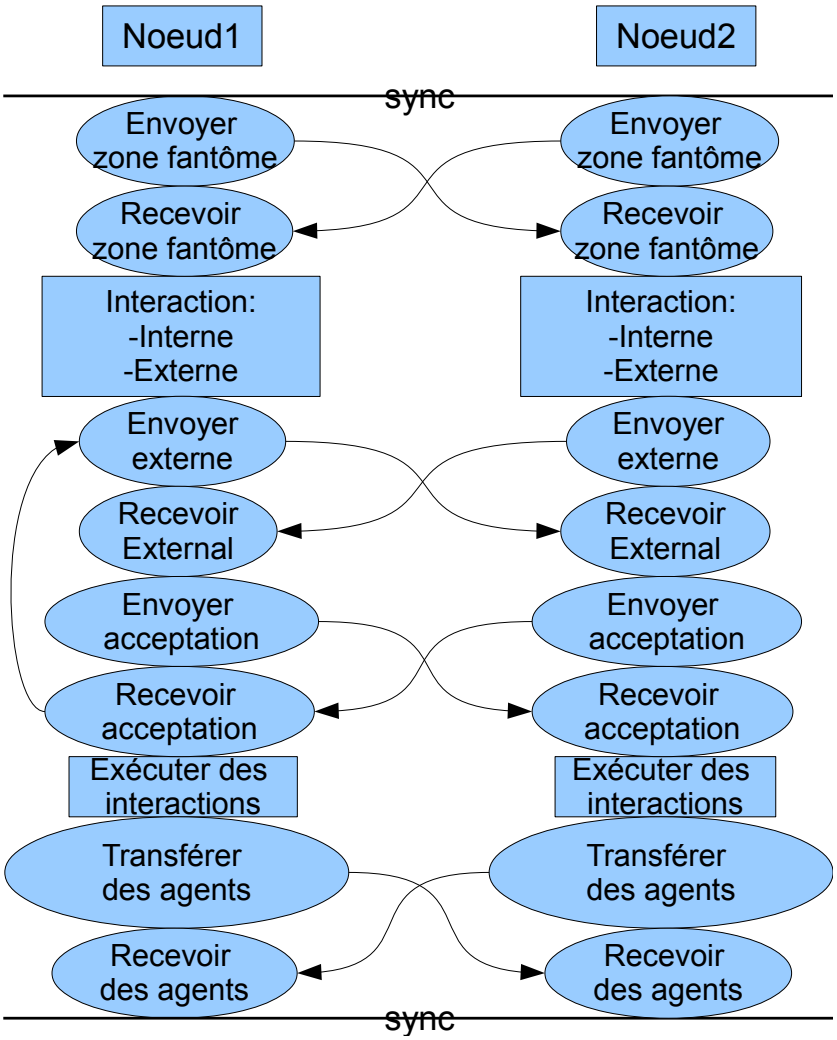

FIGURE A.5 – Pas de simulation avec 2 machines dans le cas de la distribution de l'environnement.

interactions impliquant un ou plusieurs agents situés sur un ou des nœuds adjacents). Les interactions externes sont transmises aux nœuds concernés et le nœuds courant attend aussi les interactions externes des autres nœuds. L'étape suivante consiste à accepter ou refuser (ce qui peut impliquer plusieurs échanges de messages avant de converger) les interactions externes des autres nœuds, avant d'appliquer l'ensemble des interactions. Les interactions étant évaluées, le simulateur détermine les agents se trouvant en dehors de la portion d'environnement qu'il gère et transmet ces agents aux autres nœuds. Finalement, les nœuds se synchronisent tous avant de pouvoir passer au pas de temps suivant. Pour la répartition des agents, les étapes sont quasiment

identiques, si ce n'est que la zone fantôme correspond dans ce cas à l'ensemble de l'environnement et qu'il n'y a aucun transfert d'agent entre les nœuds.

A.6 Plateforme d'évaluation

Il existe déjà plusieurs platformes dans le domaine des simulations réparties (table A.1). Néanmoins, toutes ces plateformes reposent sur une synchronisation forte. Selon les implémentations, cette synchronisation est gérée au travers d'un intergiciel, d'un machine virtuelle dédiée ou au travers de mémoire partagée entre machines virtuelles. Certaines utilisent plutôt une approche orientée maître/esclave avec synchronisation forte au niveau du maître. Nous ne pouvions donc pas aisément utiliser l'une d'entre elles pour définir et étudier d'autres formes de synchronisation.

Pour évaluer les politiques de synchronisation forte et flexible, nous avons développé un simulateur distribué qui repose sur la répartition de parcelles d'environnement ou des agents sur les machines d'un réseau. Ainsi, chaque nœuds est constitué d'une machine gérant un sous-ensemble de l'environnement avec les agents s'y trouvant. L'ensemble des nœuds sont totalement connectés. Le simulateur peut s'exécuter selon trois modes : en synchronisation forte, avec une fenêtre de temps finie ou avec une fenêtre infinie (c'est-à-dire sans aucune synchronisation entre les différents nœuds).

Les figures A.3 & A.4 illustre une simulation répartie sur 4 nœuds et schématise l'architecture de la plateforme. Chaque nœuds de calcul est constitué d'une couche de communication, d'une simulation locale et d'un environnement partiel avec ses agents. La simulation locale est l'élément principal du nœuds et a pour charge de gérer l'ensemble de la dynamique de simulation : les interactions entre les agents, l'envoi des informations concernant les zones situées à la frontière de différents nœuds ainsi que la gestion de la visualisation. La couche de communication gère les connections réseaux avec les autres nœuds, permet les échanges de messages et transmet aussi les informations liées au pas de temps courant du nœuds.

Lors de chaque pas de temps, les étapes suivantes sont effectuées :

- le nœuds envoie les informations concernant ses agents se trouvant à proximité des frontières de son environnement aux nœuds correspondants,
- le nœuds attend la réception des informations transmises par les autres nœuds concernant leurs agents proches des frontières,
- le nœuds effectue le tour de parole au niveau des agents qu'il gère,

TABLE A.2 – Temps d'exécution de proies prédateurs sur 4 nœuds.

Pas de temps	2000	4000	6000	8000	10000
Synchronisation forte	1h	2h30	3h30	4h45	6h
Sans synchronisation	50mn	1h20	2h	2h45	3h20

- les interactions faisant intervenir des agents se trouvant sur d'autres nœuds sont transmises aux nœuds concernés,
- le nœuds gère les interactions externes des autres nœuds,
- les interactions sont exécutées et les migrations d'agents sont effectuées,
- finalement, chaque nœuds met à jour son affichage et se synchronise avec les autres machines avant de pouvoir passer au prochain pas de temps.

Les différentes politiques de synchronisation partagent la majorité des étapes décrites ci-dessus avec de légères différences. En synchronisation forte, tous les envois de messages entre nœuds reposent sur des accusés de réception. Particulièrement lors de l'étape 7 qui garantit la synchronisation de tous les nœuds avant d'autoriser la transition au pas de temps suivant. Pour la politique exploitant les fenêtres de temps, tant que le nœuds n'est pas trop en avance par rapport au nœuds le plus lent, il peut passer directement au pas de temps suivant. Si ce n'est pas le cas, le nœuds se met en attente de la progression de la machine la plus lente. Par contre, les interactions entre agents situés sur deux machines différentes n'ont plus qu'une probabilité très faible de se produire (point détaillé dans les expérimentations). Avec la politique utilisant une fenêtre infinie, c'est-à-dire lorsqu'il n'y a plus de synchronisation, les nœuds progressent indépendamment et quasiment toutes les interactions sont temporellement incohérentes (ie. se produisent entre des agents appartenant à des pas de temps différents).

Pour tester la montée en charge de notre environnement d'évaluation, nous avons développé une application illustrant un comportement de *flocking* similaire à celui de Reynolds [Reynolds, 1987] et l'avons réparti sur 50 nœuds, chacun prenant en charge 100000 agents au début de la simulation (figure A.6).

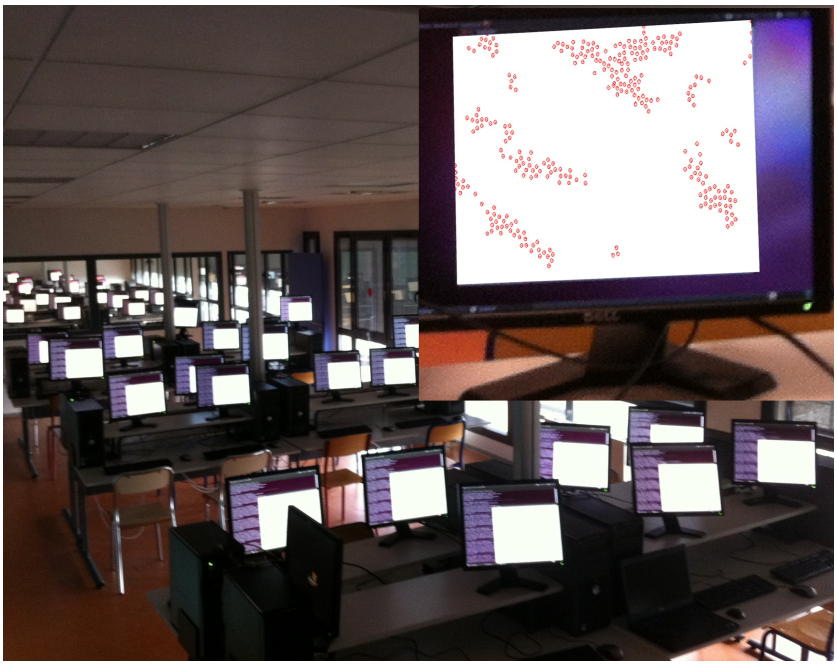

FIGURE A.6 – Simulation du phénomène de *flocking* avec 5 millions d'agents sur 50 nœuds.

A.7 Évaluation de l'efficacité des deux types de répartition

L'efficacité des différents types de répartition proposés précédemment est nécessairement dépendante de la nature des simulations effectuées. En effet, le fait de se placer dans le cadre des simulations d'agents situés induit des interrogations par rapport à l'évolution des volumes de population, mais aussi et surtout de leurs dynamique de déplacement. Dans cette section, nous étudions les deux types de répartition sur deux cas classiques de simulations d'agents situés : un modèle de *flocking* et un modèle *proies prédateurs*.

A.7.1 Un modèle de *flocking*

Le modèle de comportement de *flocking* proposé par [Reynolds, 1987] est constitué d'un ensemble d'oiseaux qui volent en groupe. Chaque oiseau détermine sa direction en fonction des oiseaux dans son champs de perception. Du point de vue de l'environnement, la répartition des agents, même si elle est homogène initialement, devient rapidement hétérogène avec la constitution de différents groupements d'agents se déplaçant en nuée. Cette dynamique

de déplacement provoque une forte variation de charge entre les nœuds dans le cas de la répartition de l'environnement, alors que le nombre d'agents par nœuds est identique dans l'autre type de répartition.

A.7.2 Un modèle *proies prédateurs*

Ce modèle est une application classique de simulation de co-évolution de population de proies et de prédateurs [Wilensky, 1997]. Les proies se reproduisent selon un certain rythme et se déplacent aléatoirement dans l'environnement alors que les prédateurs chassent les proies. Les prédateurs peuvent mourir de famine tandis que les proies survivent jusqu'à ce qu'elles soient attaquées par un prédateur. Cette simulation illustre la co-évolution des populations avec des cycles alternés de croissance/décroissance des deux populations.

A.8 Résultats expérimentaux

Dans cette section, nous évaluons les performances des deux types de répartition, de l'environnement ou des agents, par rapport au deux types d'applications, le *flocking* et le *proies prédateurs*. Les expérimentations diffèrent par l'évolution de la taille de l'environnement, le nombre de nœuds utilisés lors de la simulation ainsi que le nombre d'agents. Nous utilisons deux critères principaux pour mesurer les performances du simulateur :

- le temps moyen d'un pas de temps.
- le volume de messages entre les nœuds.

L'ensemble des expérimentations ont été réalisées dans des salles de TP constituées de PC sous Linux (Intel-R CoreTM2 Duo CPU E8400 3.00GHz, 4GB de mémoire et Ethernet 100Mb), en faisant varier le nombre de nœuds de calcul de 1 à 16 machines pour des durées de simulation d'au moins 300 pas de temps, et sur la grille de calcul Grid5000[1].

A.8.1 Performances globales de la simulation

La figure A.7 correspondent à la simulation de *flocking* pour les deux types de répartition. La figure démontre que les performances sont meilleures avec la répartition des agents car la dynamique de l'application induit une concentration d'agents sur certaines zones de l'environnement ce qui pénalise fortement les performances de la répartition de l'environnement.

[1] `http://www.grid5000.org`

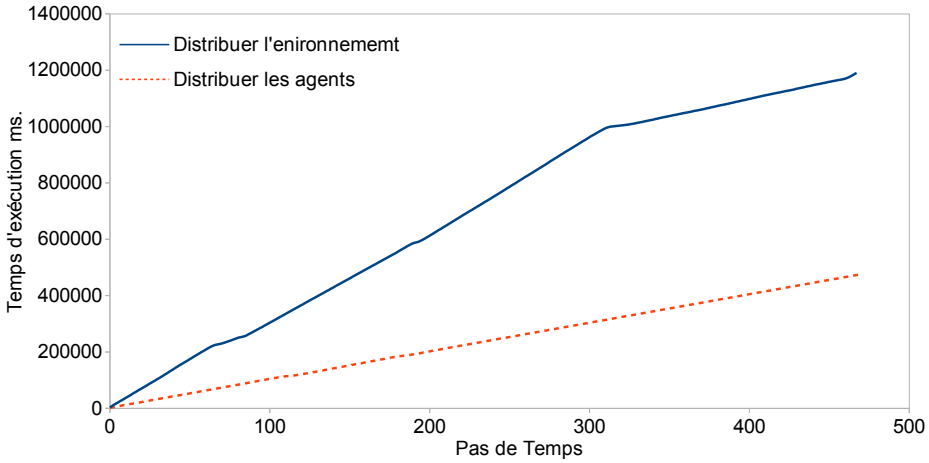

FIGURE A.7 – Temps d'exécution pour le *flocking*.

On constate donc que pour la répartition des agents, la distribution homogène de la charge de calcul compense les coûts de communication pourtant élevés (figure A.8).

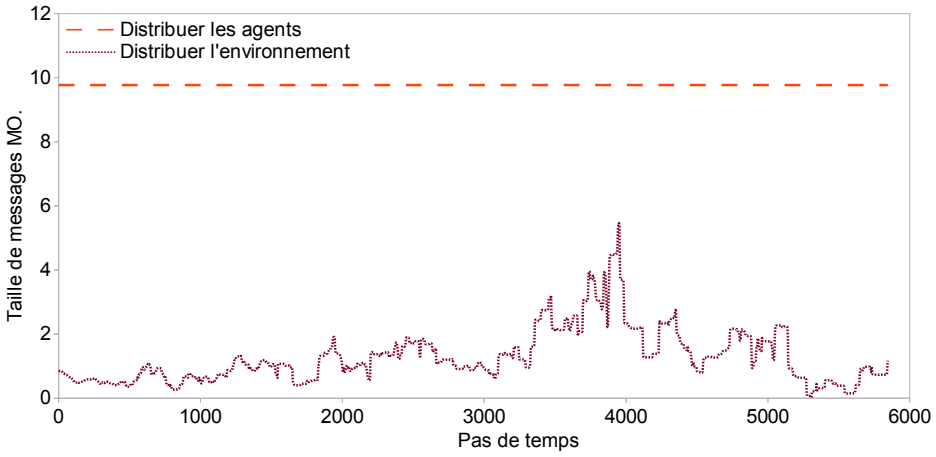

FIGURE A.8 – Comparaison des coûts de communication pour les deux types de répartition.

Par contre, dans le cas du modèle *proies prédateurs*, la performance est meilleure dans le cas de la répartition de l'environnement car la distribution des agents étant homogène, la charge se trouve mieux répartie (A.9).

Pour résumer, lorsque la dynamique des populations d'agents entraîne une répartition plutôt homogène dans l'environnement, la répartition de ce dernier obtient de meilleures performances, alors que dans une situation de

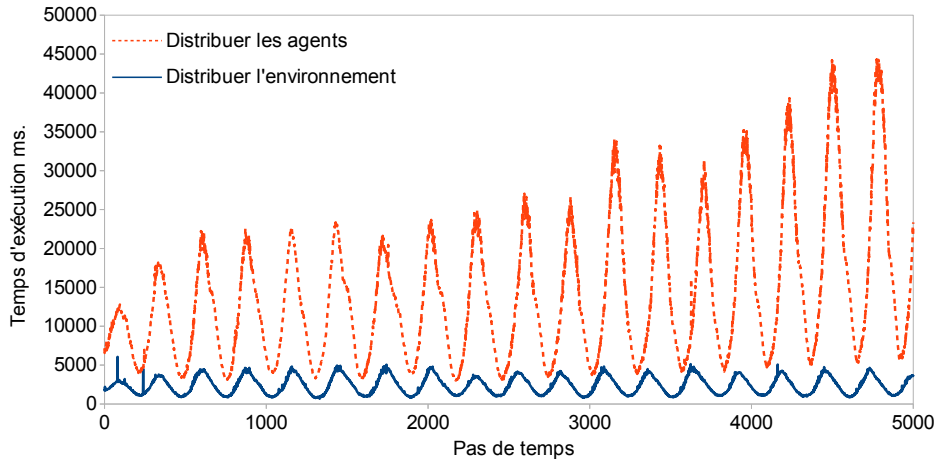

FIGURE A.9 – Temps d'exécution pour le modèle *proies prédateurs*.

fluctuation importante d'agents sur une zone restreinte, la répartition des agents est plus efficiente.

A.8.2 Evaluations à la répartition de l'environnement

Nous avons cherché dans cette section à mieux caractériser les avantages et limites de la répartition de l'environnement, particulièrement au niveau de la gestion des zones fantômes. La première expérience consiste à faire varier le nombre de nœuds de calcul et à observer l'évolution du temps moyen de calcul d'un pas de temps. L'application évaluée est le modèle *proies prédateurs* avec une configuration initiale de 42000 agents répartis respectivement sur 1, 2 et 4 nœuds. La figure A.10 montre les gains obtenu lors de l'ajout de nœuds. On constate que pour cette application, la durée nécessaire pour le calcul d'un pas de temps sur l'ensemble des nœuds décroît fortement avec l'ajout de nœuds supplémentaires, ce qui valide la parallélisation implicite induite par l'homogénéité de répartition des agents au cours de la simulation.

Pour caractériser la répartition du temps de calcul lors d'un pas de simulation entre les aspects liés à la synchronisation et ceux liés à l'évaluation des agents, nous avons instrumenté le simulateur. La figure A.11 confirme que la majorité du temps est exploité pour le calcul du comportement des agents et l'exécution de leur interaction, ce qui explique les gains de performances lors de l'ajout de nœuds. Ainsi, même si les coûts de synchronisation augmentent avec le nombre de machines, ils sont compensés par les gains sur le temps global d'exécution de la simulation.

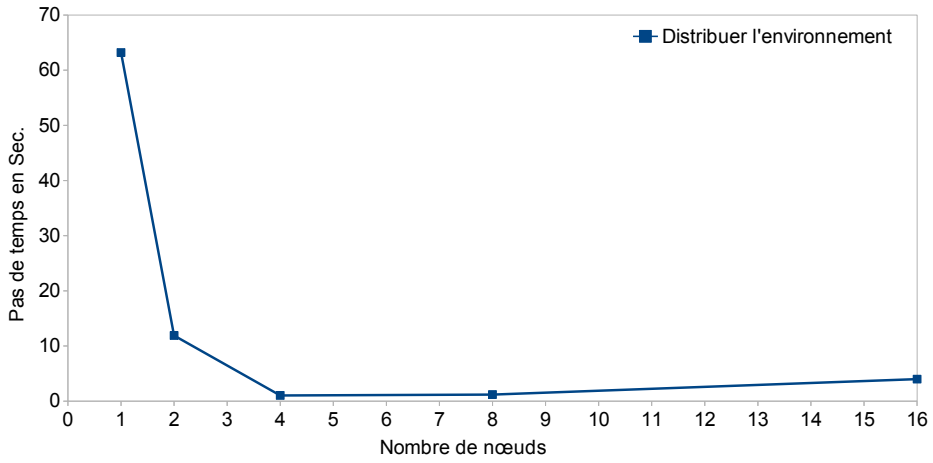

FIGURE A.10 – Temps de calcul moyen d'un pas de temps.

Expérimentations sur la zone fantôme

Cette expérience consiste à explorer le coût de la gestion des zones fantômes. En effet, ces dernières doivent être échangées entre les nœuds adjacents au début de chaque pas de simulation. Nous exécutons donc le modèle de *flocking* sur un réseau de 3x3 nœuds en faisant varier le nombre d'agents de 9 à 900000 et en utilisant dans un premier temps une taille de 10 unités pour la largeur de la la zone fantôme, puis de 100 unités (ce qui correspond à la taille de la zone de perception des oiseaux).

La figure A.12 démontre que le temps nécessaire pour le calcul d'un pas de temps s'accroît en fonction du nombre d'agents, mais le plus important est que la taille de la zone n'a pas d'impact. Ceci s'explique par le protocole utilisé qui ne nécessite qu'un seul échange de message pour transmettre l'ensemble des agents en début de pas de temps. Il faut cependant noter que si l'on utilisait une largeur de zone fantôme plus grande que la largeur gérée par un nœuds, les performances s'effondreraient. Ceci indique qu'un protocole spécifique devrait être instauré dans le cas d'applications impliquant des agents ayant une perception plus importante que la surface gérée par un nœuds.

Coût de communication

Pour cette dernière expérience, nous évaluons le volume des messages échangés dans les deux applications pour une simulation réalisée sur 2 nœuds de calcul. La figure A.13 illustre parfaitement les fluctuations importantes se produisant dans le modèle du *flocking* qui correspondent au moment où la

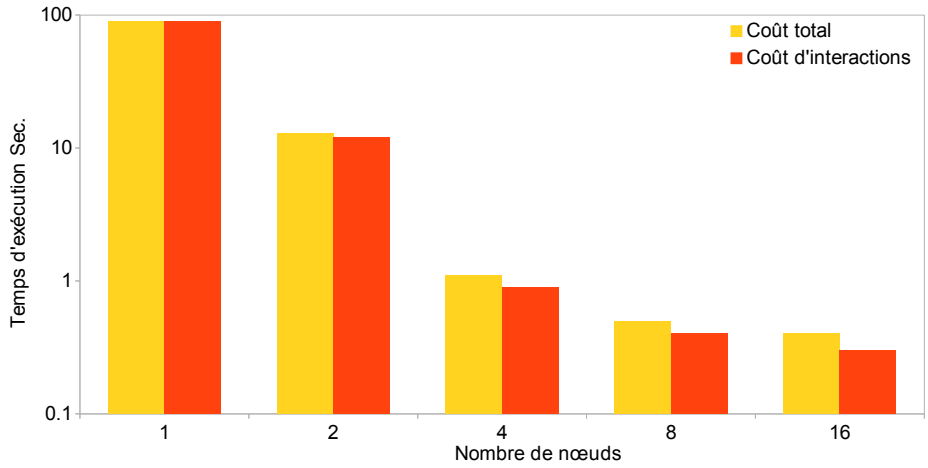

FIGURE A.11 – Temps de calcul d'un pas de temps et coût de l'évaluation des interactions.

volée transite d'un nœuds de calcul à l'autre, alors que dans le *proies prédateurs*, l'homogénéité de la répartition des agents se traduit directement en un volume quasi constant du volume des messages échangés. Cette figure visualise clairement l'impact de la dynamique de déplacement des agents sur le mode de répartition de calcul choisit.

A.9 Évaluation des politiques de synchronisation

Dans cette section, nous étudions deux applications qui ont été réalisées pour évaluer l'impact des politiques de synchronisation proposées dans les sections précédentes. Il semble évident que les incohérences temporelles introduites par la synchronisation flexible n'auront pas les mêmes effets selon les applications. Ainsi, pour l'exemple sur les *boids*, même lorsqu'il n'y a pas de synchronisation, les phénomènes macroscopiques de regroupement d'agents apparaissent rapidement. Ce que nous souhaitons déterminer, ce sont les propriétés des modèles d'application nous permettant d'anticiper l'impact des incohérences temporelles sur l'issue de la simulation. Pour cela, nous avons choisi deux applications dont l'une n'est pas trop affectée par les incohérences tandis que l'autre y est très sensible : *Proies-prédateurs* et *Capture du drapeau*.

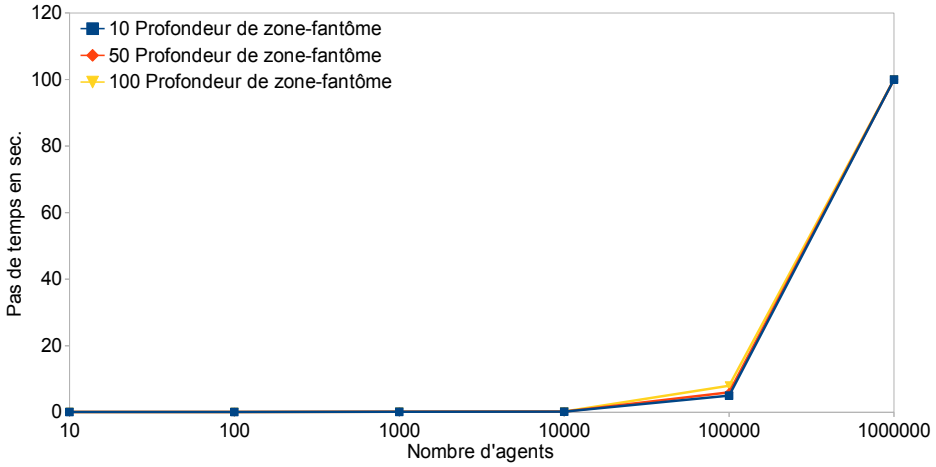

FIGURE A.12 – Variation du pas de temps en fonction de la largeur de la zone fantôme.

Capture du drapeau

[Rihawi et al., 2013c] Nous avons créé ce modèle pour illustrer les biais potentiellement introduits par les incohérences temporelles lorsque le modèle de l'application est fortement lié au déroulement temporel. Dans cette application, deux armées cherchent à capturer les drapeaux de leur adversaire. Chaque armée est constituée de deux types d'agents, des agents *soldats* qui défendent un drapeau ou partent à l'assaut de celui de l'adversaire et des agents *drapeaux* qui produisent de nouveaux soldats à chaque pas de temps. Lorsque deux soldats d'armées opposées se rencontrent, ils se neutralisent en mourant tous les deux.

Comme précisé lors de l'introduction, le choix de ces deux applications s'est effectué pour respecter la contrainte de localité des interactions : seuls des agents suffisamment proches peuvent interagir. Il n'y a pas donc pas d'échange de message entre des agents distants. Cependant, deux agents proches mais sur deux machines différentes peuvent interagir.

A.9.1 Coût de la synchronisation

L'ensemble des expérimentations ont été réalisées sur un réseau local constitué de machines homogènes (salles de TP). La plupart des expériences ont duré plus de deux millions de pas de temps. Le seul paramètre qui a été modifié est la taille W de la fenêtre de temps : $W = 0$ pour la synchronisation forte, $W = n$ avec $n = 10, 100, 1000, 10000, 100000$ pour la synchronisation flexible et $W = \infty$ pour l'absence totale de synchronisation. La table A.2

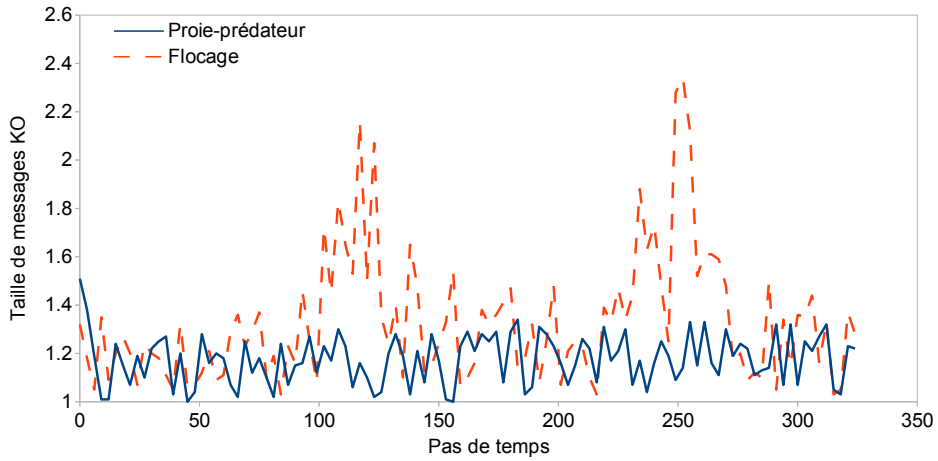

FIGURE A.13 – Variation des coûts de communication en function de la dynamique de déplacement des agents.

synthétise les résultats et démontre que sans synchronisation on obtient logiquement le temps d'exécution le plus court, cependant pour certaines applications comme la capture du drapeau, cette relaxation de la synchronisation déstabilise complètement le modèle dans certains cas.

La comparaison de la stabilité des résultats des deux applications montre que pour le *proies prédateurs*, la co-évolution des deux populations est conservée, le modèle est donc stable sur une longue durée (2 millions de pas de temps). Ainsi, cette application est fortement tolérante aux incohérences temporelles et peut tirer partie d'une synchronisation flexible. Par contre, pour la *capture du drapeau*, lorsque W devient plus grand qu'un certain nombre N de pas de temps (N étant dépendant de la configuration initiale : nombre de drapeaux et répartition sur les différents nœuds), le modèle devient instable. Cela signifie que l'une des deux armées arrive à prendre le dessus sur l'autre, ce qui mène rapidement à la capture de l'ensemble des drapeaux d'un camp.

Dans les sections qui suivent, nous cherchons à affiner ces résultats trop généraux. Pour cela, nous étudions d'abord les effets des interactions invalides temporellement sur la stabilité du modèle *proies prédateurs*, ainsi que l'évolution du volume des interactions incohérentes en fonction des politiques de synchronisation. Ensuite, nous nous intéressons plus particulièrement aux conditions de stabilité du modèle de capture du drapeau en nous concentrant sur la sensibilité à la configuration initiale lors de sa répartition sur les nœuds.

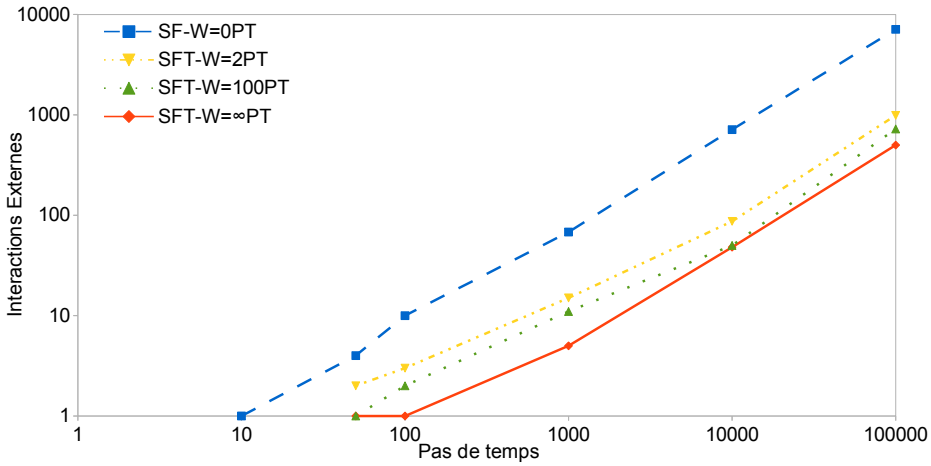

FIGURE A.14 – Volume d'interactions externes en fonction des politiques de synchronisation.

Interactions dans *proies prédateurs*

Les interactions externes (IE) sont des interactions se produisant entre des agents ne se trouvant pas sur le même nœuds. La figure A.14 illustre l'évolution du nombre d'interactions externes en fonction de la variation de la taille de la fenêtre pour une simulation contenant 5000 agents et s'exécutant durant 100000 pas de temps. On constate qu'en synchronisation forte, il y a beaucoup d'interactions externes alors que leur nombre chute drastiquement même avec une très petite fenêtre de temps ($W = 2$). Ceci s'explique car lorsque la plateforme n'est pas en synchronisation forte, il n'y a plus d'attente d'information de la part des autres nœuds par rapport aux agents proches des frontières.

Ainsi dans la plupart des cas, le nœuds courant considère que les interactions externes échouent et poursuit son cycle d'évaluation. Ceci implique qu'il y a un biais important introduit au niveau des frontières entre les parcelles d'environnement puisque la quasi-totalité des interactions échouent. Dans l'hypothèse improbable où la surface d'une parcelle d'environnement devient trop petite par rapport à zone de frontière avec les autres nœuds, nous serions amenés à une situation dans laquelle plus aucune interaction ne pourrait se produire. Cependant, cette situation est un cas limite théorique puisque l'intérêt de la répartition de l'environnement consiste justement à dédier une parcelle d'environnement fournissant une charge suffisante à chaque machine.

Les interactions invalides (II), c'est-à-dire induisant une incohérence temporelle, sont des interactions se produisant entre des agents n'appartenant

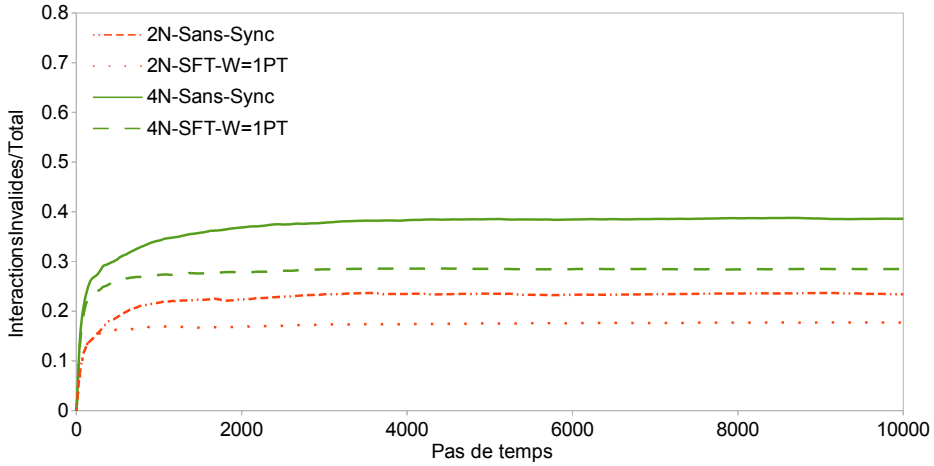

FIGURE A.15 – Proportion d'interactions invalides en fonction du nombre de nœuds.

pas au même pas de temps. En effet, lors du transfert d'agents d'un nœuds à un autre, lorsque la synchronisation n'est pas forte, il est possible d'avoir des agents de pas de temps différents qui entrent malgré tout en interaction. En effet, lorsqu'un agent a un pas de temps n est transféré sur une machine ayant un pas de temps $n + k$, l'agent transféré conserve sa propre référence temporelle (n dans ce cas). Ainsi, même deux agents se trouvant sur le même nœuds peuvent produire une interaction invalide.

Nous cherchons donc ici à quantifier la proportion que représentent ces interactions incohérentes d'un point de vue temporel par rapport à l'ensemble des interactions valides. La figure A.15 montre que le nombre de ces interactions invalides croît en fonction de la taille de la fenêtre de temps ainsi qu'en fonction du nombre de nœuds. Ainsi, que l'on soit avec une petite fenêtre de temps ou sans synchronisation, il y a une proportion non négligeable d'interactions invalides oscillant entre 20 à 30% pour 2 machines et 30 à 40% pour 4 machines.

Malgré ces mesures qui démontrent l'importance des biais introduits dans la simulation avec une forte diminution des interactions externes et l'introduction d'un volume non négligeable d'interactions temporellement incohérentes, le modèle *proies prédateurs* reste insensible au choix de la politique de synchronisation.

Instabilité du modèle *capture du drapeau*

Comme pour les expériences précédentes, ces simulations ont été exécutées sur 2 nœuds durant 2 millions de pas de temps avec des fenêtres de temps va-

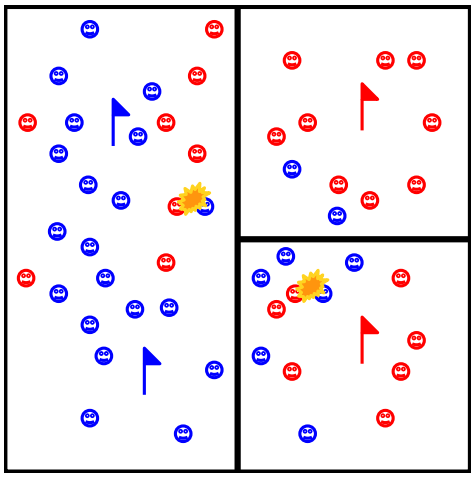

FIGURE A.16 – Configuration d'une simulation sur 3 nœuds : 2 pour l'équipe rouge et 1 pour l'équipe bleue.

riant entre 0 et l'infini. La première configuration est constituée d'un drapeau par nœuds et dans ce cas, comme pour *proies prédateurs*, le modèle est stable (aucun drapeau n'est capturé) quel que soit le mécanisme de synchronisation utilisé. Par contre, lorsque la configuration initiale repose sur un plus grand nombre de drapeaux (>20), nous obtenons des résultats différents. En effet, pour une fenêtre de taille supérieur à 10000 pas de temps, le modèle n'est plus stable et l'une des équipes l'emporte toujours sur l'autre.

Pour mieux mesurer l'évolution de la dégradation de la stabilité provoquée par les politiques de synchronisation, nous avons réalisé une dernière expérimentation. Cette simulation s'effectue sur 3 nœuds (figure A.16) : le premier contient tous les drapeaux de la première armée, alors que ceux de la deuxième armée sont répartis équitablement sur les deux nœuds restants. L'objectif est d'avoir une charge plus importante sur la première machine ce qui induit une instabilité du modèle dès que l'on n'utilise plus une synchronisation forte. Ce que nous étudions dans cette configuration particulière est la vitesse de déstabilisation du modèle en fonction de la taille de la fenêtre de temps, ou en fonction du nombre de drapeaux initiaux.

Si nous définissons le *pas de temps critique* (PTC), comme le nombre de pas de temps nécessaires pour aboutir à la destruction de l'ensemble des drapeaux d'un des deux camps lorsqu'il n'y a pas de synchronisation, la figure A.17 illustre le fait que le PTC diminue avec l'augmentation du nombre de drapeaux initiaux, ainsi $PTC = \frac{\alpha_1}{D}$, α_1 est une constante dépendante de la configuration initiale (ie. le nombre de drapeau et leur disposition sur les différents nœuds).

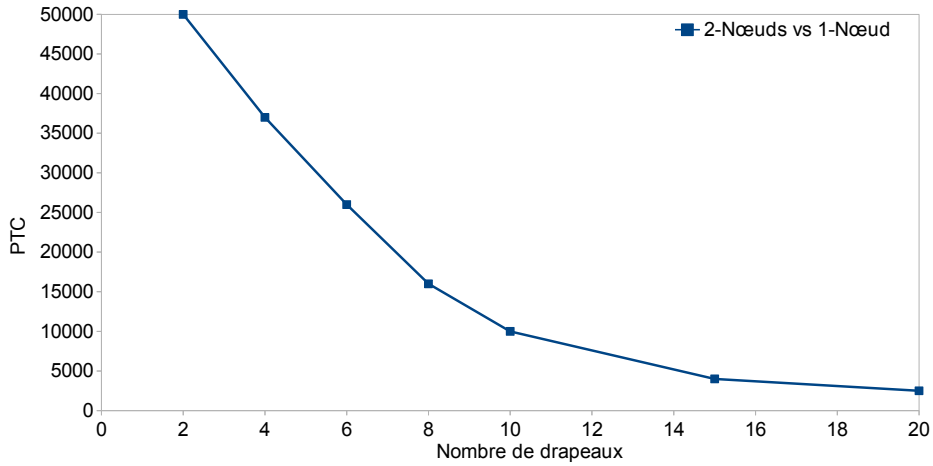

FIGURE A.17 – Évolution du pas de temps critique en fonction du nombre de drapeaux.

Si nous paramétrons le pas de temps critique par la fenêtre de temps W utilisée ($PTC(W)$), alors $PTC = PTC(W)$ lorsque $W = \infty$. La figure A.18 illustre la rapidité de convergence vers le pas de temps critique en fonction de la fenêtre de temps choisie. Cette figure montre $PTC(W)$ décroît si la taille W de la fenêtre augmente, ainsi, selon la figure, $PTC(W)/PTC = \frac{\alpha_2}{W/PTC}$, α_2 est constant. Ainsi, $PTC(W) = \frac{\alpha_2 \times PTC^2}{W}$, donc $PTC(W) = \frac{\alpha}{W \times D^2}$, $\alpha = \alpha_2 \times \alpha_1^2$ est constant. Cela signifie donc que le pas de temps critique décroît si le nombre de drapeaux ou la taille de la fenêtre augmentent.

La figure A.18 montre aussi que pour toutes les configurations, le modèle reste stable pour de petites fenêtres de temps. Néanmoins, il est possible d'analyser chaque courbe en deux parties principales : la première de 0 à 30% du pas de temps critique où le modèle est stable sur un nombre conséquent de pas de temps. Cela implique qu'il est possible d'autoriser une fenêtre de temps jusqu'à 30% du PTC pour accélérer la convergence. La deuxième partie de la courbe, pour des fenêtres de temps excédant les 30% du PTC, le $PTC(W)$ décroît plus lentement en fonction de la fenêtre de temps.

A.10 Conclusion

Lorsque l'on modélise des systèmes complexes impliquant différentes familles d'agents et de nombreuses interactions, la simulation à l'aide de systèmes multi-agents devient rapidement coûteuse. Dans ce travail, nous avons exploré différentes pistes afin de réaliser des simulations à large échelle de systèmes

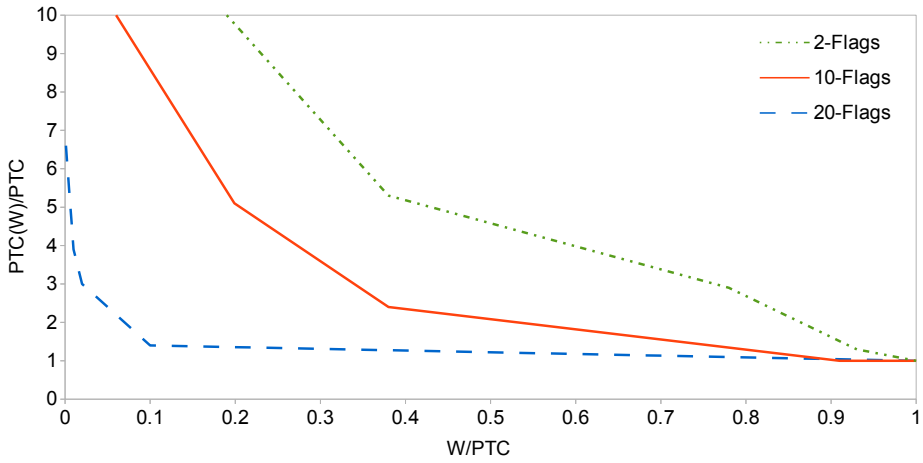

FIGURE A.18 – Vitesse de convergence vers le *PTC* en fonction de la configuration initiale.

à base d'agents situés dans un espace euclidien. Dans ce contexte, les agents interagissent classiquement au niveau individuel, mais les observations sont réalisées au niveau des populations d'agents, soit sur des propriétés agrégés des différentes familles d'agents, soit sur des phénomènes émergents, souvent liés aux dynamiques de déplacement. Afin de traiter des systèmes composés de centaines de milliers, voire de millions d'agents, nous avons choisi de répartir le simulateur sur un réseau de machines. Le passage d'un système ne tournant que sur une seule machine à un système distribué induit de nouvelles problématiques portant sur la gestion du temps et de la synchronisation, ainsi que sur la répartition de la charge de calcul.

La première partie de la thèse étudie deux types de répartition : l'une reposant sur un critère de proximité spatiale des agents et l'autre sur des critères non spatiaux. La première approche consiste à discrétiser l'environnement et à répartir ces sous-ensembles sur un réseau de machines. Ainsi, chaque nœud de calcul détient une partie de l'environnement et gère le tour de parole des agents qu'il contient. Chaque machine gère donc une même surface de l'environnement, mais le nombre d'agents contenus dans ce sous-ensemble peut fortement varier au cours de la simulation, induisant des différences de charge de calcul entre les machines. En effet, en fonction des dynamiques de déplacement des agents, la répartition des agents au sein de l'environnement peut devenir fortement hétérogène. La seconde approche consiste à répartir uniquement les agents selon des critères arbitraires comme le type de l'agent ou encore selon les coûts de calculs estimés. Avec cette approche, une vue globale de l'environnement est reconstituée lors de chaque pas de

temps sur l'ensemble des machines, même si chacune des machines ne gère qu'un sous-ensemble des agents. Dans un système constitué d'agents homogènes et ne nécessitant pas de création et destruction d'agents au cours de la simulation, la charge est correctement répartie. Par contre, si la simulation induit une évolution de la population d'agents, un mécanisme de répartition de charge devient nécessaire afin de rééquilibrer les volumes d'agents gérés par les différentes machines.

La seconde partie de la thèse repose sur la remise en cause d'une hypothèse exploitée par l'ensemble des simulateurs existants : la garantie du déterminisme et de la reproductibilité des simulations. En effet, les simulateurs existants répartissent le calcul sur un réseau de machines et s'assurent que l'ensemble des agents évolue dans la même temporalité. Cependant, lorsque l'on simule des millions d'agents, les observations portent principalement sur des données agrégées. On peut donc supposer que des incohérences au niveau de l'état de centaines, voire de milliers d'agents, n'aura pas nécessairement un impact significatif sur les mesures effectuées par agrégation des propriétés de l'ensemble de la population. Afin d'étudier cette question, nous avons proposé de relâcher la contrainte de synchronisation en bornant l'écart temporel existant entre la machine la plus lente et la machine la plus rapide au cours de la simulation. En exploitant une fenêtre de temps suffisamment grande, on obtient un système dans lequel les machines sont totalement désynchronisées. Pour certaines applications, malgré l'absence totale de synchronisation les propriétés émergentes sont conservées et le gain en temps de calcul est important.

Afin d'étudier la pertinence des différents types de répartition, ainsi que l'impact du relâchement de la contrainte de synchronisation, nous avons développé un simulateur et implémenté différentes applications. Ces applications ont été choisies pour leurs différences de propriétés en terme d'homogénéité de répartition spatiale ainsi qu'en terme de dynamique d'évolution de population. Ces expérimentations illustrent l'adéquation de certains types de répartition pour certaines propriétés des applications. Ainsi, la distribution de l'environnement est particulièrement adaptée aux systèmes constitués d'une population qui évolue en nombre et dont la dynamique de déplacement n'induit pas la constitution d'un trop faible nombre de *clusters* d'agents. A l'inverse, si la dynamique de déplacement induit la création de zones hétérogènes, certaines très denses en terme d'agents et d'autres totalement dépeuplée, la répartition selon des critères non spatiaux est plus appropriée. De même, lors du relâchement de la contrainte de synchronisation, la question de la dynamique d'évolution de la population d'agents, ainsi que la durée de vie moyenne des agents sont des critères importants pour définir la taille de la fenêtre de temps permettant à la fois un gain de temps d'exécution tout en

préservant le phénomène émergent.

Ces travaux ont permis la mise à disposition d'un environnement distribué pour la simulation à large échelle d'agents situés, jusqu'à 1 million d'agents avec un pas de temps inférieur à 2mn sur une infrastructure de type grille (Grid5000). La plateforme fournit une grande liberté à l'utilisateur en terme de stratégie de répartition ainsi que de mécanismes de synchronisation. Il reste de nombreuses pistes à explorer mais ces travaux fournissent une première base de travail et un environnement propices à de nouvelles expérimentations.

List of Figures

List of Tables

Bibliography

Bellifemine, F., Bergenti, F., Caire, G., and Poggi, A. (2005).
Jade — a java agent development framework.
In *Multi-Agent Programming*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 125–147. Springer US.

Bernon, C., Camps, V., Gleizes, M. P., and Picard, G. (2003).
Designing agents' behaviors and interactions within the framework of adelfe methodology.
In *ESAW' 03*, pages 311–327.

Bonnet, G. and Tessier, C. (2009).
An incremental adaptive organization for a satellite constellation.
In *Organized Adaption in Multi-Agent Systems*, volume 5368 of *Lecture Notes in Computer Science*, pages 108–125. Springer Berlin Heidelberg.

Bourdon, J., Vercouter, L., and Ishida, T. (2009).
A multiagent model for provider-centered trust in composite web services.
In *Principles of Practice in Multi-Agent Systems, 12th International Conference, PRIMA 2009, Nagoya, Japan, December 14-16, 2009. Proceedings*, pages 216–228.

Bromuri, S. and Stathis, K. (2008).
Situating cognitive agents in golem.
In *Engineering Environment-Mediated Multi-Agent Systems*, volume 5049 of *Lecture Notes in Computer Science*, pages 115–134. Springer Berlin Heidelberg.

Bromuri, S. and Stathis, K. (2009).
Distributed agent environments in the ambient event calculus.
In *Proc.of DEBS*, pages 12:1–12:12, New York, USA. ACM.

Brooks, R. A. (1991).
Intelligence without reason.

In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'91, pages 569–595, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Cabri, G., Ferrari, L., and Zambonelli, F. (2004).
Role-based approaches for engineering interactions in large-scale multi-agent systems.
In *Software Engineering for Multi-Agent Systems II*, volume 2940 of *Lecture Notes in Computer Science*, pages 360–361. Springer Berlin / Heidelberg.

Collier, N. and North, M. (2012).
Parallel agent-based simulation with repast for high performance computing.
In *In SIMULATION:Transactions of the Society for Modeling and Simulation International*, pages 01–21.

Cordasco, G., De Chiara, R., Mancuso, A., Mazzeo, D., Scarano, V., and Spagnuolo, C. (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 2011, Bordeaux, France.

Cosenza, B., Cordasco, G., De Chiara, R., and Scarano, V. (2011).
Distributed load balancing for parallel agent-based simulations.
In *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*.

Coulouris, G., Dollimore, J., and Kindberg, T. (2005).
Distributed Systems: Concepts and Design (International Computer Science).
Addison-Wesley Longman, Amsterdam.

Davis, W. and Moeller, G. (1999).
The high level architecture: Is there a better way?
In *Simulation Conference Proceedings, 1999 Winter*, volume 2, pages 1595–1601 vol.2.

Dyke Parunak, H., Brueckner, S., Fleischer, M., and Odell, J. (2004).
A design taxonomy of multi-agent interactions.
In *Agent-Oriented Software Engineering IV*, volume 2935 of *Lecture Notes in Computer Science*, pages 123–137. Springer Berlin Heidelberg.

Ebcioglu, K., Saraswat, V., and Sarkar, V. (2004).
X10: Programming for hierarchical parallelism and non-uniform data access.
In *Proceedings of the International Workshop on Language Runtimes, OOPSLA*.

Elnozahy, E. N. M., Alvisi, L., Wang, Y.-M., and Johnson, D. B. (2002).
A survey of rollback-recovery protocols in message-passing systems.
ACM Comput. Surv., 34(3):375–408.

Faci, N., Guessoum, Z., and Marin, O. (2006).
Dimax: A fault-tolerant multi-agent platform.
In *Proceedings of the 2006 International Workshop on Software Engineering for Large-scale Multi-agent Systems*, SELMAS '06, pages 13–20, New York, NY, USA. ACM.

Fatès, N. and Chevrier, V. (2010).
How important are updating schemes in multi-agent systems? an illustration on a multi-turmite model.
In *Proceedings of the 9th International Conference on Autonomous Agents and Multi-agent Systems: volume 1 - Volume 1*, AAMAS '10, pages 533–540, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Ferber, J. (1999).
Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence.
Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Fujimoto, R. (2000).
Parallel and distributed simulation systems.
Wiley series on parallel and distributed computing. Wiley.

Genesereth, M. R. and Ketchpel, S. P. (1994).
Software agents.
Commun. ACM, 37(7):48–53.

Gold, T. (2003).
Why time flows: The physics of past & future.
Daedalus, 132(2):pp. 37–40.

Grimm, V. and Railsback, S. F. (2013).
Individual-based Modeling and Ecology.

Princeton Series in Theoretical and Computational Biology. Princeton University Press, stu - student edition edition.

Guessoum, Z., Briot, J.-P., Marin, O., Hamel, A., and Sens, P. (2003).
Dynamic and adaptive replication for large-scale reliable multi-agent systems.
In *Software Engineering for Large-Scale Multi-Agent Systems*, volume 2603 of *Lecture Notes in Computer Science*, pages 182–198. Springer Berlin Heidelberg.

Guessoum, Z., Faci, N., and Briot, J.-P. (2005).
Adaptive replication of large-scale multi-agent systems: Towards a fault-tolerant multi-agent platform.
In *Proceedings of the Fourth International Workshop on Software Engineering for Large-scale Multi-agent Systems*, SELMAS '05, pages 1–6, New York, NY, USA. ACM.

Gupta, B., Rahimi, S., and Yang, Y. (2007).
A novel roll-back mechanism for performance enhancement of asynchronous checkpointing and recovery.
Informatica (Slovenia), 31(1):1–13.

Hayes Roth, B. (1995).
An architecture for adaptive intelligent systems.
Artificial Intelligence, 72(1–2):329–365.

Horling, B., Mailler, R., and Lesser, V. (2004).
Farm: A scalable environment for multi-agent development and evaluation.
In *Software Engineering for Multi-Agent Systems II*, volume 2940 of *Lecture Notes in Computer Science*, pages 364–367. Springer Berlin / Heidelberg.

Šišlák, D., Volf, P., Jakob, M., and Pechoucek, M. (2009).
Distributed platform for large-scale agent-based simulations.
In *Agents for Games and Simulations*, volume 5920 of *Lecture Notes in Computer Science*, pages 16–32. Springer Berlin Heidelberg.

Isenburg, M., Lindstrom, P., and Childs, H. (2010).
Parallel and streaming generation of ghost data for structured grids.
CGA, IEEE, 30(3):32 –44.

Jamali, N. and Zhao, X. (2008).
Distributed coordination of massively multi-agent systems.
In *Massively Multi-Agent Technology*, volume 5043 of *Lecture Notes in Computer Science*, pages 13–27. Springer Berlin / Heidelberg.

Jefferson, D. R. (1985).
Virtual time.
ACM Trans. Program. Lang. Syst., 7:404–425.

Jin, X., Liu, J., and Yang, Z. (2005).
The dynamics of peer-to-peer tasks: An agent-based perspective.
In *Agents and Peer-to-Peer Computing*, volume 3601 of *Lecture Notes in Computer Science*, pages 173–184. Springer Berlin / Heidelberg.

Jinkai, X. and Weihong, Y. (2010).
Study on comparison between jafmas and jade.
In *Circuits,Communications and System (PACCS), 2010 Second Pacific-Asia Conference on*, volume 1, pages 105 –108.

Judson, O. P. (1994).
The rise of the individual-based model in ecology.
Trends in Ecology & Evolution, 9(1):9 – 14.

Karmakharm, T., Richmond, P., and Romano, D. (2010).
Agent-based large scale simulation of pedestrians with adaptive realistic navigation vector fields.
In *Theory and Practice of Computer Graphics (TPCG) 2010*, pages 67–74.

Kiran, M., Richmond, P., Holcombe, M., Chin, L. S., Worth, D., and 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*, pages 1633–1636, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.
no details in this paper.

Kubera, Y., Mathieu, P., and Picault, S. (2008).
Interaction-oriented agent simulations : From theory to implementation.
In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08)*, pages 383–387. IOS Press.

Kubera, Y., Mathieu, P., and Picault, S. (2010).
Everything can be agent!
In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, pages 1547–1548, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Kubera, Y., Mathieu, P., and Picault, S. (2011).
Ioda: An interaction-oriented approach for multi-agent based simulations.
Journal of Autonomous Agents and Multi-Agent Systems, 23(3):303–343.

Kuhl, F., Weatherly, R., and Dahmann, J. (1999).
Creating computer simulation systems: an introduction to the high level architecture.
Prentice Hall PTR, Upper Saddle River, NJ, USA.

Lamport, L. (1978).
Ti clocks, and the ordering of events in a distributed system.
Commun. ACM, 21:558–565.

Logan, B. and Theodoropoulos, G. (2001).
The distributed simulation of multiagent systems.
Proceedings of the IEEE, 89(2):174 –185.

Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., and Balan, G. (2005).
Mason: A multiagent simulation environment.
Simulation, 81(7):517–527.

Maes, P. (1990).
Situated agents can have goals.
Robot. Auton. Syst., 6(1-2):49–70.

Mataric, M. J. (1994).
Interaction and intelligent behavior.
Technical report, MIT.

Mathieu, P. and Brandouy, O. (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)*, pages 185–197. Springer.

McTaggart, J. E. (1908).
The unreality of time.
Mind, 17(68):pp. 457–474.

Minar, N., Burkhart, R., and Langton, C. (1996).
The swarm simulation system: A toolkit for building multi-agent simulations.
Technical report, Santa Fe Institute.

Minson, R. and Theodoropoulos, G. K. (2004).
Distributing repast agent-based simulations with hla.
In *In European Simulation Interoperability Workshop 2004*, pages 04–046.

Miyata, N. and Ishida, T. (2008).
Community-based load balancing for massively multi-agent systems.
In *Massively Multi-Agent Technology*, volume 5043 of *Lecture Notes in Computer Science*, pages 28–42. Springer Berlin / Heidelberg.

Motshegwa, T. and Schroeder, M. (2004).
Interaction monitoring and termination detection for agent societies: Preliminary results.
In *Engineering Societies in the Agents World*, volume 3071 of *Lecture Notes in Computer Science*, pages 519–519. Springer Berlin / Heidelberg.

Muller, G. and Vercouter, L. (2005).
Decentralized monitoring of agent communications with a reputation model.
In *Trusting Agents for Trusting Electronic Societies*, volume 3577 of *Lecture Notes in Computer Science*, pages 144–161. Springer Berlin Heidelberg.

Nadiminti, K., Assunção, M. D., and Buyya, R. (2006).
Distributed systems and recent innovations: Challenges and benefits.
InfoNet Magazine, 16(3):1–5.

Navarro, L., Corruble, V., Flacher, F., and Zucker, J.-D. (2013).
A flexible approach to multi-level agent-based simulation with the mesoscopic representation.
In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pages 159–166. International Foundation for Autonomous Agents and Multiagent Systems.

Ngobye, M., de Groot, W. T., and van der Weide, T. P. (2010).
Types and priorities of multi-agent system interactions.
Interdisciplinary Description of Complex Systems - scientific journal, 8(1):49–58.

Nikolai, C. and Madey, G. (2009).
Tools of the trade: A survey of various agent based modeling platforms.
Journal of Artificial Societies and Social Simulation, 12(2):2.

North, M., Collier, N., Ozik, J., Tatara, E., Macal, C., Bragen, M., and Sydelko, P. (2013).

Complex adaptive systems modeling with repast simphony.
Complex Adaptive Systems Modeling, 1(1):1–26.

Reynolds, C. W. (1987).
Flocks, herds, and schools: a distributed behavioral model.
Computer Graphics.

Richmond, P., Coakley, S., and Romano, D. M. (2009).
A high performance agent based modelling framework on graphics card hardware with cuda.
In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 1125–1126, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Rihawi, O., Secq, Y., and Mathieu, P. (2013a).
Impact des politiques de synchronisation dans les simulations réparties d'agents situés.
In *21èmes Journées Francophones sur les Systèmes Multi-Agents (JFSMA2013)*, pages 115–124. Cépaduès.

Rihawi, O., Secq, Y., and Mathieu, P. (2013b).
Relaxing synchronization constraints in distributed agent-based simulations.
In *Jurnal Teknologi (Sciences and Engineering)*, volume 63:3 (E), pages 65–76. Penerbit UTM Press (Extended version from the selected paper in DCAI 2013).

Rihawi, O., Secq, Y., and Mathieu, P. (2013c).
Synchronization policies impact in distributed agent-based simulation.
In *Distributed Computing and Artificial Intelligence (DCAI 2013)*, volume 217 of *Advances in Intelligent Systems and Computing*, pages 19–26. Springer International Publishing (Extended version selected for publication in Jurnal Teknologi).

Rihawi, O., Secq, Y., and Mathieu, P. (2014).
Effective distribution of large scale situated agent-based simulations.
In *ICAART 2014 6th International Conference on Agents and Artificial Intelligence*, volume 1, pages 312–319. SCITEPRESS Digital Library.

Russell, S. J., Norvig, P., Candy, J. F., Malik, J. M., and Edwards, D. D. (1996).
Artificial intelligence: a modern approach.

Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Sanderson, D., Busquets, D., and Pitt, J. (2012).
A micro-meso-macro approach to intelligent transportation systems.
In *Proceedings of the 2012 IEEE Sixth International Conference on Self-Adaptive and Self-Organizing Systems Workshops*, SASOW '12, pages 77–82, Washington, DC, USA. IEEE Computer Society.

Scerri, D., Drogoul, A., Hickmott, S., and 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*, pages 541–548, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Shoham, Y. (1993).
Agent-oriented programming.
Artif. Intell., 60(1):51–92.

Siebert, J., Ciarletta, L., and 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, pages 509–516, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Suzumura, T. and Kanezashi, H. (2012a).
Highly scalable x10-based agent simulation platform and its application to large-scale traffic simulation.
In *Distributed Simulation and Real Time Applications (DS-RT), 2012 IEEE/ACM 16th International Symposium on*, pages 243–250.

Suzumura, T. and Kanezashi, H. (2012b).
Towards performance optimization of x10-based agent simulation platform with adaptive synchronization method.
In *Proceedings of the Winter Simulation Conference*, WSC '12, pages 352:1–352:2. Winter Simulation Conference.

Taillandier, P., Vo, D.-A., Amouroux, E., and Drogoul, A. (2010).
Gama: A simulation platform that integrates geographical information data, agent-based modeling and multi-scale control.

In *PRIMA*, volume 7057 of *Lecture Notes in Computer Science*, pages 242–258. Springer.

Tanenbaum, A. S. and van Steen, M. (2008).
Distributed Systems: Principles and Paradigms.
Prentice Hall International.

Timm, I. and Pawlaszczyk, D. (2005).
Large scale multiagent simulation on the grid.
Cluster Computing and the Grid, IEEE International Symposium on, 1:334–341.

Torrel, J.-c., Lattaud, C., and Heudin, J.-c. (2007).
Studying complex stellar dynamics using a hierarchical multi-agent model.
In *Modelling and Simulation in Science*, pages 307–312.

Vallee, M., Ramparany, F., and Vercouter, L. (2005).
A multi-agent system for dynamic service composition in ambient intelligence environments.
In *The 3rd International Conference on Pervasive Computing (PERVASIVE 2005)*, pages 175–182.

Weiss, G. (1999).
Multiagent systems: a modern approach to distributed artificial intelligence.
MIT Press, Cambridge, MA, USA.

Weyns, D., Helleboogh, A., and Holvoet, T. (2005).
The packet-world: A test bed for investigating situated multi-agent systems.
In *Software Agent-Based Applications, Platforms and Development Kits*, Whitestein Series in Software Agent Technologies, pages 383–408. Birkhäuser Basel.

Wilensky, U. (1997).
Netlogo wolf sheep predation model.

Wooldridge, M. (2009).
An Introduction to Multi-Agent Systems.
Wiley Publishing, 2nd edition.

Xiaoxia, S. and Qiuhai, Z. (2003).
The introduction on high level architecture (hla) and run-time infrastructure (rti).

In *SICE 2003 Annual Conference*, volume 1, pages 1136 –1139 Vol.1.

Yamamoto, G., Tai, H., and Mizuta, H. (2008).
A platform for massive agent-based simulation and its evaluation.
In *Massively Multi-Agent Technology*, volume 5043 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg.

Yoo, M.-J. and Glardon, R. (2009).
Combining jade and repast for the complex simulation of enterprise value-adding networks.
In *Agent-Oriented Software Engineering IX*, volume 5386 of *Lecture Notes in Computer Science*, pages 243–256. Springer Berlin Heidelberg.

Yurdusev, A. N. (1993).
'level of analysis' and 'unit of analysis': A case for distinction.
Millennium - Journal of International Studies, 22(1):77–88.